Tibero

tbPSM 참조 안내서

Tibero 7

대한민국 경기도 성남시 분당구 황새울로258번길 29, BS 타워 9층 우)13595

Website

http://www.tmaxtibero.com

기술서비스센터

Tel : +82-1544-8629

E-Mail : info@tmax.co.kr

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

Detailed Information related to the license can be found in the following directory : ${INSTALL_PATH}/license/oss_licenses

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다. : OpenSSL, RSA Data Security, Inc., Apache Foundation, Jean-loup Gailly and Mark Adler, Paul Hsieh's hash

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고해 주십시오. : ${INSTALL_PATH}/license/oss_licenses

안내서 정보

안내서 제목: Tibero tbPSM 참조 안내서

발행일: 2024-08-22

소프트웨어 버전: Tibero 7.2.2

안내서 버전: v7.2.2

내용 목차

예 목차

안내서에 대하여

안내서의 대상

본 안내서는 Tibero®(이하 Tibero)에서 제공하는 저장 프러시저 모듈 즉 tbPSM(Tibero 의 Persistent Stored Module) 패키지를 참조하려는 데이터베이스 관리자(Database Administrator, 이하 DBA), 애플리케이션 프로그램 개발자를 대상으로 기술한다.

안내서의 전제 조건

본 안내서를 원활히 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 데이터베이스의 이해
- RDBMS의 이해
- SQL의 이해
- tbPSM의 이해

안내서의 제한 조건

본 안내서는 Tibero를 실무에 적용하거나 운용하는 데 필요한 모든 사항을 포함하고 있지 않다. 따라서 설치, 환경설정 등 운용 및 관리에 대해서는 각 제품 안내서를 참고하기 바란다.

참고

Tibero의 설치 및 환경설정에 관한 내용은 "Tibero 설치 안내서"를 참고한다.

안내서 규약

표기	의미
<<AaBbCc123>>	프로그램 소스 코드의 파일명
<Ctrl>+C	Ctrl과 C를 동시에 누름
[Button]	GUI의 버튼 또는 메뉴 이름
진하게	강조
" "(따옴표)	다른 관련 안내서 또는 안내서 내의 다른 장 및 절 언급
'입력항목'	화면 UI에서 입력 항목에 대한 설명
하이퍼링크	메일 계정, 웹 사이트
>	메뉴의 진행 순서
+----	하위 디렉터리 또는 파일 있음
|----	하위 디렉터리 또는 파일 없음
참고	참고 또는 주의사항
주의	주의할 사항
[그림 1.1]	그림 이름
[예 1.1]	예제 이름
`AaBbCc123`	Java 코드, XML 문서
[*command argument*]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨
|	선택 사항. 예) A|B: A나 B 중 하나
…	파라미터 등이 반복되어서 나옴
${ }	환경변수

시스템 사용 환경

	요구 사항
Platform	HP-UX 11i v3(11.31)
	Solaris (Solaris 11)
	AIX (AIX 7.1/AIX 7.2/AIX 7.3)
	GNU (X86, 64, IA64)
	Red Hat Enterprise Linux 7 kernel 3.10.0 이상
	Windows(x86) 64bit
Hardware	최소 2.5GB 하드디스크 공간
	1GB 이상 메모리 공간
Compiler	PSM (C99 지원 필요)
	tbESQL/C (C99 지원 필요)

관련 안내서

안내서	설명
Tibero 설치 안내서	설치 과정에 필요한 시스템 요구사항과 설치 및 제거 방법을 기술한 안내서이다.
Tibero tbCLI 안내서	Call Level Interface인 tbCLI의 개념과 구성요소, 프로그램 구조를 소개하고 tbCLI 프로그램을 작성하는 데 필요한 데이터 타입, 함수, 에러 메시지를 기술한 안내서이다.
Tibero 애플리케이션 개발자 안내서	각종 애플리케이션 라이브러리를 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero External Procedure 안내서	External Procedure를 소개하고 이를 생성하고 사용하는 방법을 기술한 안내서이다.
Tibero JDBC 개발자 안내서	Tibero에서 제공하는 JDBC 기능을 이용하여 애플리케이션 프로그램을 개발하는 방법을 기술한 안내서이다.
Tibero tbESQL/C 안내서	C 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbESQL/COBOL 안내서	COBOL 프로그래밍 언어를 사용해 데이터베이스 작업을 수행하는 각종 애플리케이션 프로그램을 작성하는 방법을 기술한 안내서이다.
Tibero tbPSM 안내서	저장 프러시저 모듈인 tbPSM의 개념과 문법, 구성요소를 소개하고, 프로그램을 작성하는 데 필요한 제어 구조, 복합 타입, 서브 프로그램, 패키지와 SQL 문장을 실행하고 에러를 처리하는 방법을 기술한 안내서이다.
Tibero 관리자 안내서	Tibero의 동작과 주요 기능의 원활한 수행을 보장하기 위해 DBA가 알아야 할 관리 방법을 논리적 또는 물리적 측면에서 설명하고, 관리를 지원하는 각종 도구를 기술한 안내서이다.
Tibero 유틸리티 안내서	데이터베이스와 관련된 작업을 수행하기 위해 필요한 유틸리티의 설치 및 환경설정, 사용 방법을 기술한 안내서이다.
Tibero TAS 안내서	Tibero Active Storage(TAS)를 사용해서 Tibero 의 파일을 관리하고자 하는 관리자를 대상으로 기술한 안내서이다.
Tibero	Tibero를 사용하는 도중에 발생할 수 있는 각종 에러의 원인과 해결 방법을 기술한 안내서이다.

안내서	설명
에러 참조 안내서	
Tibero 참조 안내서	Tibero의 동작과 사용에 필요한 초기화 파라미터와 데이터 사전, 정적 뷰, 동적 뷰를 기술한 참조 안내서이다.
Tibero SQL 참조 안내서	데이터베이스 작업을 수행하거나 애플리케이션 프로그램을 작성할 때 필요한 SQL 문장을 기술한 참조 안내서이다.
Tibero Spatial 참조 안내서	Tibero에서 Geometry 타입에 대한 설명과 Spatial 기능 관련 프러시저 함수 목록 및 사용 방법 등을 기술한 안내서이다.
Tibero TEXT 참조 안내서	Tibero의 제공하는 Text Index를 소개하고, Text Index를 생성 하고 사용하는 방법을 기술하는 안내서이다.
Tibero TDP.NET 안내서	Tibero Data Provider for .NET 기능을 기술하는 안내서이다.
Tibero IMCS 안내서	Tibero에서 제공하는 In-Memory Column Store(이하 IMCS) 기능을 기술하는 안내서이다.

제1장 패키지 소개

본 장에서는 패키지의 기본 개념과 구성을 소개한다.

1.1. 개요

패키지(package)는 하나 이상의 tbPSM 프러시저(procedure)와 함수(function) 등을 포함하는 스키마 객체이다. 하나의 패키지에 포함되는 스키마 객체는 프러시저와 함수를 비롯하여 변수, 커서(cursor), 예외 상황(exception) 등이 있다.

패키지는 프러시저, 함수와 마찬가지로 임의의 SQL 문장에서 사용할 수 있다.

패키지를 사용함으로써 얻을 수 있는 장점은 다음과 같다.

- 연관된 프러시저와 함수의 개발과 관리가 쉽다.

  예를 들어 여러 프러시저와 함수에서 공통으로 액세스하는 변수를 패키지에 포함시킬 수 있다.

- 애플리케이션 프로그램의 개발이 쉽다.

  패키지에서 지원하는 여러 기능을 이용하면 좀 더 강력한 애플리케이션 프로그램을 작성할 수 있다.

- 효율적인 실행이 가능하다.

  권한의 부여와 회수, 메모리 로딩(memory loading) 등이 패키지 단위로 수행되기 때문에 효율적인 실행이 가능하다.

1.2. 구성

패키지는 **사용자 정의 패키지**와 Tibero에서 제공하는 **시스템 패키지**로 구성된다.

1.2.1. 사용자 정의 패키지

사용자 정의 패키지는 사용자가 원하는 목적에 맞춰 정의하는 패키지이다. 사용자 정의 패키지는 프러시저, 함수와 유사하게 다음과 같이 정의할 수 있다.

패키지 선언부

외부 인터페이스이며 공개 프러시저(public procedure)와 함수, 변수, 상수 등을 포함한다.

패키지 선언부의 세부 내용은 다음과 같다.

- 문법

```
CREATE [OR REPLACE] PACKAGE package_name [invoker_rights] AS
    -- 공개 프러시저의 선언
    -- 공개 함수의 선언
    -- 공개 변수, 상수, 예외 상황의 선언
END [package_name];
```

- 특징
  - CREATE PACKAGE 문장을 이용하여 선언부를 생성한다.
  - 패키지 이름(package_name) 다음에 패키지를 호출자 권한(invoker's rights) 또는 정의자 권한(definer's rights)으로 실행할 것인지를 설정할 수 있다. 설정이 생략되면 정의자 권한으로 실행된다.
  - END 다음에 패키지 이름은 생략해도 상관없다.

- 사용 예제

[예 1.1] EMP_MGMT 패키지의 선언부

```
CREATE PACKAGE EMP_MGMT AS   ... EMP_MGMT 패키지의 선언부 ...

    FUNCTION HIRE(ENAME VARCHAR, ADDR VARCHAR, SALARY NUMBER,
                  DEPTNO INT) RETURN INT;
    PROCEDURE FIRE(EMPNO INT);
    PROCEDURE RAISE_SALARY(EMPNO INT, AMOUNT NUMBER);
    EMP_NOT_FOUND EXCEPTION;

END EMP_MGMT;
```

위의 예에서는 EMP_MGMT 패키지를 정의자 권한으로 실행하며, 공개 프러시저(2개)와 함수(1개) 그리고 공개 예외 상황(1개)를 포함하고 있다.

패키지 구현부

실제 프로그램이 구현되는 부분이다. 공개 프러시저와 함수 등의 프로그램 구현부와 함께 비공개 프러시저(private procedure)와 함수, 변수 등을 포함한다.

패키지 구현부의 세부 내용은 다음과 같다.

- 문법

```
CREATE [OR REPLACE] PACKAGE BODY package_name AS
    -- 공개 프러시저의 구현부
```

```
    -- 공개 함수의 구현부
    -- 비공개 프러시저 및 함수의 선언부
    -- 비공개 프러시저 및 함수의 구현부
    -- 비공개 변수, 상수, 예외 상황의 정의
[BEGIN
    -- 초기화 명령어의 집합부
END [package_name];
```

- 특징
  - CREATE PACKAGE BODY 문장을 이용하여 구현부를 생성한다.
  - END 다음에 패키지 이름은 생략해도 상관없다.
- 사용 예제

[예 1.2] EMP_MGMT 패키지의 구현부

```
CREATE PACKAGE BODY EMP_MGMT AS  ... EMP_MGMT 패키지의 구현부 ...

    NUM_EMP INT;
    EMPNO NUMBER := 0;

    FUNCTION HIRE(ENAME VARCHAR, ADDR VARCHAR, SALARY NUMBER,
                  DEPTNO INT) RETURN INT AS
    BEGIN
        EMPNO := EMPNO + 1;       ... 새로운 EMPNO 생성 ...
        INSERT INTO EMP VALUES (EMPNO, ENAME, ADDR, SALARY, DEPTNO);
        NUM_EMP := NUM_EMP + 1;
        RETURN(EMPNO);
    END;

    PROCEDURE FIRE(EMPNO INT) AS
    BEGIN
        DELETE FROM EMP WHERE EMP.EMPNO = FIRE.EMPNO;
        NUM_EMP := NUM_EMP ? 1;
    END;

    PROCEDURE RAISE_SALARY(EMPNO INT, AMOUNT NUMBER) AS
    BEGIN
        UPDATE EMP SET SALARY = SALARY + AMOUNT
        WHERE EMP.EMPNO = RAISE_SALARY.EMPNO;
    END;

END EMP_MGMT;
```

위의 예에서는 [예 1.1]에 포함된 모든 프러시저와 함수에 대한 정의를 포함하고 있으며, 비공개 변수인 NUM_EMP를 선언하고 있다.

패키지 프러시저와 함수를 정의하는 방법은 패키지 변수, 상수, 예외 상황에 대한 내용 외에는 일반 프러시저 및 함수를 정의하는 방법과 동일하다.

프러시저 또는 함수 내부의 컬럼의 이름과 변수의 이름이 같은 경우 컬럼이 포함된 테이블의 이름이나 프러시저 또는 함수의 이름을 지정하여 혼동을 피할 수 있다. 예를 들어 프러시저 FIRE 함수의 내부에서 테이블 EMP에 포함된 컬럼 EMPNO와 프러시저 FIRE로 전달된 파라미터 변수 EMPNO가 같은 이름을 가지므로, 각각 테이블 이름과 프러시저 이름을 지정해 주었다.

패키지 내에 포함된 프러시저나 함수를 그 패키지의 소유자 이외의 다른 사용자가 호출하도록 할 수 있다. 프러시저나 함수를 호출하려면 해당 패키지의 EXECUTE 권한을 부여 받아야 한다. 해당 패키지의 소유자는 항상 해당 패키지를 호출할 수 있다.

뷰(view)가 패키지 프러시저나 함수를 이용하여 정의된 경우에는 뷰를 접근하기 위해 SELECT 권한을 부여 받아야 한다. 이때 해당 패키지의 EXECUTE 권한은 필요하지 않다. 패키지 프러시저나 함수를 참조하려면 패키지의 이름과 프러시저 또는 함수의 이름을 함께 써주어야 한다.

다음은 [예 1.1]에서 정의한 EMP_MGMT 패키지 내의 HIRE 함수를 호출하는 예이다. 이때 반환값은 EMPNO 변수에 저장된다.

```
DECLARE
    empno NUMBER;
BEGIN
    empno := EMP_MGMT.HIRE('John', 'New York', 45000, 5);
    DBMS_OUTPUT.PUT_LINE('Hired employee No. is ' || empno);
END;
```

1.2.2. 시스템 패키지

시스템 패키지는 Tibero에서 기본적으로 제공하는 패키지 라이브러리로, SYS 사용자의 소유로 정의된다. 또한, Tibero를 설치한 후에 일반 사용자도 사용할 수 있도록 시스템 패키지와 동일한 이름의 공용 동의어(Public Synonym)가 정의된다. 그리고 일반 사용자가 시스템 패키지의 프러시저 또는 함수를 호출하면 호출자 권한으로 실행된다.

시스템 패키지는 일반 사용자에게 여러 가지의 확장된 기능을 제공한다. 시스템 패키지를 SQL 문장에서 그대로 사용한다거나, 다른 프러시저와 함수 그리고 패키지를 정의할 때도 사용할 수 있다.

시스템 패키지의 프러시저와 함수를 호출하는 방법은 사용자 정의 패키지와 같다.

다음은 32KB 이상인 대용량 객체의 이름과 생성자를 출력하는 예이다.

```
SELECT name, creator FROM image
WHERE DBMS_LOB.GETLENGTH(data) >= 32768;
```

위 예에서 보듯이 data는 image 테이블에서 대용량 객체형으로 정의된 컬럼이다.

다음은 Tibero에서 제공하는 시스템 패키지이다.

시스템 패키지	설명
ANYDATA	임의의 타입을 담을 수 있는 타입이다.
DBMS_ALERT	ALERT은 Tibero 데이터베이스와 통합되어 있는 알림/기다림 기능을 하는 패키지이다.
DBMS_APPLICATION_INFO	애플리케이션 정보에 관련된 뷰의 값을 변경하기 위한 패키지이다.
DBMS_AQ	Advanced Queuing 기능과 관련한 프러시저와 함수를 제공하는 패키지이다.
DBMS_AQADM	Advanced Queuing 기능을 구성하고 관리하는 데 사용할 수 있는 프러시저와 함수를 제공한다.
DBMS_BACKUP_RESTORE	데이터베이스의 백업과 복구를 위해 사용되는 패키지이다.
DBMS_CRYPTO	DES, DES3, AES 알고리즘 및 체인, 패딩 기법을 이용한 확장된 데이터 암호화 및 복호화 패키지이다.
DBMS_DB_VERSION	제품의 버전 정보를 확인하는 패키지이다.
DBMS_DDL	CREATE OR REPLACE PROCEDURE, PACKAGE로 시작하는 DDL 구문에 대한 WRAPPING 기능 외 기타 DDL 관련 기능을 제공하는 패키지이다.
DBMS_DEBUG	두 개의 세션을 이용하여 PSM 프로그램의 디버깅을 할 수 있는 패키지이다.
DBMS_ERRLOG	DML의 error logging 기능을 지원하는 패키지이다.
DBMS_FGA	FGA(Fine-Grained Auditing) policy를 관리하는 패키지이다.
DBMS_FLASHBACK	Flashback 기능을 지원하는 패키지이다.
DBMS_INMEMORY	In-Memory Column Store 기능을 지원하는 패키지이다.
DBMS_JAVA	데이터베이스에서 사용하는 Java 객체에 접근하기 위한 패키지이다.
DBMS_JOB	JOB을 관리하기 위한 패키지이다.
DBMS_JOB_WITH_NAME	JOB을 name을 통해 관리하기 위한 패키지이다.
DBMS_LOB	BLOB, CLOB 타입의 대용량 데이터를 처리하기 위한 패키지이다.
DBMS_LOCK	세션을 대기시키는 패키지이다.
DBMS_METADATA	DB Object에 metadata 정보를 조회하여 생성 script를 얻을 수 있는 함수를 제공한다.
DBMS_MONITOR	인스턴스 단위, 세션 단위, 클라이언트 식별자 단위로 SQL 수행을 모니터링하기 위한 패키지이다.

시스템 패키지	설명
DBMS_MVIEW	실체화 뷰(Materialized View)와 관련된 정보를 제공하고 이 정보를 최근의 것으로 변경할 수 있는 REFRESH 기능을 사용하기 위한 패키지이다.
DBMS_OBFUSCATION_TOOLKIT	DES, DES3 알고리즘을 이용한 데이터 암호화 및 복호화 패키지이다.
DBMS_OUTPUT	메시지 버퍼에 메시지를 저장하고 읽기 위한 패키지이다.
DBMS_PARALLEL_EXECUTE	병렬 방식으로 테이블을 업데이트할 수 있는 기능을 제공하는 패키지이다.
DBMS_PIPE	동일 인스턴스에 속해 있는 세션들 간의 통신을 지원하는 패키지이다.
DBMS_RANDOM	임의의 숫자 및 문자열을 생성하는 패키지이다.
DBMS_REDACT	Data redaction을 위한 interface를 제공하는 패키지이다.
DBMS_REDEFINITION	온라인 상태(on-line state)에서 특정 테이블을 재정의하기 위한 패키지이다.
DBMS_REPAIR	테이블과 인덱스 내의 깨진 블록을 검사하고 복구하는 기능을 제공하는 패키지이다.
DBMS_RESOURCE_MANAGER	업무의 성격에 따라 그룹을 나누어 자원을 분배할 수 있다.
DBMS_RESULT_CACHE	Result Cache를 관리하는 패키지이다.
DBMS_RLS	가상 개인 데이터베이스 기능을 관리하는 패키지이다.
DBMS_ROWID	ROWID에 담긴 정보를 보거나 생성하기 위한 패키지이다.
DBMS_SCHEDULER	PSM으로 작성된 프로그램을 스케줄링하고 관리하기 위한 패키지이다.
DBMS_SESSION	세션 식별자를 관리하는 패키지이다.
DBMS_SPACE	세그먼트(segment)의 크기와 공간 사용에 대한 정보를 제공하는 패키지이다.
DBMS_SPACE_ADMIN	세그먼트를 관리하는 기능을 제공하는 패키지이다.
DBMS_SPH	SQL Plan History(이하 SPH) 기능을 사용하는데 필요로 하는 기능을 제공하기 위한 패키지이다.
DBMS_SPM	SQL Plan Baseline 기능을 사용하는데 필요로 하는 기능을 제공하기 위한 패키지이다.
DBMS_SQL	데이터베이스에 접근하는 Dynamic SQL을 사용하기 위한 패키지이다.
DBMS_SQLTUNE	실시간 SQL 모니터링 기능에 대한 보고서를 생성해 주는 함수를 제공하고 있는 패키지이다.
DBMS_SQL_TRANSLATOR	SQL 번역 프로파일을 관리 및 사용하는 패키지이다.

시스템 패키지	설명
DBMS_STATS	데이터베이스 객체에 대한 통계 정보를 관리하기 위한 패키지이다.
DBMS_SYSTEM	SYS 권한을 가진 관리자가 시스템적인 제어를 위해 사용하는 패키지이다.
DBMS_TPR	Tibero Performance Repository(이하 TPR) 기능을 사용하는데 필요로 하는 기능을 제공하기 위한 패키지이다.
DBMS_TRANSACTION	트랜잭션(Transaction) 문장을 실행하고 트랜잭션을 관리하기 위한 패키지이다.
DBMS_TYPES	데이터 타입들을 숫자로 정의한 패키지이다.
DBMS_UTILITY	여러 가지 유용한 기능들을 제공하는 패키지이다.
DBMS_VERIFY	여러 요소들에 대하여 정합성을 검사하는 기능을 제공한다.
DBMS_XMLDOM	XML 문서 처리를 위한 DOM API를 제공한다.
DBMS_XMLGEN	쿼리를 입력으로 받고 해당 쿼리의 결과 집합을 XML형식으로 생성해주는 패키지이다.
DBMS_XMLPARSER	XML 문서를 파싱하여 XML 문서의 구조 및 내용에 접근할 수 있는 API를 제공한다.
DBMS_XMLSAVE	XML-to-database-type 기능을 제공하는 패키지이다.
DBMS_XPLAN	SQL의 플랜 정보와 수행 정보를 조회하는 기능을 제공하는 패키지이다.
DBMS_XSLPROCESSOR	XML 문서의 구성과 내용을 관리할 수 있는 인터페이스를 제공하는 패키지이다.
HTF	HTML 태그를 생성하는 패키지이다.
HTP	HTML 태그를 생성하는 패키지이다. Apache 웹 서버 모듈로 개발된 mod_tbpsm과 연동되어 프러시저를 호출할 때 HTML 페이지를 생성하여 내보낸다.
OWA_UTIL	웹 에이전트에서 주로 사용하는 서브 프로그램을 담고 있는 패키지이다.
SA_COMPONENTS	Label Security 기능의 레이블들의 정의를 관리하는 패키지이다.
SA_LABEL_ADMIN	Label Security 정책의 레이블들을 관리하는 패키지이다.
SA_POLICY_ADMIN	Label Security 정책들을 관리할 수 있는 패키지이다.
SA_SYSDBA	Label Security 정책들을 생성, 변경, 작동 제어, 제거하는 패키지이다.
SA_USER_ADMIN	Label Security 기능의 사용자 레이블들을 관리하는 패키지이다.
TEXT_DDL	TEXT INDEX를 사용할 때 TEXT INDEX의 설정들을 변경하기 위한 패키지이다.
UTL_COMPRESS	데이터를 압축/압축 해제하기 위한 패키지이다.

시스템 패키지	설명
UTL_ENCODE	호스트 간의 데이터를 전송할 수 있도록 표준 인코딩 기술로 인코딩하기 위한 패키지이다.
UTL_FILE	운영체제에서 관리하는 파일에 접근하기 위한 패키지이다.
UTL_HTTP	웹 표준(RFC2616)에 따라 웹 페이지(Web page)의 요청을 처리하기 위한 패키지이다.
UTL_I18N	언어, 국가별 설정들 간의 호환 기능을 제공하는 패키지이다.
UTL_MATCH	두 문자열의 유사도를 계산하는 함수들을 제공하는 패키지이다.
UTL_RAW	RAW 타입의 데이터를 처리하기 위한 패키지이다.
UTL_RECOMP	Invalid 상태의 오브젝트를 다시 컴파일하기 위한 패키지이다.
UTL_SMTP	SMTP(Simple Mail Transfer Protocol)를 이용하여 전자 메일을 전송하기 위한 패키지이다.
UTL_TCP	tbPSM을 이용한 원격 TCP 서버와의 TCP/IP 소켓 통신을 지원하는 패키지이다.
UTL_URL	URL(Uniform Resource Locator) 주소를 ESCAPE 형태로 변환하기 위한 패키지이다.

참고

위의 패키지에 대한 자세한 내용은 해당 패키지를 설명하는 각 장을 참고한다.

제2장 ANYDATA

본 장에서는 ANYDATA 타입의 기본 개념과 메소드에 대하여 기술한다.

2.1. 개요

ANYDATA 타입은 임의의 타입을 담을 수 있는 타입이다.

2.2. 메소드

2.2.1. CONVERT

임의의 타입을 입력으로 받아, ANYDATA 타입의 인스턴스를 생성한다.

CONVERT 메소드 시리즈의 세부 내용은 다음과 같다.

- 프로토타입

```
STATIC FUNCTION ConvertNumber(num IN NUMBER)
       return AnyData;
STATIC FUNCTION ConvertDate(dat IN DATE)
       return AnyData;
STATIC FUNCTION ConvertChar(c IN CHAR)
       return AnyData;
STATIC FUNCTION ConvertVarchar(c IN VARCHAR)
       return AnyData;
STATIC FUNCTION ConvertVarchar2(c IN VARCHAR2)
       return AnyData;
STATIC FUNCTION ConvertRaw(r IN RAW)
       return AnyData;
STATIC FUNCTION ConvertBlob(b IN BLOB)
       return AnyData;
STATIC FUNCTION ConvertClob(c IN CLOB)
       return AnyData;
STATIC FUNCTION ConvertTimestamp(ts IN TIMESTAMP_UNCONSTRAINED)
       return AnyData;
STATIC FUNCTION ConvertTimestampTZ(ts IN TIMESTAMP_TZ_UNCONSTRAINED)
       return AnyData;
STATIC FUNCTION ConvertTimestampLTZ(ts IN TIMESTAMP_LTZ_UNCONSTRAINED)
```

```
       return AnyData;
STATIC FUNCTION ConvertIntervalYM(inv IN YMINTERVAL_UNCONSTRAINED)
       return AnyData;
STATIC FUNCTION ConvertIntervalDS(inv IN DSINTERVAL_UNCONSTRAINED)
       return AnyData;
STATIC FUNCTION ConvertNchar(nc IN NCHAR)
       return AnyData;
STATIC FUNCTION ConvertNVarchar2(nc IN NVARCHAR2)
       return AnyData;
STATIC FUNCTION ConvertNClob(nc IN NCLOB)
       return AnyData;
STATIC FUNCTION ConvertBFloat(fl IN BINARY_FLOAT)
       return AnyData;
STATIC FUNCTION ConvertBDouble(dbl IN BINARY_DOUBLE)
       return AnyData;
STATIC FUNCTION ConvertObject(obj IN "<object>")
       return AnyData;
STATIC FUNCTION ConvertCollection(col IN "<collection>")
       return AnyData;
```

- 파라미터

파라미터	설명
num, da, c, r, b, ts, inv, nc, fl, dbl, obj, col	ANYDATA 타입으로 변환할 임의의 타입의 입력값이다. – obj는 사용자가 정의한 임의의 object 타입이 가능하다. – col은 ddl로 생성한 전역 콜렉션 타입만 가능하다.

- 예제

```
declare
    var sys.anydata;
    nvar number := 10;
begin
    var := sys.anydata.convertNumber(nvar);
end;
/


create or replace type SIMPLE_OBJ is object ( c1 number, c2 number);
/

declare
     v1 SIMPLE_OBJ;
     result sys.anydata;
begin
```

```
     v1 := SIMPLE_OBJ (10, 20);

     result := sys.anydata.convertobject(v1);
end;
/

create or replace type SIMPLE_COLL  as table of varchar(128);
/

declare
   any1 sys.anydata;
   any2 sys.anydata;

   type LOCAL_COLL is table of varchar(128);
   var1 SIMPLE_COLL ;
   var2 LOCAL_COLL;

   res1 SIMPLE_COLL ;
   status pls_integer;
begin
    var1 := SIMPLE_COLL ('a', 'b');
    any1 := sys.anydata.convertCollection(var1);

    begin
        var2 := LOCAL_COLL('a', 'b');
        any2 := sys.anydata.convertCollection(var2);
    exception when dbms_types.incorrect_usage then
        dbms_output.put_line('Anydata does not support local collection type');
    end;

end;
/
```

2.2.2. GET

ANYDATA 타입 인스턴스를 임의의 타입으로 변환한다. 리턴값은 성공했을 때 DBMS_TYPES.SUCCESS를 리턴한다.

GET 메소드 시리즈의 세부 내용은 다음과 같다.

- 프로토타입

```
MEMBER FUNCTION GetNumber(self IN AnyData, num OUT NOCOPY NUMBER)
       return PLS_INTEGER;
MEMBER FUNCTION GetDate(self IN AnyData, dat OUT NOCOPY DATE)
```

```
       return PLS_INTEGER;
MEMBER FUNCTION GetChar(self IN AnyData, c OUT NOCOPY CHAR)
       return PLS_INTEGER;
MEMBER FUNCTION GetVarchar(self IN AnyData, c OUT NOCOPY VARCHAR)
       return PLS_INTEGER;
MEMBER FUNCTION GetVarchar2(self IN AnyData, c OUT NOCOPY VARCHAR2)
       return PLS_INTEGER;
MEMBER FUNCTION GetRaw(self IN AnyData, r OUT NOCOPY RAW)
       return PLS_INTEGER;
MEMBER FUNCTION GetBlob(self IN AnyData, b OUT NOCOPY BLOB)
       return PLS_INTEGER;
MEMBER FUNCTION GetClob(self IN AnyData, c OUT NOCOPY CLOB)
       return PLS_INTEGER;
MEMBER FUNCTION GetTimestamp(self IN AnyData,
       ts OUT NOCOPY TIMESTAMP_UNCONSTRAINED) return PLS_INTEGER;
MEMBER FUNCTION GetTimestampTZ(self IN AnyData,
       ts OUT NOCOPY TIMESTAMP_TZ_UNCONSTRAINED) return PLS_INTEGER;
MEMBER FUNCTION GetTimestampLTZ(self IN AnyData,
       ts OUT NOCOPY TIMESTAMP_LTZ_UNCONSTRAINED) return PLS_INTEGER;
MEMBER FUNCTION GetIntervalYM(self IN AnyData,
       inv IN OUT NOCOPY YMINTERVAL_UNCONSTRAINED) return PLS_INTEGER;
MEMBER FUNCTION GetIntervalDS(self IN AnyData,
       inv IN OUT NOCOPY DSINTERVAL_UNCONSTRAINED) return PLS_INTEGER;
MEMBER FUNCTION GetNchar(self IN AnyData, nc OUT NOCOPY NCHAR)
       return PLS_INTEGER;
MEMBER FUNCTION GetNVarchar2(self IN AnyData, nc OUT NOCOPY NVARCHAR2)
       return PLS_INTEGER;
MEMBER FUNCTION GetNClob(self IN AnyData, nc OUT NOCOPY NCLOB)
       return PLS_INTEGER;
MEMBER FUNCTION GetBFloat(self IN AnyData, fl OUT NOCOPY BINARY_FLOAT)
       return PLS_INTEGER;
MEMBER FUNCTION GetBDouble(self IN AnyData, dbl OUT NOCOPY BINARY_DOUBLE)
       return PLS_INTEGER;
MEMBER FUNCTION GetCollection(self IN AnyData, obj OUT NOCOPY "<collection>")
       return PLS_INTEGER;
MEMBER FUNCTION GetObject(self IN AnyData, obj OUT NOCOPY "<object>")
       return PLS_INTEGER;
```

- 파라미터

파라미터	설명
self	ANYDATA 타입 인스턴스 자신이다. 자신의 값은 변경되지 않는다.
num, da, c, r, b, ts, inv, nc, fl, dbl, obj, col	ANYDATA 타입에서 꺼내올 변수이다.

- 예외 상황

예외 상황	설명
DBMS_TYPES.TYPE_MISMATCH	생성했던 타입과 다른 타입으로 GET*을 호출하는 경우 발생한다.

- 예제

```
declare
    var sys.anydata;
    nvar number := 10;
    rvar number;
    status pls_integer;
begin
    var := sys.anydata.convertNumber(nvar);
    status := var.getnumber(rvar);
    dbms_output.put_line( rvar );
end;
/

declare
     v1 SIMPLE_OBJ;
     v2 SIMPLE_OBJ;
     result sys.anydata;
     flags pls_integer;
begin
     v1 := SIMPLE_OBJ (10, 20);

     result := sys.anydata.convertobject(v1);

     flags := result.getobject(v2);

     dbms_output.put_line (v2.c1);
     dbms_output.put_line (v2.c2);
end;
/

declare
   any1 sys.anydata;

   var1 SIMPLE_COLL ;
   res1 SIMPLE_COLL ;
   status pls_integer;
begin
   var1 := SIMPLE_COLL ('a', 'b');
   any1 := sys.anydata.convertCollection(var1);
   status := any1.getcollection(res1);
   dbms_output.put_line(res1(1));
   dbms_output.put_line(res1(2));
```

```
end;
/
```

2.2.3. ACCESS

ANYDATA 타입 인스턴스를 임의의 타입으로 변환한다. 예외가 발생하지 않으며, 잘못된 타입으로 연산한 경우 GET 시리즈와 달리 NULL을 리턴한다.

ACCESS 메소드 시리즈의 세부 내용은 다음과 같다.

- 프로토타입

```
MEMBER FUNCTION AccessNumber(self IN AnyData)
       return NUMBER DETERMINISTIC;
MEMBER FUNCTION AccessDate(self IN AnyData)
       return DATE DETERMINISTIC;
MEMBER FUNCTION AccessChar(self IN AnyData)
       return CHAR DETERMINISTIC;
MEMBER FUNCTION AccessVarchar(self IN AnyData)
       return VARCHAR DETERMINISTIC;
MEMBER FUNCTION AccessVarchar2(self IN AnyData)
       return VARCHAR2 DETERMINISTIC;
MEMBER FUNCTION AccessRaw(self IN AnyData)
       return RAW DETERMINISTIC;
MEMBER FUNCTION AccessBlob(self IN AnyData)
       return BLOB DETERMINISTIC;
MEMBER FUNCTION AccessClob(self IN AnyData)
       return CLOB DETERMINISTIC;
MEMBER FUNCTION AccessTimestamp(self IN AnyData)
       return TIMESTAMP_UNCONSTRAINED DETERMINISTIC;
MEMBER FUNCTION AccessTimestampTZ(self IN AnyData)
       return TIMESTAMP_TZ_UNCONSTRAINED DETERMINISTIC;
MEMBER FUNCTION AccessTimestampLTZ(self IN AnyData)
       return TIMESTAMP_LTZ_UNCONSTRAINED DETERMINISTIC;
MEMBER FUNCTION AccessIntervalYM(self IN AnyData)
       return YMINTERVAL_UNCONSTRAINED DETERMINISTIC;
MEMBER FUNCTION AccessIntervalDS(self IN AnyData)
       return DSINTERVAL_UNCONSTRAINED DETERMINISTIC;
MEMBER FUNCTION AccessNchar(self IN AnyData)
       return NCHAR DETERMINISTIC;
MEMBER FUNCTION AccessNVarchar2(self IN AnyData)
       return NVARCHAR2 DETERMINISTIC;
MEMBER FUNCTION AccessBFloat(self IN AnyData)
       return BINARY_FLOAT DETERMINISTIC;
```

```
MEMBER FUNCTION AccessBDouble(self IN AnyData)
       return BINARY_DOUBLE DETERMINISTIC;
```

- 파라미터

파라미터	설명
self	ANYDATA 타입 인스턴스 자신이다. 자신의 값은 변경되지 않는다.

- 예제

```
declare
    var sys.anydata;
    nvar number := 10;
    rvar number;
    status pls_integer;
begin
    var := sys.anydata.convertNumber(nvar);
    dbms_output.put_line( var.accessnumber );
    dbms_output.put_line( var.accessvarchar ); -- null 출력

end;
/
```

2.2.4. GETTYPENAME

ANYDATA 타입에서 타입이름을 반환한다.

GETTYPENAME 메소드의 세부 내용은 다음과 같다.

- 프로토타입

```
MEMBER FUNCTION GetTypeName(self IN AnyData) return VARCHAR;
```

- 파라미터

파라미터	설명
self	ANYDATA 타입 인스턴스 자신이다. 자신의 값은 변경되지 않는다.

- 예제

```
 create or replace type SIMPLE_COLL  as table of varchar(128);
/

create or replace type SIMPLE_OBJ is object ( c1 number, c2 number);
/
```

```
declare
    var1 sys.anydata;
    var2 sys.anydata;
    var3 sys.anydata;
    num number := 10;
    obj SIMPLE_OBJ;
    coll SIMPLE_COLL;
begin
    obj := SIMPLE_OBJ (10, 20);
    coll := SIMPLE_COLL('a', 'b');
    var1 := sys.anydata.convertNumber(num);
    var2 := sys.anydata.convertobject(obj);
    var3 := sys.anydata.convertCollection(coll);
    dbms_output.put_line(var1.gettypename );
    dbms_output.put_line(var2.gettypename );
    dbms_output.put_line(var3.gettypename );
end;
/
```

제3장 DBMS_ALERT

본 장에서는 DBMS_ALERT 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

3.1. 개요

DBMS_ALERT 패키지는 Tibero의 ALERT 기능과 관련한 프러시저와 함수를 제공한다. Tibero의 ALERT은 Tibero 데이터베이스와 통합되어 있는 알림/기다림 기능을 하는 패키지이다. 이는 영속적이지 않으며 서버가 재가동되면 모든 알림/기다림 정보는 사라진다.

ALERT기능은 보통 다음과 같이 사용한다.

1. 기다릴 alert을 등록한다. (register)

2. 기다린다. (waitone, waitany)

3. 필요한 이벤트가 발생하면 다른 세션에서 alert을 발생시킨다. (signal)

1번에서 등록한 alert이나 3번에서 발생시킨 alert signal은 서버를 재가동할 때 사라진다.

다음은 DBMS_ALERT 패키지 내에 정의된 상수이다.

```
MAXWAIT CONSTANT INTEGER := 86400000
```

3.2. 프러시저

본 절에서는 DBMS_ALERT 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

3.2.1. REGISTER

기다릴 ALERT을 등록하는 프러시저이다. 등록된 ALERT은 REGISTER 프러시저를 수행한 세션에서만 유효하다. 중복해서 등록할 필요는 없다(무시됨). 다른 세션에서 ALERT을 기다리려면 새로 등록해야 한다.

REGISTER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ALERT.REGISTER
(
    name         IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
name	등록/기다릴 ALERT의 이름이다.

- 예제

```
BEGIN
 DBMS_ALERT.REGISTER('ABC');
END;
/
```

3.2.2. REMOVE

등록된 ALERT을 제거하는 프러시저이다. 이 프러시저는 프러시저를 수행하는 세션에만 영향을 준다. 즉, 다른 세션에서 등록한 동일 이름에 대해서는 영향을 주지 않는다. 등록되지 않은 ALERT을 REMOVE하려고 하면 에러가 발생한다.

REMOVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ALERT.REMOVE
(
    name         IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
name	제거할 ALERT의 이름이다.

- 예제

```
BEGIN
 DBMS_ALERT.REMOVE('ABC');
END;
/
```

3.2.3. REMOVEALL

등록된 모든 ALERT을 제거하는 프러시저이다. 이 프러시저는 프러시저를 수행하는 세션에만 영향을 준다. 즉, 다른 세션에서 등록한 동일 이름에 대해서는 영향을 주지 않는다.

REMOVEALL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ALERT.REMOVEALL
(
);
```

- 예제

```
BEGIN
  DBMS_ALERT.REMOVEALL();
END;
/
```

3.2.4. SET_DEFAULTS

사용하지 않는 프러시저이다.

3.2.5. SIGNAL

ALERT을 발생시켜 이를 기다리고 있는 세션들을 깨워주는 프러시저이다.

SIGNAL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ALERT.SIGNAL
(
    name         IN   VARCHAR2,
    message      IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
name	발생시킬 ALERT의 이름이다.
message	함께 전달할 메시지이다. 최대 128bytes까지 전송된다.

- 예제

```
BEGIN
  DBMS_ALERT.SIGNAL('ABC', 'DEF');
END;
/
```

3.2.6. WAITANY

어떠한 종류든 내가 등록한 ALERT을 기다리는 프러시저이다. 세션에서 이전에 등록했던 ALERT이 발생한 경우 깨어난다. 또한 타임아웃이 지나면 깨어날 수 있다. 특정 ALERT을 받아서 깨어나도 해당 ALERT이 자동으로 REMOVE되지 않는다.

WAITANY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ALERT.WAITANY
(
    name         OUT   VARCHAR2,
    message      OUT   VARCHAR2,
    status       OUT   VARCHAR2,
    timeout      IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
name	WAITANY 프러시저가 깨어나도록 한 ALERT의 이름이다. 타임아웃으로 깨어난 경우 값이 없다.
message	WAITANY 프러시저가 깨어나도록 한 ALERT과 함께 전달된 메시지이다. 타임아웃으로 깨어난 경우 값이 없다.
status	– 1 : 타임아웃으로 깨어난 경우 – 0 : ALERT이 발생해서 깨어난 경우
tiemout	얼마나 기다릴지 설정하는 값이다. (단위: seconds)

- 예제

```
set serveroutput on
declare
    name   varchar(1000);
    msg    varchar(1000);
```

```
    status integer;
begin
    dbms_alert.register('ABC');
    dbms_alert.waitany(name, msg, status, 100000);
    dbms_output.put_line(name);
    dbms_output.put_line(msg);
    dbms_output.put_line(status);
END;
/
```

3.2.7. WAITONE

특정 ALERT을 기다리는 프러시저이다. 이전에 등록했던 ALERT만 기다릴 수 있으며, 등록하지 않은 ALERT을 지정하는 경우 에러가 발생한다. 또한 timeout이 지나면 깨어날 수 있다. 특정 ALERT을 받아서 깨어나도 해당 ALERT이 자동으로 REMOVE되지 않는다.

WAITANY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ALERT.WAITONE
(
    name         IN   VARCHAR2,
    message      OUT   VARCHAR2,
    status       OUT   VARCHAR2,
    timeout      IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
name	기다릴 ALERT의 이름이다.
message	WAITANY 프러시저가 깨어나도록 한 ALERT과 함께 전달된 메시지이다. 타임아웃으로 깨어난 경우 값이 없다.
status	– 1 : 타임아웃으로 깨어난 경우 – 0 : ALERT이 발생해서 깨어난 경우
tiemout	얼마나 기다릴지 설정하는 값이다. (단위: seconds)

- 예제

```
set serveroutput on
declare
```

```
    name   varchar(1000);
    msg    varchar(1000);
    status integer;
begin
    dbms_alert.register('ABC');
    dbms_alert.waitone('ABC', msg, status, 100000);
    dbms_output.put_line(msg);
    dbms_output.put_line(status);
END;
/
```

제4장 DBMS_APPLICATION_INFO

본 장에서는 DBMS_APPLICATION_INFO 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

4.1. 개요

DBMS_APPLICATION_INFO는 V$SESSION, SYS._VT_SESSION_LONGOPS 뷰의 값을 변경하는 패키지이다.

4.2. 프러시저

본 절에서는 DBMS_DBMS_APPLICATION_INFO 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

4.2.1. READ_CLIENT_INFO

현재 세션의 client_info 값을 읽는다.

READ_CLIENT_INFO 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO
(
    client_info         OUT         VARCHAR2
);
```

- 파라미터

파라미터	설명
client_info	세션으로부터 읽을 client info 값이다.

- 예제

```
DECLARE
   client_info VARCHAR2(64);
BEGIN
   DBMS_APPLICATION_INFO.READ_CLIENT_INFO(client_info);
```

```
   DBMS_OUTPUT.PUT_LINE(client_info);
END;
/
```

4.2.2. READ_MODULE

현재 세션의 모듈 정보와 액션 정보를 읽는 프러시저이다.

READ_MODULE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_APPLICATION_INFO.READ_MODULE
(
    module_name         OUT         VARCHAR2,
    action_name         OUT         VARCHAR2
)
```

- 파라미터

파라미터	설명
module_name	현재 세션의 모듈 값이다.
action_name	현재 세션의 액션 값이다.

- 예제

```
DECLARE
    module_name VARCHAR(64);

    action_name VARCHAR(64);
BEGIN
    DBMS_APPLICATION_INFO.READ_MODULE(module_name, action_name);
    DBMS_OUTPUT.PUT_LINE(module_name);
    DBMS_OUTPUT.PUT_LINE(action_name);

END;
/
```

4.2.3. SET_ACTION

현재 세션의 액션 이름을 지정한다.

SET_ACTION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_APPLICATION_INFO.SET_ACTION
(
    action_name         IN          VARCHAR2
)
```

- 파라미터

파라미터	설명
action_name	세션에 저장할 액션 이름이다. 64bytes 이상의 값은 잘린다.

- 예제

```
CREATE TABLE NEW_DATA_TBL(DATA NUMBER)
/

BEGIN
    DBMS_APPLICATION_INFO.SET_ACTION('데이터 추가');
    INSERT INTO NEW_DATA_TBL VALUES(1);
    INSERT INTO NEW_DATA_TBL VALUES(2);
    COMMIT;
    DBMS_APPLICATION_INFO.SET_ACTION(NULL);
END;
/
```

4.2.4. SET_CLIENT_INFO

현재 세션의 클라이언트 정보를 지정한다.

SET_CLIENT_INFO 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO
(
    client_info         IN          VARCHAR2
);
```

- 파라미터

파라미터	설명
client_info	현재 세션 내에 저장될 client_info이다. 64bytes 이상의 값은 잘린다.

- 예제

```
BEGIN
    DBMS_APPLICATION_INFO.SET_CLIENT_INFO('my_client_info');
END;
/

SELECT sid, client_info FROM V$SESSION WHERE client_info = 'my_client_info';
```

4.2.5. SET_MODULE

현재 세션의 모듈 이름을 지정한다.

SET_MODULE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_APPLICATION_INFO.SET_MODULE
(
    module_name         IN          VARCHAR2,
    action_name         IN          VARCHAR2
);
```

- 파라미터

파라미터	설명
module_name	세션에 저장할 모듈 이름이다. 64bytes 이상의 값은 잘린다.
action_name	세션에 저장할 액션 이름이다. 64bytes 이상의 값은 잘린다.

- 예제

```
CREATE TABLE NEW_DATA_TBL(DATA NUMBER)
/

BEGIN
    DBMS_APPLICATION_INFO.SET_MODULE('데이터 추가', '새로운 데이터 인서트');
    INSERT INTO NEW_DATA_TBL VALUES(1);
    INSERT INTO NEW_DATA_TBL VALUES(2);
    COMMIT;
    DBMS_APPLICATION_INFO.SET_MODULE(NULL, NULL);
END;
/
```

4.2.6. SET_SESSION_LONGOPS

SYS._VT_SESSION_LONGOPS 뷰의 값을 변경한다.

SET_SESSION_LONGOPS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS
(
     id                  IN OUT      PLS_INTEGER,
     op_name             IN          VARCHAR     DEFAULT NULL,
     sofar               IN          NUMBER      DEFAULT 0,
     totalwork           IN          NUMBER      DEFAULT 0,
     target_desc         IN          VARCHAR2    DEFAULT 'unknown target',
     units               IN          VARCHAR2    DEFAULT NULL
);
```

- 파라미터

파라미터	설명
id	사용할 longops의 ID이다. -1을 넣은 경우 새로운 ID를 할당 받는다.
op_name	SYS._VT_SESSION_LONGOPS에 저장될 opname column 값이다. 128bytes 이상은 무시된다.
sofar	SYS._VT_SESSION_LONGOPS에 저장될 sofar column 값이다.
totalwork	SYS._VT_SESSION_LONGOPS에 저장될 totalwork column 값이다.
target_desc	SYS._VT_SESSION_LONGOPS에 저장될 target_desc column 값이다. 128bytes 이상은 무시된다.
units	SYS._VT_SESSION_LONGOPS에 저장될 units column 값이다. 128bytes 이상은 무시된다.

- 예제

```
DECLARE
    idx PLS_INTERGER := -1;

BEGIN
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS(idx);
END;
/
SELECT * FROM SYS._VT_SESSION_LONGOPS;
```

제5장 DBMS_AQ

본 장에서는 DBMS_AQ 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

5.1. 개요

DBMS_AQ 패키지는 Tibero의 Advanced Queuing 기능과 관련한 프러시저와 함수를 제공한다. Tibero의 Advanced Queuing은 Tibero 데이터베이스와 통합되어 있는 메시지 Queuing 기능이다. 메시지는 데이터베이스 테이블에 영속적으로 저장되며, 필요에 따라 SQL을 통해 직접 queue 테이블의 내용을 조회하는 것도 가능하다.

queue는 단일 소비자 queue 또는 복수 소비자 queue일 수 있다. 단일 소비자 queue에 메시지를 enqueue하면 이 메시지는 오직 하나의 소비자만 dequeue할 수 있으며, 이 dequeue된 메시지를 또 다른 소비자가 dequeue하는 것은 불가능하다. 복수 소비자 queue일 경우는 디폴트 구독자들 또는 별도로 지정한 수신자들을 대상으로 enqueue를 수행할 수 있으며, 이때 각 소비자는 자신을 대상으로 enqueue된 메시지를 dequeue할 수 있다.

tbCLI 클라이언트는 자신이 관심있어 하는 queue에 enqueue가 수행될 때 그 이벤트를 알림으로 받을 수 있도록 자신을 등록할 수 있다. 이때 콜백함수를 명시해 놓으면 알림을 받을 때마다 해당 콜백함수를 수행하는 것이 가능하다.

참고

메시지의 payload 타입은 현재 RAW 타입만 가능하고, tbCLI에서 알림 등록 및 콜백함수 수행에 대해서는 "Tibero tbCLI 안내서"를 참고한다.

다음은 DBMS_AQ 패키지 내에 정의된 상수이다.

```
IMMEDIATE CONSTANT PLS_INTEGER := 0;
ON_COMMIT CONSTANT PLS_INTEGER := 1;
BROWSE CONSTANT BINARY_INTEGER := 1;
LOCKED CONSTANT BINARY_INTEGER := 2;
REMOVE CONSTANT BINARY_INTEGER := 3;
REMOVE_NODATA CONSTANT BINARY_INTEGER := 4;
FIRST_MESSAGE CONSTANT BINARY_INTEGER := 1;
NEXT_MESSAGE CONSTANT BINARY_INTEGER := 3;
FOREVER CONSTANT BINARY_INTEGER := -1;
NO_WAIT CONSTANT BINARY_INTEGER := 0;
NAMESPACE_AQ CONSTANT BINARY_INTEGER := 1;
```

5.2. 타입

본 절에서는 DBMS_AQ 패키지에서 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

5.2.1. AQ$_AGENT

메시지의 생산자 또는 소비자를 명시할 때 사용하는 타입이다.

AQ$_AGENT 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE AQ$_AGENT IS RECORD
(
     name     VARCHAR2(30),
     address  VARCHAR2(1024) DEFAULT NULL,
     protocol NUMBER DEFAULT 0
);
```

- 필드

필드 이름	설명
name	생산자 또는 소비자의 이름을 명시한다. 객체의 이름을 부여할 때와 동일한 규칙이 적용된다.
address	현재 버전에서는 사용되지 않는다.
protocol	현재 버전에서는 사용되지 않는다.

5.2.2. AQ$_RECIPIENT_LIST_T

메시지를 수신할 agent의 목록을 명시한다. 복수 소비자 queue에 대해서만 이 타입을 사용할 수 있다.

AQ$_RECIPIENT_LIST_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE AQ$_RECIPIENT_LIST_T IS TABLE OF AQ$_AGENT
        INDEX BY BINARY_INTEGER;
```

5.2.3. DEQUEUE_OPTIONS_T

dequeue할 때 적용할 수 있는 옵션들을 명시하는 타입이다.

DEQUEUE_OPTIONS_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DEQUEUE_OPTIONS_T IS RECORD
(
     consumer_name     VARCHAR2(30)    DEFAULT NULL,
     dequeue_mode      BINARY_INTEGER  DEFAULT REMOVE,
     navigation        BINARY_INTEGER  DEFAULT NEXT_MESSAGE,
     visibility        BINARY_INTEGER  DEFAULT ON_COMMIT,
     wait              BINARY_INTEGER  DEFAULT FOREVER,
     msgid             RAW(16)         DEFAULT NULL,
     correlation       VARCHAR2(128)   DEFAULT NULL,
     deq_condition     VARCHAR2(4000)  DEFAULT NULL
);
```

- 필드

필드 이름	설명
consumer_name	소비자의 이름이다. 이 소비자에게 보내진 메시지만 dequeue가 된다. 단일 소비자 queue에 대해서는 이 값이 NULL이어야 한다.
dequeue_mode	dequeue를 수행할 때 수행되는 잠금 방식을 결정한다. 설정 가능한 값은 다음과 같다. – BROWSE : 아무런 잠금도 획득하지 않고 메시지 내용을 읽는다. 단순 select와 동일하다. – LOCKED : dequeue한 메시지를 읽으면서 잠금을 획득한다. 이 잠금은 트랜잭션이 유지되는 동안 유효하다. select for update와 동일하다고 할 수 있다. – REMOVE : 메시지를 읽고 지운다. (기본값) – REMOVE_NODATA : 메시지를 읽지 않고 지운다.
navigation	가져올 메시지의 위치를 결정한다. 설정 가능한 값은 다음과 같다. – NEXT_MESSAGE : 마지막 위치 다음에 있는 메시지를 가져온다. (기본값) – FIRST_MESSAGE : 검색 조건에 맞는 첫 번째 메시지를 가져온다. 이 옵션을 사용하면 메시지의 위치가 초기화된다.

필드 이름	설명
visibility	메시지를 dequeue하는 동작을 현재 트랜잭션에 같이 포함시킬지 결정한다. dequeue_mode가 BROWSE일 경우 이 값은 무시된다. 설정 가능한 값은 다음과 같다. – ON_COMMIT : dequeue는 현재 트랜잭션에 포함된다. – IMMEDIATE : dequeue는 현재 트랜잭션에 참여하지 않고 자율 트랜잭션으로 수행되며, dequeue를 완료하면 자율 트랜잭션은 커밋된다.
wait	검색 조건에 해당하는 메시지가 없을 때 기다릴 시간을 명시한다. 설정 가능한 값은 다음과 같다. – FOREVER : 영원히 기다린다. (기본값) – NO_WAIT : 기다리지 않고 바로 리턴한다. – 숫자 : 기다릴 시간을 초 단위로 명시한다.
msgid	dequeue할 메시지 식별자를 명시한다.
correlation	dequeue할 메시지의 연관 식별자를 명시한다. 퍼센트 기호(%) 또는 언더바(_)와 같은 패턴 대조에 사용되는 문자를 넣을 수 있다.
deq_condition	메시지 속성 또는 메시지 데이터 속성에 대한 조건식을 지정한다. 이것은 SQL 문의 WHERE 절에 오는 것과 같은 형식의 참/거짓을 가지는 표현식이다. 메시지 속성은 queue 테이블에 존재하는 priority, corrid 또는 기타 다른 컬럼을 포함한다.

5.2.4. ENQUEUE_OPTIONS_T

enqueue할 때 취할 수 있는 옵션들을 명시하는 타입이다.

ENQUEUE_OPTIONS_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE ENQUEUE_OPTIONS_T IS RECORD
(
     visibility            BINARY_INTEGER  DEFAULT ON_COMMIT
);
```

- 필드

필드 이름	설명
visibility	메시지를 enqueue하는 동작을 현재 트랜잭션에 같이 포함시킬지 결정한다. 설정 가능한 값은 다음과 같다. – ON_COMMIT : enqueue는 현재 트랜잭션에 포함된다. (기본값) – IMMEDIATE : enqueue는 현재 트랜잭션에 참여하지 않고 자율 트랜잭션으로 수행되며, enqueue를 완료하면 자율 트랜잭션은 커밋된다.

5.2.5. MESSAGE_PROPERTIES_T

메시지 각각에 대한 여러 속성을 명시하는데 사용하는 타입이다. enqueue할 때는 이 속성을 채워서 enqueue하며, dequeue할 때는 채웠던 속성이 반환된다.

MESSAGE_PROPERTIES_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE MESSAGE_PROPERTIES_T IS RECORD
(
     priority          BINARY_INTEGER  NOT NULL DEFAULT 1,
     correlation       VARCHAR2(128)   DEFAULT NULL,
     recipient_list    AQ$_RECIPIENT_LIST_T,
     enqueue_time      DATE
);
```

- 필드

필드 이름	설명
priority	메시지의 우선 순위를 명시한다. 숫자가 작을수록 우선 순위가 높은 것이다. 음수를 포함해 어떤 숫자도 가능하다.
correlation	메시지를 enqueue할 때 생산자가 명시할 수 있는 연관 식별자이다.
recipient_list	메시지를 전달할 수신자를 직접 명시할 때 사용하는 AQ$_AGENT agent에 대한 인덱스 테이블이다. 복수 소비자 queue에 대해서만 이 값을 명시할 수 있고, 이 값을 명시하지 않으면 디폴트 구독자들이 수신자가 된다. 이 파라미터는 dequeue할 때 소비자에게 반환되지 않는다.
enqueue_time	메시지가 enqueue된 시점을 반환한다. 이 값은 시스템이 결정하며 enqueue할 때 사용자가 명시할 수 없다.

5.2.6. MESSAGE_PROPERTIES_ARRAY_T

이 타입은 dbms_aq.enqueue_array나 dbms_aq.dequeue_array를 호출할 때 여러 메시지에 대한 속성을 함께 명시하기 위해 사용한다. 즉, payload_array에 명시된 payload 갯수만큼 message_properties_array_t varray에 속성들이 명시되어야 한다.

MESSAGE_PROPERTIES_ARRAY_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE MESSAGE_PROPERTIES_ARRAY_T IS VARRAY (2147483647)
        OF MESSAGE_PROPERTIES_T;
```

5.2.7. MSGID_ARRAY_T

이 타입은 dbms_aq.enqueue_array나 dbms_aq.dequeue_array를 호출할 때 enqueue되거나 dequeue된 메시지 식별자들을 반환하기 위한 용도로 사용한다.

MSGID_ARRAY_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE MSGID_ARRAY_T IS TABLE OF RAW(16) INDEX BY BINARY_INTEGER;
```

5.2.8. PAYLOAD_ARRAY_T

이 타입은 dbms_aq.enqueue_array나 dbms_aq.dequeue_array를 호출할 때 payload 여러 개를 함께 명시하기 위해 사용한다. 현재 payload는 RAW 타입만 지원된다.

PAYLOAD_ARRAY_T 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE PAYLOAD_ARRAY_T IS VARRAY(2147483647) OF RAW(32767);
```

5.3. 프러시저와 함수

본 절에서는 DBMS_AQ 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

5.3.1. DEQUEUE

대상 queue로부터 메시지를 하나 dequeue한다. 어떤 메시지를 dequeue할 것인가는 dequeue_options의 consumer_name, msgid, correlation 또는 deq_condition에 무엇을 명시했느냐에 따라 결정된다. 각각에 대한 설명은 "5.2.3. DEQUEUE_OPTIONS_T"를 참고한다.

dequeue되는 메시지의 순서는 queue 테이블을 생성할 때 명시하는 정렬 순서에 의해 결정된다. dequeue_options에 msgid 또는 correlation을 명시하면 이 순서를 무시할 수 있다. enqueue 및 dequeue도 일반 데이터베이스 동작과 마찬가지로 일관성 읽기 규칙이 적용된다. 예를 들어 BROWSE가 시작되고 나서 enqueue된 메시지가 보이지 않을 수 있다.

navigation 파라미터의 기본값은 NEXT_MESSAGE이다. 이것은 반복적인 dequeue를 수행할 때 맨 처음 dequeue를 수행했을 때의 스냅샷을 기준으로 메시지들이 순차적으로 반환됨을 의미한다. 즉, 첫 번째 dequeue를 수행한 이후에 enqueue된 메시지는 현재 dequeue하고 있는 메시지들이 다 소비된 이후에만 소비가 가능하다. 일반적으로는 이렇게 동작해도 문제가 없지만 우선 순위와 같은 이유로 dequeue할 때 항상 queue의 첫 번째 메시지를 dequeue해야 할 경우는 FIRST_MESSAGE 옵션을 사용해야만 한다. 예를 들어 기존에 enqueue된 메시지를 처리하는 중에 더 높은 우선 순위의 메시지가 enqueue되었을 경우 이를 먼저 처리하기 위해서 이 옵션이 필요하다.

DEQUEUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQ.DEQUEUE
(
    queue_name         IN  VARCHAR2,
    dequeue_options    IN  DEQUEUE_OPTIONS_T,
    message_properties OUT MESSAGE_PROPERTIES_T,
    payload            OUT RAW,
    msgid              OUT RAW
);
```

- 파라미터

파라미터	설명
queue_name	메시지를 dequeue할 queue의 이름을 명시한다.
dequeue_options	dequeue할 모든 메시지에 동일하게 적용되는 옵션을 명시한다. 자세한 내용은 "5.2.3. DEQUEUE_OPTIONS_T"를 참고한다.
message_properties	반환되는 메시지 속성들을 명시한다. 자세한 내용은 "5.2.5. MESSAGE_PROPERTIES_T"를 참고한다.

파라미터	설명
payload	사용자가 지정하는 payload이다. 현재는 RAW 타입만 가능하다.
msgid	dequeue된 메시지에 대한 식별자이다.

- 예제

```
DECLARE
dequeue_options     DBMS_AQ.dequeue_options_t;
message_properties  DBMS_AQ.message_properties_t;
message_handle      RAW(16);
message             raw(1000);
BEGIN
   dequeue_options.consumer_name := 'RED';
   dequeue_options.deq_condition := 'enq_time is not null';
   DBMS_AQ.DEQUEUE(
      queue_name          =>     'my_multi_q',
      dequeue_options     =>     dequeue_options,
      message_properties  =>     message_properties,
      payload             =>     message,
      msgid               =>     message_handle);
   dbms_output.put_line('msgid: ' || message_handle);
END;
/
```

5.3.2. DEQUEUE_ARRAY

여러 메시지를 한꺼번에 dequeue하고 그것을 payload의 배열, 메시지 속성의 배열, 메시지 식별자의 배열로 반환하는 함수이다. 함수의 반환값은 성공적으로 dequeue된 메시지의 갯수이다.

dequeue_options에 명시된 wait 시간은 dequeue를 수행할 때 queue에 dequeue할 메시지가 아무것도 존재하지 않을 때만 적용된다. 만일 dequeue할 수 있는 메시지가 queue에 하나 이상 존재하면 DEQUEUE_ARRAY 함수는 최대 array_size 만큼의 메시지를 dequeue한 후 즉시 리턴된다.

dequeue_options로 msgid는 명시할 수 없다. navigation 파라미터와 관련한 내용은 “5.2.3. DEQUEUE_OPTIONS_T”를 참고한다.

DEQUEUE_ARRAY 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQ.DEQUEUE_ARRAY
(
    queue_name                IN   VARCHAR2,
    dequeue_options           IN   DEQUEUE_OPTIONS_T,
    array_size                IN   PLS_INTEGER,
```

```
    message_properties_array  OUT  MESSAGE_PROPERTIES_ARRAY_T,
    payload_array             OUT  PAYLOAD_ARRAY_T,
    msgid_array               OUT  MSGID_ARRAY_T
)
RETURN pls_integer;
```

- 파라미터

파라미터	설명
queue_name	메시지들을 dequeue할 queue의 이름을 명시한다.
dequeue_options	dequeue할 모든 메시지에 동일하게 적용되는 옵션을 명시한다. 자세한 내용은 "5.2.3. DEQUEUE_OPTIONS_T"를 참고한다.
array_size	dequeue할 메시지의 수를 지정한다.
message_properties_array	반환되는 메시지 속성들에 대한 배열이다. 자세한 내용은 "5.2.6. MESSAGE_PROPERTIES_ARRAY_T"를 참고한다.
payload_array	반환되는 payload들에 대한 배열이다. 자세한 내용은 "5.2.8. PAYLOAD_ARRAY_T"를 참고한다.
msgid_array	dequeue된 메시지들의 메시지 식별자가 반환되는 배열이다. 자세한 내용은 "5.2.7. MSGID_ARRAY_T"를 참고한다.

- 예제

```
declare
  dequeue_options       DBMS_AQ.dequeue_options_t;
  msg_prop_array        DBMS_AQ.message_properties_array_t;
  payload_array         dbms_aq.payload_array_t;
  msgid_array           DBMS_AQ.msgid_array_t;
  retval                PLS_INTEGER;
BEGIN
  dequeue_options.consumer_name := 'RED';

  retval := DBMS_AQ.DEQUEUE_ARRAY(
              queue_name              => 'my_multi_q',
              dequeue_options         => dequeue_options,
              array_size              => 10,
              message_properties_array => msg_prop_array,
              payload_array           => payload_array,
              msgid_array             => msgid_array);

  DBMS_OUTPUT.PUT_LINE('Number of messages dequeued: ' || retval);
  for i in 1..retval loop
    dbms_output.put_line ('msgid('||i||'): ' || msgid_array(i));
  end loop;
END;
/
```

5.3.3. ENQUEUE

대상 queue에 메시지를 하나 enqueue한다. 복수 소비자 queue에 enqueue를 수행할 때 이 queue에 디폴트 구독자가 하나도 등록되어 있지 않은 상태에서 수신자를 명시하지 않으면 메시지를 전달할 곳이 없으므로 에러를 반환한다.

ENQUEUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQ.ENQUEUE
(
    queue_name          IN  VARCHAR2,
    enqueue_options     IN  ENQUEUE_OPTIONS_T,
    message_properties IN  MESSAGE_PROPERTIES_T,
    payload             IN  RAW,
    msgid               OUT RAW
);
```

- 파라미터

파라미터	설명
queue_name	메시지를 enqueue할 queue의 이름을 명시한다.
enqueue_options	enqueue할 모든 메시지에 동일하게 적용되는 옵션을 명시한다. 자세한 내용은 “5.2.4. ENQUEUE_OPTIONS_T”를 참고한다.
message_properties	enqueue할 메시지들의 메시지 속성을 명시한다. 자세한 내용은 “5.2.5. MESSAGE_PROPERTIES_T”를 참고한다.
payload	사용자가 지정하는 payload이다. 현재는 RAW 타입만 가능하다.
msgid	시스템이 부여한 메시지 식별자를 반환한다. 유일성이 보장되며 dequeue할 때 메시지 식별 용도로 사용 가능하다.

- 예제

```
DECLARE
   enqueue_options     DBMS_AQ.enqueue_options_t;
   message_properties  DBMS_AQ.message_properties_t;
   message_handle      RAW(16);
BEGIN
   message_properties.priority := 2;
   message_properties.correlation := 'abc';
   DBMS_AQ.ENQUEUE(
      queue_name              => 'my_q',
      enqueue_options         => enqueue_options,
      message_properties      => message_properties,
      payload                 => hextoraw('FFFF'),
```

```
      msgid                    => message_handle);
   dbms_output.put_line('msgid: ' || message_handle);
END;
/
```

5.3.4. ENQUEUE_ARRAY

이 함수는 각각의 해당되는 메시지 속성과 함께 여러 payload를 한꺼번에 enqueue하는 함수이다. 완료와 함께 enqueue된 메시지의 식별자들을 담는 배열이 반환된다. 함수의 반환값은 성공적으로 enqueue된 메시지의 갯수이다.

ENQUEUE_ARRAY 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQ.ENQUEUE_ARRAY
(
    queue_name               IN  VARCHAR2,
    enqueue_options          IN  ENQUEUE_OPTIONS_T,
    array_size               IN  PLS_INTEGER,
    message_properties_array IN  MESSAGE_PROPERTIES_ARRAY_T,
    payload_array            IN  PAYLOAD_ARRAY_T,
    msgid_array              OUT MSGID_ARRAY_T
)
RETURN pls_integer;
```

- 파라미터

파라미터	설명
queue_name	메시지들을 enqueue할 queue의 이름을 명시한다.
enqueue_options	enqueue할 모든 메시지에 동일하게 적용되는 옵션을 명시한다. 자세한 내용은 “5.2.4. ENQUEUE_OPTIONS_T”를 참고한다.
array_size	enqueue할 메시지의 수를 지정한다.
message_properties_array	enqueue할 메시지들의 메시지 속성에 대한 배열이다. 자세한 내용은 “5.2.6. MESSAGE_PROPERTIES_ARRAY_T”를 참고한다.
payload_array	enqueue할 메시지들의 payload에 대한 배열이다. 자세한 내용은 “5.2.8. PAYLOAD_ARRAY_T”를 참고한다.
msgid_array	enqueue된 메시지들의 메시지 식별자가 반환되는 배열이다. 자세한 내용은 “5.2.7. MSGID_ARRAY_T”를 참고한다.

- 예제

```
declare
  v               dbms_aq.payload_array_t;
  msg_prop_array  dbms_Aq.message_properties_array_t;
  retval          pls_integer;
  msgid_array     dbms_aq.msgid_array_t;
  enqueue_options DBMS_AQ.enqueue_options_t;
begin
  v := dbms_aq.payload_array_t (hextoraw('FFFF'), hextoraw('EEEE'),
       hextoraw('EEEE'), hextoraw('EEEE'), hextoraw('EEEE'), hextoraw('EEEE'));
  msg_prop_array := dbms_aq.message_properties_array_t ();
  msg_prop_array.extend(6);

  for i in 1..6 loop
    msg_prop_array(i).priority := 7 - i;
  end loop;

  retval := dbms_aq.enqueue_array (
      queue_name               => 'my_multi_q',
      enqueue_options          => enqueue_options,
      array_size               => 6,
      message_properties_array => msg_prop_array,
      payload_array            => v,
      msgid_array              => msgid_array);
  dbms_output.put_line('processed: ' || retval);
  for i in 1..retval loop
    dbms_output.put_line ('msgid('||i||'): ' || msgid_array(i));
  end loop;
end;
/
```

제6장 DBMS_AQADM

본 장에서는 DBMS_AQADM 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

6.1. 개요

DBMS_AQADM 패키지는 Tibero의 Advanced Queuing 기능을 구성하고 관리하는 데 사용할 수 있는 프러시저와 함수를 제공한다.

6.2. 프러시저와 함수

본 절에서는 DBMS_AQADM 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

6.2.1. ADD_SUBSCRIBER

대상 queue에 디폴트 구독자를 추가하는 프러시저이다. 메시지를 enqueue할 때 수신자 목록을 대상으로 하거나 디폴트 구독자 목록을 대상으로 하여 enqueue할 수 있다.

이 프러시저는 복수 소비자 queue에 대해서만 수행이 가능하고 프러시저가 실행되면 이를 포함한 트랜잭션은 커밋된다. 프러시저가 성공적으로 종료된 이후의 enqueue는 해당 디폴트 구독자를 반영한다.

ADD_SUBSCRIBER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.ADD_SUBSCRIBER
(
    queue_name        IN    VARCHAR2,
    subscriber        IN    DBMS_AQ.AQ$_AGENT
);
```

- 파라미터

파라미터	설명
queue_name	추가할 queue의 이름이다.
subscriber	구독자를 명시하는 DBMS_AQ.AQ$_AGENT 타입의 agent이다.

- 예제

```
DECLARE
    subscriber dbms_aq.aq$_agent;
BEGIN
    subscriber.name := 'RED';

    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name  =>  'my_multi_q',
        subscriber  =>  subscriber);
END;
/
```

6.2.2. CREATE_QUEUE

대상 queue 테이블에 queue를 생성하는 프러시저이다. CREATE_QUEUE로 queue를 생성한 후 START_QUEUE를 호출하여 queue를 활성화시킬 수 있다. 디폴트로 enqueue와 dequeue 모두 비활성화된 채로 queue가 생성된다.

CREATE_QUEUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.CREATE_QUEUE
(
    queue_name        IN VARCHAR2,
    queue_table       IN VARCHAR2,
    queue_comment     IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
queue_name	생성할 queue의 이름이다. 스키마 내에서는 유일해야 하며 일반적인 객체의 이름을 부여할 때와 동일한 규칙이 적용된다.
queue_table	queue를 저장할 queue 테이블의 이름이다.
queue_comment	queue에 대한 사용자 지정 주석을 명시한다. 이후 정적 뷰를 통해 조회가 가능하다.

- 예제

```
begin
dbms_aqadm.create_queue_table ('my_qtbl', 'RAW', multiple_consumers=>false);
dbms_aqadm.create_queue ('my_q', 'my_qtbl');
end;
/
```

6.2.3. CREATE_QUEUE_TABLE

queue 테이블을 생성하는 프러시저이다. queue 테이블을 생성할 때 dequeue 순서를 결정하는 정렬 키를 지정할 수 있다. 저장 데이터 타입은 현재 RAW 만 가능하다.

CREATE_QUEUE_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.CREATE_QUEUE_TABLE
(
    queue_table         IN  VARCHAR2,
    queue_payload_type  IN  VARCHAR2,
    storage_clause      IN  VARCHAR2        DEFAULT NULL,
    sort_list           IN  VARCHAR2        DEFAULT 'ENQ_TIME',
    multiple_consumers  IN  BOOLEAN         DEFAULT FALSE,
    table_comment       IN  VARCHAR2        DEFAULT NULL
);
```

- 파라미터

파라미터	설명
queue_table	생성할 queue 테이블의 이름이다.
queue_payload_type	저장할 사용자 데이터의 타입을 지정한다. 현재는 RAW 만 가능하다.
storage_clause	저장 파라미터를 지정한다. queue 테이블을 생성하기 위하여 CREATE TABLE 문을 수행할 때 이 파라미터가 포함된다.
sort_list	오름차순으로 정렬할 정렬 키 컬럼을 지정한다. 이 파라미터를 명시되지 않으면 FIFO 순서와 동일하게 enqueue된 시간의 오름차순으로 정렬된다. (ENQ_TIME) 다음과 같은 형식으로 지정하며 가능한 컬럼 이름은 PRIORITY와 ENQ_TIME 이다. 두 컬럼이 동시에 명시되면 앞에 나온 컬럼값이 우선되어 정렬된다. `sort_column_1[,sort_column_2]` 위와 같이 정렬 순서를 지정하여 queue 테이블을 생성하면 정렬 순서는 바꿀 수 없고 queue 테이블에 생성되는 모든 queue는 동일한 정렬 순서를 갖는다. 단, dequeue할 때 msgid 또는 correlation을 지정하면 정렬 순서를 무시하고 특정 메시지를 dequeue할 수 있다.
multiple_consumers	– FALSE : 단일 소비자 queue를 저장하는 queue 테이블을 만든다. (기본값) – TRUE : 복수 소비자 queue를 저장하는 queue 테이블을 만든다.
table_comment	queue table에 대한 사용자 지정 주석을 명시한다. 이후 정적 뷰를 통해 조회가 가능하다.

- 예제

```
begin
dbms_aqadm.create_queue_table ('my_multi_qtbl', 'RAW', sort_list=>
    'PRIORITY,ENQ_TIME', multiple_consumers=>true);
end;
/
```

6.2.4. DROP_QUEUE

기존의 queue를 제거하는 프러시저이다. STOP_QUEUE를 통해 enqueue와 dequeue가 모두 비활성화 되기 전에는 DROP_QUEUE의 수행은 허용되지 않는다. queue가 제거되면 queue에 저장된 모든 데이터는 같이 삭제된다.

DROP_QUEUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.DROP_QUEUE
(
    queue_name          IN VARCHAR2
);
```

- 파라미터

파라미터	설명
queue_name	제거할 queue의 이름이다.

- 예제

```
begin
dbms_aqadm.stop_queue ('my_multi_q');
dbms_aqadm.drop_queue ('my_multi_q');
end;
/
```

6.2.5. DROP_QUEUE_TABLE

기존의 queue 테이블을 제거하는 프러시저이다. queue 테이블을 제거하기 전에 queue 테이블 내의 모든 queue를 정지시키고 제거한 이후에만 queue 테이블을 제거할 수 있다. 그러나 force 옵션을 주면 이러한 동작이 자동으로 수행된다.

DROP_QUEUE_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.DROP_QUEUE_TABLE
(
    queue_table         IN VARCHAR2,
    force               IN BOOLEAN      DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
queue_table	제거할 queue 테이블의 이름이다.
force	– FALSE : queue가 테이블 내에 존재할 경우 제거되지 않는다. (기본값) – TRUE : 테이블 내의 모든 queue가 자동으로 정지되고 제거된다.

- 예제

```
begin
    dbms_aqadm.drop_queue_table ('my_multi_qtbl', true);
end;
/
```

6.2.6. GRANT_SYSTEM_PRIVILEGE

Tibero Advanced Queuing 시스템 특권을 사용자 또는 역할에게 부여하는 프러시저이다. 부여 가능한 특권은 ENQUEUE_ANY, DEQUEUE_ANY, MANAGE_ANY이다. 단, 처음에는 SYS 사용자 또는 DBA 역할이 부여된 사용자만이 이 프러시저를 수행할 수 있다.

DBMS_AQADM 패키지의 프러시저를 수행하기 위해서는 DBMS_AQADM 패키지에 대해 EXECUTE 권한이 있어야 한다. 마찬가지로 dequeue와 enqueue의 수행을 위해서는 DBMS_AQ 패키지에 대해 EXECUTE 권한이 있어야 한다.

DBMS_AQADM 패키지의 프러시저가 다루는 객체가 프러시저를 호출한 스키마에 속할 경우 이 패키지에 대해 EXECUTE 권한만 있으면 더 이상의 특권이 필요없이 프러시저의 수행이 허용된다. 마찬가지로 자기 자신의 스키마에 생성한 queue에 대해 enqueue 또는 dequeue를 수행할 때는 DBMS_AQ 패키지에 대해 EXECUTE 권한만 있으면 수행이 허용된다.

GRANT_SYSTEM_PRIVILEGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE
(
    privilege           IN    VARCHAR2,
```

```
    grantee           IN    VARCHAR2,
    admin_option      IN    BOOLEAN  DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
privilege	부여할 Tibero Advanced Queuing 시스템 특권을 지정한다. 부여 가능한 특권은 다음과 같다. – ENQUEUE_ANY : 데이터베이스의 어떤 queue에 대해서도 enqueue를 수행할 수 있다. – DEQUEUE_ANY : 데이터베이스의 어떤 queue에 대해서도 dequeue를 수행할 수 있다. – MANAGE_ANY : 데이터베이스의 어떤 스키마 객체에 대해서도 DBMS_AQADM 프러시저를 수행할 수 있다.
grantee	부여받을 대상(사용자, 역할, PUBLIC)을 명시한다.
admin_option	시스템 특권을 ADMIN 옵션과 함께 부여할지를 지정한다. ADMIN 옵션과 함께 특권이 부여되면 부여받은 대상은 이 프러시저를 호출하여 권한을 다른 사용자 또는 역할에 부여해줄 수 있다. (기본값: FALSE)

- 예제

```
begin
dbms_aqadm.grant_system_privilege ('MANAGE_ANY', 'test_aq');
end;
/
```

6.2.7. REMOVE_SUBSCRIBER

디폴트 구독자를 해당 queue에서 제거하는 프러시저이다. 이 프러시저가 실행되면 이를 포함한 트랜잭션은 커밋되며 구독자와 관련된 메시지 및 정보는 삭제된다.

REMOVE_SUBSCRIBER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.REMOVE_SUBSCRIBER
(
    queue_name          IN VARCHAR2,
```

```
    subscriber          IN DBMS_AQ.AQ$_AGENT
);
```

- 파라미터

파라미터	설명
queue_name	queue의 이름이다.
subscriber	제거할 agent를 명시한다. (타입: DBMS_AQ.AQ$_AGENT)

- 예제

```
DECLARE
    subscriber         dbms_aq.aq$_agent;
BEGIN
    subscriber.name := 'RED';

    DBMS_AQADM.remove_subscriber (
        queue_name  =>  'my_multi_q',
        subscriber  =>  subscriber);
END;
/
```

6.2.8. REVOKE_SYSTEM_PRIVILEGE

Tibero Advanced Queuing 시스템 특권을 사용자 또는 역할로부터 회수하는 프러시저이다. 회수 가능한 특권은 ENQUEUE_ANY, DEQUEUE_ANY, MANAGE_ANY이다. 단, 시스템 특권의 ADMIN 옵션만 선택적으로 회수하는 것은 불가능하다.

REVOKE_SYSTEM_PRIVILEGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE
(
    privilege         IN    VARCHAR2,
    grantee           IN    VARCHAR2
);
```

- 파라미터

파라미터	설명
privilege	회수할 Tibero Advanced Queuing 시스템 특권을 지정한다. 회수 가능한 특권은 다음과 같다. 단, ADMIN 옵션만 선택적으로 회수하는 것은 불가능하다.

파라미터	설명
	– ENQUEUE_ANY – DEQUEUE_ANY – MANAGE_ANY
grantee	회수할 대상(사용자, 역할, PUBLIC)을 명시한다.

- 예제

```
begin
dbms_aqadm.revoke_system_privilege ('MANAGE_ANY', 'test_aq');
end;
/
```

6.2.9. START_QUEUE

해당 queue의 enqueue 또는 dequeue 기능을 활성화하는 프러시저이다. queue를 생성하고 나면 START_QUEUE 프러시저를 통해 queue를 활성화해야 하며, 디폴트로 enqueue와 dequeue가 동시에 활성화된다. 이 동작은 종료와 함께 즉시 반영되며 포함하고 있는 트랜잭션에 영향을 주지 않는다.

START_QUEUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.START_QUEUE
(
    queue_name          IN VARCHAR2,
    enqueue             IN BOOLEAN      DEFAULT TRUE,
    dequeue             IN BOOLEAN      DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
queue_name	활성화할 queue의 이름을 명시한다.
enqueue	enqueue 기능을 활성화할지를 명시한다. – TRUE : 활성화된다. (기본값) – FALSE : 이전의 설정을 유지한다.
dequeue	dequeue 기능을 활성화할지를 명시한다. – TRUE : 활성화된다. (기본값)

파라미터	설명
	– FALSE : 이전의 설정을 유지한다.

- 예제

```
begin
dbms_aqadm.create_queue_table ('my_qtbl', 'RAW', multiple_consumers=>false);
dbms_aqadm.create_queue ('my_q', 'my_qtbl');
dbms_aqadm.start_queue ('my_q');
end;
/
```

6.2.10. STOP_QUEUE

해당 queue의 enqueue 또는 dequeue 기능을 비활성화하는 프러시저이다. 디폴트로 enqueue와 dequeue를 동시에 비활성화시키며, queue에 대해 트랜잭션이 진행중일 때에는 queue를 비활성화시킬 수 없다. 이 동작은 종료와 함께 즉시 반영되며 포함하고 있는 트랜잭션에 영향을 주지 않는다.

STOP_QUEUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_AQADM.STOP_QUEUE
(
    queue_name          IN VARCHAR2,
    enqueue             IN BOOLEAN      DEFAULT TRUE,
    dequeue             IN BOOLEAN      DEFAULT TRUE,
    wait                IN BOOLEAN      DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
queue_name	비활성화할 queue의 이름을 명시한다.
enqueue	enqueue 기능을 비활성화할지를 명시한다. – TRUE : 비활성화된다. (기본값) – FALSE : 이전의 설정을 유지한다.
dequeue	dequeue 기능을 비활성화할지를 명시한다. – TRUE : 비활성화된다. (기본값) – FALSE : 이전의 설정을 유지한다.

파라미터	설명
wait	현재 진행 중인 트랜잭션이 있다면 그것을 기다릴 것인지를 지정한다. – TRUE : 트랜잭션을 기다리며 해당 queue에 대해 더이상의 추가적인 enqueue 또는 dequeue가 수행되지 않는다. (기본값) – FALSE : 트랜잭션이 진행 중일 경우 에러를 반환하며 즉시 리턴한다.

- 예제

```
begin
dbms_aqadm.stop_queue ('my_multi_q');
end;
/
```

제7장 DBMS_BACKUP_RESTORE

본 장에서는 DBMS_BACKUP_RESTORE 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

7.1. 개요

DBMS_BACKUP_RESTORE 패키지는 데이터베이스의 백업과 복구를 위해 사용되는 패키지이다. 그러므로 본 패키지는 오직 sys 유저만이 수행 권한을 갖는다.

7.2. 프러시저

본 절에서는 DBMS_BACKUP_RESTORE 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

7.2.1. SEARCHFILES

파라미터로 준 경로 안 모든 파일들의 리스트를 V$SEARCH_FILES 테이블에 넣어준다. V$SEARCH_FILES 테이블은 본 프러시저를 수행할 때마다 초기화된다. V$SEARCH_FILES 테이블은 오직 sys 유저만이 조회할 수 있다.

SEARCHFILES 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_BACKUP_RESTORE.SEARCHFILES
(
    dir       IN  VARCHAR2,
    ns        IN  VARCHAR2,
);
```

- 파라미터

파라미터	설명
dir	디렉터리의 절대 경로이다.
ns	빈 문자열이다.

- 예제

```
DECLARE
  dir   VARCHAR2(1024) := '/home/tibero/data';
  ns    VARCHAR2(1024);
BEGIN
  DBMS_BACKUP_RESTORE.SEARCHFILES(dir, ns);
END;
/

SQL> SELECT * FROM V$SEARCH_FILES;

FILE_NAME
--------------------------------------------------------------------------------
/home/tibero/data/log001.log
/home/tibero/data/log002.log
/home/tibero/data/log003.log
/home/tibero/data/system001.dtf
/home/tibero/data/syssub001.dtf
/home/tibero/data/temp001.dtf
/home/tibero/data/usr001.dtf
/home/tibero/data/undo001.dtf
/home/tibero/data/c1.ctl
/home/tibero/data/.passwd

10 rows selected.
```

제8장 DBMS_CRYPTO

본 장에서는 DBMS_CRYPTO 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

8.1. 개요

DBMS_CRYPTO은 데이터 암복호화 및 인증 등에 쓰이는 해시 알고리즘을 제공하는 패키지이다. 이 패키지의 암복호화 알고리즘들은 모두 키를 사용하는데, 이와 같이 키를 사용하는 암호화 알고리즘에서는 키를 안전하게 관리하는 것이 무엇보다 중요하다. 암호화 알고리즘에서 키를 관리하는 방법은 DBMS_OBFUSCATION_TOOLKIT을 참고한다.

해시 알고리즘은 임의의 길이의 데이터를 고정길이의 해시값으로 변환한다.

8.2. 암호화/복호화 알고리즘 및 체인, 패딩

DBMS_CRYPTO은 데이터의 암호화 및 복호화를 위해 DES(Data Encryption Standard), 3DES(Triple DES), AES(Advanced Encryption Standard) 등의 더욱 다양한 알고리즘 및 블록 패딩 방식을 지원하고 있다. **DBMS_OBFUSCATION_TOOLKIT보다 많은 종류의 알고리즘을 사용하므로, 이 패키지를 사용할 것을 권장한다.**

각 알고리즘, 체인, 패딩 방식은 아래와 같다.

- 알고리즘

알고리즘	키워드	설명
DES	ENCRYPT_DES	64bits의 블록 단위로 나누어 각각을 56bits의 키를 사용하는 알고리즘이다. 그동안 널리 사용되어 왔으나 최근에는 보안성이 보장되지 않아 점차 사용 빈도가 줄어들고 있다. 대칭 키를 사용하는 알고리즘이다.
3DES	EN CRYPT_3DES	하나의 데이터에 DES 알고리즘을 두 번 내지 세 번 반복하여 적용하는 알고리즘이다. 각각 112(56 * 2)bits와 168(56 * 3)bits 키를 사용한다. DES 알고리즘과 마찬가지로 대칭 키를 사용하는 알고리즘이다. 다른 대칭 키를 사용하는 알고리즘에 비해 암호화 및 복호화를 하는 시간이 많이 필요하다는 단점이 있다.
AES	ENCRYPT_AES	128bits의 블록 단위로 나누어 각각을 128, 192 또는 256bits의 키를 사용하는 알고리즘이다. DES 방식보

알고리즘	키워드	설명
		다 강력한 알고리즘으로 설계되었으며, 사실상 암호화 표준으로 자리 잡을 것으로 보인다.
RC4	ENCRYPT_RC4	RSA 보안을 위해 로널드 라이베스트가 만든 스트림 암호화 방식으로, 무작위 치환에 기반을 두고 있다. 바이트 단위 연산을 하기 때문에 비트 단위 연산을 하는 다른 스트림 암호화에 비해 속도가 빠르다. SSL/TLS(Secure Sockets Layer / Transport Layer Security) 및 무선랜 표준 IEEE 802.11 WEP(Wired Equivalent Privacy) 프로토콜 등에서 사용된다.
ARIA	EN CRYPT_ARIA	대한민국의 국가보안기술연구소에서 개발한 블록 암호화 알고리즘으로, AES처럼 128bits의 블록 단위로 나누어 각각을 128, 192 또는 256bits의 키를 사용한다. Involutional SPN 구조를 갖기 때문에 경량 환경 및 하드웨어 구현에 최적화되어 있다.

- 체인(chaining) 방식

체인 방식	키워드	설명
ECB(Electronic Codebook)	CHAIN_ECB	각 원본 데이터 블록들이 각각 별개로 암호화된다.
CBC(Cipher Block Chaining)	CHAIN_CBC	현재 블록이 바로 앞에 암호화된 블록의 데이터와 XOR 연산을 거친 후 암호화된다. ECB 방식에서처럼 동일한 원본이 있을 경우 동일한 암호문이 생성되는 위험을 방지하기 위함이다.
CFB(Cipher-Feedback)	CHAIN_CFB	CBC와 유사하지만, 비트 단위로 암호화하면서 쉬프트시키는 방식으로, 스트림 암호화 방식에 가깝다. 그러므로 블록 크기보다 작은 단위의 데이터의 암호화도 가능해진다.
OFB(Output-Feedback)	CHAIN_OFB	블록 암호가 동기화된 스트림 암호처럼 동작하게 한다. CFB와 유사하지만, 다음 블록을 위한 키배열 조합에 있어서 현재 블록의 암호문이 영향을 끼치지 않으므로, 현재 블록 암호화에서 오류가 발생해도 다음 블록에 영향을 끼치지 않는 장점이 있다.

- 패딩(padding) 방식

패딩 방식	키워드	설명
PKCS5	PAD_PKCS5	PKCS #5(Password-Based Cryptography Standard) 표준에 따르는 패딩 방식이다.
NONE	PAD_NONE	패딩을 사용하지 않는다. 이러할 경우 데이터가 단위 블록의 크기(128bits)의 배수가 아니면 암호화가 불가능하여 에러를 리턴하므로, 반드시 데이터의 크기를 확인해야 한다.
ZERO	PAD_ZERO	마지막 블록의 남는 바이트들을 0으로 채운다. 문자열과 같이 0의 값으로 끝을 체크하는 데이터에만 사용해야 한다.

8.3. 해시 알고리즘

DBMS_CRYPTO은 임의의 길이의 데이터를 고정 길이의 해시값으로 변환하는데, 해시값을 만드는 것은 쉽지만, 해시값을 원래의 데이터로 복원하는 일은 지극히 어렵게 함으로써, 데이터의 안전성을 보장한다.

데이터 변경 여부 체크 및 암호 인증 등 인증(authentication) 작업에 주로 사용된다.

MD4, MD5, SHA-1, SHA-2을 통해 더욱 다양한 알고리즘 및 블록 패딩 방식을 지원하고 있다. 각 알고리즘은 아래와 같다.

알고리즘	키워드	설명
MD5(Message Digest 5)	HASH_MD5	입력 데이터(길이에 상관없는 하나의 메시지)로부터 128bits 메시지 축약을 만듦으로써 데이터 무결성을 검증하는 데 사용되는 알고리즘이다. MD4의 확장판인데, MD4에 비해 속도가 빠르지는 않지만, 데이터 보안성에 있어 더 많은 확신을 제공한다.
MD4(Message Digest 4)	HASH_MD4	MD4는 MD5의 초기 버전으로서, 입력 데이터(길이에 상관없는 하나의 메시지)로부터 128bits 메시지 축약을 만듦으로써 데이터 무결성을 검증하는데 사용되는 알고리즘이다.
SHA-1(Secure Hash Algorithm 1)	HASH_SH1	SHA는 미국 NIST에 의해 개발된 SHS(Secure Hash Standard) 내에 정의된 알고리즘으로, 길이 264bits 이하의 메시지를 160bits 길이의 축약된 메시지로 만들어낸다. MD5보다는 다소 느리지만, 대규모 메시지 요약들이 폭력적 충돌 및 도치 공격을 받을 때 좀 더 안전하게 지켜준다.

알고리즘	키워드	설명
SHA-2(Secure Hash Algorithm 2)	HASH_SH256 HASH_SH384 HASH_SH512	SHA-2는 SHA-1을 대체하는 해시 암호이다. 해시 함수에서 사용하는 byte 수에 따라, SHA-224, SHA-256, SHA-384, SHA-512이 있고, Tibero에서는 SHA-256, SHA-384, SHA-512을 지원한다.

8.4. 프러시저와 함수

본 절에서는 DBMS_CRYPTO 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

8.4.1. DECRYPT

사용자가 명시한 암호화 알고리즘 및 체인, 패딩 기법을 이용하여 암호화된 데이터를 복호화하는 함수이다. 주어진 알고리즘에 맞는 키의 크기와 블록의 크기를 설정해야 올바르게 동작한다. 알고리즘 각각에 따른 키의 크기와 블록의 크기는 위의 표에 언급되어 있으니 참고한다.

DECRYPT 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 함수

```
DBMS_CRYPTO.DECRYPT
(
    src                 IN          RAW,
    cipher_type         IN          PLS_INTEGER,
    key                 IN          RAW,
    init_vector         IN          RAW          DEFAULT NULL
)
RETURN RAW;
```

  - 프러시저

```
DBMS_CRYPTO.DECRYPT
(
    dst                 IN OUT NOCOPY   BLOB,
    src                 IN              BLOB,
    cipher_type         IN              PLS_INTEGER,
    key                 IN              RAW,
    init_vector         IN              RAW          DEFAULT NULL
);
```

```
DBMS_CRYPTO.DECRYPT
(
    dst                 IN OUT NOCOPY   CLOB,
    src                 IN              BLOB,
    cipher_type         IN              PLS_INTEGER,
    key                 IN              RAW,
    init_vector         IN              RAW          DEFAULT NULL
);
```

- 파라미터

파라미터	설명
dst	복호화된 데이터이다.
src	복호화할 데이터이다.
cipher_type	사용할 암호화 알고리즘, 체인, 패딩 기법이다.
key	복호화하기 위해 주어진 키 값이다.
init_vector	초기화 벡터이다. NULL이면 0으로 채워진 초기화 벡터를 사용한다.

- 반환값

  복호화된 데이터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_NTH_ARGUMENT	cipher_type 값이 잘못 설정된 경우이다.
INVALID_INPUT	input_data의 길이가 8의 배수가 아닌 경우이다.
KEY_TOO_SHORT	키 값의 길이가 필요한 크기보다 작은 경우이다.

- 예제

```
DECLARE
    data RAW(256);
    key RAW(16);
    encrypted_data RAW(256);
    decrypted_data RAW(256);
    iv RAW(256);
BEGIN
    data := '0102030405AE030D0102030405AE030D';
    key := '0A123B8E002CD3FFA021B3E800C23DFF';
    iv := '00000000000000000000000000000000';

    encrypted_data := DBMS_CRYPTO.ENCRYPT(
            src => data,
```

```
    cipher_type => DBMS_CRYPTO.ENCRYPT_AES128 + DBMS_CRYPTO.CHAIN_CBC +
                   DBMS_CRYPTO.PAD_PKCS5,
            key => key,
            init_vector => iv);

    encrypted_data := DBMS_CRYPTO.DECRYPT(
            src => encrypted_data,
    cipher_type => DBMS_CRYPTO.ENCRYPT_AES128 + DBMS_CRYPTO.CHAIN_CBC +
                   DBMS_CRYPTO.PAD_PKCS5,
            key => key,
            init_vector => iv);
END;
```

8.4.2. ENCRYPT

사용자가 명시한 암호화 알고리즘 및 체인, 패딩 기법을 이용하여 데이터를 암호화하는 함수이다.

주어진 알고리즘에 맞는 키의 크기와 블록의 크기를 설정해야 올바르게 동작한다. 알고리즘 각각에 따른 키의 크기와 블록의 크기는 위의 표에 언급되어 있으니 참고한다.

ENCRYPT 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 함수

```
DBMS_CRYPTO.ENCRYPT
(
     src                 IN          RAW,
     cipher_type         IN          PLS_INTEGER,
     key                 IN          RAW,
     init_vector         IN          RAW          DEFAULT NULL
)
RETURN RAW;
```

```
DBMS_CRYPTO.HASH
(
    src                 IN          BLOB,
    hash_type           IN          PLS_INTEGER
)
RETURN RAW;
```

```
DBMS_CRYPTO.HASH
(
    src                 IN          CLOB,
    hash_type           IN          PLS_INTEGER
```

```
)
RETURN RAW;
```

– 프러시저

```
DBMS_CRYPTO.ENCRYPT
(
    dst                 IN OUT NOCOPY   BLOB,
    src                 IN              BLOB,
    cipher_type         IN              PLS_INTEGER,
    key                 IN              RAW,
    init_vector         IN              RAW          DEFAULT NULL
);
```

```
DBMS_CRYPTO.ENCRYPT
(
    dst                 IN OUT NOCOPY   BLOB,
    src                 IN              CLOB,
    cipher_type         IN              PLS_INTEGER,
    key                 IN              RAW,
    init_vector         IN              RAW          DEFAULT NULL
);
```

- 파라미터

– 함수

파라미터	**설명**
src	원본 데이터이다.
hash_type	사용할 해시 알고리즘이다. 다음은 사용 가능한 해시함수 알고리즘 상수이다. `hash_md4 CONSTANT PLS_INTEGER := 1;` `hash_md5 CONSTANT PLS_INTEGER := 2;` `hash_sh1 CONSTANT PLS_INTEGER := 3;` `hash_sh256 CONSTANT PLS_INTEGER := 4;` `hash_sh384 CONSTANT PLS_INTEGER := 5;` `hash_sh512 CONSTANT PLS_INTEGER := 6;`

– 프러시저

파라미터	**설명**
dst	암호화된 데이터이다.
src	암호화할 데이터이다.
cipher_type	사용할 암호화 알고리즘, 체인, 패딩 기법이다.

파라미터	설명
key	암호화하기 위해 주어진 키 값이다.
init_vector	초기화 벡터이다. NULL이면 0으로 채워진 초기화 벡터를 사용한다.

- 반환값

  암호화된 데이터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_NTH_ARGUMENT	cipher_type 값이 잘못 설정된 경우이다.
INVALID_INPUT	ZERO 패딩 방식일 때 input_data의 길이가 블록 크기의 배수가 아닌 경우이다.
KEY_TOO_SHORT	키 값의 길이가 필요한 크기보다 작은 경우이다.

- 예제

```
DECLARE
    data RAW(256);
    key RAW(16);
    encrypted_data RAW(256);
    decrypted_data RAW(256);
    iv RAW(256);
BEGIN
    data := '0102030405AE030D0102030405AE030D';
    key := '0A123B8E002CD3FFA021B3E800C23DFF';
    iv := '00000000000000000000000000000000';

    encrypted_data := DBMS_CRYPTO.ENCRYPT(
            src => data,
    cipher_type => DBMS_CRYPTO.ENCRYPT_AES128 + DBMS_CRYPTO.CHAIN_CBC +
                   DBMS_CRYPTO.PAD_PKCS5,
            key => key,
            init_vector => iv);
END;
```

8.4.3. HASH

사용자가 명시한 알고리즘을 이용하여 임의의 길이의 데이터를 고정길이의 해시값으로 변환하는 함수이다.

HASH 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_CRYPTO.HASH
(
    src                 IN          RAW,
    hash_type           IN          PLS_INTEGER
)
RETURN RAW;
```

- 파라미터

파라미터	설명
src	원본 데이터이다.
hash_type	사용할 해시 알고리즘이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_NTH_ARGUMENT	hash_type 값이 잘못 설정된 경우이다.

- 예제

```
DECLARE
    input varchar2(100);
    hash_val raw(20);
BEGIN
    input := 'DBMS_CRYPTO.HASH 테스트';

    hash_val := DBMS_CRYPTO.HASH(
             src => utl_raw.cast_to_raw(input),
             hash_type => DBMS_CRYPTO.HASH_SH1);
END;
```

8.4.4. MAC

사용자가 명시한 알고리즘과 임의의 길이의 데이터를 가지고 입력한 KEY에 대해 메시지 인증 코드(Message Authentication Code, 약칭 MAC)를 생성하는 함수이다.

MAC 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_CRYPTO.MAC
(
    src                 IN          RAW,
    mac_type            IN          PLS_INTEGER,
    key                 IN          RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
src	원본 데이터이다.
mac_type	사용할 MAC 알고리즘이다. 다음은 사용 가능한 MAC 함수 알고리즘 상수이다. `hmac_md5 CONSTANT PLS_INTEGER := 1;` `hmac_sh1 CONSTANT PLS_INTEGER := 2;` `hmac_sh256 CONSTANT PLS_INTEGER := 3;` `hmac_sh384 CONSTANT PLS_INTEGER := 4;` `hmac_sh512 CONSTANT PLS_INTEGER := 5;`
key	사용할 KEY 데이터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_NTH_ARGUMENT	mac_type 값이 잘못 설정된 경우이다.

- 예제

```
SET SERVEROUTPUT ON
DECLARE
    vKey     VARCHAR2(2000);
    vHashed  RAW(20);
    vText    VARCHAR2(2000);
BEGIN
    vText := 'test 테스트 1234 !@#$ + - / *';
    vKey  := 'PASSCODE';
    vHashed := DBMS_CRYPTO.MAC(
      src => UTL_I18N.STRING_TO_RAW (vText, 'MSWIN949'),
      mac_type => DBMS_CRYPTO.HMAC_SH1,
      key => UTL_I18N.STRING_TO_RAW(vKey, 'MSWIN949'));
    DBMS_OUTPUT.PUT_LINE(vText);
    DBMS_OUTPUT.PUT_LINE(vKey);
```

```
    DBMS_OUTPUT.PUT_LINE(vHashed);
END;
```

8.4.5. RANDOMBYTES

사용자가 명시한 크기만큼 임의의 BYTE 데이터를 반환하는 함수이다.

RANDOMBYTES 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_CRYPTO.RANDOMBYTES
(
    number_bytes         IN          PLS_INTEGER
)
RETURN RAW;
```

- 파라미터

파라미터	설명
number_bytes	반환할 임의의 데이터 크기이다.

- 예제

```
DECLARE
   l_key   RAW (16);
BEGIN
   l_key := DBMS_CRYPTO.randombytes (16);
   DBMS_OUTPUT.PUT_LINE(l_key);
END
```

제9장 DBMS_DB_VERSION

본 장에서는 DBMS_DB_VERSION 패키지의 기본 개념과 패키지를 사용하는 방법을 설명한다.

9.1. 개요

DBMS_DB_VERSION 제품의 버전 정보를 확인하는 패키지이다.

9.2. 상수

본 절에서는 DBMS_DB_VERSION 패키지에서 제공하는 상수를 알파벳 순으로 설명한다.

9.2.1. MAJOR

제품의 메인 버전값을 INTEGER로 나타낸다.

세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DB_VERSION.MAJOR
```

- 예제

```
BEGIN
   if DBMS_DB_VERSION.MAJOR = 6 then
      DBMS_OUTPUT.PUT_LINE(DBMS_DB_VERSION.MAJOR);
   end if;
END;
/
```

9.2.2. MINOR

제품의 마이너 버전값을 INTEGER로 나타낸다.

세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DB_VERSION.MINOR
```

- 예제

```
BEGIN
   if DBMS_DB_VERSION.MINOR < 1 then
      DBMS_OUTPUT.PUT_LINE(DBMS_DB_VERSION.MINOR);
   end if;
END;
/
```

제10장 DBMS_DDL

본 장에서는 DBMS_DDL 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

10.1. 개요

DBMS_DDL CREATE OR REPLACE PROCEDURE, PACKAGE로 시작하는 DDL 구문에 대한 WRAPPING 기능 외 기타 DDL 관련 기능을 제공하는 패키지이다.

10.2. 프러시저와 함수

본 절에서는 DBMS_DDL 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

10.2.1. CREATE_WRAPPED

CREATE OR REPLACE PROCEDURE, FUNCTION, PACKAGE 또는 PACKAGE BODY로 시작하는 DDL 구문을 WRAPPING한 후 해당 구문을 실행까지 하는 프러시저이다.

다음과 호출 결과가 동일하다.

```
EXECUTE IMMEDIATE DBMS_DDL.WRAP(ddl)
```

CREATE_WRAPPED 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DDL.CREATE_WRAPPED
(
    ddl IN VARCHAR2
)
```

- 파라미터

파라미터	설명
ddl	CREATE OR REPLACE PROCEDURE, FUNCTION, PACKAGE 또는 PACKAGE BODY로 시작하는 DDL 구문이다. tbwrap 툴과는 달리 하나의 DDL 구문만 입력할 수 있다.

- 예제

```
create or replace procedure create_wrapped_example
is
    hello_ddl varchar(1000) := 'create or replace procedure hello
                                is
                                  str varchar(10) := ''hello'';
                                begin
                                  DBMS_OUTPUT.PUT_LINE(str);
                                end;';
begin
    DBMS_DDL.CREATE_WRAPPED(hello_ddl);
end;
/
```

참고

DBMS_DDL.CREATE_WRAPPED 프러시저는 호출자 권한으로 실행되며, 해당 호출자의 Role은 참조하지 않도록 되어 있다. 그래서 호출자에게 CREATE ANY PROCEDURE 권한이 있어야 위 예제의 hello_ddl이 실행될 것이다.

10.2.2. IS_TRIGGER_FIRE_ONCE

TRIGGER의 속성 중 FireOnce 여부에 대해 리턴하는 함수이다.

IS_TRIGGER_FIRE_ONCE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DDL.IS_TRIGGER_FIRE_ONCE
(
    trig_owner IN VARCHAR2,
    trig_name  IN VARCHAR2
)
RETURN BOOLEAN
```

- 파라미터

파라미터	설명
trig_owner	TRIGGER 소유자 이름이다.
trig_name	TRIGGER 이름이다.

10.2.3. SET_TRIGGER_FIRING_PROPERTY

TRIGGER의 속성 중 FireOnce 여부를 변경하는 프러시저이다.

SET_TRIGGER_FIRING_PROPERTY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY
(
    trig_owner IN VARCHAR2,
    trig_name  IN VARCHAR2,
    fire_once  IN BOOLEAN
)
```

- 파라미터

파라미터	설명
trig_owner	TRIGGER 소유자 이름이다.
trig_name	TRIGGER 이름이다.
fire_once	변경할 FireOnce 속성값이다.

10.2.4. WRAP

CREATE OR REPLACE PROCEDURE, FUNCTION, PACKAGE 또는 PACKAGE BODY로 시작하는 DDL 구문을 WRAPPING한 결과를 리턴하는 함수이다.

WRAPPING 결과는 아래와 같이 실행할 수 있다.

```
EXECUTE IMMEDIATE DBMS_DDL.WRAP(ddl)
```

WRAP 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DDL.WRAP
(
    ddl IN VARCHAR2
)
RETURN VARCHAR2
```

- 파라미터

파라미터	설명
ddl	CREATE OR REPLACE PROCEDURE, FUNCTION, PACKAGE 또는 PACKAGE BODY로 시작하는 DDL 구문이다. tbwrap 툴과는 달리 하나의 DDL 구문만 입력할 수 있다.

- 예제

```
create or replace procedure wrap_example
is
    hello_ddl varchar(1000) := 'create or replace procedure hello
                                is
                                  str varchar(10) := ''hello'';
                                begin
                                  DBMS_OUTPUT.PUT_LINE(str);
                                end;';
begin
    EXECUTE IMMEDIATE DBMS_DDL.WRAP(hello_ddl);
end;
/
```

제11장 DBMS_DEBUG

본 장에서는 DBMS_DEBUG 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

11.1. 개요

DBMS_DEBUG 패키지는 두 개의 세션을 이용하여 PSM 프로그램의 디버깅을 할 수 있는 패키지이다.

PSM 프로그램을 수행하는 세션을 타겟 세션(Target session), 디버깅 정보를 타겟 세션에서 받는 세션을 디버그 세션(Debug session)이라고 한다.

다음은 DBMS_DEBUG 패키지 내에 정의된 상수이다.

```
SUCCESS                     CONSTANT BINARY_INTEGER := 0;
ERROR_BOGUS_FRAME           CONSTANT BINARY_INTEGER := 1;
ERROR_NO_DEBUG_INFO         CONSTANT BINARY_INTEGER := 2;
ERROR_NO_SUCH_OBJECT        CONSTANT BINARY_INTEGER := 3;
ERROR_UNKNOWN_TYPE          CONSTANT BINARY_INTEGER := 4;
ERROR_INDEXED_TABLE         CONSTANT BINARY_INTEGER := 5;
ERROR_ILLEGAL_INDEX         CONSTANT BINARY_INTEGER := 6;
ERROR_NULLCOLLECTION        CONSTANT BINARY_INTEGER := 7;
ERROR_NULLVALUE             CONSTANT BINARY_INTEGER := 8;
ERROR_ILLEGAL_VALUE         CONSTANT BINARY_INTEGER := 9;
ERROR_ILLEGAL_NULL          CONSTANT BINARY_INTEGER := 10;
ERROR_VALUE_MALFORMED       CONSTANT BINARY_INTEGER := 11;
ERROR_OTHER                 CONSTANT BINARY_INTEGER := 12;
ERROR_NAME_INCOMPLETE       CONSTANT BINARY_INTEGER := 13;
ERROR_NO_SUCH_BREAKPT       CONSTANT BINARY_INTEGER := 14;
ERROR_IDLE_BREAKPT          CONSTANT BINARY_INTEGER := 15;
ERROR_BAD_HANDLE            CONSTANT BINARY_INTEGER := 16;
ERROR_UNIMPLEMENTED         CONSTANT BINARY_INTEGER := 17;
ERROR_DEFERRED              CONSTANT BINARY_INTEGER := 18;
ERROR_EXCEPTION             CONSTANT BINARY_INTEGER := 19;
ERROR_COMMUNICATION         CONSTANT BINARY_INTEGER := 20;
ERROR_TIMEOUT               CONSTANT BINARY_INTEGER := 21;
ILLEGAL_INIT                CONSTANT BINARY_INTEGER := 22;
PIPE_CREATION_FAILURE       CONSTANT BINARY_INTEGER := 23;
PIPE_SEND_FAILURE           CONSTANT BINARY_INTEGER := 24;
PIPE_RECEIVE_FAILURE        CONSTANT BINARY_INTEGER := 25;
PIPE_DATATYPE_MISMATCH      CONSTANT BINARY_INTEGER := 26;
PIPE_DATA_ERROR             CONSTANT BINARY_INTEGER := 27;
```

- Breakflags
  - 프로토타입

```
BREAK_DEFAULT               CONSTANT BINARY_INTEGER := 0;
BREAK_NEXT_LINE             CONSTANT BINARY_INTEGER := 1;
BREAK_ANY_CALL              CONSTANT BINARY_INTEGER := 2;
BREAK_ANY_RETURN            CONSTANT BINARY_INTEGER := 3;
BREAK_RETURN                CONSTANT BINARY_INTEGER := 4;
BREAK_EXCEPTION             CONSTANT BINARY_INTEGER := 5;
BREAK_HANDLER               CONSTANT BINARY_INTEGER := 6;
ABORT_EXECUTION             CONSTANT BINARY_INTEGER := 7;
```

  - 인자

    DBMS_DEBUG.CONTINUE 함수의 인자로 사용하는 breakflags에 대한 설명이다.

인자	설명
BREAK_DEFAULT	설정된 다음 브레이크 포인트로 진행한다.
BREAK_NEXT_LINE	다음 라인으로 진행한다(step over).
BREAK_ANY_CALL	다음 라인으로 진행한다(step into).
BREAK_ANY_RETURN	현재 entrypoint를 완료하는 시점까지 진행한다(step out).
BREAK_RETURN	현재 버전에서는 사용되지 않는다.
BREAK_EXCEPTION	현재 버전에서는 사용되지 않는다.
BREAK_HANDLER	현재 버전에서는 사용되지 않는다.
ABORT_EXECUTION	프로그램을 즉시 종료한다.

- Information Flags
  - 프로토타입

```
INFO_GET_STACK_DEPTH        CONSTANT BINARY_INTEGER := 0;
INFO_GET_BREAK_POINT        CONSTANT BINARY_INTEGER := 1;
INFO_GET_LINE_INFO          CONSTANT BINARY_INTEGER := 2;
INFO_GET_ALL                CONSTANT BINARY_INTEGER := 3;
```

  - 인자

    rumtime information을 가져올 때 어느 필드를 가져올지에 대한 flag이다.

인자	설명
INFO_GET_STACK_DEPTH	현재 스택의 depth를 가져온다.
INFO_GET_BREAK_POINT	지금 타겟 프로그램이 멈춰있는 브레이크 포인트의 ID를 가져온다.
INFO_GET_LINE_INFO	현재 수행 중인 라인 정보를 가져온다.

인자	설명
INFO_GET_ALL	STACK_DEPTH, _BREAK_POINT, LINE_INFO를 모두 가져온다.

11.2. 타입

본 절에서는 DBMS_DEBUG 패키지에서 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

11.2.1. breakpoint_info

브레이크 포인트의 정보를 가져오기위한 타입이다.

breakpoint_info 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE breakpoint_info IS RECORD
(
    name        VARCHAR2(30),
    owner       VARCHAR2(30),
    line#       BINARY_INTEGER,
    status      BINARY_INTEGER
);
```

- 필드

필드 이름	설명
name	PSM 프로그램의 이름이다.
owner	PSM 프로그램의 owner이다.
line#	라인 번호이다.
status	브레이크 포인트의 현재 상태이다. – breakpoint_status_active – breakpoint_status_disabled

11.2.2. program_info

PSM 프로그램의 위치를 나타내기 위한 타입이다. 프로그램의 특정 라인을 가리켜서 스택 backtrace, break point 설정 등에 사용한다.

program_info 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE program_info IS RECORD
(
    namespace   BINARY_INTEGER,
    name        VARCHAR2(30),
    owner       VARCHAR2(30),
    line#       BINARY_INTEGER,
    entrypointname   VARCHAR2(30)
);
```

- 필드

필드 이름	설명
namespace	현재 버전에서는 사용되지 않는다.
name	PSM 프로그램의 이름이다.
owner	PSM 프로그램의 owner이다.
line#	라인 번호이다.
entrypointname	현재 버전에서는 사용되지 않는다.

11.2.3. runtime_info

현재 수행 중인 PSM 프로그램 정보를 나타내는 타입이다.

runtime_info 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE runtime_info IS RECORD
(
     line#            BINARY_INTEGER,
     terminated       BINARY_INTEGER,
     breakpoint       BINARY_INTEGER,
     stackdepth       BINARY_INTEGER,
     program          program_info
);
```

- 필드

필드 이름	설명
line#	라인 번호이다.

필드 이름	설명
terminated	해당 프로그램이 종료되었는지의 여부이다.
breakpoint	브레이크 포인트의 유일한 ID이다.
stackdepth	현재 수행 중인 프로그램의 스택 depth이다.
program	소스코드 상 프로그램의 위치이다.

11.3. 프러시저와 함수

본 절에서는 DBMS_DEBUG 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

11.3.1. ATTACH_SESSION

디버그 세션에서 타겟 세션에 접속하기 위한 함수이다.

ATTACH_SESSION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.ATTACH_SESSION
(
    debug_session_id  IN VARCHAR2,
    diagnostics       IN BINARY_INTEGER := 0
);
```

- 파라미터

파라미터	설명
debug_session_id	타겟 세션의 INITIALIZE 함수에서 반환하는 타겟 세션의 ID이다. 디버그 세션에서 특정 타겟 세션을 찾을 때 사용한다.
diagnostics	현재 버전에서는 사용되지 않는다.

11.3.2. CONTINUE

breakflags를 이용하여 타겟 세션의 PSM 프로그램을 진행시키거나 완료시킬 수 있는 함수이다.

CONTINUE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.CONTINUE
(
    run_info          IN OUT  runtime_info,
    breakflags        IN      BINARY_INTEGER,
    info_requested    IN      BINARY_INTEGER := NULL
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
run_info	현재 수행 중인 PSM 프로그램의 정보를 설정한다.
breakflags	Continue 함수의 동작을 명시하는 flag이다. 자세한 설명은 "Breakflags"를 참조한다.
info_requested	run_info 인자에 어떤 정보를 요청하는지의 flag이다. 자세한 설명은 "Information Flags"를 참조한다.

- 반환값

반환값	설명
0	성공

11.3.3. DEBUG_OFF

Target session이 디버깅을 종료하는 프러시저이다.

DEBUG_OFF 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.DEBUG_OFF
```

11.3.4. DEBUG_ON

Target session이 디버깅을 시작하는 프러시저이다.

DEBUG_ON 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.DEBUG_ON
(
    no_client_side_plsql_engine     BOOLEAN := TRUE,
    immediate                       BOOLEAN := FALSE
);
```

- 파라미터

파라미터	설명
no_client_side_plsql_engine	현재 버전에서는 사용되지 않는다.
immediate	현재 버전에서는 사용되지 않는다.

11.3.5. DELETE_BREAKPOINT

브레이크 포인트를 삭제하는 함수이다.

DELETE_BREAKPOINT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.DELETE_BREAKPOINT
(
    breakpoint      IN      BINARY_INTEGER
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
breakpoint	삭제할 브레이크 포인트의 번호이다.

- 반환값

반환값	설명
0	성공

11.3.6. DETACH_SESSION

타겟 PSM 프로그램의 디버깅을 종료한다. 이 프러시저는 어느 시점에서든 호출될 수 있고 이 프러시저를 수행하여도 타겟 프로그램이 종료되지 않는다. 그러므로 타겟 프로그램이 수행 중일 경우 이 프러시저를 호출하면 타겟 프로그램은 hang 상태가 되므로 주의해야 한다.

DETACH_SESSION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.DETACH_SESSION
```

11.3.7. DISABLE_BREAKPOINT

기존의 브레이크 포인트를 삭제하지 않고 동작하지 않는 상태로 변경할 수 있는 함수이다.

DISABLE_BREAKPOINT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.DISABLE_BREAKPOINT
(
    breakpoint     IN      BINARY_INTEGER
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
breakpoint	SET_BREAKPOINT 함수에서 설정된 브레이크 포인트의 번호이다.

- 반환값

반환값	설명
0	성공

11.3.8. ENABLE_BREAKPOINT

기존의 동작하지 않는 상태의 브레이크 포인트를 동작하는 상태로 변경할 수 있는 함수이다.

ENABLE_BREAKPOINT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.ENABLE_BREAKPOINT
(
    breakpoint     IN      BINARY_INTEGER
```

```
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
breakpoint	SET_BREAKPOINT 함수에서 설정된 브레이크 포인트의 번호이다.

- 반환값

반환값	설명
0	성공

11.3.9. GET_RUNTIME_INFO

현재 수행 중인 프로그램의 정보를 가져오는 함수이다. info_requested 인자의 값에 따라 가져오는 정보가 달라진다.

GET_RUNTIME_INFO 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.GET_RUNTIME_INFO
(
    info_requested  IN  BINARY_INTEGER,
    run_info        OUT runtime_info
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
info_requested	run_info 인자에 어떤 정보를 요청하는지에 대한 flag이다. 자세한 내용은 "Information Flags"를 참조한다.
run_info	현재 수행 중인 PSM 프로그램의 정보를 설정한다.

- 반환값

반환값	설명
0	성공

11.3.10. GET_VALUE

현재 수행 중인 프로그램에서 변수의 값을 가져오는 함수이다.

GET_VALUE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.GET_VALUE
(
    variable_name  IN  VARCHAR2,
    frame#         IN  BINARY_INTEGER,
    scalar_value   OUT VARCHAR2,
    format         IN  VARCHAR2 := NULL
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
variable_name	변수의 이름이다.
frame#	현재 버전에서는 사용되지 않는다.
scalar_value	현재 버전에서는 사용되지 않는다.
format	현재 버전에서는 사용되지 않는다.

- 반환값

반환값	설명
0	성공
ERROR_ILLEGAL_VALUE	가져오려는 변수가 잘못된 경우에 반환된다.

11.3.11. INITIALIZE

디버깅을 위해서 타겟 세션을 초기화하는 함수이다. 디버그 세션이 접속할 세션의 ID를 반환한다.

INITIALIZE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.INITIALIZE
(
```

```
    debug_session_id  IN VARCHAR2,
    diagnostics       IN BINARY_INTEGER := 0
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
debug_session_id	디버그 세션이 접속할 세션의 ID이다. NULL인 경우 유일한 ID가 생성된다.
diagnostics	현재 버전에서는 사용되지 않는다.

- 반환값

반환값	설명
새롭게 등록된 Debug session ID	디버그 세션에서 접속할 세션의 ID를 반환한다.

11.3.12. PRINT_BACKTRACE

현재 수행 중인 프로그램의 콜스택을 반환하는 프러시저이다. 프로그램이 수행 중일 경우에만 호출 가능하다.

PRINT_BACKTRACE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.PRINT_BACKTRACE
(
    listing IN OUT VARCHAR2
);
```

- 파라미터

파라미터	설명
listing	VARCHAR2 타입의 버퍼에 newline이 포함된 형식으로 출력한다.

11.3.13. SET_BREAKPOINT

현재 수행 중인 타겟 프로그램에 브레이크 포인트를 설정하는 함수이다. 설정된 브레이크 포인트에 도달하면 타겟 프로그램은 수행을 중단하고 대기한다.

SET_BREAKPOINT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.SET_BREAKPOINT
(
    program           IN  program_info,
    line#             IN  BINARY_INTEGER,
    breakpoint#       OUT BINARY_INTEGER,
    fuzzy             IN  BINARY_INTEGER := 0,
    iterations        IN  BINARY_INTEGER := 0
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
program	브레이크 포인트가 설정될 PSM 프로그램의 정보를 설정한다.
line#	브레이크 포인트가 설정될 라인 번호이다.
breakpoint#	설정된 브레이크 포인트의 유일한 ID이다.
fuzzy	현재 버전에서는 사용되지 않는다.
iterations	현재 버전에서는 사용되지 않는다.

- 반환값

반환값	설명
0	성공
ERROR_IDLE_BREAKPT	브레이크 포인트가 설정되지 않은 경우 반환한다.

11.3.14. SHOW_BREAKPOINTS

현재 설정된 브레이크 포인트의 목록을 반환하는 프러시저이다. 2개의 오버로드된 프러시저가 존재한다.

SHOW_BREAKPOINTS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.SHOW_BREAKPOINTS
(
    listing IN OUT VARCHAR2
);
```

```
DBMS_DEBUG.SHOW_BREAKPOINTS
(
    listing    OUT breakpoint_table
);
```

- 파라미터

파라미터	설명
listing	VARCHAR2 타입의 버퍼에 newline이 포함된 형식으로 출력한다. 브레이크 포인트 entry의 indexed-table의 형식으로 출력한다. 브레이크 포인트의 유일한 ID가 index가 된다.

11.3.15. SYNCHRONIZE

타겟 프로그램의 signal을 기다리는 함수이다.

SYNCHRONIZE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_DEBUG.SYNCHRONIZE
(
    run_info         OUT     runtime_info,
    info_requested   IN      BINARY_INTEGER := NULL
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
run_info	현재 수행 중인 PSM 프로그램의 정보를 설정한다.
info_requested	run_info 인자에 어떤 정보를 요청하는지에 대한 flag이다. 자세한 내용은 "Information Flags"를 참조한다.

- 반환값

반환값	설명
0	성공

제12장 DBMS_ERRLOG

본 장에서는 DBMS_ERRLOG 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

12.1. 개요

DBMS_ERRLOG는 에러 로깅 테이블(error logging table)을 생성하는 하나의 프러시저를 제공한다.

일반적인 DML에서는 에러가 발생하면 실패하고 롤백이 이루어진다. 하지만 DML문에 error_logging_clause를 명시하면 에러를 발생시키지 않고 에러의 내용과 데이터를 에러 로깅 테이블에 기록하고 수행을 계속하게 된다.

12.2. 프러시저

12.2.1. CREATE_ERROR_LOG

DML의 에러 로깅에 사용할 테이블을 만드는 함수이다.

CREATE_ERROR_LOG 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_ERRLOG.CREATE_ERROR_LOG
(
    dml_table_name              IN          VARCHAR2,
    err_log_table_name          IN          VARCHAR2  DEFAULT NULL,
    err_log_table_owner         IN          VARCHAR2  DEFAULT NULL,
    err_log_table_space         IN          VARCHAR2  DEFAULT NULL,
    skip_unsupported            IN          BOOLEAN   DEFAULT FALSE
)
RETURN RAW
```

- 파라미터

파라미터	설명
dml_table_name	DML의 대상 테이블명이다. 현재 계정 소유의 테이블이라면 테이블명만으로도 사용 가능하고, 다른 유저 소유의 테이블이라면 USER.TABLE과 같은 형태로 사용할 수 있다.

파라미터	설명
err_log_table_name	에러 로깅 테이블명이다. 명시하지 않으면 ERR$_를 dml_table_name 앞에 붙여서 사용한다.
err_log_table_owner	에러 로깅 테이블을 생성할 스키마명이다.
err_log_table_space	에러 로깅 테이블을 생성할 테이블 스페이스명이다.
skip_unsupported	현재 TRUE 값을 지원하지 않는다. FALSE 값으로 고정되어 있다.

- 예제

```
SQL> create table p (a number primary key);

Table 'P' created.

SQL> insert into p values (1);

1 row inserted.

SQL> insert into p values (2);

1 row inserted.

SQL> insert into p values (3);

1 row inserted.

SQL> insert into p values (4);

1 row inserted.

SQL> create table f (a number references p(a));

Table 'F' created.

SQL> insert into f values (1);

1 row inserted.

SQL> insert into f values (3);

1 row inserted.

SQL> exec dbms_errlog.create_error_log('f');

PSM completed.
```

```
SQL> commit;

Commit completed.

SQL> insert into f (select 1 from dual union all select 5 from dual)
     log errors reject limit 1;

1 row inserted.

SQL> select * from f;

         A
----------
         1
         3
         1

3 rows selected.

SQL> select * from err$_f;

TIB_ERR_NUMBER$
---------------
TIB_ERR_MESG$
--------------------------------------------------------------------------------
TIB_ERR_ROWID$     TIB_ERR_OPTYP$
------------------ --------------------
TIB_ERR_TAG$
--------------------------------------------------------------------------------
A
--------------------------------------------------------------------------------
         -10008
 INTEGRITY constraint ('SYS'.'SYS_CON25700497') violated: primary key not found.

                  I

5

1 row selected.
```

제13장 DBMS_FGA

본 장에서는 DBMS_FGA 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

13.1. 개요

DBMS_FGA패키지는 FGA(Fine-Grained Auditing) policy를 관리하는 프러시저를 제공한다.

FGA는 테이블 혹은 뷰에서 특정 조건을 만족하는 데이터에 대한 SQL 동작을 감시하는 보안 기능이다. 감시 대상 데이터와 SQL 동작은 FGA policy 단위로 정의하고 관리한다. 감시 대상 데이터는 지정된 SQL WHERE 절의 조건식 또는 컬럼 이름을 통해 각각 행 단위 또는 열 단위로 특정할 수 있다. FGA policy가 지원하는 SQL 구문 타입은 SELECT, INSERT, UPDATE, DELETE, MERGE 구문이다.

DBMS_FGA 패키지에 대한 EXECUTE 권한만 있으면 제공하는 프러시저들을 사용할 수 있다.

다음은 DBMS_FGA 패키지 내 정의된 상수이다.

```
EXTENDED    CONSTANT NUMBER := 1;
DB          CONSTANT NUMBER := 2;
XML         CONSTANT NUMBER := 4;

ANY_COLUMNS CONSTANT NUMBER := 0;
ALL_COLUMNS CONSTANT NUMBER := 1;
```

13.2. 프러시저

본 절에서는 DBMS_FGA 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

13.2.1. ADD_POLICY

FGA policy를 생성하는 프러시저이다.

한 오브젝트 당 최대 256개의 policy를 생성할 수 있고, 각 policy 당 최대 1개의 감시 기록을 남긴다.

ADD_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE ADD_POLICY
(
```

```
    object_schema       IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2,
    policy_name         IN VARCHAR2,
    audit_condition     IN VARCHAR2 DEFAULT NULL,
    audit_column        IN VARCHAR2 DEFAULT NULL,
    handler_schema      IN VARCHAR2 DEFAULT NULL,
    handler_module      IN VARCHAR2 DEFAULT NULL,
    enable              IN BOOLEAN  DEFAULT  TRUE,
    statement_types     IN VARCHAR2 DEFAULT 'SELECT',
    audit_trail         IN NUMBER   DEFAULT DB+EXTENDED,
    audit_column_opts   IN NUMBER   DEFAULT ANY_COLUMNS
);
```

- 파라미터

파라미터	설명
object_schema	감시 대상 오브젝트의 스키마 이름이다. (기본값: NULL, NULL은 현재 세션의 스키마 이름을 의미한다.)
object_name	감시 대상 오브젝트의 이름이다.
policy_name	생성할 policy 이름이다. 설정 값은 유일해야 한다.
audit_condition	오브젝트 내 감시 대상 데이터의 행 조건을 특정하기 위한 SQL WHERE 절의 조건식이다. 검색, 삽입, 갱신, 또는 삭제되는 행의 값을 사용하여 계산할 수 있는 조건식이어야 한다. SYS_CONTEXT와 같은 함수를 사용할 수 있다. 조건식을 만족하는 행을 참고하면 감시 대상이 된다. (기본값: NULL, NULL은 TRUE를 의미이다.)
audit_column	오브젝트 내 감시 대상 데이터의 열 조건을 특정하기 위한 컬럼 이름의 리스트이다. (기본값: NULL, NULL은 참조하는 컬럼에 관계 없이 감시한다는 의미이다.)
handler_schema	감시 기록을 남김과 동시에 1번만 수행할 프러시저의 스키마 이름이다. (현재 미지원 기능) (기본값: NULL, NULL은 현재 세션의 스키마 이름을 의미한다.)
handler_module	감시 기록을 남김과 동시에 1번만 수행할 프러시저의 이름이다. 패키지 이름을 포함할 수 있다. (현재 미지원 기능)
enable	생성할 policy의 활성화 여부이다. (기본값: TRUE)
statement_types	생성할 policy가 감시할 SQL 구문 타입이다. – INSERT

파라미터	설명
	– UPDATE – DELETE – SELECT (기본값) MERGE 구문은 내부에서 사용된 SQL 구문 타입에 따라 감시 여부가 결정된다.
audit_trail	감시 기록을 남길 위치와 SQL 구문 및 바인드 변수 기록 여부를 결정한다. – DBMS_FGA.DB : SQL 구문과 바인드 변수를 제외한 감시 기록을 SYS.FGA_LOG$ 테이블에 남긴다. – DBMS_FGA.DB + DBMS_FGA.EXTENDED : SQL 구문과 바인드 변수를 포함한 감시 기록을 SYS.FGA_LOG$ 테이블에 남긴다. (기본값) – DBMS_FGA.XML : SQL 구문과 바인드 변수를 제외한 감시 기록을 XML 파일로 남긴다. (현재 미지원 기능) – DBMS_FGA.XML + DBMS_FGA.EXTENDED : SQL 구문과 바인드 변수를 포함한 감시 기록을 XML 파일로 남긴다. (현재 미지원 기능)
audit_column_opts	SQL 동작이 audit_column 인자에 명시된 모든 컬럼을 참고할 때만 감시 기록을 남길지 일부만 접근해도 감시 기록을 남길지 결정한다. – DBMS_FGA.ALL_COLUMNS : audit_column 인자에 명시된 모든 컬럼을 참고할 때 감시 기록을 남긴다. – DBMS_FGA.ANY_COLUMNS : audit_column 인자에 명시된 일부 컬럼을 참고할 때 감시 기록을 남긴다. (기본값)

- 예제

```
BEGIN
    DBMS_FGA.ADD_POLICY (
       object_schema      =>  'SCOTT',
       object_name        =>  'EMP',
       policy_name        =>  'p1,
       audit_condition    =>  'deptno in (10, 20)',
       audit_column       =>  'sal, mgr',
       handler_schema     =>   NULL,
       handler_module     =>   NULL,
       enable             =>   TRUE,
       statement_types    =>  'INSERT, UPDATE',
       audit_trail        =>  DBMS_FGA.DB + DBMS_FGA.EXTENDED,
```

```
       audit_column_opts  =>  DBMS_FGA.ANY_COLUMNS);
END; /
```

13.2.2. DISABLE_POLICY

FGA policy를 비활성화하는 프러시저이다.

DISABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE DISABLE_POLICY
(
    object_schema       IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2,
    policy_name         IN VARCHAR2
);
```

- 파라미터

파라미터	설명
object_schema	감시 대상 오브젝트의 스키마 이름이다. (기본값: NULL, NULL은 현재 세션의 스키마 이름을 의미한다.)
object_name	감시 대상 오브젝트의 이름이다.
policy_name	비활성화할 policy 이름이다.

- 예제

```
BEGIN
    DBMS_FGA.DISABLE_POLICY (
       object_schema      =>  'SCOTT',
       object_name        =>  'EMP',
       policy_name        =>  'p1');
END; /
```

13.2.3. DROP_POLICY

FGA policy를 삭제하는 프러시저이다.

DROP_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE DROP_POLICY
(
    object_schema       IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2,
    policy_name         IN VARCHAR2
);
```

- 파라미터

파라미터	설명
object_schema	감시 대상 오브젝트의 스키마 이름이다. (기본값: NULL, NULL은 현재 세션의 스키마 이름을 의미한다.)
object_name	감시 대상 오브젝트의 이름이다.
policy_name	삭제할 policy 이름이다.

- 예제

```
BEGIN
    DBMS_FGA.DROP_POLICY (
       object_schema      =>  'SCOTT',
       object_name        =>  'EMP',
       policy_name        =>  'p1');
END; /
```

13.2.4. ENABLE_POLICY

FGA policy를 활성화하는 프러시저이다.

ENABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE ENABLE_POLICY
(
    object_schema       IN VARCHAR2 DEFAULT NULL,
    object_name         IN VARCHAR2,
    policy_name         IN VARCHAR2,
    enable              IN BOOLEAN  DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
object_schema	감시 대상 오브젝트의 스키마 이름이다. (기본값: NULL, NULL은 현재 세션의 스키마 이름을 의미한다.)
object_name	감시 대상 오브젝트의 이름이다.
policy_name	활성화할 policy 이름이다.
enable	policy의 활성화 여부이다. (기본값: TRUE)

- 예제

```
BEGIN
    DBMS_FGA.ENABLE_POLICY (
       object_schema      =>  'SCOTT',
       object_name        =>  'EMP',
       policy_name        =>  'p1',
       enable             =>   TRUE);
END; /
```

제14장 DBMS_FLASHBACK

본 장에서는 DBMS_FLASHBACK 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

14.1. 개요

DBMS_FLASHBACK은 현재 GET_SYSTEM_CHANGE_NUMBER 기능만 구현되어 있는 패키지이다. 추후 FLASHBACK 모드를 관리하는 기능이 추가될 예정이다.

14.2. 프러시저와 함수

본 절에서는 DBMS_FLASHBACK 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

14.2.1. GET_SYSTEM_CHANGE_NUMBER

데이터베이스의 현재 TSN을 반환한다.

GET_SYSTEM_CHANGE_NUMBER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER
```

- 예제

```
DECLARE
   tsn NUMBER;
BEGIN
   tsn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
END;
/
```

제15장 DBMS_INMEMORY

본 장에서는 DBMS_INMEMORY 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

15.1. 개요

DBMS_INMEMORY는 In-Memory Column Store 기능을 사용하기 위한 인터페이스를 제공하는 패키지이다.

15.2. 프러시저

본 절에서는 DBMS_INMEMORY 패키지에서 제공하는 프러시저를 설명한다.

15.2.1. POPULATE

테이블에 대한 populate를 수행할 수 있도록 하는 프러시저이다.

POPULATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_INEMORY.POPULATE
(
    schema_name     IN          VARCHAR2,
    table_name      IN          VARCHAR2,
    subobject_name  IN          VARCHAR2     DEFAULT NULL
);
```

- 파라미터

파라미터	설명
schema_name	populate할 대상 테이블의 스키마 이름이다.
table_name	populate할 대상 테이블 이름이다.
subobject_name	populate할 대상 파티션 혹은 서브파티션 이름이다. NULL인 경우, 모든 파티션 및 서브파티션에 대해 populate를 수행한다.

- 예제

```
CREATE TABLE tibero.t_inmemory (c1 NUMBER) INMEMORY;
INSERT INTO tibero.t_inmemory VALUES (1);
commit;
BEGIN
    DBMS_INMEMORY.POPULATE('TIBERO', 'T_INMEMORY');
END;
/
```

15.2.2. REPOPULATE

테이블에 대한 repopulate를 수행할 수 있도록 하는 프러시저이다.

REPOPULATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_INEMORY.REPOPULATE
(
    schema_name     IN          VARCHAR2,
    table_name      IN          VARCHAR2,
    subobject_name  IN          VARCHAR2     DEFAULT NULL
);
```

- 파라미터

파라미터	설명
schema_name	repopulate할 대상 테이블의 스키마 이름이다.
table_name	repopulate할 대상 테이블 이름이다.
subobject_name	repopulate할 대상 파티션 혹은 서브파티션 이름이다. NULL인 경우, 모든 파티션 및 서브파티션에 대해 repopulate를 수행한다.

- 예제

```
CREATE TABLE tibero.t_inmemory (c1 NUMBER) INMEMORY;
INSERT INTO tibero.t_inmemory VALUES (1);
commit;
BEGIN
    DBMS_INMEMORY.POPULATE('TIBERO', 'T_INMEMORY');
END;
/
INSERT INTO tibero.t_inmemory values (2);
commit;
BEGIN
    DBMS_INMEMORY.REPOPULATE('TIBERO', 'T_INMEMORY');
END;
/
```

15.2.3. FLUSH

populate된 테이블을 In-Memory 공간에서 강제로 비워주도록 하는 프러시저이다.

FLUSH 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_INEMORY.FLUSH
(
    schema_name     IN          VARCHAR2,
    table_name      IN          VARCHAR2,
    subobject_name  IN          VARCHAR2     DEFAULT NULL
);
```

- 파라미터

파라미터	설명
schema_name	flush할 대상 테이블의 스키마 이름이다.
table_name	flush할 대상 테이블 이름이다.
subobject_name	flush할 대상 파티션 혹은 서브파티션 이름이다. NULL인 경우, 모든 파티션 및 서브파티션에 대해 flush를 수행한다.

- 예제

```
CREATE TABLE tibero.t_inmemory (c1 NUMBER) INMEMORY;
INSERT INTO tibero.t_inmemory VALUES (1);
commit;
BEGIN
    DBMS_INMEMORY.POPULATE('TIBERO', 'T_INMEMORY');
    DBMS_INMEMORY.FLUSH('TIBERO', 'T_INMEMORY');
END;
/
```

제16장 DBMS_JAVA

본 장에서는 DBMS_JAVA 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

16.1. 개요

DBMS_JAVA는 데이터베이스에서 사용하는 Java 객체에 접근하기 위한 함수를 제공하는 패키지이다.

16.2. 프러시저

본 절에서는 DBMS_JAVA 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

16.2.1. SET_OUTPUT

Java 객체의 출력을 화면으로 보여주고, 입력 값으로 주어진 크기만큼 버퍼를 사용하는 프러시저이다.

SET_OUTPUT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SET_OUTPUT
(
    bufsize IN NUMBER DEFAULT 2000
);
```

- 파라미터

파라미터	설명
bufsize	출력을 저장할 버퍼의 크기이다.

- 예제

```
call DBMS_JAVA.SET_OUTPUT();
call DBMS_JAVA.SET_OUTPUT(5000);
```

16.3. 함수

본 절에서는 DBMS_JAVA 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

16.3.1. LONGNAME

Java 객체의 완전한 이름을 반환하는 함수이다.

LONGNAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION LONGNAME
(
    shortname   IN   VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
shortname	짧게 줄인 Java 객체의 이름이다.

- 예제

```
SELECT dbms_java.longname(object_name)
FROM user_objects
WHERE object_type='JAVA CLASS' AND status='VALID';
```

16.3.2. SHORTNAME

Java 객체의 이름으로 저장할 수 없는 긴 이름을 짧은 형식으로 바꿔주는 함수이다.

SHORTNAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION SHORTNAME
(
    longname IN VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
longname	Java 객체의 원래 이름이다.

- 예제

```
DECLARE
  short_java_object_name VARCHAR2(128);
  long_laga_object_name VARCHAR2(128) ;
BEGIN
  ...
  short_java_object_name := dbms_java.shortname(long_java_object_name);
END;
```

제17장 DBMS_JOB

본 장에서는 DBMS_JOB 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

17.1. 개요

Tibero에서는 주기적으로 데이터베이스에 추가된 JOB을 검사하여, 사용자가 설정한 시각이 되면 해당하는 JOB을 실행한다.

DBMS_JOB은 PSM에서 사용 가능한 문장을 JOB으로 등록하고, 이 JOB을 실행할 수 있는 연산을 제공하는 패키지이다. DBMS_JOB 패키지 내의 프러시저를 이용하여, JOB을 데이터베이스에 추가하고 바로 실행하거나 정해진 시각에 실행되도록 설정할 수 있다.

다음은 DBMS_JOB 패키지의 특징이다.

- DBMS_JOB 패키지의 사용할 때 DBA 권한은 필요하지 않으며, 추가된 JOB은 오직 JOB의 소유자만 실행하거나 변경할 수 있다.
- JOB을 추가 또는 변경하는 경우 커밋을 실행하지 않아도 자동으로 커밋되며, JOB 내에서 실행한 작업도 자동으로 커밋된다.
- 현재 데이터베이스에 추가된 JOB은 DBA_JOBS, ALL_JOBS, USER_JOBS 뷰를 통해 확인할 수 있다.
- JOB 실행이 실패한 경우에는 재실행되며, 16번 실패하게 되면 해당 JOB은 broken 상태가 된다.
- 실행 중인 JOB을 정지할 수 있는 기능은 제공하지 않는다.

17.2. 프러시저

본 절에서는 DBMS_JOB 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

17.2.1. BROKEN

저장된 JOB의 상태를 정상 또는 broken 상태로 설정하는 프러시저이다. broken되어 있던 JOB을 정상 상태로 만들 경우 JOB의 다음 실행 시각을 설정할 수 있다.

BROKEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.BROKEN
(
    job         IN   BINARY_INTEGER,
    broken      IN   BOOLEAN,
    next_date   IN   DATE DEFAULT SYSDATE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
broken	– TRUE : JOB이 broken된 경우 – FALSE : 정상 상태인 경우
next_date	– broken이 TRUE인 경우 무시한다. – broken이 FALSE인 경우 다음 실행 시각을 설정한다.

- 예제

```
BEGIN
    /* broken된 JOB을 정상 상태로 변경하고, 한 시간 뒤 실행하도록 수정한다. */
    DBMS_JOB.BROKEN(100, false, sysdate + 1/24);
END;
/
```

17.2.2. CHANGE

저장된 JOB의 필드를 변경하는 프러시저이다.

CHANGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.CHANGE
(
    job           IN   BINARY_INTEGER,
    what          IN   VARCHAR2,
    next_date     IN   DATE,
    interval      IN   VARCHAR2,
    instance      IN   BINARY_INTEGER DEFAULT 0,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다.
next_date	JOB을 실행할 다음 시각이다.
interval	다음 JOB이 실행될 시각을 계산하기 위한 연산식이다. 자세한 내용은 “17.2.4. INTERVAL”을 참고한다.
instance	JOB을 수행할 INSTANCE이다. (기본값: 0, ANY_INSTANCE의 의미이다.)
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_JOB.CHANGE(100, null, null, 'sysdate + 1');
END;
/
```

17.2.3. INSTANCE

JOB을 수행할 INSTANCE를 변경하는 프러시저이다.

INSTANCE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.INSTANCE
(
    job           IN   BINARY_INTEGER,
    instance      IN   BINARY_INTEGER,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
instance	JOB을 수행할 INSTANCE이다. (0은 ANY_INSTANCE의 의미이다.)
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_JOB.INSTANCE(1);
END;
/
```

17.2.4. INTERVAL

JOB을 얼마나 자주 실행할지 정하는 파라미터를 변경하는 프러시저이다.

INTERVAL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.INTERVAL
(
    job         IN   BINARY_INTEGER,
    interval    IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
interval	다음에 JOB을 실행할 시각인 next_date를 업데이트하기 위한 연산식이다. NULL 또는 날짜형으로 evaluate될 수 있는 연산식의 문자열이어야 한다. 문자열 입력 시 반드시 작은따옴표(' ')로 감싸야 한다. (최소 초 단위까지 가능) JOB을 실행하기 전에 evaluate된다. JOB이 성공적으로 실행되고, interval의 계산 값이 NULL이면 해당 JOB은 삭제된다. 다음은 interval 파라미터의 사용 예이다. – 예1) `'sysdate + 1'` 하루에 한 번씩 실행한다. – 예2) `'next_day(sysdate,''MONDAY'')'` 매주 월요일에 실행한다. – 예3)

파라미터	설명
	`'null'` 한 번만 실행한다.

- 예제

```
BEGIN
    /* 하루에 한 번씩 실행한다. */
    DBMS_JOB.INTERVAL(100, 'sysdate + 1');
END;
/
```

17.2.5. NEXT_DATE

JOB이 스케줄되어 자동으로 실행될 시각을 변경하는 프러시저이다. 실행된 후에는 interval 값에 의해 업데이트된다.

NEXT_DATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.NEXT_DATE
(
    job         IN   BINARY_INTEGER,
    next_date   IN   DATE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
next_date	JOB이 스케줄되어 실행될 시각이다.

- 예제

```
BEGIN
    /* 10분 후에 실행하도록 변경한다. */
    DBMS_JOB.NEXT_DATE(100, sysdate + 10/24/60);
END;
/
```

17.2.6. REMOVE

데이터베이스에 추가된 JOB을 삭제하는 프러시저이다.

REMOVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.REMOVE
(
    job         IN   BINARY_INTEGER
);
```

- 파라미터

파라미터	설명
job	삭제할 JOB의 번호이다.

- 예제

```
BEGIN
    DBMS_JOB.REMOVE(100);
END;
/
```

17.2.7. RUN

JOB을 현재 세션에서 즉시 실행하는 프러시저이다. JOB이 broken되어 있어도 실행하고, 실행에 성공한 경우 JOB을 정상 상태로 변경한다.

RUN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.RUN
(
    job      IN   BINARY_INTEGER,
    force    IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.

파라미터	설명
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_JOB.RUN(100);
END;
/
```

17.2.8. SUBMIT

데이터베이스에 새로운 JOB을 추가하는 프러시저이다.

SUBMIT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.SUBMIT
(
    job           OUT  BINARY_INTEGER,
    what          IN   VARCHAR2,
    next_date     IN   DATE DEFAULT sysdate,
    interval      IN   VARCHAR2 DEFAULT 'null',
    no_parse      IN   BOOLEAN DEFAULT FALSE,
    instance      IN   BINARY_INTEGER DEFAULT 0,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다.
next_date	JOB을 다음에 실행할 시각이다.
interval	다음 JOB이 실행될 시각을 계산하기 위한 연산식이다. 자세한 내용은 “17.2.4. INTERVAL”을 참고한다.
no_parse	– TRUE : submit을 할 때 JOB을 파싱하지 않고, JOB이 실행될 때 파싱을 하게 된다. 따라서 파싱의 실패 여부가 최초 실행 시점에 보고된다. – FALSE : JOB에 관련된 프러시저를 미리 파싱한다.
instance	JOB을 수행할 INSTANCE이다. (기본값: 0, ANY_INSTANCE의 의미이다.)

파라미터	설명
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
DECLARE
    job_no number;
BEGIN
    DBMS_JOB.SUBMIT(job_no,'dbms_output.put_line(''ok'');', SYSDATE,
                    'SYSDATE + 1');
END;
/
```

17.2.9. WHAT

JOB이 실행하는 작업을 변경하는 프러시저이다.

WHAT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB.WHAT
(
    job      IN   BINARY_INTEGER,
    WHAT     IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다. 항상 세미콜론(;)으로 끝나는 문자열이 와야 한다. 예를 들어 다음과 같은 문자열이 올 수 있다. – 예1) `psm_proc(''abc'', 10);` – 예2) `dbms_output.put_line(''ok'');` – 예3)

파라미터	설명
	declare x number; begin x := x + 1; dbms_outout.put_line(x); end;

- 예제

```
BEGIN
    /* job번호가 100인 job을 psm_proc 프러시저 호출로 대체한다. */
    DBMS_JOB.WHAT(100, 'psm_proc(''abc'', 10);');
END;
/
```

제18장 DBMS_JOB_WITH_NAME

본 장에서는 DBMS_JOB_WITH_NAME 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

18.1. 개요

Tibero에서는 주기적으로 데이터베이스에 추가된 JOB을 검사하여, 사용자가 설정한 시각이 되면 해당하는 JOB을 실행한다.

DBMS_JOB_WITH_NAME은 PSM에서 사용 가능한 문장을 JOB으로 등록하고, 이 JOB을 실행할 수 있는 연산을 제공하는 패키지이다. DBMS_JOB_WITH_NAME 패키지 내의 프러시저를 이용하여, JOB을 데이터베이스에 추가하고 바로 실행하거나 정해진 시각에 실행되도록 설정할 수 있다. 기본적으로 DBMS_JOB 패키지와 동일한 기능을 제공하며, JOB ID 대신 name을 통한 관리를 지원한다.

다음은 DBMS_JOB_WITH_NAME 패키지의 특징이다.

- DBMS_JOB_WITH_NAME 패키지의 사용할 때 DBA 권한은 필요하지 않으며, 추가된 JOB은 오직 JOB의 소유자만 실행하거나 변경할 수 있다.
- JOB을 추가 또는 변경하는 경우 커밋을 실행하지 않아도 자동으로 커밋되며, JOB 내에서 실행한 작업도 자동으로 커밋된다.
- 현재 데이터베이스에 추가된 JOB은 DBA_JOBS_WITH_NAME, ALL_JOBS_WITH_NAME, USER_JOBS_WITH_NAME 뷰를 통해 확인할 수 있다.
- JOB 실행이 실패한 경우에는 재실행되며, 16번 실패하게 되면 해당 JOB은 broken 상태가 된다.
- 실행 중인 JOB을 정지할 수 있는 기능은 제공하지 않는다.

18.2. 프러시저

본 절에서는 DBMS_JOB_WITH_NAME 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

18.2.1. BROKEN

저장된 JOB의 상태를 정상 또는 broken 상태로 설정하는 프러시저이다. broken되어 있던 JOB을 정상 상태로 만들 경우 JOB의 다음 실행 시각을 설정할 수 있다.

BROKEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.BROKEN
(
    job         IN   VARCHAR2,
    broken      IN   BOOLEAN,
    next_date   IN   DATE DEFAULT SYSDATE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
broken	– TRUE : JOB이 broken된 경우 – FALSE : 정상 상태인 경우
next_date	– broken이 TRUE인 경우 무시한다. – broken이 FALSE인 경우 다음 실행 시각을 설정한다.

- 예제

```
BEGIN
    /* broken된 JOB을 정상 상태로 변경하고, 한 시간 뒤 실행하도록 수정한다. */
    DBMS_JOB_WITH_NAME.BROKEN('job_name_here', false, sysdate + 1/24);
END;
/
```

18.2.2. CHANGE

저장된 JOB의 필드를 변경하는 프러시저이다.

CHANGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.CHANGE
(
    job           IN   VARCHAR2,
    what          IN   VARCHAR2,
    next_date     IN   DATE,
```

```
    interval      IN   VARCHAR2,
    instance      IN   BINARY_INTEGER DEFAULT 0,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다.
next_date	JOB을 실행할 다음 시각이다.
interval	다음 JOB이 실행될 시각을 계산하기 위한 연산식이다. 자세한 내용은 “18.2.4. INTERVAL”을 참고한다.
instance	JOB을 수행할 INSTANCE이다. (기본값: 0, ANY_INSTANCE의 의미이다.)
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_JOB_WITH_NAME.CHANGE('job_name_here', null, null, 'sysdate + 1');
END;
/
```

18.2.3. INSTANCE

JOB을 수행할 INSTANCE를 변경하는 프러시저이다.

INSTANCE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.INSTANCE
(
    job           IN   BINARY_INTEGER,
    instance      IN   BINARY_INTEGER,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.

파라미터	설명
instance	JOB을 수행할 INSTANCE이다. (0은 ANY_INSTANCE의 의미이다.)
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_JOB_WITH_NAME.INSTANCE(1);
END;
/
```

18.2.4. INTERVAL

JOB을 얼마나 자주 실행할지 정하는 파라미터를 변경하는 프러시저이다.

INTERVAL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.INTERVAL
(
    job         IN   VARCHAR2,
    interval    IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
interval	다음에 JOB을 실행할 시각인 next_date를 업데이트하기 위한 연산식이다. NULL 또는 날짜형으로 evaluate될 수 있는 연산식의 문자열이어야 한다(최소 초 단위까지 가능). 문자열 입력 시 반드시 작은따옴표(' ')로 감싸야 한다. JOB을 실행하기 전에 evaluate된다. JOB이 성공적으로 실행되고, interval의 계산 값이 NULL이면 해당 JOB은 삭제된다. 다음은 interval 파라미터의 사용 예이다. – 예1) `'sysdate + 1'` 하루에 한 번씩 실행한다.

파라미터	설명
	– 예2) `'next_day(sysdate,''MONDAY'')'` 매주 월요일에 실행한다. – 예3) `'null'` 한 번만 실행한다.

- 예제

```
BEGIN
    /* 하루에 한 번씩 실행한다. */
    DBMS_JOB_WITH_NAME.INTERVAL('job_name_here', 'sysdate + 1');
END;
/
```

18.2.5. NAME

JOB의 이름을 변경하는 프러시저이다. 실행된 후에는 new_name 값에 의해 업데이트된다.

NAME 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.NAME
(
    job         IN   VARCHAR2,
    new_name    IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
new_name	실행할 JOB의 새로운 이름이다.

- 예제

```
BEGIN
    DBMS_JOB_WITH_NAME.NAME('job_name_here','new_name_here');
```

```
END;
/
```

18.2.6. NEXT_DATE

JOB이 스케줄되어 자동으로 실행될 시각을 변경하는 프러시저이다. 실행된 후에는 interval 값에 의해 업데이트된다.

NEXT_DATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.NEXT_DATE
(
    job         IN   VARCHAR2,
    next_date   IN   DATE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
next_date	JOB이 스케줄되어 실행될 시각이다.

- 예제

```
BEGIN
    /* 10분 후에 실행하도록 변경한다. */
    DBMS_JOB_WITH_NAME.NEXT_DATE('job_name_here', sysdate + 10/24/60);
END;
/
```

18.2.7. REMOVE

데이터베이스에 추가된 JOB을 삭제하는 프러시저이다.

REMOVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.REMOVE
(
```

```
    job         IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
job	삭제할 JOB의 이름이다.

- 예제

```
BEGIN
    DBMS_JOB_WITH_NAME.REMOVE('job_name_here');
END;
/
```

18.2.8. RUN

JOB을 현재 세션에서 즉시 실행하는 프러시저이다. JOB이 broken되어 있어도 실행하고, 실행에 성공한 경우 JOB을 정상 상태로 변경한다.

RUN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.RUN
(
    job      IN   VARCHAR2,
    force    IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_JOB_WITH_NAME.RUN('job_name_here');
END;
/
```

18.2.9. SUBMIT

데이터베이스에 새로운 JOB을 추가하는 프러시저이다.

SUBMIT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.SUBMIT
(
    job           OUT  BINARY_INTEGER,
    name          IN   VARCHAR2,
    what          IN   VARCHAR2,
    next_date     IN   DATE DEFAULT sysdate,
    interval      IN   VARCHAR2 DEFAULT 'null',
    no_parse      IN   BOOLEAN DEFAULT FALSE,
    instance      IN   BINARY_INTEGER DEFAULT 0,
    force         IN   BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 번호이다.
name	실행할 JOB의 이름이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다.
next_date	JOB을 다음에 실행할 시각이다.
interval	다음 JOB이 실행될 시각을 계산하기 위한 연산식이다. 자세한 내용은 "18.2.4. INTERVAL"을 참고한다.
no_parse	– TRUE : submit을 할 때 JOB을 파싱하지 않고, JOB이 실행될 때 파싱을 하게 된다. 따라서 파싱의 실패 여부가 최초 실행 시점에 보고된다. – FALSE : JOB에 관련된 프러시저를 미리 파싱한다.
instance	JOB을 수행할 INSTANCE이다. (기본값: 0, ANY_INSTANCE의 의미이다.)
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
DECLARE
    job_no number;
BEGIN
    DBMS_JOB_WITH_NAME.SUBMIT(job_no, 'job_name_here',
'dbms_output.put_line(''ok'');',
```

```
                    SYSDATE, 'SYSDATE + 1');
END;
/
```

18.2.10. WHAT

JOB이 실행하는 작업을 변경하는 프러시저이다.

WHAT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_JOB_WITH_NAME.WHAT
(
    job      IN   VARCHAR2,
    WHAT     IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
job	실행할 JOB의 이름이다.
what	실행할 PL/SQL 프러시저 또는 PSM 문장의 시퀀스이다. 항상 세미콜론(;)으로 끝나는 문자열이 와야 한다. 예를 들어 다음과 같은 문자열이 올 수 있다. – 예1) `psm_proc(''abc'', 10);` – 예2) `dbms_output.put_line(''ok'');` – 예3) `declare x number;` `begin` `x := x + 1;` `dbms_outout.put_line(x);` `end;`

- 예제

```
BEGIN
    /* job이름이 'job_name_here'인 job을 psm_proc 프러시저 호출로 대체한다. */
    DBMS_JOB_WITH_NAME.WHAT('job_name_here', 'psm_proc(''abc'', 10);');
END;
/
```

제19장 DBMS_LOB

본 장에서는 DBMS_LOB 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

19.1. 개요

DBMS_LOB은 BLOB, CLOB 또는 BFILE 타입의 컬럼 데이터에 여러 가지 연산을 제공하는 패키지이다. 또한, DBMS_LOB 패키지 내의 프러시저와 함수를 이용하여 대용량 객체형의 전체 또는 일부에 읽기, 쓰기 등의 작업을 수행할 수 있다.

다음은 DBMS_LOB 패키지를 사용할 때 유의해야 할 사항이다.

- 호출자 권한
  - 이 패키지는 호출자 권한으로 실행된다. 만약 다른 프러시저나 함수에서 호출되는 경우 해당 프러시저나 함수의 실행자 권한으로 실행된다.
- 프러시저와 함수의 파라미터의 길이(length)와 오프셋(offset) 단위
  - 대상(destination) 데이터가 BLOB 타입이면 바이트(Byte), CLOB 타입이면 문자(character) 단위이다. 함수의 반환값도 이와 같다.
- 프러시저와 함수의 파라미터 값을 전달할 때 값의 유효 범위
  - LOB 데이터 내의 오프셋, 길이, 크기 등을 나타내는 값은 항상 1 이상이어야 하며 LOBMAXSIZE 상수의 값보다 크면 안 된다.
  - LOBMAXSIZE 상수는 18446744073709551615(2^64 – 1)의 값을 가진다.
  - LOB 데이터 내의 오프셋, 길이, 크기 등이 1보다 작거나 LOBMAXSIZE보다 큰 값이면 INVALID_ARGVAL 예외 상황이 발생한다.
- CLOB 타입의 컬럼 데이터
  - 이 타입은 항상 유니코드 UTF-16 문자 집합의 문자열이 저장되며, 2bytes로 하나의 문자를 나타낸다.
  - 길이 또는 크기 값은 **LOBMAXSIZE 상수를 2로 나눈 값**보다 작아야 한다.
  - 만약 이 값보다 크면 마찬가지로 INVALID_ARGVAL 예외 상황이 발생한다.
- APPEND, COPY, TRIM, WRITE
  - LOB 데이터를 갱신하려는 프러시저와 함수의 파라미터로 크기와 오프셋 값의 합이 최댓값(LOBMAXSIZE 상수 값)을 초과하면 안 된다. 만약 초과하면 예외 상황이 발생한다. 그리고 LOB 데이터를 갱신

하려면 먼저 그 데이터를 포함하는 로우에 잠금(LOCK)을 설정해야 한다. 왜냐하면 프러시저나 함수는 자동으로 잠금을 설정해 주지 않기 때문이다.

- READ, COMPARE, INSTR, SUBSTR
  - 읽기 전용의 프러시저와 함수는 LOB 데이터의 마지막까지만 읽기를 수행한다.
- COMPARE, INSTR, SUBSTR
  - 문자열 패턴의 작업을 수행하는 함수의 파라미터로 LIKE 함수에서 사용되는 퍼센트(%)와 언더바(_)와 같은 와일드 카드(wild card) 문자를 사용할 수 없다.

LOB 데이터를 수행할 때에는 먼저 대상 LOB 데이터를 OPEN 프러시저를 이용하여 열고, 작업이 끝나면 CLOSE 프러시저를 이용하여 닫는다. 이때 OPEN 프러시저로 열린 LOB 데이터를 닫으면 데이터베이스에 갱신된 내용이 반영된다.

반면에 OPEN 프러시저를 실행하여 열지 않은 LOB 데이터에 대해 갱신 연산을 수행하면 바로 데이터베이스에 반영된다. 대개의 경우 LOB 데이터를 갱신하면 많은 디스크 작업이 수반되므로, 여러 번에 걸쳐 데이터베이스에 반영하는 것보다 한번에 모든 갱신을 데이터베이스에 반영하는 것이 효율적이다.

OPEN 프러시저를 실행하여 LOB 데이터를 연 경우에는 COMMIT 문장을 실행하기 전에 반드시 CLOSE 프러시저를 실행하여 닫아야 한다. 만약 열려있는 LOB 데이터가 있는데 COMMIT을 실행하면, 에러가 발생한다. 열려 있는 LOB 데이터가 있을 때 ROLLBACK을 실행하면 모든 갱신은 취소되고 열려 있는 LOB 데이터에 대한 정보도 없어진다.

다음은 DBMS_LOB 패키지 내에 정의된 상수이다.

- LOBMAXSIZE

  LOB 데이터의 최대 크기이다.

  ```
  LOBMAXSIZE CONSTANT BINARY_INTEGER := 18446744073709551615
  ```

  데이터 타입은 BINARY_INTEGER이고 LOB 데이터의 최대 크기는 18446744073709551615이다.

- LOB_READONLY

  LOB 데이터에 대한 읽기 전용의 사용 여부를 설정하는 모드이다.

  ```
  LOB_READONLY CONSTANT BINARY_INTEGER := 0
  ```

  데이터 타입은 BINARY_INTEGER이고 값이 0이면 읽기 전용으로 설정된다.

- LOB_READWRITE

  LOB 데이터의 읽기 및 쓰기를 설정하는 모드이다.

  ```
  LOB_READWRITE CONSTANT BINARY_INTEGER := 1
  ```

  데이터 타입은 BINARY_INTEGER이고 값이 1이면 읽기 및 쓰기로 설정된다.

- FILE_READONLY

  BFILE 을 열때 사용하는 모드이다.

  ```
  FILE_READONLY CONSTANT BINARY_INTEGER := 0
  ```

  데이터 타입은 BINARY_INTEGER이고, 파일 열기 모드이며, 읽기 모드만 존재한다.

19.2. 예외

다음은 DBMS_LOB 패키지에서 미리 제공된 예외이다. 자세한 내용은 “19.3. 프러시저”와 “19.4. 함수”의 "예외 사항" 항목을 참고한다.

- INVALID_ARGVAL
- ACCESS_ERROR
- NOTEXIST_DIRECTORY
- NOPRIV_DIRECTORY
- INVALID_DIRECTORY
- OPERATION_FAILED
- UNOPENED_FILE
- OPEN_TOOMANY

19.3. 프러시저

본 절에서는 DBMS_LOB 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

19.3.1. APPEND

원본 LOB 데이터의 전체를 대상 LOB 데이터의 끝에 붙여 넣는 프러시저이다.

APPEND 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

    ```
    DBMS_LOB.APPEND
    (
        dest_lob            IN OUT NOCOPY       BLOB,
    ```

```
    src_lob             IN                  BLOB
);
```

– CLOB 타입인 경우

```
DBMS_LOB.APPEND
(
     dest_lob            IN OUT NOCOPY       CLOB,
     src_lob             IN                  CLOB
);
```

CLOB 데이터를 전달하는 경우 dest_lob, src_lob 파라미터의 LOB 데이터의 문자 집합(character set)이 같아야 한다.

- 파라미터

파라미터	설명
dest_lob	대상 LOB Locator이다.
src_lob	원본 LOB Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	대상 또는 원본 LOB 데이터가 NULL인 경우이다.

- 예제

```
DECLARE
    dest_lob CLOB := 'All''s fair in ';
    src_lob CLOB := 'love and war';
BEGIN
    DBMS_LOB.APPEND(dest_lob, src_lob);
    DBMS_OUTPUT.PUT_LINE('Result = ' || dest_lob);
END;
/
Result = All's fair in love and war

PSM completed
SQL>
```

19.3.2. COPY

원본 LOB 데이터의 전체 또는 일부를 대상 LOB 데이터에 복사하는 프러시저이다. 이때 복사할 원본 LOB 데이터의 오프셋과 대상 LOB 데이터의 오프셋을 지정할 수 있다. 만약 대상 LOB 데이터의 오프셋이 대상 LOB 데이터의 길이보다 짧으면 오프셋 위치에 존재하는 이전 데이터는 갱신된다.

이와는 반대로 대상 LOB 데이터 오프셋이 대상 LOB 데이터의 길이보다 길면 중간에 0(BLOB 데이터) 또는 공백(CLOB 데이터)으로 채워진다.

COPY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.COPY
(
    dest_lob        IN OUT NOCOPY   BLOB,
    src_lob         IN              BLOB,
    amount          IN              INTEGER,
    dest_offset     IN              INTEGER := 1,
    src_offset      IN              INTEGER := 1
);
```

  - CLOB 타입인 경우

```
DBMS_LOB.COPY
(
    dest_lob        IN OUT NOCOPY   CLOB,
    src_lob         IN              CLOB,
    amount          IN              INTEGER,
    dest_offset     IN              INTEGER := 1,
    src_offset      IN              INTEGER := 1
);
```

    CLOB 데이터를 전달하는 경우 dest_lob, src_lob 파라미터의 LOB 데이터의 문자 집합이 같아야 한다.

- 파라미터

파라미터	**설명**
dest_lob	대상 LOB Locator이다.
src_lob	원본 LOB Locator이다.
amount	복사할 Byte(BLOB 데이터) 또는 문자(CLOB 데이터)의 개수이다.
dest_offset	대상 LOB 데이터 내의 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))
src_offset	원본 LOB 데이터 내의 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터 중 하나라도 NULL인 경우이다.
INVALID_POS	src_offset, dest_offset의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우이다.
INVALID_LEN	amount의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우이다.

- 예제

```
DECLARE
    dest_lob CLOB := 'It you would be loved, ';
    src_lob CLOB := 'be worthy to be loved';
BEGIN
    DBMS_LOB.COPY(dest_lob, src_lob, length(src_lob),
                  length(dest_lob) + 1, 1);
    DBMS_OUTPUT.PUT_LINE('Result = ' || dest_lob);
END;
/
Result = It you would be loved, be worthy to be loved

PSM completed
SQL>
```

19.3.3. CONVERTTOBLOB

입력으로 받은 CLOB을 BLOB으로 변환하는 프러시저이다.

CONVERTTOBLOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.CONVERTTOBLOB
(
    dest_lob        IN OUT NO COPY  BLOB,
    src_lob         IN              CLOB,
    amount          IN              INTEGER,
    dest_offset     IN OUT          INTEGER,
    src_offset      IN OUT          INTEGER,
    blob_csid       IN              NUMBER,
    lang_context    IN OUT          INTEGER,
    warning         OUT             INTEGER
);
```

- 파라미터

파라미터	설명
dest_lob	대상 LOB Locator이다.
src_lob	원본 LOB Locator이다.
amount	복사할 문자의 개수이다.
dest_offset	대상 LOB 데이터 내의 오프셋이다. 출력은 대상 LOB이 저장된 마지막 주소이다. (단위: Byte(BLOB 데이터))
src_offset	원본 LOB 데이터 내의 오프셋이다. 출력은 원본 LOB을 읽은 마지막 주소이다. (단위: 문자(CLOB 데이터))
blob_csid	CLOB에서 변환된 데이터를 BLOB을 저장할 때 사용할 캐릭터 셋의 ID이다.
lang_context	미사용
warning	미사용

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터 중 하나라도 NULL인 경우이다.

- 예제

```
declare
    src_clob                clob;
    dest_blob               blob;
    dest_clob               clob;

    amount                  integer;
    dest_offset             integer;
    src_offset              integer;
    blob_csid               integer;
    lang_context            integer;
    warning                 integer;

begin
    src_clob := 'abcdefghijklmn';
    dbms_lob.createtemporary(dest_blob, false);
    dbms_lob.createtemporary(dest_clob, false);

    amount := 10;
    dest_offset := 1;
    src_offset := 1;
    blob_csid := 0;
    lang_context := 0;
    warning := 0;
```

```
    dbms_lob.CONVERTTOBLOB(dest_blob, src_clob, amount,
                           dest_offset, src_offset, blob_csid,
                           lang_context, warning);
end;
/

PSM completed
SQL>
```

19.3.4. CONVERTTOCLOB

입력으로 받은 BLOB을 CLOB으로 변환하는 프러시저이다.

CONVERTTOCLOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.CONVERTTOCLOB
(
    dest_lob        IN OUT NO COPY  CLOB,
    src_lob         IN              BLOB,
    amount          IN              INTEGER,
    dest_offset     IN OUT          INTEGER,
    src_offset      IN OUT          INTEGER,
    blob_csid       IN              NUMBER,
    lang_context    IN OUT          INTEGER,
    warning         OUT             INTEGER
);
```

- 파라미터

파라미터	설명
dest_lob	대상 LOB Locator이다.
src_lob	원본 LOB Locator이다.
amount	복사할 문자의 개수이다.
dest_offset	대상 LOB 데이터 내의 오프셋이다. 출력은 대상 LOB이 저장된 마지막 주소이다. (단위: Byte(BLOB 데이터))
src_offset	원본 LOB 데이터 내의 오프셋이다. 출력은 원본 LOB을 읽은 마지막 주소이다. (단위: 문자(CLOB 데이터))
blob_csid	CLOB에서 변환된 데이터를 BLOB로 저장할 때 사용할 캐릭터 셋의 ID이다.
lang_context	미사용
warning	미사용

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터 중 하나라도 NULL인 경우이다.

- 예제

```
declare
    src_clob                clob;
    dest_blob               blob;
    dest_clob               clob;
    amount                  integer;
    dest_offset             integer;
    src_offset              integer;
    blob_csid               integer;
    lang_context            integer;
    warning                 integer;

begin
    src_clob := 'abcdefghijklmn';
    dbms_lob.createtemporary(dest_blob, false);
    dbms_lob.createtemporary(dest_clob, false);

    amount := 10;
    dest_offset := 1;
    src_offset := 1;
    blob_csid := 0;
    lang_context := 0;
    warning := 0;

    dbms_output.put_line('org clob : ' || src_clob);

    dbms_lob.CONVERTTOBLOB(dest_blob, src_clob, amount,
                           dest_offset, src_offset, blob_csid,
                           lang_context, warning);

    dbms_output.put_line('dest offset : ' || dest_offset);
    dbms_output.put_line('srct offset : ' || src_offset);

    amount := 20;
    dest_offset := 1;
    src_offset := 1;
    blob_csid := 0;
    lang_context := 0;
    warning := 0;

    dbms_lob.CONVERTTOCLOB(dest_clob, dest_blob, amount,
```

```
                          dest_offset, src_offset, blob_csid,
                          lang_context, warning);

    dbms_output.put_line('dest clob : ' || dest_clob);
    dbms_output.put_line('dest offset : ' || dest_offset);
    dbms_output.put_line('srct offset : ' || src_offset);
end;
/

org clob : abcdefghijklmn
dest offset : 11
srct offset : 11
dest clob : abcdefghij
dest offset : 11
srct offset : 11

PSM completed.

SQL>
```

19.3.5. CREATETEMPORARY

임시 CLOB 또는 임시 BLOB을 생성하는 프러시저이다.

CREATETEMPORARY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.CREATETEMPORARY
(
    lob         IN OUT NOCOPY   BLOB,
    cache       IN              BOOLEAN,
    dur         IN              PLS_INTEGER := 10
);
```

  - CLOB 타입인 경우

```
DBMS_LOB.CREATETEMPORARY
(
    lob         IN OUT NOCOPY   CLOB,
    cache       IN              BOOLEAN,
    dur         IN              PLS_INTEGER := 10
);
```

- 파라미터

파라미터	설명
lob	대상 LOB Locator이다.
cache	LOB 데이터를 읽을 때 버퍼 캐시(buffer cache)에 저장할지 여부이다.
dur	현재 이 파라미터는 CREATETEMPORARY 프러시저의 수명을 조정하는 기능을 지원하지 않는다. 단, 기본은 세션이 완료되면 CREATETEMPORARY 프러시저가 자동으로 삭제된다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	캐시 파라미터가 NULL인 경우이다.

- 예제

```
DECLARE
    lob_1 CLOB;
    lob_2 CLOB := 'tibero';
BEGIN
    DBMS_LOB.CREATETEMPORARY(lob_1, false);
    DBMS_LOB.APPEND(lob_1, lob_2);
    DBMS_OUTPUT.PUT_LINE(lob_1);
END;
/
tibero

PSM completed
SQL>
```

19.3.6. ERASE

LOB 데이터의 일부 또는 전체를 삭제하는 프러시저이다. 삭제된 영역은 0(BLOB 데이터) 또는 공백(CLOB 데이터)으로 채워진다. 대상 LOB 데이터의 길이가 오프셋과 삭제할 영역 크기의 합보다 짧다면 실제로 삭제된 영역의 크기는 삭제할 영역으로 입력한 크기보다 작을 수 있다. 실제로 삭제된 영역의 크기는 amount 파라미터로 출력된다.

ERASE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.ERASE
(
    lob         IN OUT NOCOPY   BLOB,
    amount      IN OUT NOCOPY   INTEGER,
    offset      IN              INTEGER := 1
);
```

– CLOB 타입인 경우

```
DBMS_LOB.ERASE
(
    lob         IN OUT NOCOPY   CLOB,
    amount      IN OUT NOCOPY   INTEGER,
    offset      IN              INTEGER := 1
);
```

- 파라미터

파라미터	설명
lob	대상 LOB Locator이다.
amount	삭제할 Byte 또는 문자 개수이다.
offset	삭제할 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터가 하나라도 NULL인 경우이다.
INVALID_POS	offset의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우이다.
INVALID_LEN	amount의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우이다.

- 예제

```
DECLARE
    lob CLOB := 'Tmaxsoft Tibero';
    amount NUMBER := 7;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Length of original LOB = ' || length(lob));
    DBMS_LOB.ERASE(lob, amount, 9);
    DBMS_OUTPUT.PUT_LINE('Value of erased LOB = ' || lob);
    DBMS_OUTPUT.PUT_LINE('Length of erased LOB = ' || length(lob));
END;
/
Length of original LOB = 15
```

```
Value of erased LOB = Tmaxsoft
Length of erased LOB = 15

PSM completed
SQL>
```

19.3.7. FILECLOSE

File Locator로 파일을 닫는다.

FILECLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.FILECLOSE
(
    file_loc     IN OUT NOCOPY   BFILE
);
```

- 파라미터

파라미터	설명
file_loc	닫을 파일의 Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.

- 예제

  FILEOPEN의 예제를 참조한다.

19.3.8. FILECLOSEALL

현 세션에 DBMS_LOB패키지를 사용해 열려있는 모든 파일을 닫다.

FILECLOSEALL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.FILECLOSEALL;
```

- 예제

```
BEGIN
    DBMS_LOB.FILECLOSEALL;
END;
/
```

19.3.9. FILEGETNAME

File Locator로부터 디렉터리명과 파일명을 가져온다.

FILEGETNAME 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.FILEGETNAME
(
    file_loc     IN   BFILE,
    dir_alias    OUT  VARCHAR2,

    filename     OUT  VARCHAR2
);
```

- 파라미터

파라미터	설명
file_loc	파일의 Locator이다.
dir_alias	File Locator로부터 읽어올 디렉터리명이다.
filename	File Locator로부터 읽어올 파일명이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.

- 예제

```
declare
    f1 bfile;
    dir_alias varchar(128);
    filename varchar(128);
begin
   f1 := bfilename('BFILETESTDIR', 'test001.bin');
   dbms_lob.filegetname(f1, dir_alias, filename);
```

```
   dbms_output.put_line(dir_alias);
   dbms_output.put_line(filename);
end;
/

BFILETESTDIR
test001.bin
```

19.3.10. FILEOPEN

File Locator로 읽기 전용, 바이너리 모드로 파일을 연다.

FILEOPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.FILEOPEN
(
    file_loc     IN OUT NOCOPY   BFILE,
    open_mode    IN              BINARY_INTEGER := file_readonly
);
```

- 파라미터

파라미터	설명
file_loc	열 파일의 Locator이다.
open_mode	파일 읽기 모드(file_readonly) 파라미터만 허용한다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.
INVALID_ARGVAL	open_mode에 file_readonly 값외 다른 값이 온 경우이다.
INVALID_DIRECTORY	File locator에 지정된 디렉터리가 유효하지 않나 존재하지 않는 경우이다.
NOEXIST_DIRECTORY	File locator에 지정된 디렉터리 경로가 존재하 않는 경우이다.
OPERATION_FAILED	디렉터리 경로에 파일이 존재하지 않거나, 파일 열수 없는 경우이다.

- 예제

```
create directory BFILETESTDIR as '/mybasepath/bfiletest/';

declare
    f utl_file.file_type;
```

```
    buffer raw(16) := '000102030405060708090A0B0C0D0E0F';
begin
    f := utl_file.fopen('BFILETESTDIR', 'test001.bin', 'wb');
    utl_file.put_raw(f, buffer);
    utl_file.fclose(f);
end;
/

declare
    f1 bfile;
    amount number;
    buffer raw(16);
    result raw(16) := '000102030405060708090A0B0C0C0E0F';
begin
   f1 := bfilename('BFILETESTDIR', 'test001.bin');

   dbms_lob.fileopen(f1, dbms_lob.file_readonly);

   amount := 16;
   dbms_lob.read(f1, amount, 1, buffer);

   dbms_output.put_line(amount);
   dbms_output.put_line(buffer);

   dbms_lob.fileclose(f1);
end;
/

16
000102030405060708090A0B0C0D0E0F
```

19.3.11. FREETEMPORARY

이미 생성된 임시 BLOB 또는 임시 CLOB을 삭제하는 프러시저이다.

FREETEMPORARY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.FREETEMPORARY
(
```

```
    lob         IN OUT NOCOPY   BLOB
);
```

– CLOB 타입인 경우

```
DBMS_LOB.FREETEMPORARY
(
    lob         IN OUT NOCOPY   CLOB
);
```

- 파라미터

파라미터	설명
lob	대상 LOB Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터가 NULL인 경우이다.

- 예제

```
DECLARE
    lob CLOB;
BEGIN
    DBMS_LOB.CREATETEMPORARY(lob, false);
    DBMS_LOB.FREETEMPORARY(lob);
END;
/

PSM completed
SQL>
```

19.3.12. LOADFROMFILE

파일 내용을 읽어서 LOB 데이터에 쓰기를 한다. 파일은 열어져 있어야 하며, LOB도 생성된 상태여야 한다. 본 프러시저에서는 문자집합 변환이 이뤄지지 않는다(CLOB의 데이터 내부 형태는 UCS2 인코딩이므로, CLOB으로 읽어들일 파일 인코딩또한 UCS2이어야 한다. 그렇지 않는 경우 동작은 정의되지 않는다).

LOADFROMFILE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

– BLOB 타입인 경우

```
DBMS_LOB.LOADFROMFILE
(
    dest_lob    IN OUT NOCOPY BLOB,
    src_file    IN            BFILE,
    amount      IN            INTEGER,
    dest_offset IN            INTEGER := 1,
    src_offset  IN            INTEGER := 1
);
```

– CLOB 타입인 경우

```
DBMS_LOB.LOADFROMFILE
(
    dest_lob    IN OUT NOCOPY CLOB,
    src_file    IN            BFILE,
    amount      IN            INTEGER,
    dest_offset IN            INTEGER := 1,
    src_offset  IN            INTEGER := 1
);
```

- 파라미터

파라미터	설명
dest_lob	대상 LOB Locator이다.
src_file	LOB으로 읽어들일 bfile 타입의 파일 핸들이다. BLOB 및 CLOB 버전의 경우 문자집합의 변환이 없다(CLOB 버전을 사용하는 경우 입력 텍스트 파일은 UCS2 인코딩만을 지원한다).
amount	파일로부터 읽어들일 크기이다. BLOB 버전의 경우에는 바이트를 의미하고, CLOB 버전인 경우는 문자를 의미한다(문자는 UCS2 인코딩을 가정하므로, 1문자는 2Bytes이다).
dest_offset	쓸 LOB의 offset이다. (기본값: 1, 1 베이스)
src_offset	읽을 파일의 offset이다. (기본값: 1, 1 베이스)

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터가 적절하지 않는 경우 발생한다.

- 예제

```
SQL> create directory BFILETESTDIR as '/home/tests/test_bfile/';

Directory 'BFILETESTDIR' created.
```

```
SQL> DECLARE
    f UTL_FILE.FILE_TYPE;

    buffer RAW(16) := '000102030405060708090A0B0C0C0E0F';
begin
    f := UTL_FILE.FOPEN('BFILETESTDIR', 'test001.bin', 'wb');
    UTL_FILE.PUT_RAW(f, buffer);
    UTL_FILE.FCLOSE(f);
end;
/

PSM completed.

SQL> SET SERVEROUTPUT ON

SQL> DECLARE
    f1 BFILE;
    bl1 BLOB := HEXTORAW('AAAAAA');
    amount NUMBER;
    buffer RAW(16);
    result RAW(16) := '000102030405060708090A0B0C0C0E0F';
    buffer2 RAW(32);
    result2 RAW(32) := '000102030405060708090A0B0C0D0E0F';
begin
   f1 := BFILENAME('BFILETESTDIR', 'test001.bin');

   DBMS_LOB.FILEOPEN(f1, DBMS_LOB.FILE_READONLY);

   amount := 16;
   DBMS_LOB.LOADFROMFILE(bl1, f1, amount);
   DBMS_LOB.READ(bl1, amount, 1, buffer);

   DBMS_OUTPUT.PUT_LINE(amount);
   DBMS_LOB.FILECLOSE(f1);
end;
/
16

PSM completed.

SQL> DECLARE
    lob CLOB;
BEGIN
    DBMS_LOB.CREATETEMPORARY(lob, false);
    DBMS_LOB.FREETEMPORARY(lob);
END;
```

```
/

PSM completed.

SQL> DECLARE
    f UTL_FILE.FILE_TYPE;

    buffer RAW(16) := '00610062006300640065006600670068';
BEGIN
    f := UTL_FILE.FOPEN('BFILETESTDIR', 'ucs2_file.bin', 'wb');
    UTL_FILE.PUT_RAW(f, buffer);
    UTL_FILE.FCLOSE(f);
END;
/

PSM completed.

SQL> SET SERVEROUTPUT ON
SQL> DECLARE
    f1 BFILE;
    bl1 CLOB := 'init';
    amount NUMBER;
BEGIN
   F1 := BFILENAME('BFILETESTDIR', 'ucs2_file.bin');

   DBMS_LOB.FILEOPEN(f1, DBMS_LOB.FILE_READONLY);

   AMOUNT := 8;
   DBMS_LOB.LOADFROMFILE(bl1, f1, amount);

   DBMS_OUTPUT.PUT_LINE (bl1);

   DBMS_LOB.FILECLOSE(f1);
END;
/
abcdefgh

PSM completed.

SQL>
```

19.3.13. LOADBLOBFROMFILE

파일 내용을 읽어서, BLOB 데이터에 쓰기를 한다. 파일은 열어져 있어야 하며, LOB도 생성된 상태여야 한다.

LOADBLOBFROMFILE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.LOADBLOBFROMFILE
(
    dest_lob    IN OUT NOCOPY BLOB,
    src_file    IN            BFILE,
    amount      IN            INTEGER,
    dest_offset IN OUT        INTEGER,
    src_offset  IN OUT        INTEGER
);
```

- 파라미터

파라미터	설명
dest_lob	대상 LOB Locator이다.
src_file	LOB으로 읽어들일 bfile 타입의 파일 핸들이다. 문자집합의 변환이 없다.
amount	파일로부터 읽어들일 크기(바이트)이다.
dest_offset	쓸 LOB의 offset이다(1 베이스). 프러시저가 완료된 후 offset은 쓰여진 만큼 수정된다.
src_offset	읽을 파일의 offset이다(1 베이스). 프러시저가 완료된 후 offset은 읽혀진 만큼 수정된다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터가 적절하지 않는 경우 발생한다.

- 예제

```
SQL> SET SERVEROUTPUT ON

SQL> create directory BFILETESTDIR as '/home/tests/test_bfile/';

Directory 'BFILETESTDIR' created.

SQL> DECLARE
    f UTL_FILE.FILE_TYPE;

    buffer RAW(16) := '000102030405060708090A0B0C0C0E0F';
begin
    f := UTL_FILE.FOPEN('BFILETESTDIR', 'test001.bin', 'wb');
    UTL_FILE.PUT_RAW(f, buffer);
    UTL_FILE.FCLOSE(f);
```

```
end;
/

PSM completed.

SQL> DECLARE
    f1 BFILE;
    bl1 BLOB := HEXTORAW('AAAAAA');
    amount NUMBER;
    buffer RAW(16);
    result RAW(16) := '000102030405060708090A0B0C0C0E0F';
    buffer2 RAW(32);
    result2 RAW(32) := '000102030405060708090A0B0C0D0E0F';
    src_ofst NUMBER;
    dst_ofst NUMBER;
begin
   f1 := BFILENAME('BFILETESTDIR', 'test001.bin');

   DBMS_LOB.FILEOPEN(f1, dbms_lob.file_readonly);

   amount := 16;
    src_ofst := 1;
    dst_ofst := 1;

   DBMS_LOB.LOADBLOBFROMFILE(bl1, f1, amount, dst_ofst,  src_ofst);
   DBMS_LOB.READ(bl1, amount, 1, buffer);

   DBMS_OUTPUT.PUT_LINE(amount);
   DBMS_OUTPUT.PUT_LINE(src_ofst);
   DBMS_OUTPUT.PUT_LINE(dst_ofst);


   DBMS_LOB.FILECLOSE(f1);
end;
/
16
17
17

PSM completed.

SQL>
```

19.3.14. OPEN

File Locator로 읽기 전용, 바이너리 모드로 파일을 연다.

OPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.OPEN
(
    file_loc     IN OUT NOCOPY   BFILE,
    open_mode    IN              BINARY_INTEGER := file_readonly
);
```

- 파라미터

파라미터	설명
file_loc	열 파일의 Locator이다.
open_mode	파일 읽기 모드(file_readonly) 파라미터만 허용한다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.
INVALID_ARGVAL	open_mode에 file_readonly 값외 다른 값이 온 우이다.
INVALID_DIRECTORY	File locator에 지정된 디렉터리가 유효하지 않나 존재하지 않는 경우이다.
NOEXIST_DIRECTORY	File locator에 지정된 디렉터리 경로가 존재하 않는 경우이다.
OPERATION_FAILED	디렉터리 경로에 파일이 존재하지 않거나, 파일 열수 없는 경우이다.

- 예제

```
create directory BFILETESTDIR as '/mybasepath/bfiletest/';

declare
    f utl_file.file_type;

    buffer raw(16) := '000102030405060708090A0B0C0D0E0F';
begin
    f := utl_file.fopen('BFILETESTDIR', 'test001.bin', 'wb');
    utl_file.put_raw(f, buffer);
    utl_file.fclose(f);
end;
/
```

```
declare
    f1 bfile;
    amount number;
    buffer raw(16);
    result raw(16) := '000102030405060708090A0B0C0C0E0F';
begin
   f1 := bfilename('BFILETESTDIR', 'test001.bin');

   dbms_lob.open(f1, dbms_lob.file_readonly);

   amount := 16;
   dbms_lob.read(f1, amount, 1, buffer);

   dbms_output.put_line(amount);
   dbms_output.put_line(buffer);

   dbms_lob.close(f1);
end;
/

16
000102030405060708090A0B0C0D0E0F
```

19.3.15. READ

대상 LOB 데이터의 일부 또는 전체를 읽어 출력 파라미터의 버퍼에 저장하는 프러시저이다. 만약 읽기를 시작하는 오프셋과 읽을 크기의 합이 대상 LOB 데이터의 크기보다 커서 LOB 데이터의 끝을 지나가게 되면, 실제 읽은 데이터의 크기는 파라미터로 주어진 읽을 크기보다 작을 수 있다. 이때 실제로 읽어온 데이터의 크기는 입출력 파라미터 amount에 저장되어 반환된다. 만약 읽을 오프셋의 위치가 대상 LOB 데이터의 크기보다 크다면 NO_DATA_FOUND 예외 상황이 발생한다.

CLOB 데이터로부터 읽은 데이터를 클라이언트에 전송하는 경우 클라이언트의 문자 집합으로 자동 변환된다. 이 경우 실제로 읽어온 데이터와 달라질 수 있다.

READ 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.READ
(
    lob_loc     IN              BLOB,
    amount      IN OUT NOCOPY   BINARY_INTEGER,
    offset      IN              INTEGER,
```

```
    buffer      OUT             RAW
);
```

– CLOB 타입인 경우

```
DBMS_LOB.READ
(
    lob_loc     IN              CLOB,
    amount      IN OUT NOCOPY   BINARY_INTEGER,
    offset      IN              INTEGER,
    buffer      OUT             VARCHAR2
);
```

– CLOB 타입인 경우

```
DBMS_LOB.READ
(
    file_loc    IN              CLOB,
    amount      IN OUT NOCOPY   BINARY_INTEGER,
    offset      IN              INTEGER,
    buffer      OUT             VARCHAR2
);
```

- 파라미터

파라미터	**설명**
lob_loc	읽을 대상 LOB Locator이다.
file_loc	읽을 대상 File Locator이다.
amount	읽을 크기를 입력하고 실제로 읽어온 크기를 출력한다. (단위: Byte(BLOB 데이터, FILE 데이터) 또는 문자(CLOB 데이터) 개수)
offset	읽을 대상 LOB 데이터 내의 오프셋이다. (단위: Byte(BLOB 데이터, FILE 데이터) 또는 문자(CLOB 데이터))
buffer	읽을 데이터를 저장하는 출력 버퍼이다.

- 예외 상황

예외 상황	**설명**
VALUE_ERROR	입력 파라미터 중 하나라도 NULL인 경우이다.
INVALID_POS	offset의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우이다.
INVALID_LEN	– amount의 값이 1보다 작거나 MAXBUFSIZE보다 큰 경우이다. – amount의 값이 버퍼의 크기보다 큰 경우이다.

예외 상황	설명
NO_DATA_FOUND	offset이 대상 LOB 데이터의 크기보다 큰 경우이다.
UNOPENED_FILE	열리지 않은 File Locator로 연산을 수행하려 한 경우이다.

- 예제

```
DECLARE
    lob CLOB := 'TIBERO fighting!!!';
    buffer VARCHAR2(256);
    amount BINARY_INTEGER := 8;
BEGIN
    DBMS_LOB.READ(lob, amount, 8, buffer);
    DBMS_OUTPUT.PUT_LINE('Value to be read = ' || buffer);
END;
/
Value to be read = fighting

PSM completed
SQL>
```

19.3.16. TRIM

대상 LOB 데이터의 길이를 지정된 길이로 설정하는 함수이다. BLOB 데이터의 경우 Byte 단위로, CLOB 데이터의 경우 문자 단위로 길이를 지정한다. 지정된 길이를 넘는 데이터는 소멸된다.

만약 이 프러시저를 길이가 0인 LOB 데이터로 실행하면 아무런 값이 반환되지 않는다.

TRIM 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.TRIM
(
    lob         IN OUT NOCOPY   BLOB,
    newlen      IN              INTEGER
);
```

  - CLOB 타입인 경우

```
DBMS_LOB.TRIM
(
    lob         IN OUT NOCOPY   CLOB,
    newlen      IN              INTEGER
);
```

- 파라미터

파라미터	설명
lob	대상 LOB Locator이다.
newlen	새롭게 지정된 LOB 데이터의 길이이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력 파라미터 중 하나라도 NULL인 경우이다.
INVALID_LEN	newlen 값이 0보다 작거나 LOBMAXSIZE보다 큰 경우이다.

- 예제

```
DECLARE
    lob CLOB := 'A pity beyond all telling is in the heart of love';
BEGIN
    DBMS_LOB.TRIM(lob, 25);
    DBMS_OUTPUT.PUT_LINE('Value = ' || lob);
    DBMS_OUTPUT.PUT_LINE('Length = ' || length(lob));
END;
/
Value = A pity beyond all telling
Length = 25

PSM completed
SQL>
```

19.3.17. WRITE

대상 LOB 데이터의 지정된 오프셋 위치에 주어진 데이터를 지정된 크기만큼 저장하는 프러시저이다. 새로운 데이터가 저장되는 위치에 존재하는 이전 데이터는 소멸된다. BLOB 데이터에 대한 오프셋 및 크기는 Byte 단위이며, CLOB 데이터에 대해서는 문자 단위이다. 주어진 데이터의 실제 크기보다 지정된 크기가 큰 경우에는 에러가 발생하며, 지정된 크기가 작은 경우에는 지정된 크기만큼만 저장된다. 지정된 오프셋이 대상 LOB 데이터의 크기보다 큰 경우에는 중간에 0(BLOB 데이터) 또는 공백(CLOB 데이터)으로 채워진다.

이 프러시저를 CLOB 데이터에 대하여 실행할 때 CLOB 데이터의 문자 집합과 저장할 데이터의 문자 집합은 같아야 한다. 이 프러시저를 클라이언트 쪽에서 호출하는 경우 저장할 데이터가 클라이언트 쪽의 문자 집합으로부터 CLOB 데이터 문자 집합으로 변환된 후에 저장된다.

WRITE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.WRITE
(
    lob             IN OUT NOCOPY   BLOB,
    amount          IN              BINARY_INTEGER,
    offset          IN              INTEGER,
    buffer          IN              RAW
);
```

  - CLOB 타입인 경우

```
DBMS_LOB.WRITE
(
    lob             IN OUT NOCOPY   CLOB,
    amount          IN              BINARY_INTEGER,
    offset          IN              INTEGER,
    buffer          IN              VARCHAR
);
```

- 파라미터

파라미터	**설명**
lob	대상 LOB Locator이다.
amount	저장할 데이터의 크기이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))
offset	데이터를 저장할 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))
buffer	저장할 데이터이다.

- 예외 상황

예외 상황	**설명**
VALUE_ERROR	파라미터 중 하나라도 NULL인 경우이다.
INVALID_POS	offset의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우이다.
INVALID_LEN	amount의 값이 1보다 작거나 MAXBUFSIZE보다 큰 경우이다.

- 예제

```
DECLARE
    lob CLOB;
    buffer VARCHAR2(100);
BEGIN
    DBMS_LOB.CREATETEMPORARY(lob, false);
    buffer := 'Love is friendship set on fire';
    DBMS_LOB.WRITE(lob, length(buffer), 1, buffer);
    DBMS_OUTPUT.PUT_LINE(lob);
END;
/
Love is friendship set on fire

PSM completed
SQL>
```

19.3.18. WRITEAPPEND

대상 LOB 데이터의 끝에 주어진 데이터를 지정된 크기만큼 저장하는 프러시저이다. 이 프러시저는 WRITE 프러시저의 파라미터 오프셋을 대상 LOB 데이터의 길이로 설정한 것과 같다. BLOB 데이터의 크기는 Byte 단위이며, CLOB 데이터는 문자 단위이다. 주어진 데이터의 실제 크기보다 지정된 크기가 큰 경우에는 에러가 발생하며, 지정된 크기가 작은 경우에는 지정된 크기만큼만 저장된다.

이 프러시저를 CLOB 데이터에 대하여 실행할 때에 CLOB 데이터의 문자 집합과 저장할 데이터의 문자 집합은 같아야 한다. 이 프러시저를 클라이언트 쪽에서 호출하는 경우 저장할 데이터가 클라이언트 쪽의 문자 집합에서 CLOB 데이터 문자 집합으로 변환된 후에 저장된다.

WRITEAPPEND 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.WRITEAPPEND
(
    lob         IN OUT NOCOPY   BLOB,
    amount      IN              BINARY_INTEGER,
    buffer      IN              RAW
);
```

  - CLOB 타입인 경우

```
DBMS_LOB.WRITEAPPEND
(
    lob         IN OUT NOCOPY   CLOB,
    amount      IN              BINARY_INTEGER,
```

```
    buffer      IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
lob	대상 LOB Locator이다.
amount	저장할 데이터의 크기이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터) 개수)
buffer	저장할 데이터이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))

- 예외 상황

예외 상황	설명
VALUE_ERROR	파라미터 중 하나라도 NULL인 경우이다.
INVALID_LEN	amount 값이 1보다 작거나 MAXBUFSIZE보다 큰 경우이다.

- 예제

```
DECLARE
    lob CLOB := 'Parting is such ';
    buffer VARCHAR2(100) := 'sweet sorrow';
BEGIN
    DBMS_LOB.WRITEAPPEND(lob, length(buffer), buffer);
    DBMS_OUTPUT.PUT_LINE('Result = ' || lob);
END;
/
Result = Parting is such sweet sorrow

PSM completed
SQL>
```

19.4. 함수

본 절에서는 DBMS_LOB 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

19.4.1. CLOSE

File Locator로 파일을 닫는다.

CLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.CLOSE
(
    file_loc     IN OUT NOCOPY   BFILE
);
```

- 파라미터

파라미터	설명
file_loc	닫을 파일의 Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.

- 예제

  OPEN의 예제를 참조한다.

19.4.2. COMPARE

두 개의 LOB 데이터의 전체 또는 일부를 비교하는 함수이다. 같은 타입의 LOB 데이터 간에만 비교가 가능하다.

COMPARE 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.COMPARE
(
    lob_1           IN          BLOB,
    lob_2           IN          BLOB,
    amount          IN          INTEGER := 4294967295,
    offset_1        IN          INTEGER := 1,
    offset_2        IN          INTEGER := 1
)
RETURN INTEGER;
```

  - CLOB 타입인 경우

```
DBMS_LOB.COMPARE
(
    lob_1           IN          CLOB,
    lob_2           IN          CLOB,
    amount          IN          INTEGER := 4294967295,
    offset_1        IN          INTEGER := 1,
    offset_2        IN          INTEGER := 1
)
RETURN INTEGER;
```

CLOB 데이터를 전달하는 경우 lob_1, lob_2 파라미터의 LOB 데이터의 문자 집합이 같아야 한다.

- 파라미터

파라미터	설명
lob_1	비교 대상인 첫 번째 LOB Locator이다.
lob_2	비교 대상인 두 번째 LOB Locator이다.
amount	비교할 Byte의 크기(BLOB 데이터) 또는 문자(CLOB 데이터) 개수이다.
offset_1	비교를 시작할 첫 번째 LOB 데이터 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))
offset_2	비교를 시작할 두 번째 LOB 데이터 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))

- 반환값

반환값	설명
0	lob_1, lob_2의 LOB 데이터가 동일한 경우에 반환한다.
N < 0 or N > 0	lob_1, lob_2의 LOB 데이터가 동일하지 않은 경우에 반환한다.
NULL	– amount가 1보다 작은 경우에 반환한다. – amount가 LOBMAXSIZE보다 큰 경우에 반환한다. – offset_1 또는 offset_2가 1보다 작은 경우에 반환한다. – offset_1 또는 offset_2가 LOBMAXSIZE보다 큰 경우에 반환한다.

- 예제

```
DECLARE
    lob_1 CLOB := 'abcdefgh';
    lob_2 CLOB := 'abcdefgg';
BEGIN
    IF DBMS_LOB.COMPARE(lob_1, lob_2) = 0 then
```

```
        DBMS_OUTPUT.PUT_LINE('LOB_1 equals LOB_2');
    ELSE
        DBMS_OUTPUT.PUT_LINE('LOB_1 does not equals LOB_2');
    END IF;
END;
/
LOB_1 does not equals LOB_2

PSM completed
SQL>
```

19.4.3. FILEEXISTS

File Locator로 해당 파일이 존재하는지 여부를 확인한다.

FILEEXISTS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.FILEEXISTS
(
    file_loc     IN OUT NOCOPY   BFILE,
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
file_loc	존재 여부를 확인할 File Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.
INVALID_DIRECTORY	File locator에 지정된 디렉터리가 유효하지 않나 존재하지 않는 경우이다.
NOEXIST_DIRECTORY	File locator에 지정된 디렉터리 경로가 존재하 않는 경우이다.

- 예제

```
create directory BFILETESTDIR as '/mybasepath/bfiletest/';

declare
    f utl_file.file_type;
```

```
    buffer raw(16) := '000102030405060708090A0B0C0D0E0F';
begin
    f := utl_file.fopen('BFILETESTDIR', 'test001.bin', 'wb');
    utl_file.put_raw(f, buffer);
    utl_file.fclose(f);
end;
/

declare
    f1 bfile;
    f2 bfile;
begin
   f1 := bfilename('BFILETESTDIR', 'test001.bin');
   f2 := bfilename('BFILETESTDIR', 'test002.bin');

   dbms_output.put_line(dbms_lob.fileexists(f1));
   dbms_output.put_line(dbms_lob.fileexists(f2));
end;
/

1
0
```

19.4.4. FILEISOPEN

File Locator로 해당 파일에 세션에서 열어져 있는지를 확인한다.

FILEISOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.FILEISOPEN
(
    file_loc     IN OUT NOCOPY   BFILE,
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
file_loc	파일의 Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.

- 예제

```
create directory BFILETESTDIR as '/mybasepath/bfiletest/';

declare
    f utl_file.file_type;

    buffer raw(16) := '000102030405060708090A0B0C0D0E0F';
begin
    f := utl_file.fopen('BFILETESTDIR', 'test001.bin', 'wb');
    utl_file.put_raw(f, buffer);
    utl_file.fclose(f);
end;
/

declare
    f1 bfile;
begin
   f1 := bfilename('BFILETESTDIR', 'test001.bin');

   dbms_lob.open(f1, dbms_lob.file_readonly);
   dbms_output.put_line(dbmb_lob.fileisopen(f1));
   dbms_lob.close(f1);
   dbms_output.put_line(dbmb_lob.fileisopen(f1));
end;
/

1
0
```

19.4.5. GETCHUNKSIZE

대상 LOB 객체의 chunk 크기를 반환하는 함수이다. chunk란 Tibero에서 내부적으로 Lob 데이터를 나누어 관리하는 단위이며, chunk의 배수로 read / write 작업을 요청할 경우 서버의 처리 로직이 최적화되어 수행된다.

GETCHUNKSIZE 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.GETCHUNKSIZE
(
    lob         IN          BLOB
)
RETURN INTEGER;
```

– CLOB 타입인 경우

```
DBMS_LOB.GETCHUNKSIZE
(
    lob         IN          CLOB
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
lob	대상 LOB 데이터이다.

- 반환값

반환값	설명
INTEGER	GETCHUNKSIZE 함수가 성공적으로 종료된 경우에 반환한다.

- 예제

```
create table CHUNKY(a clob, b blob);
insert into CHUNKY select DBMS_RANDOM.STRING('L', 10000), empty_blob() from dual;

DECLARE
    temp_clob CLOB;
    temp_blob BLOB;
    cons_clob CLOB;
    cons_blob BLOB;
BEGIN
    select a into cons_clob from test;
    select b into cons_blob from test;
    dbms_output.put_line(dbms_lob.getchunksize(temp_clob));
    dbms_output.put_line(dbms_lob.getchunksize(temp_blob));
    dbms_output.put_line(dbms_lob.getchunksize(cons_clob));
    dbms_output.put_line(dbms_lob.getchunksize(cons_blob));
END;
/
8140
8140
8140
8140
```

```
PSM completed
SQL>
```

19.4.6. GETLENGTH

대상 LOB 데이터의 길이를 반환하는 함수이다. 이때 길이는 입력 LOB 데이터의 타입에 따라 Byte 또는 문자 단위의 값으로 반환된다.

GETLENGTH 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.GETLENGTH
(
        lob_loc        IN          BLOB
)
RETURN INTEGER;
```

  - CLOB 타입인 경우

```
DBMS_LOB.GETLENGTH
(
        lob_loc        IN          CLOB
)
RETURN INTEGER;
```

  - BFILE 타입인 경우

```
DBMS_LOB.GETLENGTH
(
        file_loc        IN          BFILE
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
lob_loc	대상 LOB Locator이다.
file_loc	대상 File Locator이다.

- 반환값

반환값	설명
0	빈 LOB 데이터인 경우에 반환한다.
NULL	입력 LOB 데이터의 파라미터가 NULL인 경우에 반환한다.

COPY, ERASE, WRITE 등의 프러시저에 의해 채워진 0(BLOB 데이터) 또는 공백(CLOB 데이터) 문자도 대상 LOB 데이터의 길이에 포함된다.

- 예제

```
DECLARE
    lob CLOB := 'architecture';
BEGIN
    DBMS_OUTPUT.PUT_LINE(DBMS_LOB.GETLENGTH(lob));
END;
/
12

PSM completed
SQL>
```

19.4.7. INSTR

대상 LOB 데이터 내에서 주어진 패턴이 n 번째로 나타나는 오프셋을 반환하는 함수이다. 패턴을 탐색하는 위치는 대상 LOB 데이터의 처음일 수도 있고 입력 파라미터로 지정된 위치일 수도 있다. 이때 탐색할 대상 패턴은 LIKE 연산자에서 사용되는 퍼센트(%) 또는 언더바(_)와 같은 와일드 카드 문자 등을 포함할 수 없다.

INSTR 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.INSTR
(
    lob         IN      BLOB,
    pattern     IN      RAW,
    offset      IN      INTEGER := 1,
    nth         IN      INTEGER := 1
)
RETURN INTEGER;
```

  - CLOB 타입인 경우

```
DBMS_LOB.INSTR
(
    lob         IN      CLOB,
    pattern     IN      VARCHAR,
    offset      IN      INTEGER := 1,
    nth         IN      INTEGER := 1
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
lob	패턴을 탐색할 대상 LOB Locator이다.
pattern	탐색할 패턴이다.
offset	LOB 데이터 내의 탐색을 시작할 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))
nth	탐색할 패턴의 개수이다. 1 이상의 값을 가지며, 1이면 첫 번째 패턴의 오프셋을 반환한다.

- 반환값

반환값	설명
INTEGER	패턴과 일치하는 부분을 찾은 경우 해당 패턴이 시작되는 오프셋을 반환한다. 만약 찾지 못한 경우에는 0을 반환한다.
NULL	– IN 파라미터 중 하나라도 NULL인 경우에 반환한다. – offset이 1보다 작거나 LOBMAXSIZE보다 큰 경우에 반환한다. – nth가 1보다 작거나 LOBMAXSIZE보다 큰 경우에 반환한다.

- 예제

```
DECLARE
    lob CLOB := 'Corporate floor';
    result NUMBER;
BEGIN
    result := DBMS_LOB.INSTR(lob, 'or', 3, 2);
    DBMS_OUTPUT.PUT_LINE('Result offset = ' || result);
END;
/
Result offset = 14
```

```
PSM completed
SQL>
```

19.4.8. ISOPEN

File Locator로 해당 파일에 세션에서 열어져 있는지를 확인한다.

ISOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOB.ISOPEN
(
    file_loc     IN OUT NOCOPY   BFILE,
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
file_loc	파일의 Locator이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	File locator가 NULL인 경우이다.

- 예제

```
create directory BFILETESTDIR as '/mybasepath/bfiletest/';

declare
    f utl_file.file_type;

    buffer raw(16) := '000102030405060708090A0B0C0D0E0F';
begin
    f := utl_file.fopen('BFILETESTDIR', 'test001.bin', 'wb');
    utl_file.put_raw(f, buffer);
    utl_file.fclose(f);
end;
/

declare
    f1 bfile;
begin
```

```
   f1 := bfilename('BFILETESTDIR', 'test001.bin');

   dbms_lob.open(f1, dbms_lob.file_readonly);
   dbms_output.put_line(dbmb_lob.isopen(f1));
   dbms_lob.close(f1);
   dbms_output.put_line(dbmb_lob.isopen(f1));
end;
/

1
0
```

19.4.9. ISTEMPORARY

주어진 LOB이 임시 LOB인지 아닌지의 여부를 확인하는 함수이다.

ISTEMPORARY 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.ISTEMPORARY
(
    lob         IN          BLOB
)
RETURN INTEGER;
```

  - CLOB 타입인 경우

```
DBMS_LOB.ISTEMPORARY
(
    lob         IN          CLOB
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
lob	대상 LOB Locator이다.

- 반환값

반환값	설명
1	임시 LOB인 경우에 반환한다.

반환값	설명
0	임시 LOB이 아닌 경우에 반환한다.

- 예제

```
DECLARE
    lob CLOB;
BEGIN
    DBMS_LOB.CREATETEMPORARY(lob, false);
    IF DBMS_LOB.ISTEMPORARY(lob) = 1THEN
        DBMS_OUTPUT.PUT_LINE('TRUE');
    ELSE
        DBMS_OUTPUT.PUT_LINE('FALSE');
    END IF;
END;
/
TRUE

PSM completed
SQL>
```

19.4.10. SUBSTR

대상 LOB 데이터의 지정된 오프셋의 위치로부터 지정된 크기만큼의 데이터를 반환하는 함수이다. 반환될 최대 크기는 32767bytes이며, CLOB 데이터의 경우 반환될 최대 문자 개수는 32767을 2로 나눈 값이다. CLOB 데이터에 대하여 이 함수를 호출하면 같은 문자 집합의 문자열이 반환된다.

이 함수는 하나의 파라미터라도 NULL이 넘겨지거나, 파라미터 amount 또는 offset의 값이 1보다 작거나 LOBMAXSIZE보다 큰 경우에 NULL을 반환한다.

이 함수를 CLOB 데이터에 대해 클라이언트 쪽에서 호출한 경우 만약 클라이언트 쪽에 설정된 문자 집합가 CLOB 데이터 문자 집합과 다르다면 데이터 전송 중에 자동적으로 문자 집합의 변환이 일어난다.

SUBSTR 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB 타입인 경우

```
DBMS_LOB.SUBSTR
(
    lob         IN          BLOB,
    amount      IN          INTEGER := 32767,
    offset      IN          INTEGER := 1
```

```
)
RETURN RAW;
```

– CLOB 타입인 경우

```
DBMS_LOB.SUBSTR
(
    lob         IN          CLOB,
    amount      IN          INTEGER := 32767,
    offset      IN          INTEGER := 1
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
lob	대상 LOB 데이터이다.
amount	읽을 데이터의 크기이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터) 개수)
offset	데이터를 읽을 LOB 데이터 내의 오프셋이다. (단위: Byte(BLOB 데이터) 또는 문자(CLOB 데이터))

- 반환값

반환값	설명
RAW/VARCHAR2 데이터	SUBSTR 함수가 성공적으로 종료된 경우에 반환한다.
NULL	– 입력 파라미터가 하나라도 NULL인 경우에 반환한다. – amount가 1보다 작거나 32767bytes보다 큰 경우에 반환한다. – offset이 1보다 작거나 LOBMAXSIZE보다 큰 경우에 반환한다.

- 예제

```
DECLARE
    lob CLOB := 'Your friend is too young';
    buffer VARCHAR2(100);
BEGIN
    buffer := DBMS_LOB.SUBSTR(lob, 6, 6);
    DBMS_OUTPUT.PUT_LINE('My favorite word is ' || UPPER(buffer));
END;
/
My favorite word is FRIEND
```

```
PSM completed
SQL>
```

제20장 DBMS_LOCK

본 장에서는 DBMS_LOCK 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

20.1. 개요

DBMS_LOCK은 유저에게 Lock 관리 및 SLEEP 기능을 제공하는 패키지이다. 이 패키지는 디폴트로 관리자만 수행 가능하다.

DBMS_LOCK 패키지에서는 아래 테이블에 정의된 상수를 사용한다.

이름	별칭	타입	값
NS_MODE	Null	INTEGER	1
SS_MODE	Sub shared	INTEGER	2
SX_MODE	Sub eXclusive or Row Exclusive Mode	INTEGER	3
S_MODE	Shared or Row Exclusive Mode or Intended Exclusive	INTEGER	4
SSX_MODE	Shared Sub eXclusive or Shared Row Exclusive Mode	INTEGER	5
X_MODE	Exclusive	INTEGER	6

다른 스레드에서 lock을 잡고 있는 경우, 아래 호환성 표에 따라 요청 결과가 결정된다.

모드	NL 요청	SS 요청	SX 요청	S 요청	SSX 요청	X 요청
NL	성공	성공	성공	성공	성공	성공
SS	성공	성공	성공	성공	성공	실패
SX	성공	성공	성공	성공	실패	실패
S	성공	성공	실패	성공	실패	실패
SSX	성공	성공	실패	실패	실패	실패
X	성공	실패	실패	실패	실패	실패

상수 MAXWAIT으로 설정할 경우 영원히 기다린다.

```
maxwait constant pls_integer := 32767;
```

20.2. 프러시저

본 절에서는 DBMS_LOCK 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

20.2.1. ALLOCATE_UNIQUE

지정된 lockname에 Lock ID를 할당하는 프러시저이다. Lock ID 값의 범위는 1073741824 에서 1999999999 까지이다. 사용자에게 숫자가 아닌 이름을 통해서 관리를 쉽게하기 위한 목적으로 제공한다.

ALLOCATE_UNIQUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOCK.ALLOCATE_UNIQUE
(
    lockname            IN          VARCHAR2,

    lockhandle          OUT         VARCHAR2,

    expiration_secs     IN          PLS_INTEGER     DEFAULT 864000
)
```

- 파라미터

파라미터	설명
lockname	유일한 lockhandle 값을 생성할 Lock 이름을 지정한다.
lockhandle	생성된 Lock ID에 대한 handle을 리턴받는다. 이 handle 값을 통해 REQEST, CONVERT, RELEASE 프러시저를 호출할 수 있다. Lock ID를 잘못 사용했을 때의 문제를 방지하기 위해 Lock ID를 직접 리턴하는 대신 VARCHAR2 (128) 변수로 설정된 handle 값을 리턴한다. handle은 한 세션에서만 유효하므로 다른 세션에서 해당 값을 사용해서는 안 된다.
expiration_secs	DBMS_LOCK_ALLOCATED 테이블을 재활용하기 위한 유효 시간을 초단위로 지정한 값이다. (기본값: 10일) 마지막 ALLOCATE_UNIQUE 프러시저를 호출한 시점이후로 지나간 시간을 계산한다. **DBMS_LOCK_ALLOCATED 테이블에서 Lock을 직접 삭제해서는 안 된다. _DBMS_LOCK_REUSE 초기화 파라미터를 Y로 설정한 경우에만 동작한다.**

- 예제

```
DECLARE
    lock_handle VARCHAR2(128);
BEGIN
    DBMS_LOCK.ALLOCATE_UNIQUE('my_lock', lock_handle);
END;
/
```

20.2.2. CONVERT

기존 Lock 모드에서 다른 Lock 모드로 변경하기 위한 프러시저이다.

CONVERT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOCK.CONVERT
(
    id         IN INTEGER ||
    lockhandle IN VARCHAR2,
    lockmode   IN INTEGER,
    timeout    IN NUMBER DEFAULT MAXWAIT
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
id or lockhandle	Lock 모드를 변경하기 위한 Lock ID 또는 handle이다.
lockmode	변경할 새로운 Lock 모드이다.
timeout	Lock 모드 변경을 위해 기다릴 시간을 설정한다. 만약 Lock을 얻을 수 없다면 timeout 에러 값을 리턴한다. (단위: 초)

- 반환값

반환값	설명
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Owner error
5	Illegal lock handle

- 예제

```
BEGIN
    DBMS_LOCK.CONVERT(lock_handle, DBMS_LOCK.X_MODE, 600);
END;
/
```

20.2.3. RELEASE

기존 잡았던 Lock을 명시적으로 해제하기 위한 프러시저이다. 기본적으로 Lock은 세션이 종료되면 자동적으로 해제된다.

RELEASE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOCK.RELEASE
(
    id         IN INTEGER ||
    lockhandle IN VARCHAR2
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
id or lockhandle	Lock을 해제하기 위한 Lock ID 또는 handle이다.

- 반환값

반환값	설명
0	Success
3	Parameter error
4	Owner error
5	Illegal lock handle

- 예제

```
BEGIN
    DBMS_LOCK.RELEASE(lock_handle);
END;
/
```

20.2.4. REQUEST

원하는 Lock 모드를 지정하여 Lock을 요청하기 위한 프러시저이다.

REQUEST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOCK.REQUEST
(
    id                 IN  INTEGER ||
    lockhandle         IN  VARCHAR2,
    lockmode           IN  INTEGER DEFAULT X_MODE,
    timeout            IN  INTEGER DEFAULT MAXWAIT,
    release_on_commit  IN  BOOLEAN DEFAULT FALSE
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
id or lockhandle	Lock을 요청하기 위한 Lock ID 또는 handle이다.
lockmode	요청하는 Lock 모드이다.
timeout	Lock 얻기 위해 기다릴 시간이다. (단위: 초) 만약 Lock을 얻을 수 없다면 timeout 에러 값을 리턴한다.
release_on_commit	– TRUE : commit이나 rollback할 때 Lock이 해제된다. – FALSE : 명시적으로 Lock을 해제하거나 세션이 종료될 때까지 Lock을 해제하지 않는다.

- 반환값

반환값	설명
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Already owned by id or lockhandle
5	Illegal lock handle

- 예제

```
BEGIN
    DBMS_LOCK.REQUEST(lock_handle, DBMS_LOCK.S_MODE, DBMS_LOCK.MAXWAIT, TRUE);
END;
/
```

20.2.5. SLEEP

현재 세션을 일정 시간만큼 대기시킨다.

SLEEP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_LOCK.SLEEP
(
    seconds        IN         NUMBER
)
```

- 파라미터

파라미터	설명
seconds	대기 시간(초)이다.

- 예제

```
DECLARE
   second NUMBER;
BEGIN
   second := 0.11;
   DBMS_LOCK.SLEEP(second);
END;
/
```

제21장 DBMS_METADATA

본 장에서는 DBMS_METADATA 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

21.1. 개요

DBMS_METADATA는 DB 객체를 재생성하기 위해 사용할 수 있는 XML 문서나 DDL 구문을 획득하거나 수정할 수 있는 패키지이다.

- 타입

```
CREATE OR REPLACE TYPE sys.ku$_parsed_item AS OBJECT (
    item            VARCHAR2(30),
    value           VARCHAR2(4000),
    object_row      NUMBER );
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_parsed_item FOR sys.ku$_parsed_item;

CREATE OR REPLACE TYPE sys.ku$_parsed_items IS TABLE OF sys.ku$_parsed_item;
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_parsed_items FOR sys.ku$_parsed_items;

CREATE OR REPLACE TYPE sys.ku$_ddl AS OBJECT (
    ddlText        CLOB,
    parsedItems sys.ku$_parsed_items );
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_ddl FOR sys.ku$_ddl;

CREATE OR REPLACE TYPE sys.ku$_ddls IS TABLE OF sys.ku$_ddl;
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_ddls FOR sys.ku$_ddls;

CREATE OR REPLACE TYPE sys.ku$_multi_ddl AS OBJECT (
    object_row     NUMBER,
    ddls           sys.ku$_ddls );
/
```

```
CREATE OR REPLACE PUBLIC SYNONYM ku$_multi_ddl FOR sys.ku$_multi_ddl;

CREATE OR REPLACE TYPE sys.ku$_multi_ddls IS TABLE OF sys.ku$_multi_ddl;
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_multi_ddls FOR sys.ku$_multi_ddls;

CREATE OR REPLACE TYPE sys.ku$_ErrorLine IS OBJECT (
    errorNumber    NUMBER,
    errorText      VARCHAR2(2000) );
/

CREATE OR REPLACE TYPE sys.ku$_SubmitResult AS OBJECT (
    ddl          sys.ku$_ddl,
    errorLines   sys.ku$_ErrorLines );
/

CREATE OR REPLACE TYPE sys.ku$_SubmitResults IS TABLE OF sys.ku$_SubmitResult;
/

CREATE OR REPLACE PUBLIC SYNONYM ku$_SubmitResults FOR sys.ku$_SubmitResults;
```

21.2. 프러시저와 함수

21.2.1. ADD_TRANSFORM

CONVERT, FETCH_DDL, FETCH_XML 등에 적용될 transform을 추가한다.

ADD_TRANSFORM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.ADD_TRANSFORM (
   handle       IN NUMBER,
   name         IN VARCHAR2,
   encoding     IN VARCHAR2 DEFAULT NULL,
   object_type  IN VARCHAR2 DEFAULT NULL)
 RETURN NUMBER;
```

- 파라미터

파라미터	설명
handle	OPEN 또는 OPENW 함수를 통해 획득한 handle이다.

파라미터	설명
name	DBMS_METADATA에서 제공하는 transform 이름이다. 현재는 'DDL', 'MODIFY'를 제공함 – 'DDL' : XML을 DDL구문으로 변환하는 transform – 'MODIFY' : XML 내용을 수정하는 transform
encoding	현재 사용 불가
object_type	현재 사용 불가

- 반환값

  OPEN, OPENW의 반환값과는 다른 transform handle(SET_TRANSFORM_PARAM, SET_REMAP_PARAM의 파라미터로 사용됨)

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음

- 예제

```
SQL> declare

   2    handle number;

   3    transform_handle number;

   4 begin

   5    handle := dbms_metadata.open('TABLE');

   6    transform_handle := dbms_metadata.add_transform(handle, 'DDL');

   7    dbms_metadata.close(handle);

   8 end;

   9 /

PSM completed.
```

21.2.2. CLOSE

OPEN, OPENW의 반환값인 handle을 정리한다.

CLOSE 프러시저 의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.CLOSE (
   handle  IN NUMBER);
```

- 파라미터

파라미터	설명
handle	OPEN 또는 OPENW 함수를 통해 획득한 handle이다.

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid

- 예제

```
SQL> declare

   2     handle number;

   3 begin

   4     handle := dbms_metadata.open('TABLE');

   5     dbms_metadata.close(handle);


   6 end;

   7 /

PSM completed.
```

21.2.3. CONVERT

입력된 XML 문서를 DDL로 변환한다.

CONVERT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.CONVERT (
  handle   IN NUMBER,
  document IN XMLType,
  result   IN OUT NOCOPY CLOB);
```

- 파라미터

파라미터	설명
handle	OPENW 함수를 통해 획득한 handle이다.
document	입력 XML 문서이다.
result	결과 DDL이다.

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid
INCONSISTENT_OPERATION	handle에 'DDL' transform이 추가되어 있지 않음

- 예제

```
SQL> create table t (c1 number);


Table 'T' created.

SQL> declare


   2     handle  number;

   3     handlew number;

   4     transform_handle number;

   5     ret_xml xmltype;

   6     ret_ddl clob;

   7 begin
```

```
  8     handle := dbms_metadata.open('TABLE');
  9     dbms_metadata.set_filter(handle, 'NAME', 'T');
 10     ret_xml := dbms_metadata.fetch_xml(handle);
 11     dbms_metadata.close(handle);
 12
 13     handlew := dbms_metadata.openw('TABLE');
 14     transform_handle := dbms_metadata.add_transform(handlew, 'DDL');
 15     dbms_metadata.convert(handlew, ret_xml, ret_ddl);
 16     dbms_metadata.close(handlew);
 17 end;
 18 /
PSM completed.
```

21.2.4. FETCH_DDL

DDL을 출력한다.

FETCH_DDL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.FETCH_DDL (
   handle  IN NUMBER)
RETURN sys.ku$_ddls;
```

- 파라미터

파라미터	설명
handle	OPEN 함수를 통해 획득한 handle이다.

- 반환값

  결과 DDL(모든 DDL이 return된 후에는 NULL)

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid
INCONSISTENT_OPERATION	handle에 'DDL' transform이 추가되어 있지 않음

- 예제

```
SQL> create table t (c1 number);


Table 'T' created.

SQL> declare


   2     handle  number;

   3     transform_handle number;

   4     ret_ddl sys.ku$_ddls;

   5 begin

   6     handle := dbms_metadata.open('TABLE');

   7     transform_handle := dbms_metadata.add_transform(handle, 'DDL');

   8     dbms_metadata.set_filter(handle, 'NAME', 'T');

   9     loop

  10         ret_ddl := dbms_metadata.fetch_ddl(handle);

  11         exit when ret_ddl is null;

  12     end loop;

  13     dbms_metadata.close(handle);

  14 end;

  15 /

PSM completed.
```

21.2.5. FETCH_XML

XML을 출력한다.

FETCH_XML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.FETCH_XML (
   handle  IN NUMBER)
RETURN XMLType;
```

- 파라미터

파라미터	설명
handle	OPEN 함수를 통해 획득한 handle이다.

- 반환값

  결과 XML(모든 XML이 return된 후에는 NULL)

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid
INCONSISTENT_OPERATION	handle에 'DDL' transform이 추가되어 있음

- 예제

```
SQL> create table t (c1 number);

Table 'T' created.

SQL> declare


   2     handle  number;

   3     ret_xml xmltype;

   4 begin

   5     handle := dbms_metadata.open('TABLE');

   6     dbms_metadata.set_filter(handle, 'NAME', 'T');
```

```
   7     loop
   8         ret_xml := dbms_metadata.fetch_xml(handle);
   9         exit when ret_xml is null;
  10     end loop;
  11     dbms_metadata.close(handle);
  12 end;
  13 /
PSM completed.
```

21.2.6. GET_DDL

객체의 DDL을 출력한다.

GET_DDL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.GET_DDL (
    object_type         IN VARCHAR2,
    name                IN VARCHAR2,
    schema              IN VARCHAR2 DEFAULT NULL,
    version             IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model               IN VARCHAR2 DEFAULT 'TIBERO',
    transform           IN VARCHAR2 DEFAULT 'DDL')
RETURN CLOB;
```

- 파라미터

파라미터	설명
object_type	출력하고자 하는 객체 타입이다. OPEN, OPENW에서 사용하는 object_type 파라미터와 동일한 값을 사용하면 된다.
name	객체의 이름이다.
schema	객체의 스키마 이름이다.
version	현재 사용 불가

파라미터	설명
model	현재 사용 불가
transform	현재 사용 불가

- 반환값

  결과 DDL

- 예외 상황

예외 상황	설명
OBJECT_NOT_FOUND	객체를 찾지 못함

- 예제

```
SQL> set long 90000;
SQL> create table t (c1 number);

Table 'T' created.

SQL> select dbms_metadata.get_ddl('TABLE', 'T') from dual;

DBMS_METADATA.GET_DDL('TABLE','T')
--------------------------------------------------------------------------------
CREATE TABLE "TIBERO"."T" (
"C1" NUMBER)

 PCTFREE 10 INITRANS 2 NOCOMPRESS LOGGING
 STORAGE(INITIAL 131072 NEXT 131072 MINEXTENTS 1 MAXEXTENTS 4294967295)
 TABLESPACE "USR"

1 row selected.
```

21.2.7. GET_DEPENDENT_DDL

종속 객체의 DDL을 출력한다.

GET_DEPENDENT_DDL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.GET_DEPENDENT_DDL (

    object_type         IN VARCHAR2,

    base_object_name    IN VARCHAR2,
```

```
    base_object_schema  IN VARCHAR2 DEFAULT NULL,
    version             IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model               IN VARCHAR2 DEFAULT 'TIBERO',
    transform           IN VARCHAR2 DEFAULT 'DDL',
    object_count        IN NUMBER   DEFAULT 10000)
RETURN CLOB;
```

- 파라미터

파라미터	설명
object_type	출력하고자 하는 객체 타입이다. OPEN, OPENW에서 사용하는 object_type 파라미터와 동일한 값을 사용하면 된다.
base_object_name	부모 객체의 이름이다.
base_object_schema	부모 객체의 스키마 이름이다.
version	현재 사용 불가
model	현재 사용 불가
transform	현재 사용 불가
object_count	현재 사용 불가

- 반환값

  결과 DDL

- 예외 상황

예외 상황	설명
OBJECT_NOT_FOUND	객체를 찾지 못함

- 예제

```
SQL> set long 90000;
SQL> create table t (c1 number);

Table 'T' created.

SQL> create index t_idx on t(c1);
```

```
Index 'T_IDX' created.

SQL> select dbms_metadata.get_dependent_ddl('INDEX', 'T') from dual;

DBMS_METADATA.GET_DEPENDENT_DDL('INDEX','T')
--------------------------------------------------------------------------------
CREATE INDEX "TIBERO"."T_IDX" ON "TIBERO"."T" ("C1")  PCTFREE 10 INITRANS 2  LOG
GING
 STORAGE(INITIAL 131072 NEXT 131072 MINEXTENTS 1 MAXEXTENTS 4294967295)
 TABLESPACE "USR"

1 row selected.
```

21.2.8. GET_DEPENDENT_XML

종속 객체의 XML을 출력한다.

GET_DEPENDENT_XML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.GET_DEPENDENT_XML (

    object_type         IN VARCHAR2,

    base_object_name    IN VARCHAR2,

    base_object_schema  IN VARCHAR2 DEFAULT NULL,

    version             IN VARCHAR2 DEFAULT 'COMPATIBLE',

    model               IN VARCHAR2 DEFAULT 'TIBERO',

    transform           IN VARCHAR2 DEFAULT NULL,

    object_count        IN NUMBER   DEFAULT 10000)

RETURN CLOB;
```

- 파라미터

파라미터	설명
object_type	출력하고자 하는 객체 타입이다.

파라미터	설명
	OPEN, OPENW에서 사용하는 object_type 파라미터와 동일한 값을 사용하면 된다.
base_object_name	부모 객체의 이름이다.
base_object_schema	부모 객체의 스키마 이름이다.
version	현재 사용 불가
model	현재 사용 불가
transform	현재 사용 불가
object_count	현재 사용 불가

- 반환값

  결과 XML

- 예외 상황

예외 상황	설명
OBJECT_NOT_FOUND	객체를 찾지 못함

- 예제

```
SQL> set long 90000;
SQL> create table t (c1 number);

Table 'T' created.

SQL> create index t_idx on t(c1);

Index 'T_IDX' created.

SQL> select dbms_metadata.get_dependent_xml('INDEX', 'T') from dual;



DBMS_METADATA.GET_DEPENDENT_XML('INDEX','T')
--------------------------------------------------------------------------------

<ROWSET>

  <INDEX>
            <OBJ_ID>3113</OBJ_ID>
            <SGMT_ID>3113</SGMT_ID>
            <TS_ID>3</TS_ID>
            <BO_ID>3112</BO_ID>
```

```
        <SHDRDBA>8388697</SHDRDBA>
        <COL_CNT>1</COL_CNT>
        <ROW_CNT>0</ROW_CNT>
        <BLK_CNT>0</BLK_CNT>
        <DISTINCT_KEY>0</DISTINCT_KEY>
        <DBLK_KEY>0</DBLK_KEY>
        <LBLK_KEY>0</LBLK_KEY>
        <LEAF_BLK_CNT>0</LEAF_BLK_CNT>
        <BLEVEL>0</BLEVEL>
        <PCTFREE>10</PCTFREE>
        <PCTFREE_BYTES>819</PCTFREE_BYTES>
        <INITRANS>2</INITRANS>
        <TYPE_NO>1</TYPE_NO>
        <FLAGS>0</FLAGS>
        <PCTTHRES>0</PCTTHRES>
        <INCLCOL>-1</INCLCOL>
        <CLUFAC>0</CLUFAC>
        <ANALYZETIME NULL="TRUE"/>
        <RESERVED1>0</RESERVED1>
        <RESERVED2>0</RESERVED2>
        <RESERVED3>0</RESERVED3>
        <RESERVED4 NULL="TRUE"/>
        <OWNER_NAME>TIBERO</OWNER_NAME>
      <OBJ>
          <OBJ_ID>3113</OBJ_ID>
          <OWNER_ID>18</OWNER_ID>
          <NAME>T_IDX</NAME>
          <SUBNAME NULL="TRUE"/>
          <BO_ID>3112</BO_ID>
          <TYPE_NO>0</TYPE_NO>
          <FLAGS>0</FLAGS>
          <CTIME>2022/01/12</CTIME>
          <MTIME>2022/01/12</MTIME>
          <STIME>2022/01/12</STIME>
          <RESERVED1>0</RESERVED1>
          <RESERVED2>0</RESERVED2>
          <RESERVED3>0</RESERVED3>
          <RESERVED4 NULL="TRUE"/>
          <OWNER_NAME>TIBERO</OWNER_NAME>
    </OBJ>  <ICOL_LIST>
  <ICOL>
            <OBJ_ID>3113</OBJ_ID>
            <BO_ID>3112</BO_ID>
            <COL_NO>0</COL_NO>
            <POS_NO>0</POS_NO>
            <PROPERTY>0</PROPERTY>
            <RESERVED1>0</RESERVED1>
```

```
              <RESERVED2>0</RESERVED2>
              <RESERVED3>0</RESERVED3>
              <RESERVED4 NULL="TRUE"/>
              <EXPRESSION NULL="TRUE"/>
            <COL>
                <OBJ_ID>3112</OBJ_ID>
                <COL_NO>0</COL_NO>
                <SGMTCOL_NO>0</SGMTCOL_NO>
                <NAME>C1</NAME>
                <TYPE_NO>1</TYPE_NO>
                <LEN>22</LEN>
                <PRECISION>38</PRECISION>
                <SCALE>-128</SCALE>
                <NOT_NULL>0</NOT_NULL>
                <PROPERTY>0</PROPERTY>


                <RESERVED1>0</RESERVED1>
                <RESERVED2>0</RESERVED2>
                <RESERVED3>0</RESERVED3>
                <RESERVED4 NULL="TRUE"/>
                <DFLTVAL_EXP_TXT NULL="TRUE"/>
    </COL></ICOL>  </ICOL_LIST>
<TS>
            <TS_ID>3</TS_ID>
            <NAME>USR</NAME>
            <TYPE_NO>0</TYPE_NO>
            <UNIT>16</UNIT>
            <INC>0</INC>
            <FILE_CNT>1</FILE_CNT>
            <FLAGS>20</FLAGS>
            <RESERVED1>0</RESERVED1>
            <RESERVED2>0</RESERVED2>
            <RESERVED3>0</RESERVED3>
            <RESERVED4 NULL="TRUE"/>
    </TS>  <BASE_OBJ>
            <OBJ_ID>3112</OBJ_ID>
            <OWNER_ID>18</OWNER_ID>
            <NAME>T</NAME>
            <SUBNAME NULL="TRUE"/>
            <BO_ID>4294967295</BO_ID>
            <TYPE_NO>1</TYPE_NO>
            <FLAGS>0</FLAGS>
            <CTIME>2022/01/12</CTIME>
            <MTIME>2022/01/12</MTIME>
            <STIME>2022/01/12</STIME>
            <RESERVED1>0</RESERVED1>
```

```
            <RESERVED2>0</RESERVED2>
            <RESERVED3>0</RESERVED3>
            <RESERVED4 NULL="TRUE"/>
            <OWNER_NAME>TIBERO</OWNER_NAME>
        </BASE_OBJ>  <TS>
            <TS_ID>3</TS_ID>
            <NAME>USR</NAME>
            <TYPE_NO>0</TYPE_NO>
            <UNIT>16</UNIT>
            <INC>0</INC>
            <FILE_CNT>1</FILE_CNT>
            <FLAGS>20</FLAGS>
            <RESERVED1>0</RESERVED1>
            <RESERVED2>0</RESERVED2>
            <RESERVED3>0</RESERVED3>
            <RESERVED4 NULL="TRUE"/>


        </TS>  <SGMT>
            <SGMT_ID>3113</SGMT_ID>
            <TS_ID>3</TS_ID>
            <SHDRDBA>8388697</SHDRDBA>
            <NEXT_EXT>0</NEXT_EXT>
            <TYPE_NO>2</TYPE_NO>
            <USER_ID>18</USER_ID>
            <MAX_EXTS>4294967295</MAX_EXTS>
            <BUFPOOL>0</BUFPOOL>
            <RESERVED1>0</RESERVED1>
            <RESERVED2>0</RESERVED2>
            <RESERVED3>0</RESERVED3>
            <RESERVED4 NULL="TRUE"/>
        </SGMT></INDEX>

</ROWSET>

1 row selected.
```

21.2.9. GET_GRANTED_DDL

부여된 객체의 DDL을 출력한다.

GET_GRANTED_DDL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.GET_GRANTED_DDL (
    object_type         IN VARCHAR2,
    grantee             IN VARCHAR2 DEFAULT NULL,
    version             IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model               IN VARCHAR2 DEFAULT 'TIBERO',
    transform           IN VARCHAR2 DEFAULT 'DDL',
    object_count        IN NUMBER   DEFAULT 10000)
RETURN CLOB;
```

- 파라미터

파라미터	설명
object_type	출력하고자 하는 객체 타입이다. OPEN, OPENW에서 사용하는 object_type 파라미터와 동일한 값을 사용하면 된다.
grantee	수혜자의 이름이다.
version	현재 사용 불가
model	현재 사용 불가
transform	현재 사용 불가
object_count	현재 사용 불가

- 반환값

  결과 DDL

- 예외 상황

예외 상황	설명
OBJECT_NOT_FOUND	객체를 찾지 못함.

- 예제

```
SQL> create user usr1 identified by 'tibero';

User 'USR1' created.

SQL> grant dba to usr1;

Granted.

SQL> select dbms_metadata.get_granted_ddl('ROLE_GRANT', 'USR1') from dual;

DBMS_METADATA.GET_GRANTED_DDL('ROLE_GRANT','USR1')
--------------------------------------------------------------------------------
```

```
GRANT "DBA" TO "USR1"

1 row selected.
```

21.2.10. GET_GRANTED_XML

부여된 객체의 XML을 출력한다.

GET_GRANTED_XML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.GET_GRANTED_XML (
    object_type         IN VARCHAR2,
    grantee             IN VARCHAR2 DEFAULT NULL,
    version             IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model               IN VARCHAR2 DEFAULT 'TIBERO',
    transform           IN VARCHAR2 DEFAULT NULL,
    object_count        IN NUMBER   DEFAULT 10000)
RETURN CLOB;
```

- 파라미터

파라미터	설명
object_type	출력하고자 하는 객체 타입이다. OPEN, OPENW에서 사용하는 object_type 파라미터와 동일한 값을 사용하면 된다.
grantee	수혜자의 이름이다.
version	현재 사용 불가
model	현재 사용 불가
transform	현재 사용 불가
object_count	현재 사용 불가

- 반환값

  결과 XML

- 예외 상황

예외 상황	설명
OBJECT_NOT_FOUND	객체를 찾지 못함

- 예제

```
SQL> create user usr1 identified by 'tibero';

User 'USR1' created.

SQL> grant dba to usr1;

Granted.

SQL> select dbms_metadata.get_granted_xml('ROLE_GRANT', 'USR1') from dual;

DBMS_METADATA.GET_GRANTED_XML('ROLE_GRANT','USR1')
--------------------------------------------------------------------------------

<ROWSET>

  <ROLE_GRANT>
            <GRANTEE_ID>23</GRANTEE_ID>
            <PRIV_NO>11</PRIV_NO>
            <FLAGS>0</FLAGS>
            <GRANTEE_NAME>USR1</GRANTEE_NAME>
            <ROLE_NAME>DBA</ROLE_NAME>
        </ROLE_GRANT>
</ROWSET>

1 row selected.
```

21.2.11. GET_XML

객체의 XML을 출력한다.

GET_XML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.GET_XML (
    object_type         IN VARCHAR2,
    name                IN VARCHAR2,
    schema              IN VARCHAR2 DEFAULT NULL,
    version             IN VARCHAR2 DEFAULT 'COMPATIBLE',
    model               IN VARCHAR2 DEFAULT 'TIBERO',
    transform           IN VARCHAR2 DEFAULT NULL)
RETURN CLOB;
```

- 파라미터

파라미터	설명
object_type	출력하고자 하는 객체 타입이다. OPEN, OPENW에서 사용하는 object_type 파라미터와 동일한 값을 사용하면 된다.
name	객체의 이름이다.
schema	객체의 스키마 이름이다.
version	현재 사용 불가
model	현재 사용 불가
transform	현재 사용 불가

- 반환값

  결과 XML

- 예외 상황

예외 상황	설명
OBJECT_NOT_FOUND	객체를 찾지 못함.

- 예제

```
SQL> set long 90000;
SQL> create table t (c1 number);

Table 'T' created.

SQL> select dbms_metadata.get_xml('TABLE', 'T') from dual;



DBMS_METADATA.GET_XML('TABLE','T')
--------------------------------------------------------------------------------

<ROWSET>

  <TABLE>
            <OBJ_ID>3114</OBJ_ID>
            <SGMT_ID>3114</SGMT_ID>
            <TS_ID>3</TS_ID>
            <COL_CNT>1</COL_CNT>
            <SHDRDBA>8388681</SHDRDBA>
            <PCTFREE>10</PCTFREE>
            <PCTFREE_BYTES>819</PCTFREE_BYTES>
            <INITRANS>2</INITRANS>
```

```
          <ROW_CNT>0</ROW_CNT>
          <BLK_CNT>0</BLK_CNT>
          <EMP_CNT>0</EMP_CNT>
          <AVG_SPC>0</AVG_SPC>
          <CHN_CNT>0</CHN_CNT>
          <FLAGS>0</FLAGS>
          <AUDIT_FLAGS>----------------------------------</AUDIT_FLAGS>
          <DOP>0</DOP>
          <AVG_RLN>0</AVG_RLN>
          <ANALYZETIME NULL="TRUE"/>
          <STATUS>0</STATUS>
          <RPB>744</RPB>
          <RESERVED1>0</RESERVED1>
          <RESERVED2>0</RESERVED2>
          <RESERVED3>0</RESERVED3>
          <RESERVED4 NULL="TRUE"/>
          <OWNER_NAME>TIBERO</OWNER_NAME>
       <OBJ>
           <OBJ_ID>3114</OBJ_ID>
           <OWNER_ID>18</OWNER_ID>
           <NAME>T</NAME>
           <SUBNAME NULL="TRUE"/>
           <BO_ID>4294967295</BO_ID>
           <TYPE_NO>1</TYPE_NO>
           <FLAGS>0</FLAGS>
           <CTIME>2022/01/12</CTIME>
           <MTIME>2022/01/12</MTIME>
           <STIME>2022/01/12</STIME>
           <RESERVED1>0</RESERVED1>
           <RESERVED2>0</RESERVED2>
           <RESERVED3>0</RESERVED3>
           <RESERVED4 NULL="TRUE"/>
           <OWNER_NAME>TIBERO</OWNER_NAME>
    </OBJ>  <COL_LIST>
<LOB_LIST/>
<CON_LIST/>
<REF_CON_LIST/>
<TS>
          <TS_ID>3</TS_ID>
          <NAME>USR</NAME>
          <TYPE_NO>0</TYPE_NO>
          <UNIT>16</UNIT>
          <INC>0</INC>
          <FILE_CNT>1</FILE_CNT>
          <FLAGS>20</FLAGS>
          <RESERVED1>0</RESERVED1>
          <RESERVED2>0</RESERVED2>
```

```
              <RESERVED3>0</RESERVED3>
              <RESERVED4 NULL="TRUE"/>
        </TS>  <SGMT>
              <SGMT_ID>3114</SGMT_ID>
              <TS_ID>3</TS_ID>
              <SHDRDBA>8388681</SHDRDBA>
              <NEXT_EXT>0</NEXT_EXT>
              <TYPE_NO>1</TYPE_NO>
              <USER_ID>18</USER_ID>
              <MAX_EXTS>4294967295</MAX_EXTS>
              <BUFPOOL>0</BUFPOOL>
              <RESERVED1>0</RESERVED1>
              <RESERVED2>0</RESERVED2>
              <RESERVED3>0</RESERVED3>
              <RESERVED4 NULL="TRUE"/>
        </SGMT></TABLE>

</ROWSET>

1 row selected.
```

21.2.12. OPEN

DBMS_METADATA 패키지에서 사용할 handle을 생성한다.

OPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.OPEN (
   object_type  IN VARCHAR2,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model        IN VARCHAR2 DEFAULT 'TIBERO',
   network_link IN VARCHAR2 DEFAULT NULL)
 RETURN NUMBER;
```

- 파라미터

파라미터	설명
object_type	객체 타입은 "[객체 타입]"을 참고한다.
version	현재 사용 불가
model	현재 사용 불가
network_link	현재 사용 불가

- 반환값

  내부에서 정의된 handle(다른 함수나 프로시저의 파라미터로 사용됨)

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	잘못된 파라미터가 입력됨
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음

- 예제

```
SQL> declare

   2     handle number;

   3 begin

   4     handle := dbms_metadata.open('TABLE');

   5     dbms_metadata.close(handle);

   6 end;


   7 /

PSM completed.
```

21.2.13. OPENW

패키지에서 사용할 쓰기 전용 handle을 생성한다.

OPENW 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.OPENW (
   object_type  IN VARCHAR2,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE',
   model        IN VARCHAR2 DEFAULT 'TIBERO')
 RETURN NUMBER;
```

- 파라미터

파라미터	설명
object_type	객체 타입은 "[객체 타입]"을 참고한다.
version	현재 사용 불가
model	현재 사용 불가

[객체 타입]

객체 타입	설명	객체 타입 유형
CONSTRAINT	제약조건	SND
FUNCTION	사용자 함수	SN
INDEX	인덱스	SND
JAVA_SOURCE	자바 객체	SN
JOB	작업	S
LIBRARY	라이브러리	SN
MATERIALIZED_VIEW	실체화 뷰	SN
MATERIAL IZED_VIEW_LOG	실체화 뷰 로그	D
OBJECT_GRANT	스키마 객체 특권	DG
PACKAGE	패키지(패키지 선언과 본문을 모두 포함)	SN
PACKAGE_SPEC	패키지 선언	SN
PACKAGE_BODY	패키지 본문	SN
PROCEDURE	프러시저	SN
PROFILE	프로파일	N
REF_CONSTRAINT	참조 무결성 제약조건	SND
ROLE	역할	N
ROLE_GRANT	역할 특권	G
SEQUENCE	시퀀스	SN
SYNONYM	동의어(PUBLIC도 스키마처럼 사용)	SN
SYSTEM_GRANT	시스템 특권	G
TABLE	테이블	SN
TABLESPACE	테이블 스페이스	N
TRIGGER	트리거	SND
TYPE	사용자 정의 타입(사용자 정의 타입 선언과 본문을 모두 포함)	SN
TYPE_SPEC	사용자 정의 타입 선언	SN

객체 타입	설명	객체 타입 유형
TYPE_BODY	사용자 정의 타입 본문	SN
USER	사용자	N
VIEW	뷰	SN

다음은 객체 타입 유형의 의미이다.

- S : 스키마 객체 (Schema objects). 스키마에 속한 객체 타입
- N : 이름이 있는 객체 (Named objects)
- D : 종속 객체 (Dependent objects). 부모 객체가 존재하는 객체 타입
- G : 부여된 객체 (Granted objects). 권한과 관련된 객체 타입

- 반환값

  내부에서 정의된 쓰기 전용 handle(CONVERT의 파라미터로 사용됨)

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	잘못된 파라미터가 입력됨
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음

- 예제

```
SQL> declare

   2     handle number;

   3 begin

   4     handle := dbms_metadata.openw('TABLE');

   5     dbms_metadata.close(handle);

   6 end;


   7 /

PSM completed.
```

21.2.14. SET_FILTER

대상 객체를 특정할 수 있는 필터를 설정한다.

SET_FILTER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.SET_FILTER (



    handle              IN NUMBER,
    name                IN VARCHAR2,
    value               IN VARCHAR2,
    object_type_path    IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_FILTER (
    handle              IN NUMBER,
    name                IN VARCHAR2,
    value               IN BOOLEAN DEFAULT TRUE,
    object_type_path    IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_FILTER (
    handle              IN NUMBER,
    name                IN VARCHAR2,
    value               IN NUMBER,
    object_type_path    IN VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
handle	OPEN 또는 OPENW 함수를 통해 획득한 handle이다.
name	필터 이름이다("[객체 타입]" 참고). 각 필터의 값의 타입에 맞는 프로시저를 사용해야 한다.
value	필터 값이다.
object_type_path	현재 사용 불가

[객체 타입]

아래 표는 필터 이름과 대응하는 객체 타입을 표시한다.

객체 타입 유형	필터 이름	타입	설명
S	SCHEMA	Varchar2	스키마 이름이다(PUBLIC도 사용 가능)

객체 타입 유형	필터 이름	타입	설명
N	NAME	Varchar2	객체 이름이다.
N	NAME_EXPR	Varchar2	객체 이름과 관련된 expression이다. 객체 특정 시 WHERE 절에 사용된다.
D	BASE_OBJECT_NAME	Varchar2	부모 객체 이름이다.
D	BASE_OB JECT_SCHEMA	Varchar2	부모 객체의 스키마 이름이다.
D	BASE_OB JECT_NAME_EXPR	Varchar2	부모 객체 이름과 관련된 expression 이다. 부모 객체 특정 시 WHERE 절에 사용된다.
D	BASE_OBJECT_TYPE	Varchar2	부모 객체 타입이다.
INDEX, TRIGGER	SYSTEM_GENERATED	Boolean	TRUE이면 시스템이 생성한 인덱스/트리거를 포함한다. FALSE이면 생략한다. (기본값 : TRUE)
G	GRANTEE	Varchar2	수혜자 이름이다.

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음
INCONSISTENT_ARGS	handle 생성 시 입력된 object_type 파라미터와 대응하지 않는 필터를 설정하려 함

- 예제

```
SQL> declare

   2     handle number;

   3 begin

   4     handle := dbms_metadata.open('TABLE');

   5     dbms_metadata.set_filter(handle, 'NAME', 'T');

   6     dbms_metadata.close(handle);
```

```
   7 end;

   8 /

PSM completed.
```

21.2.15. SET_PARSE_ITEM

추출된 XML 문서에서 별도로 추출하고자 하는 정보를 설정한다.

SET_PARSE_ITEM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.SET_PARSE_ITEM (
    handle              IN NUMBER,
    name                IN VARCHAR2,
    object_type         IN VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
handle	OPEN 또는 OPENW 함수를 통해 획득한 handle이다.
name	Parse item 이름이다("[객체 타입]" 참고).
object_type	객체 타입

[객체 타입]

아래 표는 Parse item 이름과 대응하는 객체 타입 종류를 표시한다.

객체 타입 유형	Parse item 이름	설명
모든 객체 (SNDG)	VERB	DDL 구문의 동사 (CREATE, GRANT 등)
모든 객체(SNDG)	OBJECT_TYPE	DDL 구문의 객체 타입 (TABLE, INDEX 등)
N	NAME	DDL 구문의 객체 이름
G	GRANTEE	DDL 구문의 수혜자 이름

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	handle이 NULL이거나 invalid
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음

예외 상황	설명
INCONSISTENT_ARGS	handle 생성 시 입력된 object_type 파라미터와 대응하지 않는 Parse item을 설정하려 함

- 예제

```
SQL> create table t (c1 number);

Table 'T' created.

SQL> set serveroutput on;
SQL> DECLARE

   2     handle NUMBER;

   3     transform_handle NUMBER;

   4     ddls sys.ku$_ddls;

   5     pi sys.ku$_parsed_items;


   6 BEGIN

   7     handle := dbms_metadata.open('TABLE');

   8     dbms_metadata.set_filter(handle, 'NAME', 'T');

   9

  10     dbms_metadata.set_parse_item(handle, 'OBJECT_TYPE');

  11     dbms_metadata.set_parse_item(handle, 'VERB');

  12

  13     transform_handle := dbms_metadata.add_transform(handle, 'DDL');

  14     ddls := dbms_metadata.fetch_ddl(handle);

  15     if ddls.count > 0 then

  16         for cur_ddl in ddls.first..ddls.last loop

  17             pi := ddls(cur_ddl).parseditems;

  18             if pi.count > 0 then
```

```
  19                 for i in pi.first..pi.last loop
  20                     dbms_output.put_line(pi(i).item || ' : ' || pi(i).value);
  21                 end loop;
  22             end if;
  23             dbms_output.put_line(' ');
  24         end loop;
  25     end if;
  26     dbms_metadata.close(handle);
  27 END;
  28 /
VERB : CREATE
OBJECT_TYPE : TABLE


PSM completed.
```

21.2.16. SET_REMAP_PARAM

추출된 XML 문서 중 특정 값을 변경할 때 사용하는 파라미터를 설정한다.

SET_REMAP_PARAM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.SET_REMAP_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    old_value           IN VARCHAR2,
    new_value           IN VARCHAR2,
    object_type         IN VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
transform_handle	ADD_TRANSFORM 함수를 통해 획득한 transform handle이다.

파라미터	설명
	'MODIFY' transform만 사용 가능하다.
name	파라미터 이름이다("[객체 타입]" 참고).
old_value	변경 대상이 되는 값이다.
new_value	변경 하고자 하는 값이다.
object_type	현재 사용 불가

[객체 타입]

아래 표는 파라미터 이름과 대응하는 객체 타입을 표시한다.

객체 타입 유형	파라미터 이름	설명
N 부모 객체가 이름이 있는 객체인 종속 객체	REMAP_NAME	객체 이름을 변경한다. 종속 객체의 경우 부모 객체의 이름을 변경한다.
SDG User	REMAP_SCHEMA	스키마 이름을 변경한다. 종속 객체의 경우 부모 객체의 스키마를 변경한다. 부여된 객체의 경우 수혜자 이름을 변경한다.

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	transform handle이 NULL이거나 invalid
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음
INCONSISTENT_ARGS	handle 생성 시 입력된 object_type 파라미터와 대응하지 않는 파라미터를 설정하려 함

- 예제

```
SQL> create table t (c1 number);

Table 'T' created.

SQL> declare

   2     handle number;

   3     transform_handle number;

   4     ret_xml xmltype;
```

```
  5 begin

  6     handle := dbms_metadata.open('TABLE');

  7     dbms_metadata.set_filter(handle, 'NAME', 'T');

  8     transform_handle := dbms_metadata.add_transform(handle, 'MODIFY');

  9     dbms_metadata.set_remap_param(transform_handle, 'REMAP_SCHEMA',
 'TIBERO', 'TTT');
 10     ret_xml := dbms_metadata.fetch_xml(handle);

 11     dbms_metadata.close(handle);

 12 end;

 13 /

PSM completed.
```

21.2.17. SET_TRANSFORM_PARAM

추출된 DDL의 형태를 결정하는 파라미터를 설정한다.

SET_TRANSFORM_PARAM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_METADATA.SET_TRANSFORM_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    value               IN VARCHAR2,
    object_type         IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_TRANSFORM_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
    value               IN BOOLEAN,
    object_type         IN VARCHAR2 DEFAULT NULL);

DBMS_METADATA.SET_TRANSFORM_PARAM (
    transform_handle    IN NUMBER,
    name                IN VARCHAR2,
```

```
    value                IN NUMBER,
    object_type          IN VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
transform_handle	ADD_TRANSFORM 함수를 통해 획득한 transform handle이다. 'DDL' transform만 사용 가능하다. SESSION_TRANSFORM은 해당 세션에서 기본적으로 사용하는 DDL transform을 의미한다. SESSION_TRANSFORM은 DBMS_METADATA 패키지에 정의되어 있다. GET_DDL 등의 함수에서도 사용된다.
name	파라미터 이름이다("[객체 타입]" 참고). 각 파라미터값의 타입에 맞는 프러시저를 사용해야 한다.
value	파라미터 값이다.
object_type_path	현재 사용 불가

[객체 타입]

아래 표는 파라미터 이름과 대응하는 객체 타입 및 파라미터 값의 타입을 표시한다.

객체 타입 유형	파라미터 이름	타입	설명
모든 객체 (SNDG)	PRETTY	Boolean	TRUE이면 DDL 추출 시 들여쓰기와 줄바꿈이 추가된다. (기본값 : TRUE)
모든 객체 (SNDG)	SQLTERMINATOR	Boolean	TRUE이면 DDL 추출 시 종결부호(;) 포함한다. (기본값 : FALSE)
TABLE	CONSTRAINTS	Boolean	TRUE이면 TABLE 추출 시 참조 무결성 제약조건을 제외한 모든 제약조건 포함한다. (기본값 : TRUE)
TABLE	REF_CONSTRAINTS	Boolean	TRUE이면 TABLE 추출 시 참조 무결성 제약조건을 포함한다. (기본값 : TRUE)
TABLE	CONSTRAINTS_AS_ALTER	Boolean	TRUE이면 TABLE 추출 시 제약조건을 CREATE TABLE 구문에 포함시키지 않고 별도의 ALTER TABLE 구문으로 제공한다.

객체 타입 유형	파라미터 이름	타입	설명
			FALSE이면 TABLE 추출 시 제약조건을 CREATE TABLE 구문에 포함시킨다. TRUE 설정을 위해선 CONSTRAINTS 파라미터가 TRUE로 설정되어 있어야 한다. (기본값 : FALSE)
TABLE	TABLE_COMPRESSION_CLAUSE	Varchar2	TABLE 추출 시 compression clause를 지정할 수 있다. NONE 설정 시 compression clause를 생략한다(소속 테이블스페이스의 설정에 따름).
INDEX CONSTRAINT TABLE TABLESPACE	SEGMENT_ATTRIBUTES	Boolean	TRUE이면 DDL 추출 시 segment attribute clause를 포함시킨다. (기본값 : TRUE)
INDEX CONSTRAINT TABLE	STORAGE	Boolean	TRUE이면 DDL 추출 시 storage clause를 포함시킨다. SEGMENT_ATTRIBUTES 파라미터가 FALSE이면 해당 파라미터는 무시된다. (기본값 : TRUE)
INDEX CONSTRAINT TABLE	TABLESPACE	Boolean	TRUE이면 DDL 추출 시 tablespace clause를 포함시킨다. SEGMENT_ATTRIBUTES 파라미터가 FALSE이면 해당 파라미터는 무시된다. (기본값 : TRUE)

- 예외 상황

예외 상황	설명
INVALID_ARGVAL	transform handle이 NULL이거나 invalid
INVALID_OPERATION	FETCH_DDL, FETCH_XML 수행 후에는 호출할 수 없음
INCONSISTENT_ARGS	handle 생성 시 입력된 object_type 파라미터와 대응하지 않는 파라미터를 설정하려 함

- 예제

```
SQL> exec dbms_metadata.set_transform_param(dbms_metadata.session_transform,
'SQLTERMINATOR',true);

PSM completed.
```

제22장 DBMS_MONITOR

본 장에서는 DBMS_MONITOR 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

22.1. 개요

DBMS_MONITOR는 인스턴스, 세션, 클라이언트 식별자 단위의 성능 모니터링을 지원하는 패키지이다.

해당 단위로 SQL 트레이스를 제어함으로써 성능 모니터링을 할 수 있다.

22.2. 프러시저와 함수

본 절에서는 DBMS_MONITOR 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

22.2.1. CLIENT_ID_TRACE_DISABLE

주어진 클라이언트 식별자를 가진 세션들에 대하여 SQL 트레이스를 비활성화하는 프러시저이다.

CLIENT_ID_TRACE_DISABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE
(
    client_id       IN      VARCHAR2
)
```

- 파라미터

파라미터	설명
client_id	SQL 트레이스를 비활성화할 세션들의 클라이언트 식별자이다.

22.2.2. CLIENT_ID_TRACE_ENABLE

주어진 클라이언트 식별자를 가진 세션들에 대하여 SQL 트레이스를 활성화하는 프러시저이다.

CLIENT_ID_TRACE_ENABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE
(
    client_id       IN      VARCHAR2,
    binds           IN      BOOLEAN DEFAULT NULL
)
```

- 파라미터

파라미터	설명
client_id	SQL 트레이스를 활성화할 세션들의 클라이언트 식별자이다.
binds	SQL 트레이스를 활성화할 때 바인드 정보를 남길 지 여부를 결정한다. NULL이면 바인드 정보를 남기지 않는다.

22.2.3. DATABASE_TRACE_DISABLE

데이터베이스 서버 인스턴스 전체에 대하여 SQL 트레이스를 비활성화하는 프러시저이다.

DATABASE_TRACE_DISABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MONITOR.DATABASE_TRACE_DISABLE
```

22.2.4. DATABASE_TRACE_ENABLE

데이터베이스 서버 인스턴스 전체에 대하여 SQL 트레이스를 활성화하는 프러시저이다.

DATABASE_TRACE_ENABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MONITOR.DATABASE_TRACE_ENABLE
(
    binds           IN      BOOLEAN DEFAULT NULL
)
```

- 파라미터

파라미터	설명
binds	SQL 트레이스를 활성화할 때 바인드 정보를 남길 지 여부를 결정한다. NULL이면 바인드 정보를 남기지 않는다.

22.2.5. SESSION_TRACE_DISABLE

주어진 세션에 대하여 SQL 트레이스를 비활성화하는 프러시저이다.

SESSION_TRACE_DISABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MONITOR.SESSION_TRACE_DISABLE
(
    session_id      IN      BINARY_INTEGER DEFAULT NULL,
    serial_num      IN      BINARY_INTEGER DEFAULT NULL
)
```

- 파라미터

파라미터	설명
session_id	SQL 트레이스를 비활성화할 세션의 식별자이다. NULL이면 현재 세션을 가리킨다.
serial_num	SQL 트레이스를 비활성화할 세션의 시리얼 번호이다. NULL이면 시리얼 번호에 무관하게 해당 세션 식별자를 사용하는 세션들에 대하여 지속적으로 SQL 트레이스를 비활성화한다.

22.2.6. SESSION_TRACE_ENABLE

주어진 세션에 대하여 SQL 트레이스를 활성화하는 프러시저이다.

SESSION_TRACE_ENABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MONITOR.SESSION_TRACE_ENABLE
(
    session_id      IN      BINARY_INTEGER DEFAULT NULL,
```

```
    serial_num      IN      BINARY_INTEGER DEFAULT NULL,
    binds           IN      BOOLEAN DEFAULT NULL
)
```

- 파라미터

파라미터	설명
session_id	SQL 트레이스를 활성화할 세션의 식별자이다. NULL이면 현재 세션을 가리킨다.
serial_num	SQL 트레이스를 활성화할 세션의 시리얼 번호이다. NULL이면 시리얼 번호에 무관하게 해당 세션 식별자를 사용하는 세션들에 대하여 지속적으로 SQL 트레이스를 활성화한다.
binds	SQL 트레이스를 활성화할 때 바인드 정보를 남길 지 여부를 결정한다. NULL이면 바인드 정보를 남기지 않는다.

제23장 DBMS_MVIEW

본 장에서는 DBMS_MVIEW 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

23.1. 개요

DBMS_MVIEW는 실체화 뷰와 관련된 정보를 제공하고 이 정보를 최근의 것으로 변경할 수 있는 REFRESH 기능을 사용하기 위한 프러시저를 제공하는 패키지이다.

23.2. 프러시저

본 절에서는 DBMS_MVIEW 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

23.2.1. EXPLAIN_REWRITE

SQL의 **질의 다시 쓰기**(Query Rewrite) 정보를 설명하는 프러시저이다. 이 프러시저는 왜 SQL 질의가 재작성에 실패했는지 또는 어떤 실체화 뷰를 사용해서 재작성 되었는지, 재작성된 SQL 질의는 무엇인지를 설명해준다.

이러한 정보를 밑바탕으로 사용자는 SQL 질의가 재작성될 수 있도록 처리할 수 있다. query 파라미터로 전달받은 SQL 질의문은 실제로 수행되지 않으며, 결과는 현재 스키마의 REWRITE_TABLE 테이블에 저장된다. 단, 이 프러시저를 호출하기 전에 $TB_HOME/scripts/rewrite_table.sql 스크립트를 실행하여 REWRITE_TABLE 테이블을 미리 생성해야 한다.

EXPLAIN_REWRITE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MVIEW.EXPLAIN_REWRITE
(
    query         IN   VARCHAR2,
    statement_id  IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
query	SQL 질의문이다.
statement_id	SQL 질의문의 결과를 구별하기 위해 사용자가 정한 고유 식별자이다.

파라미터	설명
	이 식별자는 REWRITE_TABLE 테이블의 STATEMENT_ID 컬럼에 저장된다.

- 예제

```
SQL> create table base as (select mod(level, 100) a, level*10 b from dual
        connect by level<=100);
SQL> create materialized view mv enable query rewrite as
        select sum(a+b) s, count(b+a) c from base;
SQL> @rewrite_table.sql
SQL> exec dbms_mview.explain_rewrite('select avg(a+b) from base')
SQL> select MV_OWNER, MV_NAME, QUERY, REWRITTEN_TXT, MESSAGE
     from rewrite_table;

MV_OWNER MV_NAME QUERY                  REWRITTEN_TXT            MESSAGE
-------- ------- -------------- ----------------------- ------------------------
SYS      MV      SELECT AVG(A+B) SELECT ("MV"."S"         010: query was rewritten

                 FROM BASE         / "MV"."C") "AVG(A+B)"   with materialized view

                                  FROM "SYS"."MV"

                 select avg(a+b) SELECT ("MV"."S"         000: whole query was
                 from base         / "MV"."C") "AVG(A+B)"   written
                                  FROM "SYS"."MV"
```

23.2.2. REFRESH

실체화 뷰를 Refresh하는 프러시저이다.

REFRESH 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_MVIEW.REFRESH
(
    qualified_obj     IN   VARCHAR2,
    refresh_method    IN   VARCHAR2
);
```

- 파라미터

파라미터	설명
qualified_obj	Refresh할 실체화 뷰이다.
refresh_method	Refresh에 사용될 방법이다. – C 또는 c : 완전 Refresh를 사용한다. – F 또는 f : 빠른 Refresh를 사용한다. – ? : 빠른 Refresh가 가능한 경우 빠른 Refresh를 사용하고, 그렇지 않으면 완전 Refresh를 사용한다. (기본값)

- 예제

```
call dbms_mview.refresh('myuser.mv_t','f');
call dbms_mview.refresh('mv_t','c');
```

제24장 DBMS_OBFUSCATION_TOOLKIT

본 장에서는 DBMS_OBFUSCATION_TOOLKIT 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

24.1. 개요

DBMS_OBFUSCATION_TOOLKIT은 데이터를 암호화(encryption)하고 복호화(decryption)하는 패키지이다. 이 패키지에서는 데이터의 암호화 및 복호화를 위해 DES(Data Encryption Standard) 또는 3DES(Triple DES) 알고리즘을 이용한다. DES와 3DES 알고리즘은 대칭 키(symmetric key)를 사용하는 알고리즘이다.

알고리즘	설명
DES	56bits의 키를 사용하는 알고리즘이다. 그동안 널리 사용되어 왔으나 최근에는 보안성이 보장되지 않아 점차 사용 빈도가 줄어들고 있다. 대칭 키를 사용하는 알고리즘이다.
3DES	하나의 데이터에 DES 알고리즘을 두 번 내지 세 번 반복하여 적용하는 알고리즘이다. 각각 112(56 * 2)bits와 168(56 * 3)bits 키를 사용한다. DES 알고리즘과 마찬가지로 대칭 키를 사용하는 알고리즘이다. 다른 대칭 키를 사용하는 알고리즘에 비해 암호화 및 복호화를 하는 시간이 많이 필요하다는 단점이 있다.

이러한 대칭 키를 사용하는 암호화 알고리즘에서는 키를 안전하게 관리하는 것이 무엇보다 중요하다.

암호화 알고리즘에서 키를 관리하는 방법은 다음과 같다.

- **데이터베이스에 키를 저장하는 방법**

  특정 테이블의 컬럼에 키를 저장하는 방법이다. 이때 키를 저장하는 테이블은 암호화된 데이터 컬럼과 같은 테이블일 수도 있고, 다른 테이블일 수도 있다. 같은 테이블에 키를 저장하는 경우에는 해당 테이블 전체에 대한 액세스 특권을 부여하는 대신 뷰 또는 tbPSM 프로그램 등을 통하여 해당 테이블에 액세스해야 한다.

- **운영체제 파일에 저장하는 방법**

  키를 파일에 저장하고, 데이터베이스 내에서 tbPSM 프로그램 등을 통하여 운영체제 파일로부터 키를 읽어 데이터를 암호화하거나 복호화하는 방법이다. 이때 키를 저장하는 운영체제 파일은 패스워드 파일과 같이 다른 사람이 직접 액세스할 수 없도록 보안을 유지해야 한다.

- **사용자가 키를 직접 입력하는 방법**

  사용자 또는 애플리케이션 프로그램 내에서 필요할 때마다 키를 데이터베이스로 전송하여 암호화 및 복호화를 수행하는 방법이다. 이 방법은 사용자가 키를 항상 암기하고 있거나 애플리케이션 프로그램의 소스 내에 키가 포함되어 있어야 한다. 또한, 키는 네트워크를 통하여 데이터베이스 서버로 전달되므로, 네트워크 보안에도 유의해야 한다.

24.2. 프러시저와 함수

본 절에서는 DBMS_OBFUSCATION_TOOLKIT 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

24.2.1. DES3DECRYPT

3DES 알고리즘을 이용하여 암호화된 데이터를 복호화하는 프러시저와 함수이다. 주어진 암호화된 데이터에 대하여 두 번 또는 세 번의 DES 복호화를 수행할 수 있다.

복호화를 위해 주어진 키에 대해 복호화를 두 번 수행한다면 16bytes(128bits), 세 번을 수행한다면 24bytes(192bits)이어야 하며, 그렇지 않으면 예외 상황이 발생한다. 디폴트는 두 번의 DES 복호화를 수행한다. 암호화된 데이터와 같은 횟수만큼 DES 알고리즘을 수행해야 한다.

DES3DECRYPT 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT
(
    input               IN          RAW,
    key                 IN          RAW,
    decrypted_data      OUT         RAW,
    which               IN          PLS_INTEGER    DEFAULT 0,
    iv                  IN          RAW            DEFAULT NULL
);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT
(
    input_string        IN          VARCHAR2,
    key_string          IN          VARCHAR2,
    decrypted_string    OUT         VARCHAR2,
    which               IN          PLS_INTEGER    DEFAULT 0,
    iv_string           IN          VARCHAR2       DEFAULT NULL
);
```

– 함수

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT
(
    input               IN          RAW,
    key                 IN          RAW,
    which               IN          PLS_INTEGER    DEFAULT 0,
    iv                  IN          RAW            DEFAULT NULL
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT
(
    input_string        IN          VARCHAR2,
    key_string          IN          VARCHAR2,
    which               IN          PLS_INTEGER    DEFAULT 0,
    iv_string           IN          VARCHAR2       DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
input, input_string	복호화할 데이터이다.
key	복호화하기 위해 주어진 키 값이다.
decrypted_data	복호화된 결과 데이터이다.
which	– 0이면 DES 복호화를 2번 수행한다. – 1이면 DES 복호화를 3번 수행한다.
iv	초기화 벡터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_INPUT	input_data의 길이가 8의 배수가 아닌 경우이다.
KEY_TOO_SHORT	키 값의 길이가 8보다 작은 경우이다.
INVALID_DES_MODE	which 값이 0 또는 1이 아닌 경우이다.

- 예제

```
DECLARE
    data RAW(256);
    key RAW(16);
```

```
    encrypted_data RAW(256);
    decrypted_data RAW(256);
BEGIN
    data := '0102030405AE030D0123456789ABCDEF';
    key := '0A123B8E002CD3FF01DE2389A4567BCF';
    DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(input => data, key => key,
            encrypted_data => encrypted_data);
    data := encrypted_data;
    DBMS_OBFUSCATION_TOOLKIT.DES3Decrypt(input => data, key => key,
            decrypted_data => decrypted_data);
END;
```

24.2.2. DES3ENCRYPT

3DES 알고리즘을 이용하여 데이터를 암호화하는 프러시저와 함수이다. 주어진 데이터에 대하여 두 번 또는 세 번의 DES 암호화를 수행할 수 있다.

암호화를 위해 주어진 키에 대해 암호화를 두 번을 수행한다면 16bytes(128bits), 세 번을 수행한다면 24bytes(192bits)이어야 하며, 그렇지 않으면 예외 상황이 발생한다. 디폴트는 두 번의 DES 암호화를 수행한다.

DES3ENCRYPT 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.DES3ENCRYPT
(
    input               IN          RAW,
    key                 IN          RAW,
    encrypted_data      OUT         RAW,
    which               IN          PLS_INTEGER    DEFAULT 0,
    iv                  IN          RAW            DEFAULT NULL
);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3ENCRYPT
(
     input_string        IN          VARCHAR2,
     key_string          IN          VARCHAR2,
     encrypted_string    OUT         VARCHAR2,
     which               IN          PLS_INTEGER    DEFAULT 0,
     iv_string           IN          VARCHAR2       DEFAULT NULL
);
```

  - 함수

```
DBMS_OBFUSCATION_TOOLKIT.DES3ENCRYPT
(
     input               IN          RAW,
     key                 IN          RAW,
     which               IN          PLS_INTEGER    DEFAULT 0,
     iv                  IN          RAW            DEFAULT NULL
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3ENCRYPT
(
    input_string        IN          VARCHAR2,
    key_string          IN          VARCHAR2,
    which               IN          PLS_INTEGER    DEFAULT 0,
    iv_string           IN          VARCHAR2       DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
input, input_string	암호화할 데이터이다.
key	암호화하기 위해 주어진 키 값이다.
encrypted_data	암호화된 결과 데이터이다.
which	– 0이면 DES 암호화를 2번 수행한다. – 1이면 DES 암호화를 3번 수행한다.
iv	초기화 벡터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_INPUT	input_data의 길이가 8의 배수가 아닌 경우이다.
KEY_TOO_SHORT	키 값의 길이가 8보다 작은 경우이다.
INVALID_DES_MODE	which 값이 0 또는 1이 아닌 경우이다.

- 예제

```
DECLARE
    data RAW(256);
    key RAW(16);
    encrypted_data RAW(256);
    decrypted_data RAW(256);
```

```
BEGIN
    data := '0102030405AE030D0123456789ABCDEF';
    key := '0A123B8E002CD3FF01DE2389A4567BCF';
    DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(input => data, key => key,
            encrypted_data => encrypted_data);
    data := encrypted_data;
    DBMS_OBFUSCATION_TOOLKIT.DES3Decrypt(input => data, key => key,
            decrypted_data => decrypted_data);
END;
```

24.2.3. DES3GETKEY

임의의 값을 입력 값으로 받아 DES3 알고리즘을 위한 키를 생성하는 프러시저와 함수이다.

DES3GETKEY 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.DES3GETKEY
(
    which           IN          PLS_INTEGER     DEFAULT 0,
    seed            IN          RAW,
    key             OUT         RAW
);
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3GETKEY
(
    which           IN          PLS_INTEGER     DEFAULT 0,
    seed_string     IN          VARCHAR2,
    key             OUT         VARCHAR2
);
```

  - 함수

```
DBMS_OBFUSCATION_TOOLKIT.DES3GETKEY
(
    which           IN          PLS_INTEGER     DEFAULT 0,
    seed            IN          RAW
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DES3GETKEY
(
    which           IN          PLS_INTEGER     DEFAULT 0,
```

```
    seed_string     IN          VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
which	– 0이면 DES 암호화를 2번 수행한다. – 1이면 DES 암호화를 3번 수행한다.
seed	80자 이상의 임의의 값이다.
key	암호화하기 위해 주어진 키 값이다.

- 예외 상황

예외 상황	설명
NO_SEED	seed의 값이 NULL인 경우이다.
SEED_TOO_SHORT	seed의 길이가 80보다 작은 경우이다.
INVALID_INPUT	which가 NULL인 경우이다.
INVALID_DES_MODE	which의 값이 0 또는 1이 아닌 경우이다.

- 예제

```
DECLARE
    data VARCHAR2(4096);
    key VARCHAR2(4096);
 key_seed VARCHAR2(4096);
    encrypted_data VARCHAR2(4096);
    decrypted_data VARCHAR2(4096);
BEGIN
    data := '0102030405AE030D';
    key_seed := '1234567890'||'1234567890'||'1234567890'||'1234567890';
    key_seed := rpad(key_seed,80);
    key := dbms_obfuscation_toolkit.DES3GetKey(seed_string => key_seed);
    DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(input_string => data, key_string => key,
            encrypted_string=> encrypted_data);
    data := encrypted_data;
    DBMS_OBFUSCATION_TOOLKIT.DES3Decrypt(input_string => data, key_string => key,
            decrypted_string => decrypted_data);
END;
/
```

24.2.4. DESDECRYPT

DES 알고리즘을 이용하여 암호화된 데이터를 복호화하는 프러시저와 함수이다. 복호화를 위해 주어진 키는 반드시 8bytes(64bits)이어야 하며, 그렇지 않으면 예외 상황이 발생한다.

DESDECRYPT 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.DESDECRYPT
(
    input               IN          RAW,
    key                 IN          RAW,
    decrypted_data      OUT         RAW
);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESDECRYPT
(
    input_string        IN          VARCHAR2,
    key_string          IN          VARCHAR2,
    decrypted_string    OUT         VARCHAR2
);
```

  - 함수

```
DBMS_OBFUSCATION_TOOLKIT.DESDECRYPT
(
    input               IN          RAW,
    key                 IN          RAW
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DESDECRYPT
(
    input_string        IN          VARCHAR2,
    key_string          IN          VARCHAR2
)
 RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
input, input_string	복호화할 데이터이다.
key	복호화하기 위해 주어진 키 값이다.
decrypted_data	복호화된 결과 데이터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_INPUT	input_data의 길이가 8의 배수가 아닌 경우이다.
KEY_TOO_SHORT	키 값의 길이가 8보다 작은 경우이다.

- 예제

```
DECLARE
    data RAW(256);
    key RAW(16);
    encrypted_data RAW(256);
    decrypted_data RAW(256);
BEGIN
    data := '0102030405AE030D';
    key := '0A123B8E002CD3FF';
    DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(input => data, key => key,
            encrypted_data => encrypted_data);
    data := encrypted_data;
    DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(input => data, key => key,
            decrypted_data => decrypted_data);
END;
```

24.2.5. DESENCRYPT

DES 알고리즘을 이용하여 데이터를 암호화하는 프러시저와 함수이다. 암호화를 위해 주어진 키는 반드시 8bytes(64bits)이어야 하며, 그렇지 않으면 예외 상황이 발생한다.

DESENCRYPT 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.DESENCRYPT
(
    input               IN          RAW,
    key                 IN          RAW,
    encrypted_data      OUT         RAW
);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESENCRYPT
(
    input_string        IN          VARCHAR2,
```

```
    key_string          IN          VARCHAR2,
    encrypted_string    OUT         VARCHAR2
);
```

– 함수

```
DBMS_OBFUSCATION_TOOLKIT.DESENCRYPT
(
    input               IN          RAW,
    key                 IN          RAW
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DESENCRYPT
(
    input_string        IN          VARCHAR2,
    key_string          IN          VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
input, input_string	암호화할 데이터이다.
key	암호화하기 위해 주어진 키 값이다.
encrypted_data	암호화된 결과 데이터이다.

- 예외 상황

예외 상황	설명
INVALID_ARGUMENT	파라미터 중 하나라도 NULL인 경우이다.
INVALID_INPUT	input_data의 길이가 8의 배수가 아닌 경우이다.
KEY_TOO_SHORT	키 값의 길이가 8보다 작은 경우이다.

- 예제

```
DECLARE
    data RAW(256);
    key RAW(16);
    encrypted_data RAW(256);
    decrypted_data RAW(256);
BEGIN
    data := '0102030405AE030D';
    key := '0A123B8E002CD3FF';
    DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(input => data, key => key,
            encrypted_data => encrypted_data);
```

```
    data := encrypted_data;
    DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(input => data, key => key,
            decrypted_data => decrypted_data);
END;
```

24.2.6. DESGETKEY

임의의 값을 입력 값으로 받아 DES 알고리즘을 위한 키를 생성하는 프러시저와 함수이다.

DESGETKEY 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.DESGETKEY
(
    seed            IN          RAW,
    key             OUT         RAW
);
```

```
DBMS_OBFUSCATION_TOOLKIT.DESGETKEY
(
    seed_string     IN          VARCHAR2,
    key             OUT         VARCHAR2
);
```

  - 함수

```
DBMS_OBFUSCATION_TOOLKIT.DESGETKEY
(
    seed            IN          RAW
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.DESGETKEY
(
    seed_string     IN          VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
seed	80자 이상의 임의의 값이다.
key	암호화하기 위해 주어진 키 값이다.

- 예외 상황

예외 상황	설명
NO_SEED	seed의 값이 NULL인 경우이다.
SEED_TOO_SHORT	seed의 길이가 80보다 작은 경우이다.

- 예제

```
DECLARE
    data VARCHAR2(4096);
    key VARCHAR2(4096);
 key_seed VARCHAR2(4096);
    encrypted_data VARCHAR2(4096);
    decrypted_data VARCHAR2(4096);
BEGIN
    data := '0102030405AE030D';
    key_seed := '1234567890'||'1234567890'||'1234567890'||'1234567890';
    key_seed := rpad(key_seed,80);
    key := dbms_obfuscation_toolkit.DESGetKey(seed_string => key_seed);
    DBMS_OBFUSCATION_TOOLKIT.DESEncrypt(input_string => data, key_string => key,
            encrypted_string=> encrypted_data);
    data := encrypted_data;
    DBMS_OBFUSCATION_TOOLKIT.DESDecrypt(input_string => data, key_string => key,
            decrypted_string => decrypted_data);
END;
/
```

24.2.7. MD5

임의의 값을 입력 값으로 받아 MD5 알고리즘의 checksum을 생성하는 함수이다.

MD5 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 프러시저

```
DBMS_OBFUSCATION_TOOLKIT.MD5
(
    input           IN          RAW,
    checksum        OUT         RAW
);
```

```
DBMS_OBFUSCATION_TOOLKIT.MD5
(
```

```
    input_string            IN          VARCHAR2,
    checksum_string         OUT         VARCHAR2
);
```

– 함수

```
DBMS_OBFUSCATION_TOOLKIT.MD5
(
    input           IN          RAW
)
RETURN RAW;
```

```
DBMS_OBFUSCATION_TOOLKIT.MD5
(
    input_string          IN          VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
input	임의의 암호화할 값이다.
checksum	input으로부터 구한 checksum 값이다.

- 예외 상황

예외 상황	설명
NO_DATA_PASSED	input의 값이 NULL인 경우이다.

제25장 DBMS_OUTPUT

본 장에서는 DBMS_OUTPUT 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

25.1. 개요

DBMS_OUTPUT은 메시지를 버퍼에 저장하고 버퍼로부터 메시지를 읽어오기 위한 인터페이스를 제공하는 패키지이다. 하나의 프러시저, 함수, 트리거 등에 의해 저장된 메시지는 다른 프러시저, 함수, 트리거 등에서 읽어올 수 있다.

DBMS_OUTPUT 패키지 내의 ENABLE 프러시저를 실행하여 메시지를 주고받기 위한 버퍼를 지정된 크기로 할당한다. 만약 메시지를 지정된 버퍼 크기 이상으로 저장하려고 하면 예외 상황이 발생한다.

할당된 버퍼는 DISABLE 프러시저를 실행하여 제거할 수 있으며, 다시 ENABLE 프러시저를 실행하기 전에는 GET_LINE 및 GET_LINES 또는 PUT 및 PUT_LINE, NEW_LINE 프러시저를 호출해도 무시된다. 버퍼 내의 메시지는 여러 라인으로 구성되어 있으며, 라인마다 라인 끝(End of Line, 이하 EOL) 문자로 끝난다.

PUT_LINE 프러시저를 통해 버퍼에 저장된 메시지는 GET_LINE이나 GET_LINES 프러시저를 실행하여 읽어올 수 있다.

다음은 DBMS_OUTPUT 패키지 내에 정의된 타입이다.

- CHARARR

  메시지 버퍼에서 읽어온 내용을 저장하기 위한 공간이다.

```
TYPE CHARARR IS TABLE OF VARCHAR2(32767)
```

25.2. 프러시저

본 절에서는 DBMS_OUTPUT 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

25.2.1. DISABLE

할당된 메시지 버퍼를 제거하고, DBMS_OUTPUT 패키지 내의 다른 프러시저를 사용할 수 없게 하는 프러시저이다. 이 프로서지를 실행하면 메시지 버퍼에 남아있는 모든 메시지가 함께 제거된다.

DISABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_OUTPUT.DISABLE;
```

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Before DISABLE');
    DBMS_OUTPUT.DISABLE;
    DBMS_OUTPUT.PUT_LINE('After DISABLE');
END;
/

PSM completed
SQL>
```

25.2.2. ENABLE

지정된 크기의 메시지 버퍼를 할당하고 DBMS_OUTPUT 패키지 내의 다른 프러시저를 사용할 수 있게 하는 프러시저이다. 이 프러시저를 여러 번 호출하면 가장 크게 지정한 크기로 메시지 버퍼를 할당한다. 이때 메시지 버퍼의 크기는 최소 2KB, 최대 1MB이다.

ENABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_OUTPUT.ENABLE
(
    buffer_size     IN          INTEGER     DEFAULT 20000
);
```

- 파라미터

파라미터	설명
buffer_size	할당할 메시지 버퍼의 크기이다. (단위: Byte)

- 예제

```
BEGIN
    DBMS_OUTPUT.DISABLE;
    DBMS_OUTPUT.PUT_LINE('Before ENABLE');
    DBMS_OUTPUT.ENABLE(32768);
    DBMS_OUTPUT.PUT_LINE('After ENABLE');
END;
```

```
/
After ENABLE

PSM completed
SQL>
```

25.2.3. GET_LINE, GET_LINES

메시지 버퍼로부터 라인 단위로 메시지를 읽어오는 프러시저이다. 이 프러시저는 라인 단위로 메시지를 읽으며, 하나의 라인을 형성하지 않은 메시지는 읽지 않는다.

GET_LINE 또는 GET_LINES 프러시저를 호출한 후에 PUT 또는 PUT_LINE 프러시저를 호출하면, 현재까지 메시지 버퍼에 남아 있던 메시지는 모두 제거된다.

한 라인의 메시지는 최대 255bytes의 크기를 가지므로, 출력 파라미터의 크기를 충분하게 설정해야 한다.

GET_LINE, GET_LINES 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - GET_LINE

    GET_LINE 프러시저는 한 번 호출될 때마다 하나의 라인만을 읽어온다.

    ```
    DBMS_OUTPUT.GET_LINE
    (
        line            OUT         VARCHAR,
        status          OUT         INTEGER
    );
    ```

  - GET_LINES

    GET_LINES 프러시저는 지정된 수만큼, 한 번에 여러 라인의 메시지를 읽어온다. 이때 메시지 버퍼로부터 실제로 읽어온 메시지 라인의 수를 반환한다. 만약 메시지 버퍼 내에 충분한 수의 메시지 라인이 없어서 지정된 수만큼의 메시지 라인을 가져오지 못하면, 가져온 메시지 라인의 수만큼만 저장된다.

    ```
    DBMS_OUTPUT.GET_LINES
    (
        lines           OUT         CHARARR,
        numlines       IN OUT      INTEGER
    );
    ```

- 파라미터

파라미터	설명
line, lines	메시지 버퍼로부터 읽어 온 한 라인 또는 여러 라인의 메시지이다.

파라미터	설명
status	– 메시지를 성공적으로 읽어 온 경우에는 0을 반환한다. – 메시지를 성공적으로 읽어 오지 못한 경우에는 1을 반환한다.
numlines	읽어 올 메시지 라인의 수를 입력하고, 실제로 읽어 온 메시지 라인의 수를 출력한다.

- 예제

  – GET_LINE

```
DECLARE
    message VARCHAR(1024);
    status INTEGER;
BEGIN
    DBMS_OUTPUT.PUT_LINE('A poet is the painter of the soul');
    DBMS_OUTPUT.PUT_LINE('Faith without deeds is useless');
    DBMS_OUTPUT.PUT_LINE('Forgiveness is better than revenge');
    DBMS_OUTPUT.GET_LINE(message, status);
    DBMS_OUTPUT.PUT_LINE(message);
END;
/
A poet is the painter of the soul

PSM completed
SQL>
```

  – GET_LINES

```
DECLARE
    message_arr DBMS_OUTPUT.CHARARR;
    num_lines INTEGER := 4;
BEGIN
    DBMS_OUTPUT.PUT('A poet is ');
    DBMS_OUTPUT.PUT('the painter of the soul');
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('Faith without deeds is useless');
    DBMS_OUTPUT.PUT_LINE('Forgiveness is better than revenge');
    DBMS_OUTPUT.GET_LINES(message_arr, num_lines);
    DBMS_OUTPUT.PUT_LINE(message_arr(2));
END;
/
Faith without deeds is useless

PSM completed
SQL>
```

25.2.4. NEW_LINE

메시지 버퍼에 EOL 문자를 저장하는 프러시저이다.

PUT 프러시저로 메시지를 저장한 경우 EOL 문자가 없으므로 라인 단위로 메시지를 읽는 GET_LINE, GET_LINES 프러시저로 문자열을 읽어 올 수 없다. 단, NEW_LINE 프러시저를 사용하면 GET_LINE, GET_LINES 프러시저로 문자열을 읽어 올 수 있다.

NEW_LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_OUTPUT.NEW_LINE;
```

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('The will of a man is his happiness');
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('Love your neighbor as yourself');
END;
/
The will of a man is his happiness

Love your neighbor as yourself

PSM completed
SQL>
```

25.2.5. PUT, PUT_LINE

메시지 버퍼에 메시지를 저장하는 프러시저이다.

PUT, PUT_LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

  – PUT

    PUT 프러시저는 하나의 라인을 여러 번에 걸쳐 저장할 수 있다. 버퍼에 저장되는 메시지의 마지막 라인 끝에 EOL 문자가 첨부되지 않는다.

```
DBMS_OUTPUT.PUT
(
    data IN NUMBER
);
```

```
DBMS_OUTPUT.PUT
(
   data IN VARCHAR
);
```

– PUT_LINE

PUT_LINE 프러시저는 PUT 프러시저와는 반대로 버퍼에 저장되는 메시지의 마지막 라인 끝에 EOL 문자를 첨부한다. 만약 EOL 문자만을 저장하려면 NEW_LINE 프러시저를 호출한다.

```
DBMS_OUTPUT.PUT_LINE
(
   data IN NUMBER
);
```

```
DBMS_OUTPUT.PUT_LINE
(
   data IN VARCHAR
);
```

- 파라미터

입력 값으로 주어진 파라미터의 데이터 타입은 NUMBER, VARCHAR, DATE 세 가지이며, 메시지로 저장되기 전에 항상 VARCHAR 타입으로 변환된다. NUMBER와 DATE 타입의 데이터는 TO_CHAR 함수를 적용하여 반환된 결과로 저장된다.

파라미터	설명
data	메시지 버퍼에 저장할 메시지 데이터이다.

- 예외 상황

예외 상황	설명
BUF_OVERFLOW	메시지 버퍼의 최댓값을 초과한 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('PUT_LINE_EXAMPLE');
    DBMS_OUTPUT.PUT('PUT_EXAMPLE');
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT(10000);
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('TIBERO');
    DBMS_OUTPUT.PUT_LINE(10000);
END;
/
PUT_LINE_EXAMPLE
PUT_EXAMPLE
```

```
10000
TIBERO
10000

PSM completed.
SQL>
```

제26장 DBMS_PARALLEL_EXECUTE

본 장에서는 DBMS_PARALLEL_EXECUTE 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

26.1. 개요

DBMS_PARALLEL_EXECUTE는 병렬 방식으로 테이블을 업데이트할 수 있는 기능을 제공하는 패키지이다.

이 패키지는 테이블의 행 집합을 CHUNK 단위로 그룹화한 뒤, 각 CHUNK에 대해 사용자 지정 SQL을 병렬로 수행하고, 각 CHUNK의 처리가 완료되면 COMMIT을 수행한다.

다음은 DBMS_PARALLEL_EXECUTE 패키지를 사용할 때 유의해야 할 사항이다.

- 호출자 권한 : 해당 패키지는 호출자 권한으로 실행된다. 만약 다른 프러시저나 함수에서 호출되는 경우 해당 프러시저나 함수의 실행자 권한으로 실행된다.
- 병렬 실행 시 권한 : 병렬 실행은 DBMS_SCHEDULER의 CREATE_JOB을 통해 수행된다. 호출자는 CREATE_JOB 권한이 있어야 한다.
- SQL 수행 권한 : CREATE_CHUNKS_BY_SQL, RUN_TASK, RESUME_TASK는 SQL을 필요로 하며 내부적으로 DBMS_SQL을 사용한다.

26.2. 상수

본 절에서는 DBMS_PARALLEL_EXECUTE 패키지에서 제공하는 상수를 알파벳 순으로 설명한다.

26.2.1. CHUNK 상태 상수

CHUNK 상태를 표현한 상수는 아래와 같다.

상수 이름	상수 타입	상수 값	설명
ASSIGNED	NUMBER	1	CHUNK가 작업 처리를 위해 할당된 상태이다.
PROCESSED	NUMBER	2	CHUNK에 대한 작업 처리가 성공적으로 수행되었다.

상수 이름	상수 타입	상수 값	설명
PROCESSED_WITH_ERROR	NUMBER	3	CHUNK에 대한 작업 처리가 완료되었으나, 수행 도중 에러가 발생하였다.
UNASSIGNED	NUMBER	0	CHUNK가 작업 처리에 할당되지 않은 상태이다.

26.2.2. TASK 상태 상수

TASK 상태를 표현한 상수는 아래와 같다.

상수 이름	상수 타입	상수 값	설명
CHUNKED	NUMBER	5	CHUNK 생성에 성공하였으나, 아직 CHUNK들이 ASSIGN되거나 PROCESS되지 않았다.
CHUNKING	NUMBER	2	CHUNK 생성 작업이 진행 중이다.
CHUNKING_FAILED	NUMBER	3	CHUNK 생성 작업에 실패하였다.
CRASHED	NUMBER	9	CHUNK의 상태가 ASSIGNED 혹은 UNASSIGNED인 상태에서 JOB이 CRASH되었다. 이 상태 상수는 병렬 수행 시에만 발생 가능하다.
CREATED	NUMBER	1	TASK가 생성되었다.
FINISHED	NUMBER	7	모든 CHUNK의 작업 처리가 에러 없이 수행되었다.
FINISHED_WITH_ERROR	NUMBER	8	모든 CHUNK의 작업 처리가 완료되었으나, 에러가 발생한 CHUNK들이 있다.
NO_CHUNKS	NUMBER	4	TASK와 연관된 테이블로 생성된 CHUNK가 없다.
PROCESSING	NUMBER	6	CHUNK에 대한 작업 처리가 수행 중이다.

26.3. 예외

다음은 DBMS_PARALLEL_EXECUTE 패키지에서 미리 제공된 예외를 알파벳 순으로 설명한다.

예외 이름	예외 코드	설명
CHUNK_NOT_FOUND	14329	지시된 CHUNK를 찾지 못했다.
DUPLICATE_TASK_NAME	14327	동일한 TASK 이름이 이미 같은 사용자에게서 생성되어 있다.
INVALID_STATE_FOR_CHUNK	14322	CREATED 혹은 CHUNKING_FAILED 상태가 아닌 TASK에 대해 CHUNK 생성 작업을 시도하였다.
INVALID_STATE_FOR_RESUME	14325	작업 처리를 재개하려 했으나, TASK의 상태가 FINISHED_WITH_ERROR 혹은 CRASHED가 아니다.
INVALID_STATE_FOR_RUN	14324	작업 처리를 수행하려 했으나, TASK의 상태가 CHUNKED가 아니다.
INVALID_STATUS	14323	CHUNK의 상태 변경에 잘못된 변수가 지정되었다.
INVALID_STRING_PARAM	14326	입력받은 변수가 NULL이거나, 허용된 길이를 초과하였다.
INVALID_TABLE	14321	CHUNK 생성을 시도한 테이블이 물리적인 테이블이 아니거나 Index-Organized Table이다.
TASK_NOT_FOUND	14328	명시된 task_name을 찾지 못하였다.

26.4. 실체화 뷰

다음은 DBMS_PARALLEL_EXECUTE 패키지에서 사용되는 뷰를 알파벳 순으로 나열한다.

- DBA_PARALLEL_EXECUTE_CHUNKS
- DBA_PARALLEL_EXECUTE_TASKS
- USER_PARALLEL_EXECUTE_CHUNKS
- USER_PARALLEL_EXECUTE_TASKS

26.5. 프러시저와 함수

본 절에서는 DBMS_PARALLEL_EXECUTE 패키지에서 제공하는 프러시저와 함수를 설명한다.

26.5.1. CREATE_TASK

호출자에 대한 TASK를 생성하는 프러시저이다. TASK 이름과 이 프러시저를 호출하는 호출자는 고유해야 한다.

CREATE_TASK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.CREATE_TASK
(
    task_name                   IN          VARCHAR2,
    comment                     IN          VARCHAR2    DEFAULT NULL
);
```

- 파라미터

파라미터	설명
task_name	생성할 TASK의 이름이다. 128바이트까지 허용된다.
comment	생성할 TASK를 설명하는 정보이다. 4000바이트까지 허용된다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
DUPLI CATE_TASK_NAME	해당 유저로 동일한 TASK 이름이 이미 존재하는 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.CREATE_TASK(
   2      'SAMPLE_TASK',
   3      'This is a sample task'
   4 );

PSM completed.

SQL> select
   2     task_owner,
   3     task_name,
   4     status,
   5     task_comment
   6 from DBA_PARALLEL_EXECUTE_TASKS;

TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS
--------------------
TASK_COMMENT
--------------------------------------------------------------------------------
```

```
TIBERO
SAMPLE_TASK
CREATED
This is a sample task


1 row selected.
```

26.5.2. CREATE_CHUNKS_BY_NUMBER_COL

지정된 컬럼에 따라 CHUNKS를 생성하는 프러시저이다. 지정된 컬럼은 NUMBER 타입이어야 한다.

이 프러시저는 지정된 컬럼의 최솟값과 최댓값을 확인한 다음, chunk_size에 따라 범위를 균등하게 나눈다.

CREATE_CHUNKS_BY_NUMBER_COL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_NUMBER_COL
(
    task_name                   IN          VARCHAR2,
    table_owner                 IN          VARCHAR2,
    table_name                  IN          VARCHAR2,
    table_column                IN          VARCHAR2,
    chunk_size                  IN          NUMBER
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
table_owner	테이블의 소유자 이름이다.
table_name	테이블의 이름이다.
table_column	테이블의 컬럼 이름이다. NUMBER 타입이어야 한다.
chunk_size	각 CHUNK의 범위이다.

- 예외 상황

예외 상황	설명
IN VALID_STATE_FOR_CHUNK	현재 TASK의 상태가 CREATED 혹은 CHUNKING_FAILED가 아닌 경우이다.
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.

예외 상황	설명
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_NUMBER_COL(
   2      'SAMPLE_TASK',
   3      'TIBERO',
   4      'SAMPLE_TABLE',
   5      'SAMPLE_COLUMN',
   6      1000
   7 );

PSM completed.

SQL> select
   2     task_owner,
   3     task_name,
   4     chunk_type,
   5     status,
   6     table_owner,
   7     table_name,
   8     number_column
   9 from DBA_PARALLEL_EXECUTE_TASKS;

TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
CHUNK_TYPE   STATUS
------------ --------------------
TABLE_OWNER
--------------------------------------------------------------------------------
TABLE_NAME
--------------------------------------------------------------------------------
NUMBER_COLUMN
--------------------------------------------------------------------------------
TIBERO
SAMPLE_TASK
NUMBER_RANGE CHUNKED
TIBERO
SAMPLE_TABLE
SAMPLE_COLUMN


1 row selected.
```

```
SQL> select
   2     chunk_id,
   3     task_owner,
   4     task_name,
   5     status,
   6     start_id,
   7     end_id
   8 from DBA_PARALLEL_EXECUTE_CHUNKS;

  CHUNK_ID
----------
TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS                 START_ID     END_ID
-------------------- ---------- ----------
       283
TIBERO
SAMPLE_TASK
UNASSIGNED                    1       1000

       284
TIBERO
SAMPLE_TASK
UNASSIGNED                 1001       2000

       285
TIBERO
SAMPLE_TASK
UNASSIGNED                 2001       3000

       286
TIBERO
SAMPLE_TASK
UNASSIGNED                 3001       4000

...
```

26.5.3. CREATE_CHUNKS_BY_ROWID

ROWID별로 CHUNK를 생성하는 프러시저이다.

테이블의 행 수와 블럭 수는 개별 CHUNK를 생성하는 근사치 기준이다. 이때 테이블은 물리적인 ROWID를 가진 테이블이어야 한다. 단, IOT는 허용하지 않는다.

CREATE_CHUNKS_BY_ROWID 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID
(
    task_name                   IN          VARCHAR2,
    table_owner                 IN          VARCHAR2,
    table_name                  IN          VARCHAR2,
    by_row                      IN          BOOLEAN,
    chunk_size                  IN          NUMBER
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
table_owner	테이블의 소유자 이름이다.
table_name	테이블의 이름이다.
by_row	TRUE일 경우 chunk_size가 row 수에 기반한다. FALSE일 경우 block 수에 기반한다.
chunk_size	각 CHUNK의 범위이다. by_row 인자에 따라 row/block이 달라지며, 근사치로 계산된다.

- 예외 상황

예외 상황	설명
INVALID_TABLE	입력된 테이블이 물리적 테이블이 아니거나 IOT인 경우이다.
IN VALID_STATE_FOR_CHUNK	현재 TASK의 상태가 CREATED 혹은 CHUNKING_FAILED가 아닌 경우이다.
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_ROWID(
   2      'SAMPLE_TASK',
   3      'TIBERO',
   4      'SAMPLE_TABLE',
   5      TRUE,
   6      1000
   7 );
```

```
PSM completed.

SQL> select
   2     task_owner,
   3     task_name,
   4     chunk_type,
   5     status,
   6     table_owner,
   7     table_name
   8 from DBA_PARALLEL_EXECUTE_TASKS;

TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
CHUNK_TYPE   STATUS
------------ --------------------
TABLE_OWNER
--------------------------------------------------------------------------------
TABLE_NAME
--------------------------------------------------------------------------------
TIBERO
SAMPLE_TASK
ROWID_RANGE  CHUNKED
TIBERO
SAMPLE_TABLE


1 row selected.

SQL> select
   2     chunk_id,
   3     task_owner,
   4     task_name,
   5     status,
   6     start_rowid,
   7     end_rowid
   8 from DBA_PARALLEL_EXECUTE_CHUNKS;

  CHUNK_ID
----------
TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS               START_ROWID        END_ROWID
-------------------- ------------------ ------------------
```

```
       192
TIBERO
SAMPLE_TASK
UNASSIGNED           AAAAqlAACAAAAA3AAA AAAAqlAACAAAAAqP//

       193
TIBERO
SAMPLE_TASK
UNASSIGNED           AAAAqlAACAAAAArAAA AAAAqlAACAAAAAtP//

       194
TIBERO
SAMPLE_TASK
UNASSIGNED           AAAAqlAACAAAAAuAAA AAAAqlAACAAAAAwP//

       195
TIBERO
SAMPLE_TASK
UNASSIGNED           AAAAqlAACAAAAAxAAA AAAAqlAACAAAAAzP//

...
```

26.5.4. CREATE_CHUNKS_BY_SQL

사용자 지정 SELECT 문에 의해 CHUNK를 생성하는 프러시저이다.

SELECT 문은 start_id 및 end_id의 두 컬럼으로 각 CHUNK의 범위를 반환해야 한다. by_rowid가 TRUE인 경우 두 컬럼은 ROWID를 반환해야 하며, FALSE인 경우 NUMBER 타입을 반환해야 한다.

CREATE_CHUNKS_BY_SQL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL
(
    task_name                   IN          VARCHAR2,
    sql_stmt                    IN          CLOB,
    by_rowid                    IN          BOOLEAN
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
sql_stmt	CHUNK의 범위를 반환하는 SQL 문이다.

파라미터	설명
by_rowid	CHUNK 생성이 ROWID 기반일 경우 TRUE, NUMBER 기반일 경우 FALSE 이다.

- 예외 상황

예외 상황	설명
IN VALID_STATE_FOR_CHUNK	현재 TASK의 상태가 CREATED 혹은 CHUNKING_FAILED가 아닌 경우이다.
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.CREATE_CHUNKS_BY_SQL(
   2      'SAMPLE_TASK',
   3      'select
   4           ((level * 10000) - 9999) as start_id,
   5           (level * 10000) as end_id
   6       from dual connect by level <= 10',
   7      FALSE
   8 );

PSM completed.

SQL> select
   2     task_owner,
   3     task_name,
   4     chunk_type,
   5     status
   6 from DBA_PARALLEL_EXECUTE_TASKS;

TASK_OWNER
------------------------------------------------------------------------------
TASK_NAME
------------------------------------------------------------------------------
CHUNK_TYPE   STATUS
------------ ------------------
TIBERO
SAMPLE_TASK
NUMBER_RANGE CHUNKED


1 row selected.
```

```
SQL> select
   2     chunk_id,
   3     task_owner,
   4     task_name,
   5     status,
   6     start_id,
   7     end_id
   8 from DBA_PARALLEL_EXECUTE_CHUNKS;

  CHUNK_ID
----------
TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS                 START_ID     END_ID
-------------------- ---------- ----------
       383
TIBERO
SAMPLE_TASK
UNASSIGNED                    1      10000

       384
TIBERO
SAMPLE_TASK
UNASSIGNED                10001      20000

       385
TIBERO
SAMPLE_TASK
UNASSIGNED                20001      30000

       386
TIBERO
SAMPLE_TASK
UNASSIGNED                30001      40000

...
```

26.5.5. DROP_TASK

해당 TASK와 연관된 CHUNK까지 모두 삭제하는 프러시저이다.

DROP_TASK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.DROP_TASK
(
    task_name                   IN          VARCHAR2
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.DROP_TASK('SAMPLE_TASK');

PSM completed.

SQL> select count(*) from DBA_PARALLEL_EXECUTE_TASKS;

  COUNT(*)
----------
         0

1 row selected.
```

26.5.6. DROP_CHUNKS

TASK에 연관된 모든 CHUNK들을 삭제하는 프러시저이다.

DROP_CHUNKS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.DROP_CHUNKS
(
    task_name                   IN          VARCHAR2
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.DROP_CHUNKS('SAMPLE_TASK');

PSM completed.

SQL> select count(*) from DBA_PARALLEL_EXECUTE_CHUNKS;

  COUNT(*)
----------
         0

1 row selected.
```

26.5.7. GENERATE_TASK_NAME

고유한 TASK 이름을 생성하여 반환한다.

GENERATE_TASK_NAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.GENERATE_TASK_NAME
(
    prefix                      IN          VARCHAR2    DEFAULT 'TASK$_'
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
prefix	TASK 이름을 생성할 때 접두사로 쓰일 문자열이다.

- 예제

```
SQL> set serveroutput on;
SQL> DECLARE
   2     TASK_NAME VARCHAR2(128);
   3 BEGIN
   4     TASK_NAME := DBMS_PARALLEL_EXECUTE.GENERATE_TASK_NAME('SAMPLE_NAME$_');
   5     DBMS_OUTPUT.PUT_LINE(TASK_NAME);
   6 END;
   7 /
SAMPLE_NAME$_393

PSM completed.
```

26.5.8. GET_NUMBER_COL_CHUNK

이 프러시저를 수행하면 할당되지 않은 NUMBER 타입의 CHUNK 하나를 골라 상태를 ASSIGNED로 변경한다.

할당할 수 있는 CHUNK가 없으면 any_rows를 FALSE로 반환하며, 그렇지 않을 경우에는 chunk_id, start_id, end_id의 값이 반환된다.

DBA_PARALLEL_EXECUTE_CHUNKS, USER_PARALLEL_EXECUTE_CHUNKS 뷰에 대한 CHUNK 정보 업데이트는 다음의 순서로 이루어진다.

1. STATUS를 ASSIGNED로 변경

2. START_TIMESTAMP를 현재 시각으로 변경

3. END_TIMESTAMP는 삭제

GET_NUMBER_COL_CHUNK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.GET_NUMBER_COL_CHUNK
(
    task_name                   IN          VARCHAR2,
    chunk_id                    OUT         NUMBER,
    start_id                    OUT         NUMBER,
    end_id                      OUT         NUMBER,
    any_rows                    OUT         BOOLEAN
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
chunk_id	반환될 CHUNK의 chunk_id 값이다.
start_id	반환될 CHUNK의 start_id 값이다.
end_id	반환될 CHUNK의 end_id 값이다.
any_rows	할당할 수 있는 CHUNK가 있는지 여부를 반환한다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> set serveroutput on;
SQL> DECLARE
   2     CHUNK_ID NUMBER;
   3     START_ID NUMBER;
   4     END_ID NUMBER;
   5     ANY_ROWS BOOLEAN;
   6 BEGIN
   7     DBMS_PARALLEL_EXECUTE.GET_NUMBER_COL_CHUNK(
   8         'SAMPLE_TASK', CHUNK_ID, START_ID, END_ID, ANY_ROWS);
   9     IF ANY_ROWS = TRUE THEN
  10         DBMS_OUTPUT.PUT_LINE('CHUNK ID : ' || CHUNK_ID);
  11         DBMS_OUTPUT.PUT_LINE('START_ID : ' || START_ID);
  12         DBMS_OUTPUT.PUT_LINE('END_ID : ' || END_ID);
  13     ELSE
  14         DBMS_OUTPUT.PUT_LINE('NO UNASSIGNED CHUNKS');
  15     END IF;
  16 END;
  17 /
CHUNK ID : 514
START_ID : 1
END_ID : 1000

PSM completed.
```

26.5.9. GET_ROWID_CHUNK

이 프러시저를 수행하면 할당되지 않은 ROWID 타입의 CHUNK 하나를 골라 상태를 ASSIGNED로 변경한다.

할당할 수 있는 CHUNK가 없으면 any_rows를 FALSE로 반환하며, 그렇지 않을 경우에는 chunk_id, start_rowid, end_rowid의 값이 반환된다.

DBA_PARALLEL_EXECUTE_CHUNKS, USER_PARALLEL_EXECUTE_CHUNKS 뷰에 대한 CHUNK 정보 업데이트는 다음의 순서로 이루어진다.

1. STATUS를 ASSIGNED로 변경

2. START_TIMESTAMP를 현재 시각으로 변경

3. END_TIMESTAMP는 삭제

GET_ROWID_CHUNK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.GET_ROWID_CHUNK
(
    task_name                   IN          VARCHAR2,
    chunk_id                    OUT         NUMBER,
    start_rowid                 OUT         ROWID,
    end_rowid                   OUT         ROWID,
    any_rows                    OUT         BOOLEAN
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
chunk_id	반환될 CHUNK의 chunk_id 값이다.
start_rowid	반환될 CHUNK의 start_rowid 값이다.
end_rowid	반환될 CHUNK의 end_rowid 값이다.
any_rows	할당할 수 있는 CHUNK가 있는지 여부를 반환한다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> set serveroutput on;
SQL> DECLARE
   2     CHUNK_ID NUMBER;
   3     START_ROWID ROWID;
   4     END_ROWID ROWID;
   5     ANY_ROWS BOOLEAN;
   6 BEGIN
   7     DBMS_PARALLEL_EXECUTE.GET_ROWID_CHUNK(
   8         'SAMPLE_TASK', CHUNK_ID, START_ROWID, END_ROWID, ANY_ROWS);
   9     IF ANY_ROWS = TRUE THEN
  10         DBMS_OUTPUT.PUT_LINE('CHUNK ID : ' || CHUNK_ID);
  11         DBMS_OUTPUT.PUT_LINE('START_ROWID : ' || START_ROWID);
  12         DBMS_OUTPUT.PUT_LINE('END_ROWID : ' || END_ROWID);
  13     ELSE
  14         DBMS_OUTPUT.PUT_LINE('NO UNASSIGNED CHUNKS');
  15     END IF;
  16 END;
  17 /
CHUNK ID : 2514
START_ROWID : AAAAqmAACAAAAE3AAA
END_ROWID : AAAAqmAACAAAAEpP//

PSM completed.
```

26.5.10. PURGE_PROCESSED_CHUNKS

PROCESSED 혹은 PROCESSED_WITH_ERROR 상태의 CHUNK들을 모두 삭제한다.

PURGE_PROCESSED_CHUNKS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.PURGE_PROCESSED_CHUNKS
(
    task_name                   IN          VARCHAR2
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> select count(*) from DBA_PARALLEL_EXECUTE_CHUNKS
   2 where status != 'UNASSIGNED';

  COUNT(*)
----------
       100

1 row selected.

SQL> exec DBMS_PARALLEL_EXECUTE.PURGE_PROCESSED_CHUNKS('SAMPLE_TASK');

PSM completed.

SQL> select count(*) from DBA_PARALLEL_EXECUTE_CHUNKS
   2 where status != 'UNASSIGNED';

  COUNT(*)
----------
         0

1 row selected.
```

26.5.11. RESUME_TASK

이 프러시저는 RUN_TASK 프러시저가 에러와 함께 완료되었거나 충돌이 발생한 경우 사용한다. 즉 TASK의 상태 정보가 PROCESSED_WITH_ERROR 혹은 CRASHED일 경우에만 사용할 수 있다.

RUN_TASK가 병렬 수행이 아닌 경우 충돌이 발생하면 상태 정보가 PROCESSING으로 유지된다. 이 경우 force 옵션을 TRUE로 사용하면 처리 작업을 재개할 수 있다. 그러나 충돌이 발생했는지 확인하는 것은 사용자의 책임 하에 있다.

이 프러시저는 처리되지 않은 CHUNK들의 처리 작업을 재개한다. PROCESSED_WITH_ERROR 혹은 ASSIGNED 상태의 CHUNK들 또한 COMMIT되지 않았으므로 처리 작업이 재개된다.

이 프러시저는 RUN_TASK 프러시저와 동일한 인자를 사용하며, task_name만을 사용하도록 오버로드된 함수의 경우 이전 RUN_TASK 혹은 RESUME_TASK 호출 시 제공된 변수를 재사용한다.

RESUME_TASK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.RESUME_TASK
(
    task_name                   IN          VARCHAR2,
    force                       IN          BOOLEAN     DEFAULT FALSE
);
```

```
DBMS_PARALLEL_EXECUTE.RESUME_TASK
(
    task_name                   IN          VARCHAR2,
    sql_stmt                    IN          CLOB,
    language_flag               IN          NUMBER,
    edition                     IN          VARCHAR2    DEFAULT NULL,
    apply_crossedition_trigger  IN          VARCHAR2    DEFAULT NULL,
    fire_apply_trigger          IN          BOOLEAN     DEFAULT TRUE,
    parallel_level              IN          NUMBER      DEFAULT 0,
    job_class                   IN          VARCHAR2    DEFAULT
                                                            'DEFAULT_JOB_CLASS',
    force                       IN          BOOLEAN     DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
sql_stmt	수행할 SQL 문장이다. 이 SQL은 :start_id와 :end_id 매개 변수를 포함해야 한다.
language_flag	지원되지 않는 기능이므로 값을 무시한다.
edition	지원되지 않는 기능이므로 값을 무시한다.
apply_crossedition_trigger	지원되지 않는 기능이므로 값을 무시한다.
fire_apply_trigger	지원되지 않는 기능이므로 값을 무시한다.
parallel_level	수행될 병렬 작업 수이다. 0이면 serial하게 수행되며, NULL일 경우 기본 병렬 수행 정도를 따른다.
job_class	병렬 수행될 경우 속할 JOB CLASS이다.
force	TRUE로 설정하면 상태 정보가 PROCESSING일 때 에러를 발생시키지 않고 처리 작업을 재개한다.

- 예외 상황

예외 상황	설명
IN VALID_STATE_FOR_RESUME	TASK의 상태가 FINISHED_WITH_ERROR 혹은 CRASHED가 아닌 경우이다.
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
[SESSION #1]
SQL> select sample_column2, count(*) from sample_table group by sample_column2;

SAMPLE_COLUMN2   COUNT(*)
-------------- ----------
             1     100000

SQL> exec DBMS_PARALLEL_EXECUTE.RUN_TASK(
   2      'SAMPLE_TASK',
   3      'update sample_table set sample_column2 = -1
   4       where sample_column between :start_id and :end_id',
   5      DBMS_SQL.NATIVE,
   6      NULL,
   7      NULL,
   8      FALSE,
   9      0,
  10      'DEFAULT_JOB_CLASS'
  11 );

PSM completed.


[SESSION #2]
SQL> exec DBMS_PARALLEL_EXECUTE.STOP_TASK('SAMPLE_TASK');

PSM completed.

SQL> select sample_column2, count(*) from sample_table group by sample_column2;

SAMPLE_COLUMN2   COUNT(*)
-------------- ----------
            -1      38000
             1      62000

2 rows selected.
```

```
SQL> exec DBMS_PARALLEL_EXECUTE.RESUME_TASK(
   2      'SAMPLE_TASK',
   3      'update sample_table set sample_column2 = -2
   4       where sample_column between :start_id and :end_id',
   5      DBMS_SQL.NATIVE,
   6      NULL,
   7      NULL,
   8      FALSE,
   9      0,
  10      'DEFAULT_JOB_CLASS',
  11      TRUE
  12 );

PSM completed.

SQL> select sample_column2, count(*) from sample_table group by sample_column2;

SAMPLE_COLUMN2   COUNT(*)
-------------- ----------
            -2      62000
            -1      38000

2 rows selected.
```

26.5.12. RUN_TASK

지정된 SQL 문장(sql_stmt)을 수행한다. 해당 SQL은 호출자의 권한으로 수행된다.

지정된 SQL 문장에는 CHUNK의 범위를 나타내는 :start_id와 :end_id 매개 변수가 있어야 한다. 각 매개 변수는 ROWID(ROWID 타입의 CHUNK를 생성한 경우) 혹은 NUMBER(NUMBER 타입의 CHUNK를 생성한 경우) 타입이어야 한다.

병렬 수행이 요청되면(parallel_level) DBMS_SCHEDULER를 통해 JOB 세션을 할당받아 수행된다. 때문에 병렬 수행을 위해서는 DBMS_SCHEDULER 패키지의 CREATE_JOB 프러시저 권한이 있어야 한다. parallel_level에 NULL을 입력하는 경우 기본 병렬 수행 수치를 계산하며, 이는 CPU_COUNT 파라미터와 PARALLEL_THREADS_PER_CPU 파라미터를 사용하여 계산된다.

이 프러시저는 모든 CHUNK들의 수행이 완료된 후에 종료되며, 병렬 수행의 경우 JOB 세션들의 작업이 완료된 후 종료된다.

RUN_TASK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.RUN_TASK
(
```

```
    task_name                    IN          VARCHAR2,
    sql_stmt                     IN          CLOB,
    language_flag                IN          NUMBER,
    edition                      IN          VARCHAR2    DEFAULT NULL,
    apply_crossedition_trigger   IN          VARCHAR2    DEFAULT NULL,
    fire_apply_trigger           IN          BOOLEAN     DEFAULT TRUE,
    parallel_level               IN          NUMBER      DEFAULT 0,
    job_class                    IN          VARCHAR2    DEFAULT
                                                             'DEFAULT_JOB_CLASS'
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
sql_stmt	수행할 SQL 문장이다. 이 SQL은 :start_id와 :end_id 매개 변수를 포함해야 한다.
language_flag	지원되지 않는 기능이므로 값을 무시한다.
edition	지원되지 않는 기능이므로 값을 무시한다.
apply_crossedition_trigger	지원되지 않는 기능이므로 값을 무시한다.
fire_apply_trigger	지원되지 않는 기능이므로 값을 무시한다.
parallel_level	수행될 병렬 작업 수이다. 0이면 serial하게 수행되며, NULL일 경우 기본 병렬 수행 정도를 따른다.
job_class	병렬 수행될 경우 속할 JOB CLASS이다.

- 예외 상황

예외 상황	설명
IN VALID_STATE_FOR_RUN	TASK의 상태가 CHUNKED가 아닌 경우이다.
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> select count(*) from DBA_PARALLEL_EXECUTE_CHUNKS
   2 where status = 'UNASSIGNED';

  COUNT(*)
----------
```

```
       100

1 row selected.

SQL> select task_owner, task_name, status from DBA_PARALLEL_EXECUTE_TASKS;

TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS
-------------------
TIBERO
SAMPLE_TASK
CHUNKED


1 row selected.

SQL> exec DBMS_PARALLEL_EXECUTE.RUN_TASK(
   2      'SAMPLE_TASK',
   3      'update sample_table set sample_column2 = -1
   4       where sample_column between :start_id and :end_id',
   5      DBMS_SQL.NATIVE,
   6      NULL,
   7      NULL,
   8      FALSE,
   9      4,
  10      'DEFAULT_JOB_CLASS'
  11 );

PSM completed.

SQL> select count(*) from DBA_PARALLEL_EXECUTE_CHUNKS
   2 where status = 'UNASSIGNED';

  COUNT(*)
----------
         0

1 row selected.

SQL> select task_owner, task_name, status from DBA_PARALLEL_EXECUTE_TASKS;

TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
```

```
--------------------------------------------------------------------------------
STATUS
------------------
TIBERO
SAMPLE_TASK
FINISHED


1 row selected.
```

26.5.13. SET_CHUNK_STATUS

CHUNK 하나의 상태 정보를 변경한다.

SET_CHUNK_STATUS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS
(
    task_name                   IN          VARCHAR2,
    chunk_id                    IN          NUMBER,
    status                      IN          NUMBER,
    err_num                     IN          NUMBER      DEFAULT NULL,
    err_msg                     IN          VARCHAR2    DEFAULT NULL,
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.
chunk_id	상태 정보를 변경할 CHUNK의 chunk_id 값이다.
status	변경할 상태 정보이다.
err_num	변경할 에러 코드 번호이다.
err_msg	변경할 에러 문장이다.

- 예외 상황

예외 상황	설명
CHUNK_NOT_FOUND	chunk_id에 해당하는 CHUNK를 찾지 못한 경우이다.
INVALID_STATUS	허용되지 않은 STATUS 값을 입력한 경우이다.
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.

예외 상황	설명
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> select
   2     chunk_id,
   3     task_owner,
   4     task_name,
   5     status,
   6     error_code,
   7     error_message
   8 from DBA_PARALLEL_EXECUTE_CHUNKS
   9 where chunk_id = 2600;

  CHUNK_ID
----------
TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS               ERROR_CODE
-------------------- ----------
ERROR_MESSAGE
--------------------------------------------------------------------------------
      2600
TIBERO
SAMPLE_TASK
UNASSIGNED



1 row selected.

SQL> exec DBMS_PARALLEL_EXECUTE.SET_CHUNK_STATUS(
   2      'SAMPLE_TASK',
   3      2600,
   4      DBMS_PARALLEL_EXECUTE.PROCESSED_WITH_ERROR,
   5      -12345,
   6      'Sample Error occured'
   7 );

PSM completed.

SQL> select
   2     chunk_id,
   3     task_owner,
```

```
   4     task_name,
   5     status,
   6     error_code,
   7     error_message
   8 from DBA_PARALLEL_EXECUTE_CHUNKS
   9 where chunk_id = 2600;

  CHUNK_ID
----------
TASK_OWNER
--------------------------------------------------------------------------------
TASK_NAME
--------------------------------------------------------------------------------
STATUS               ERROR_CODE
-------------------- ----------
ERROR_MESSAGE
--------------------------------------------------------------------------------
      2600
TIBERO
SAMPLE_TASK
PROCESSED_WITH_ERROR     -12345
Sample Error occured


1 row selected.
```

26.5.14. STOP_TASK

TASK의 수행을 중지한다.

STOP_TASK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.STOP_TASK
(
    task_name                   IN          VARCHAR2
);
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> exec DBMS_PARALLEL_EXECUTE.STOP_TASK('SAMPLE_TASK');

PSM completed.
```

26.5.15. TASK_STATUS

TASK의 상태 정보를 반환한다.

TASK_STATUS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PARALLEL_EXECUTE.TASK_STATUS
(
    task_name                   IN          VARCHAR2
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
task_name	TASK의 이름이다.

- 예외 상황

예외 상황	설명
IN VALID_STRING_PARAM	입력된 변수가 NULL이 허용되지 않거나, 허용된 길이를 초과한 경우이다.
TASK_NOT_FOUND	지정된 task_name에 맞는 TASK를 찾지 못한 경우이다.

- 예제

```
SQL> set serveroutput on;
SQL> DECLARE
   2     STATUS NUMBER;
   3 BEGIN
   4     STATUS := DBMS_PARALLEL_EXECUTE.TASK_STATUS('SAMPLE_TASK');
   5     DBMS_OUTPUT.PUT_LINE(STATUS);
```

```
   6 END;
   7 /
4

PSM completed.
```

제27장 DBMS_PIPE

본 장에서는 DBMS_PIPE 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

27.1. 개요

DBMS_PIPE은 동일한 인스턴스 내에 속해 있는 세션들 간에 통신할 수 있는 기능을 지원하는 패키지이다. UNIX 계열에서 사용되는 파이프와 유사한 개념이지만 tbPSM RDBMS 내의 자료 구조를 기반으로 운영체제와는 다른 메카니즘으로 구현되어 있다. 동일 인스턴스 내에서의 통신만 가능하므로, TAC 환경에서는 사용할 수 없다.

27.2. 프러시저와 함수

본 절에서는 DBMS_PIPE 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

27.2.1. CREATE_PIPE

명시적으로 공개 또는 사용자 소유의 통신용 파이프를 생성하는 함수이다.

CREATE_PIPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.CREATE_PIPE
(
    pipename        IN  VARCHAR2,
    maxpipesize     IN  PLS_INTEGER DEFAULT 8192,
    private         IN  BOOLEAN     DEFAULT TRUE
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
pipename	생성하는 파이프의 이름으로 인스턴스 내에서 고유한 값이다. SEND_MESSAGE, RECEIVE_MESSAGE를 호출할 때 이 이름을 사용하여 송수신할 수 있다.

파라미터	설명
maxpipesize	메시지들을 파이프에 저장할 수 있는 최대 크기이다. (기본값: 8192Bytes)
private	– TRUE : 사용자 소유의 파이프가 생성된다. (기본값) – FALSE : 모든 사용자가 공유하는 파이프가 생성된다.

- 반환값

반환값	설명
0	파이프 생성이 성공한 경우에 반환한다.

- 예제

```
DECLARE
    status pls_integer;
BEGIN
    status := DBMS_PIPE.CREATE_PIPE('tbpipe', 1000, false);

END;
```

27.2.2. NEXT_ITEM_TYPE

로컬 메시지 버퍼에 저장되어 있는 다음 아이템의 데이터 타입을 알려주는 함수이다. RECEIVE_MESSAGE를 호출하여 로컬 버퍼에 가져온 다음 NEXT_ITEM_TYPE을 호출하고 반환된 타입에 맞는 UNPACK_MESSAGE 프러시저를 호출하여 데이터를 읽어올 수 있다.

NEXT_ITEM_TYPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.NEXT_ITEM_TYPE
RETURN PLS_INTEGER;
```

- 반환값

반환값	설명
0	더 이상 아이템이 없는 경우 반환한다.
9	아이템의 데이터 타입이 VARCHAR2일 경우 반환한다.

- 예제

```
DECLARE
    status  pls_integer;
```

```
    type    pls_integer;
BEGIN
    status := DBMS_PIPE.RECEIVE_MESSAGE('tbpipe');
    type := DBMS_PIPE.NEXT_ITEM_TYPE;
    -- 타입에 맞는 UNPACK_MESSAGE 프러시저 호출
END;
```

27.2.3. PACK_MESSAGE

로컬 메시지 버퍼 안에서 메시지를 만드는 프러시저이다. 이 프러시저를 호출하여 로컬 버퍼에 단위 데이터(아이템)를 저장하고, 1개 이상의 아이템이 저장되면 SEND_MESSAGE 함수를 호출하여 하나의 메시지로 합쳐서 파이프에 보낼 수 있다.

PACK_MESSAGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.PACK_MESSAGE
(
    item  IN  VARCHAR2
);
```

- 파라미터

파라미터	설명
item	로컬 버퍼에 저장할 아이템이다.

- 예제

```
DECLARE
    msg VARCHAR2(2000);
BEGIN
    msg := 'tibero';
    DBMS_PIPE.PACK_MESSAGE(msg);
END;
```

27.2.4. PURGE

파이프에 저장되어 있는 모든 내용을 비우는 프러시저이다.

PURGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.PURGE
(
    pipename     IN  VARCHAR2
);
```

- 파라미터

파라미터	설명
pipename	저장된 모든 메시지를 비울 파이프의 이름이다.

- 예제

```
BEGIN
    DBMS_PIPE.PURGE('tbpipe');
END;
```

27.2.5. RECEIVE_MESSAGE

파이프에 저장되어 있는 메시지 한 개를 로컬 버퍼에 읽어오는 함수이다.

RECEIVE_MESSAGE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.RECEIVE_MESSAGE
(
    pipename        IN      VARCHAR2,
    timeout         IN      INTEGER DEFAULT MAXWAIT
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
pipename	저장된 메시지를 읽어올 파이프의 이름이다.
timeout	메시지를 기다릴 시간을 초 단위로 설정한다. (기본값: MAXWAIT, 86400000(1000일))

- 반환값

반환값	설명
0	파이프에 저장되어 있는 메시지 한 개를 로컬 버퍼에 읽어온 경우에 반환한다.

반환값	설명
1	지정된 타임아웃 시간을 초과한 경우 반환한다.

- 예제

```
DECLARE
    status  pls_integer;
BEGIN
    status := DBMS_PIPE.RECEIVE_MESSAGE('tbpipe');
END:
```

27.2.6. RESET_BUFFER

로컬에 있는 읽기 버퍼 및 쓰기 버퍼의 모든 내용을 비우는 프러시저이다. 새로운 파이프에 대한 읽기 및 쓰기를 위해 현재 로컬에 남아 있는 메시지를 비울 때 사용할 수 있다.

RESET_BUFFER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.RESET_BUFFER;
```

- 예제

```
BEGIN
    DBMS_PIPE.RESET_BUFFER;
END:
```

27.2.7. REMOVE_PIPE

명시적으로 생성된 파이프 및 파이프에 저장되어 있던 모든 내용을 제거하는 함수이다. 단, 이 함수를 호출하지 않고 인스턴스가 종료되어도 파이프는 제거된다.

파이프가 제거되는 경우 0을 반환한다. 이미 제거된 파이프에 대해서 해당 함수를 호출한 경우에도 0을 반환한다.

REMOVE_PIPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.REMOVE_PIPE
(
    pipename    IN      VARCHAR2
```

```
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
pipename	제거하려는 파이프의 이름이다.

- 반환값

반환값	설명
0	생성된 파이프 및 파이프에 저장되어 있던 모든 내용이 제거된 경우 반환한다.

- 예제

```
DECLARE
    status pls_integer;
BEGIN
    status := DBMS_PIPE.REMOVE_PIPE('tbpipe');
END;
```

27.2.8. SEND_MESSAGE

로컬 버퍼에 저장되어 있던 메시지를 파이프에 저장하는 함수이다. PACK_MESSAGE를 한 번 이상 호출하여 누적된 아이템들이 하나의 메시지로 구성되어 전송된다. 존재하지 않는 파이프의 이름이 인자로 들어오는 경우, 묵시적으로 private 파이프를 생성한다.

SEND_MESSAGE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.SEND_MESSAGE
(
    pipename        IN      VARCHAR2,
    timeout         IN      INTEGER DEFAULT MAXWAIT,
    maxpipesize  IN INTEGER DEFAULT 8192
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
pipename	메시지를 송신할 파이프의 이름이다.

파라미터	설명
timeout	메시지를 기다릴 시간을 초 단위로 설정한다. (기본값: MAXWAIT, 86400000(1000일))
maxpipesize	Byte 단위이며, 파이프에 해당 메시지를 보낼 때 사용가능한 최대 크기이다. 최대 사이즈를 초과하는 경우, 메시지를 파이프에 쓰지 못한다. (기본값: 8192Bytes)

- 반환값

반환값	설명
0	로컬 버퍼에 저장되어 있던 메시지가 파이프에 저장된 경우 반환한다.
1	지정된 타임아웃 시간을 초과한 경우 반환한다.

- 예제

```
DECLARE
    msg VARCHAR2(2000);
    status  pls_integer;
BEGIN
    msg := 'tibero1';
    DBMS_PIPE.PACK_MESSAGE(msg);
    msg := 'tibero2';
    DBMS_PIPE.PACK_MESSAGE(msg);
    status := DBMS_PIPE.SEND_MESSAGE('tbpipe');
END;
```

27.2.9. UNIQUE_SESSION_NAME

데이터베이스에 연결되어 있는 모든 세션들 중에 식별할 수 있는 유일한 이름을 반환하는 함수이다. 동일한 세션에서 이 함수를 호출할 때마다 항상 같은 값을 반환하며, 이 함수를 이용해 자신 세션 안에서만 사용할 수 있는 파이프의 이름을 지정할 수 있다.

UNIQUE_SESSION_NAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.UNIQUE_SESSION_NAME
RETURN VARCHAR2;
```

- 반환값

  세션 단위로 유일한 이름이다.

- 예제

```
DECLARE
    status pls_integer;
begin
    status := DBMS_PIPE.CREATE_PIPE(DBMS_PIPE.UNIQUE_SESSION_NAME, 1000, false);
    status := DBMS_PIPE.SEND_MESSAGE(DBMS_PIPE.UNIQUE_SESSION_NAME);
    status := DBMS_PIPE.RECEIVE_MESSAGE(DBMS_PIPE.UNIQUE_SESSION_NAME);
    status := DBMS_PIPE.REMOVE_PIPE(DBMS_PIPE.UNIQUE_SESSION_NAME);
END;
```

27.2.10. UNPACK_MESSAGE

로컬 메시지 버퍼 안에 저장되어 있는 아이템 하나를 읽어오는 프러시저이다. RECEIVE_MESSAGE 함수를 호출하여 파이프로부터 읽어온 메시지를 로컬 읽기 버퍼에 저장하고, UNPACK_MESSAGE 프러시저를 호출하여 메시지 안의 아이템들을 하나씩 가져올 수 있다.

UNPACK_MESSAGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_PIPE.UNPACK_MESSAGE
(
    item    OUT VARCHAR2
);
```

- 파라미터

파라미터	설명
item	로컬 버퍼에서 읽어올 아이템이다.

- 예제

```
DECLARE
    msg VARCHAR2(2000);
BEGIN
    DBMS_PIPE.UNPACK_MESSAGE(msg);
END;
```

제28장 DBMS_RANDOM

본 장에서는 DBMS_RANDOM 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

28.1. 개요

DBMS_RANDOM은 임의의 숫자를 생성하는 기능을 제공한다.

28.2. 프러시저와 함수

본 절에서는 DBMS_RANDOM 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

28.2.1. NORMAL

표준 정규 분포(가우스 분포)에서의 임의값을 추출하는 함수이다.

NORMAL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RANDOM.NORMAL
RETURN NUMBER;
```

- 예제

```
DECLARE
        a NUMBER;
BEGIN
        a := DBMS_RANDOM.NORMAL;
END;
/
```

28.2.2. RANDOM

-(2의 31제곱)보다 크거나 같고, 2의 31제곱보다 작은 임의의 정수를 생성한다.

RANDOM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RANDOM.RANDOM
RETURN BINARY_INTEGER;
```

- 예제

```
DECLARE
        a BINARY_INTEGER;
BEGIN
        a := DBMS_RANDOM.RANDOM;
END;
/
```

28.2.3. SEED

새로운 seed를 지정하여 임의값 생성 시퀀스를 새로 생성한다. 기존과 동일한 seed를 지정하면 동일 시퀀스 내의 첫 번째 값으로 돌아가므로 매 호출 후 임의값을 추출하면 같은 값이 나온다.

SEED 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - seed가 번호인 경우

```
DBMS_RANDOM.SEED
(
    val             IN              BINARY_INTEGER
);
```

  - seed가 문자열인 경우

```
DBMS_RANDOM.SEED
(
    val             IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
val	임의값을 생성할 seed 번호나 문자열이다.

- 예제

```
DECLARE
        a BINARY_INTEGER;
BEGIN
```

```
        DBMS_RANDOM.SEED(3);
        a := DBMS_RANDOM.NORMAL;

        DBMS_RANDOM.SEED('abc');
        a := DBMS_RANDOM.RANDOM;
END;
/
```

28.2.4. STRING

임의의 문자열을 생성한다.

STRING 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RANDOM.STRING
(
    opt             IN              CHAR,
    len             IN              NUMBER
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
opt	생성하고 싶은 문자열의 형태이고, 항목은 아래와 같다. – 'u' 또는 'U' : 대문자 알파벳의 문자열 – 'l' 또는 'L' : 소문자 알파벳의 문자열 – 'a' 또는 'A' : 대소문자 구분 없이 임의의 알파벳의 문자열 – 'x' 또는 'X' : 임의의 대문자 알파벳 또는 숫자의 문자열 – 'p' 또는 'P' : 임의의 출력 가능한 문자들의 배열 그 외의 값일 경우에는 기본적으로 대문자 알파벳의 문자열이다.
len	생성할 문자열의 길이이다.

- 예제

```
DECLARE
        a varchar2(1000);
BEGIN
```

```
        a := DBMS_RANDOM.STRING('x', 20);
        dbms_output.put_line(a);
END;
/
```

28.2.5. VALUE

범위 내에서 임의의 숫자를 생성한다. 범위가 지정되지 않은 경우는 0보다 크거나 같고 1보다 작은 숫자를 생성한다.

VALUE 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 범위가 지정되지 않은 경우

```
DBMS_RANDOM.VALUE
RETURN NUMBER;
```

  - 범위가 지정된 경우

```
DBMS_RANDOM.VALUE
(
    low             IN              NUMBER,
    high            IN              NUMBER
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
low	임의값 범위 내에서의 최솟값이다.
high	임의값 범위 내에서의 최댓값은 이 값보다 크거나 같을 수 없다.

- 예제

```
DECLARE
        a number;
BEGIN
        a := DBMS_RANDOM.VALUE;
        a := DBMS_RANDOM.VALUE(-30, 100);
END;
/
```

제29장 DBMS_REDACT

본 장에서는 DBMS_REDACT 패키지의 기본 개념과 패키지 내의 상수 및 프러시저들의 의미를 설명한다.

29.1. 개요

DBMS_REDACT은 Tibero에서의 Data redaction을 위한 interface를 제공하는 패키지이다. Data redaction은 낮은 권한 레벨을 가진 유저 또는 애플리케이션으로부터 입력된 쿼리의 결과 데이터를 masking(Redaction)할 수 있는 기능이다.

29.2. 상수

DBMS_REDACT에서는 패키지 내의 프러시저의 파라미터로서 사용할 수 있는 상수들을 정의하고 있다.

- DBMS_REDACT.ADD_POLICY의 function_type 파라미터 상수

상수	값	타입	설명
NONE	0	BINARY_INTEGER	Data redaction을 하지 않는다.
FULL	1	BINARY_INTEGER	고정된 값들에 대해 redact한다.
PARTIAL	2	BINARY_INTEGER	Partial redaction으로 컬럼 데이터의 일부분을 redact한다.

- DBMS_REDACT.ALTER_POLICY의 action 파라미터 상수

상수	값	타입	설명
ADD_COLUMN	1	BINARY_INTEGER	Redaction policy에 컬럼을 추가한다.
DROP_COLUMN	2	BINARY_INTEGER	Redaction policy에 컬럼을 제거한다.
MODIFY_EXPRESSION	3	BINARY_INTEGER	Redaction policy의 표현을 수정한다.
MODIFY_COLUMN	4	BINARY_INTEGER	Redaction policy의 컬럼을 수정함으로써 redaction의 function_type 또는 function_parameters를 수정한다.
SET_POLICY_DESCRIPTION	5	BINARY_INTEGER	Redaction policy에 대한 설명을 추가한다.
SET_COLUMN_DESCRIPTION	6	BINARY_INTEGER	Redaction policy의 컬럼에 대한 설명을 추가한다.

29.3. 프러시저

본 절에서는 DBMS_REDACT 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

29.3.1. ADD_POLICY

테이블 또는 뷰에 대한 Data redaction policy를 정의한다.

ADD_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REDACT.ADD_POLICY (
   object_schema                IN    VARCHAR2 := NULL,
   object_name                  IN    VARCHAR2,
   policy_name                  IN    VARCHAR2,
   policy_description           IN    VARCHAR2 := NULL,
   column_name                  IN    VARCHAR2 := NULL,
   column_description           IN    VARCHAR2 := NULL);
   function_type                IN    BINARY_INTEGER := DBMS_REDACT.FULL,
   function_parameters          IN    VARCHAR2 := NULL,
   expression                   IN    VARCHAR2,
   enable                       IN    BOOLEAN := TRUE,
   regexp_pattern               IN    VARCHAR2 := NULL,
   regexp_replace_string        IN    VARCHAR2 := NULL,
   regexp_position              IN    BINARY_INTEGER := 1,
   regexp_occurrence            IN    BINARY_INTEGER := 0,
   regexp_match_parameter       IN    VARCHAR2 := NULL,
)
```

- 파라미터

파라미터	설명
object_schema	테이블을 소유하고 있는 스키마이다. NULL인 경우 현재 유저이다.
object_name	Data redaction policy를 추가할 테이블 또는 뷰의 이름이다.
policy_name	정책의 이름이다.
policy_description	Redaction policy에 대한 설명이다.
column_name	Redaction policy를 적용할 컬럼의 이름이다. 하나보다 많은 컬럼에 대해 적용하는 경우 ALTER_POLICY 프러시저를 이용하도록 한다.
column_description	Redaction이 적용된 컬럼에 대한 설명이다.
function_type	사용할 redaction 함수의 정류로 가능한 값들은 다음과 같다. – DBMS_REDACT.NONE

파라미터	설명
	– DBMS_REDACT.FULL (기본값) – DBMS_REDACT.PARTIAL 만약 function_type이 DBMS_REDACT.REGEXP인 경우 function_parameters를 생략해야 한다. Data redaction policy를 정의하기 위해 regexp_* parameter를 사용해야 한다.
function_parameters	Redaction 함수를 위한 파라미터로 function_type이 어떤 값인지에 따라 가능한 값이 바뀐다. 만약 function_type이 DBMS_REDACT.REGEXP인 경우 function_parameters를 생략해야 한다. Data redaction policy를 정의하기 위해 regexp_* parameter를 사용해야 한다. – DBMS_REDACT.NONE : 생략 가능하며 NULL이 기본값이다. – DBMS_REDACT.FULL : 생략 가능하며 NULL이 기본값이다. – DBMS_REDACT.PARTIAL : Masking parameter를 값으로 넣어준다. 다음은 각 데이터 타입별 Masking parameter에 대한 설명이다. – Character Character 데이터 타입에 대한 Masking parameter는 다음 5가지 요소들로 구성되어 있고 이 요소들이 콤마(,)로 분리된 리스트 형태이다. • Input 형식 : Masking이 될 수도 있으면 'V', 형식 문자에 대해서는 무시하므로 'F'로 설정한다. • Output 형식 : Masking이 될 수도 있으면 'V', 형식 문자는 그대로 유지한다. • Mask character : 실제 값을 대신하게 될 글자이다. '*' 또는 'x'로 설정한다. • 시작 숫자 위치 : String 상에서 masking을 시작할 위치를 설정한다. String의 시작점은 1이다. 위치를 계산할 때는 형식 문자들은 고려되지 않는다. • 끝 숫자 위치 : String 상에서 masking이 끝나는 위치를 설정한다. 예를 들어 'VVVFVVFVVVV,VVV-VV-VVVV,X,1,5'가 masking paramter로 들어오면 123-45-6789는 XXX-XX-6789가 된다. – Number Number 데이터 타입에 대해서는 다음 3가지 요소들로 구성되어 있고 이 요소들이 콤마(,)로 분리된 리스트 형태로 masking parameter를 만든다.

파라미터	설명
	• Mask character : '0'과 '9' 사이의 글자로 실제 값을 대신하게 된다. • 시작 숫자 위치 : String 상에서 masking을 시작할 위치를 설정한다. String의 시작점은 1이다. 위치를 계산할 때는 소수점은 고려되지 않는다. • 끝 숫자 위치 : String 상에서 masking이 끝나는 위치를 설정한다. 예를 들어 masking parameter가 '9,1,5'인 경우 123456789는 999996789로 된다. – Datetime Datetime 데이터 타입의 경우 다음 6가지 요소를 구분자 없이 순차적으로 붙여 masking parameter를 만든다. • 월 - 'M' : 월에 대해 masking하지 않음 - 'm#' : 가능하면 명시된 숫자로 월을 mask, #은 1부터 12 사이 숫자 • 일 - 'D' : 일에 대해 masking하지 않음 - 'd#' : 가능하면 명시된 숫자로 일을 mask, #은 1부터 31 사이 숫자 • 년도 - 'Y' : 년도에 대해 masking하지 않음 - 'y#' : 가능하면 명시된 숫자로 년도을 mask, #은 1부터 9999 사이 숫자 • 시 - 'H' : 시에 대해 masking하지 않음 - 'h#' : 가능하면 명시된 숫자로 시을 mask, #은 0부터 23 사이 숫자 • 분 - 'M' : 분에 대해 masking하지 않음 - 'm#' : 가능하면 명시된 숫자로 분을 mask, #은 0부터 59 사이 숫자 • 초 - 'S' : 초에 대해 masking하지 않음 - 's#' : 가능하면 명시된 숫자로 초을 mask, #은 0부터 59 사이 숫자

파라미터	설명
	예를 들어 masking parameter가 'm12DYHMS'인 경우 01-May-01 01:01:01가 01-Dec-01 01:01:01이 된다.
expression	테이블 또는 뷰에 대한 default boolean expression이다. 이 파라미터로 들어온 expression의 값이 TRUE일 때만 redaction을 수행한다. 다음 함수들을 지원한다. – SYS_CONTEXT – SUBSTR 관련 함수들 – LENGTH 관련 함수들
enable	Data redaction policy의 자동 적용 여부를 결정하는 boolean 값이다. TRUE인 경우 자동으로 정책을 적용한다.
regexp_*	function_type이 DBMS_REDACT.REGEXP일 때 사용되는 파라미터들이지만 현재 function_type으로 DBMS_REDACT.REGEXP가 지원되지 않는 상태이다.

- 예제

  – function_type이 DBMS_REDACT.PARTIAL인 경우

```
BEGIN
  DBMS_REDACT.ADD_POLICY(
    object_schema           =>  'TIBERO',
    object_name             =>  'T155289',
    policy_name             =>  'T155289_POLICY',
    column_name             =>  'NUMBER_COL1',
    expression              =>  '1=1',
    function_type           =>  DBMS_REDACT.PARTIAL,
    function_parameters     =>  '7,1,5',
);
END;
```

  – function_type이 DBMS_REDACT.FULL인 경우

```
BEGIN
  DBMS_REDACT.ADD_POLICY(
    object_schema   =>  'TIBERO',
    object_name     =>  'T155289',
    policy_name     =>  'T155289_POLICY',
    column_name     =>  'NUMBER_COL1',
    expression     =>  'SYS_CONTEXT(''USERENV'',''SESSION_USER'') = ''TIBERO''',
```

```
    function_type   =>  DBMS_REDACT.FULL
);
END;
```

29.3.2. ALTER_POLICY

테이블 또는 뷰에 대한 Data redaction policy를 바꾼다.

다음과 같은 방식을 통해 정책을 바꾼다.

- 정책의 expression을 변경
- 특정 컬럼의 redaction의 종류를 변경
- 특정 컬럼에 대한 redaction 함수의 파라미터를 변경
- Redaction policy에 컬럼을 추가
- Redaction policy에서 컬럼을 제거

ALTER_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REDACT.ALTER_POLICY (
   object_schema                IN    VARCHAR2 := NULL,
   object_name                  IN    VARCHAR2,
   policy_name                  IN    VARCHAR2,
   action                       IN    BINARY_INTEGER := DBMS_REDACT.ADD_COLUMN,
   column_name                  IN    VARCHAR2 := NULL,
   function_type                IN    BINARY_INTEGER := DBMS_REDACT.FULL,
   function_parameters          IN    VARCHAR2 := NULL,
   expression                   IN    VARCHAR2,
   regexp_pattern               IN    VARCHAR2 := NULL,
   regexp_replace_string        IN    VARCHAR2 := NULL,
   regexp_position              IN    BINARY_INTEGER := 1,
   regexp_occurrence            IN    BINARY_INTEGER := 0,
   regexp_match_parameter       IN    VARCHAR2 := NULL,
   policy_description           IN    VARCHAR2 := NULL,
   column_description           IN    VARCHAR2 := NULL
);
```

- 파라미터

파라미터	설명
object_schema	테이블을 소유하고 있는 스키마이다. NULL인 경우 현재 유저이다.
object_name	Data redaction policy를 추가할 테이블 또는 뷰의 이름이다.
policy_name	정책의 이름이다.
action	자세한 내용은 “29.2. 상수”를 참고한다.
column_name	Redaction policy를 적용할 컬럼의 이름이다. 하나보다 많은 컬럼에 대해 적용하는 경우 ALTER_POLICY 프러시저를 이용하도록 한다.
function_type	사용할 redaction 함수의 정류로 가능한 값들은 다음과 같다. – DBMS_REDACT.NONE – DBMS_REDACT.FULL (기본값) – DBMS_REDACT.PARTIAL 만약 function_type이 DBMS_REDACT.REGEXP인 경우 function_parameters를 생략해야 한다. Data redaction policy를 정의하기 위해 regexp_* parameter를 사용해야 한다.
function_parameters	Redaction 함수를 위한 파라미터로 function_type이 어떤 값인지에 따라 가능한 값이 바뀐다. 만약 function_type이 DBMS_REDACT.REGEXP인 경우 function_parameters를 생략해야 한다. Data redaction policy를 정의하기 위해 regexp_* parameter를 사용해야 한다. – DBMS_REDACT.NONE : 생략 가능하며 NULL이 기본값이다. – DBMS_REDACT.FULL : 생략 가능하며 NULL이 기본값이다. – DBMS_REDACT.PARTIAL : masking parameter를 값으로 넣어준다. 각 데이터 타입에 대한 masking parameter에 대한 자세한 내용은 “29.3.1. ADD_POLICY”의 "function_parameters" 항목을 참고한다.
expression	테이블 또는 뷰에 대한 default boolean expression. 이 파라미터로 들어온 expression의 값이 TRUE일 때만 redaction을 수행한다. 다음 함수들을 지원한다. – SYS_CONTEXT – SUBSTR 관련 함수들 – LENGTH 관련 함수들

파라미터	설명
enable	Data redaction policy의 자동 적용 여부를 결정하는 boolean 값이다. TRUE인 경우 자동으로 정책을 적용한다.
regexp_*	function_type이 DBMS_REDACT.REGEXP일 때 사용되는 파라미터들이지만 현재 function_type으로 DBMS_REDACT.REGEXP가 지원되지 않는 상태이다.
policy_description	Redaction policy에 대한 설명이다.
column_description	Redaction이 적용된 컬럼에 대한 설명이다.

- 예제

```
BEGIN
   DBMS_REDACT.ALTER_POLICY(
     object_schema          => 'TIBERO',
     object_name            => 'T155289',
     policy_name            => 'T155289_POLICY',
     action                 => DBMS_REDACT.DROP_COLUMN,
     column_name            => 'NUMBER_COL1');
END;
```

29.3.3. DISABLE_POLICY

Data redaction policy가 적용되미 않게 한다.

DISABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REDACT.DISABLE_POLICY (
   object_schema                IN    VARCHAR2 := NULL,
   object_name                  IN    VARCHAR2,
   policy_name                  IN    VARCHAR2);
```

- 파라미터

파라미터	설명
object_schema	테이블 또는 뷰를 소유하고 있는 스키마이다. NULL인 경우 현재 유저가 된다.
object_name	Data redaction policy를 제거할 테이블 또는 뷰의 이름이다.
policy_name	제거될 정책의 이름이다.

- 예제

```
BEGIN
   DBMS_REDACT.DISABLE_POLICY (
     object_schema          => 'TIBERO',
     object_name            => 'T155289',
     policy_name            => 'T155289_POLICY');
END;
```

29.3.4. DROP_POLICY

테이블 또는 뷰로부터 masking policy를 제거한다.

DROP_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REDACT.DROP_POLICY (
   object_schema                IN    VARCHAR2 := NULL,
   object_name                  IN    VARCHAR2,
   policy_name                  IN    VARCHAR2
);
```

- 파라미터

파라미터	설명
object_schema	테이블 또는 뷰를 소유하고 있는 스키마이다. NULL인 경우 현재 유저가 된다.
object_name	Data redaction policy를 제거할 테이블 또는 뷰의 이름이다.
policy_name	제거될 정책의 이름이다.

- 예제

```
BEGIN
   DBMS_REDACT.DROP_POLICY (
     object_schema          => 'TIBERO',
     object_name            => 'T155289',
     policy_name            => 'T155289_POLICY');
END;
```

29.3.5. ENABLE_POLICY

Data redaction policy가 다시 적용되게 한다.

ENABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REDACT.ENABLE_POLICY (
   object_schema                IN    VARCHAR2 := NULL,
   object_name                  IN    VARCHAR2,
   policy_name                  IN    VARCHAR2
);
```

- 파라미터

파라미터	설명
object_schema	테이블 또는 뷰를 소유하고 있는 스키마이다. NULL인 경우 현재 유저가 된다.
object_name	Data redaction policy를 제거할 테이블 또는 뷰의 이름이다.
policy_name	제거될 정책의 이름이다.

- 예제

```
BEGIN
   DBMS_REDACT.ENABLE_POLICY (
     object_schema          => 'TIBERO',
     object_name            => 'T155289',
     policy_name            => 'T155289_POLICY');
END;
```

29.3.6. UPDATE_FULL_REDACTION_VALUES

Full redactation에 대한 Data redaction policy에 대해 기본적으로 보여지는 값들을 변경한다.

UPDATE_FULL_REDACTION_VALUES 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES (
   number_val       IN NUMBER                    :=  NULL,
   binfloat_val     IN BINARY_FLOAT              :=  NULL,
   bindouble_val    IN BINARY_DOUBLE             :=  NULL,
   char_val         IN CHAR                      :=  NULL,
```

```
   varchar_val      IN VARCHAR2                  :=  NULL,
   nchar_val        IN NCHAR                     :=  NULL,
   nvarchar_val     IN NVARCHAR2                 :=  NULL,
   date_val         IN DATE                      :=  NULL,
   ts_val           IN TIMESTAMP                 :=  NULL,
   tswtz_val        IN TIMESTAMP WITH TIME ZONE  :=  NULL,
   blob_val         IN BLOB                      :=  NULL,
   clob_val         IN CLOB                      :=  NULL,
   nclob_val        IN NCLOB                     NULL
);
```

- 파라미터

파라미터	설명
number_val	NUMBER 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
binfloat_val	BINARY_FLOAT 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
bindouble_val	BINARY_DOUBLE 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
char_val	CHAR 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
varchar_val	VARCHAR2 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
nchar_val	NCHAR 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
nvarchar_val	NVARCHAR2 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
date_val	DATE 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
ts_val	TIMESTAMP 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
tswtz_val	TIMESTAMP WITH TIME ZONE 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
blob_val	BLOB 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
clob_val	CLOB 데이터 타입에 대한 컬럼들의 기본값을 변경한다.
nclob_val	NCLOB 데이터 타입에 대한 컬럼들의 기본값을 변경한다.

제30장 DBMS_REDEFINITION

본 장에서는 DBMS_REDEFINITION 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

30.1. 개요

DBMS_REDEFINITION은 온라인 상태에서 특정 테이블을 재정의하기 위한 프러시저를 제공하는 패키지이다.

재정의할 대상인 원본 테이블을 지정된 컬럼 대응 정보에 따라 새 테이블로 재정의할 수 있다. 새 테이블은 재정의 수행 이전에 미리 생성되어 있어야 하며, 재정의가 끝나면 원본 테이블의 구조와 새 테이블의 구조가 서로 바뀐다.

30.2. 제약사항

현재 재정의를 지원하지 않는 원본 테이블 스펙은 다음과 같다.

- options_flag == 1일 때(PK 옵션), PK가 존재하지 않는 경우
- 실체화 뷰가 정의되어 있는 경우
- 전역 임시 테이블인 경우
- SYS 계정 소유의 테이블인 경우
- Virtual 테이블인 경우
- 외부 테이블인 경우
- Read-only인 경우
- XML index 내부 테이블인 경우
- CTXCAT index 내부 테이블인 경우
- IOT 테이블이고 options_flag == 2 (ROWID 옵션)인 경우
- VPD가 걸려있는 경우
- 이미 재정의 중인 테이블인 경우
- 특정 파티션에 대해 재정의할 때 lob 컬럼이 있는 경우

30.3. 프러시저

본 절에서는 DBMS_REDEFINITION 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

30.3.1. ABORT_REDEF_TABLE

온라인 테이블 재정의를 취소하는 프러시저이다. 이 프러시저에 원본 테이블과 새 테이블의 이름을 지정하면, 온라인 테이블을 재정의하기 위해 생성한 내부 객체는 제거되고 재정의가 취소된다. 이때 원본 테이블과 새 테이블은 취소 시점의 상태로 남는다.

ABORT_REDEF_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE ABORT_REDEF_TABLE
(
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    part_name IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
part_name	재정의 중인 파티션의 이름이다. 한 테이블의 단 하나의 파티션이 재정의 중일 경우에는 파티션의 이름을 해당 파라미터로 명시해야 한다. NULL은 전체 테이블이 재정의 중이라는 것을 의미한다. 테이블의 여러 파티션이 재정의 중일 경우에는 각 파티션 이름을 콤마로 구분해서 명시한다.

- 예제

```
BEGIN
    DBMS_REDEFINITION.ABORT_REDEF_TABLE('TIBERO', 'ORIG_TABLE', 'INT_TABLE');
END; /
```

30.3.2. CAN_REDEF_TABLE

온라인 테이블 재정의가 가능한지 확인하는 프러시저이다. 이 테이블에 테이블 이름과 옵션을 주면 온라인 테이블 재정의가 가능한지 확인한 후, 가능하지 않은 경우 에러를 발생시킨다.

CAN_REDEF_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CAN_REDEF_TABLE
(
    uname IN VARCHAR2,
    tname IN VARCHAR12,
    options_flag IN BINARY_INTEGER := 1,
    part_name IN VARCHAR2 := NULL
);
```

- 파라미터

파라미터	설명
uname	테이블이 속한 스키마의 이름이다.
tname	온라인 테이블 재정의가 가능한지 확인할 테이블의 이름이다.
options_flag	원본 테이블에서 새 테이블로 데이터를 옮길 때 어떤 정보를 사용할 것인지를 지정한다. – cons_use_pk : 기본 키의 정보를 사용한다. (지정하지 않을 경우 기본값) – cons_use_rowid : ROWID를 사용한다.
part_name	재정의할 파티션의 이름이다. 한 테이블의 단 하나의 파티션을 재정의하는 경우에는 파티션의 이름을 해당 파라미터로 명시해야 한다. NULL은 전체 테이블이 재정의된다는 것을 의미한다. 테이블의 여러 파티션을 재정의하는 경우에는 각 파티션 이름을 콤마로 구분해서 명시한다.

- 예제

```
CREATE TABLE T1 (A NUMBER PRIMARY KEY);
BEGIN
    DBMS_REDEFINITION.CAN_REDEF_TABLE('TIBERO', 'T1');
END; /
```

30.3.3. COPY_TABLE_DEPENDENTS

테이블을 온라인 재정의를 수행하면서, 테이블과 관련된 다른 객체들도 복사해주는 프러시저이다. 인덱스, 제약사항, 트리거, 권한 등에 대해서 기존 테이블의 관련 객체를 새 테이블에 복사해 줄지 파라미터로 명시할 수 있다.

COPY_TABLE_DEPENDENTS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE COPY_TABLE_DEPENDENTS
(
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    copy_indexes IN PLS_INTEGER := 1,
    copy_triggers IN BOOLEAN := TRUE,
    copy_constraints IN BOOLEAN := TRUE,
    copy_privileges IN BOOLEAN := TRUE,
    ignore_errors IN BOOLEAN := FALSE,
    num_errors OUT PLS_INTEGER,
    copy_statistics IN BOOLEAN := FALSE,
    copy_mvlog IN BOOLEAN := FALSE
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
copy_indexes	인덱스들도 복사해 줄 것인지 명시한다. – 0 : 인덱스를 복사하지 않는다. – dbms_redefinition.cons_orig_params : 인덱스를 복사한다. 단, domain 인덱스, global partitioned 인덱스, function-based 인덱스가 있을 경우, 수행이 불가능하다.
copy_triggers	– TRUE : 트리거를 복사한다. – FALSE : 트리거를 복사하지 않는다.
copy_constraints	– TRUE : 제약 사항들을 복사한다. – FALSE : 제약 사항들을 복사하지 않는다.

파라미터	설명
copy_privileges	– TRUE : 테이블의 권한을 복사한다. – FALSE : 테이블의 권한을 복사하지 않는다.
ignore_errors	– TRUE : 관련있는 객체를 복사하며 발생하는 에러는 무시하고 넘어간다. – FALSE : 관련있는 객체를 복사하며 에러가 발생하면 멈춘다.
num_errors	에러가 발생한 횟수를 담아 준다.
copy_statistics	테이블의 통계 정보를 복사한다. 미지원 기능이다.
copy_mvlog	테이블의 Materialized View LOG들을 복사한다. 미지원 기능이다.

- 예제

```
CREATE TABLE T1 (A NUMBER PRIMARY KEY, B NUMBER);
CREATE INDEX I1 ON T1(B);
CREATE TABLE INT_T1 (A NUMBER, B NUMBER);

BEGIN
   DBMS_REDEFINITION.START_REDEF_TABLE('TIBERO', 'T1', 'INT_T1',
      'A A, B B');
END;
/

DECLARE num_errors pls_integer;
BEGIN
   DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('TIBERO', 'T1', 'INT_T1',
       DBMS_REDEFINITION.CONS_ORIG_PARAMS,
       TRUE, TRUE, TRUE, TRUE, num_errors, FALSE);
END;
/
```

30.3.4. FINISH_REDEF_TABLE

온라인 테이블 재정의를 종료하는 프러시저이다. 이 프러시저에 원본 테이블과 새 테이블의 이름을 지정하면, 재정의 중에 변경된 데이터를 서로 교체한 후 원본 테이블과 새 테이블의 이름을 동기화한다. 이로써 재정의가 종료된다. 단, 시스템이 객체를 바꾸는 등의 작업을 하기 때문에 해당 테이블에서 수행되는 트랜잭션이 없는 경우에만 프러시저가 종료 작업을 수행한다.

FINISH_REDEF_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FINISH_REDEF_TABLE
(
```

```
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    part_name IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
part_name	재정의 중인 파티션의 이름이다. 한 테이블의 단 하나의 파티션이 재정의 중일 경우에는 파티션의 이름을 해당 파라미터로 명시해야 한다. NULL은 전체 테이블이 재정의 중이라는 것을 의미한다. 테이블의 여러 파티션이 재정의 중일 경우에는 각 파티션 이름을 콤마로 구분해서 명시한다.

- 예제

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE('TIBERO',
 'ORIG_TABLE', 'INT_TABLE');
END; /
```

30.3.5. REGISTER_DEPENDENT_OBJECT

기존 테이블의 관련 객체와 새 테이블의 관련 객체를 매핑시킨다. 온라인 테이블 재정의가 끝나면 매핑시킨 객체들의 이름이 서로 바뀌게 된다. 단, 관련 객체가 제약조건이고 제약조건의 이름과 테이블 이름이 같을 경우 수행이 불가능하다.

REGISTER_DEPENDENT_OBJECT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REGISTER_DEPENDENT_OBJECT
(
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    dep_type IN PLS_INTEGER DEFAULT 0,
```

```
    dep_owner IN VARCHAR2,
    dep_orig_name IN VARCHAR2,
    dep_int_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
dep_type	관련 객체의 타입이다. 현재 2(인덱스), 3(제약조건), 4(트리거) 외에는 미지원 기능이다.
dep_owner	관련 객체를 소유한 사용자 이름이다.
dep_orig_name	원본 테이블의 관련 객체 이름이다.
dep_int_name	새 테이블의 관련 객체 이름이다.

- 예제

```
CREATE TABLE T1 (A NUMBER PRIMARY KEY, B NUMBER);
CREATE INDEX I1 ON T1(B);
CREATE TABLE INT_T1 (A NUMBER, B NUMBER);
CREATE INDEX INT_I1 ON INT_T1(B);

BEGIN
   DBMS_REDEFINITION.START_REDEF_TABLE('TIBERO', 'T1', 'INT_T1',
     'A A, B B');
END;
/

BEGIN
   DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT('TIBERO', 'T1', 'INT_T1',
           DBMS_REDEFINITION.CONS_INDEX, 'TIBERO', 'I1', 'INT_I1');
END;
/
```

30.3.6. START_REDEF_TABLE

온라인 테이블 재정의를 시작하는 프러시저이다. 이 프러시저에 원본 테이블과 새 테이블의 이름 그리고 컬럼 대응 정보를 지정하면, 원본 테이블의 데이터를 새 테이블에 옮기기 시작한다.

여기서 새 테이블은 이 프러시저를 실행하기 전에 미리 생성해야 하며, 원본 테이블과 같은 스키마에 존재해야 한다. 단, 파티셔닝, 인덱스, 제약조건, 트리거와 같은 테이블의 구조와 관련된 객체는 원본 테이블과 같지 않아도 된다.

참고

컬럼 대응 정보는 원본 테이블의 어떤 컬럼을 새 테이블의 어떤 컬럼에 저장할지를 지정하는 것을 말한다.

START_REDEF_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE START_REDEF_TABLE
(
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    col_mapping IN VARCHAR2 DEFAULT NULL,
    options_flag IN BINARY_INTEGER DEFAULT cons_use_pk,
    part_name IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
col_mapping	원본 테이블과 새 테이블의 컬럼 대응 정보이다. 현재 미지원 기능이다.
options_flag	원본 테이블에서 새 테이블로 데이터를 옮길 때 어떤 정보를 사용할 것인지를 지정한다. – cons_use_pk : 기본 키의 정보를 사용한다. (지정하지 않을 경우 기본값) – cons_use_rowid : ROWID를 사용한다.
part_name	재정의할 파티션의 이름이다. 한 테이블의 단 하나의 파티션을 재정의하는 경우에는 파티션의 이름을 해당 파라미터로 명시해야 한다. NULL은 전체 테이블이 재정의된다는 것을 의미한다. 테이블의 여러 파티션을 재정의하는 경우에는 각 파티션 이름을 콤마로 구분해서 명시한다.

- 예제

```
create table ORIG_TABLE (
   PRODUCT_ID NUMBER primary key,
   PRODUCT_NAME VARCHAR2(20),
   PRICE NUMBER,
   SOLD_DATE DATE );

insert into ORIG_TABLE values (1, 'Tibero', 10000, SYSDATE);

commit;
create table INT_TABLE (
   ID NUMBER primary key,
   NAME VARCHAR2(30),
   PRICE NUMBER(8,2),
   SOLD_DATE DATE )
partition by range (SOLD_DATE)
(
   partition P_2008 values less than ('2009-01-01'),
   partition P_2009 values less than (maxvalue)
);

BEGIN
   DBMS_REDEFINITION.START_REDEF_TABLE('TIBERO', 'ORIG_TABLE',
   'INT_TABLE', 'PRODUCT_ID ID, PRODUCT_NAME NAME, PRICE PRICE,
    SOLD_DATE SOLD_DATE');
END;
/
```

30.3.7. SYNC_INTERIM_TABLE

원본 테이블과 새 테이블의 데이터를 동기화하는 프러시저이다. 물론 FINISH_REDEF_TABLE 프러시저를 실행해도 동기화가 되지만, 온라인 테이블을 재정의하는 중에 해당 테이블에 DML 문장이 많이 수행되는 경우라면 변경된 데이터를 한꺼번에 반영해야 하므로 처리 속도가 늦어질 수 있다. 또한, 재정의를 종료하는 동안에도 해당 테이블에 트랜잭션이 없어야 하기 때문에 DML 문장의 수행이 제한된다.

SYNC_INTERIM_TABLE 프러시저는 이러한 경우와는 다르게 재정의 중에도 동기화를 수행할 수 있으며, 재정의를 종료하는 시간을 줄일 수 있다.

SYNC_INTERIM_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SYNC_INTERIM_TABLE
(
```

```
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    part_name IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
part_name	재정의 중인 파티션의 이름이다. 한 테이블의 단 하나의 파티션이 재정의 중일 경우에는 파티션의 이름을 해당 파라미터로 명시해야 한다. NULL은 전체 테이블이 재정의 중이라는 것을 의미한다. 테이블의 여러 파티션이 재정의 중일 경우에는 각 파티션 이름을 콤마로 구분해서 명시한다.

- 예제

```
BEGIN
    DBMS_REDEFINITION.SYNC_INTERIM_TABLE('TIBERO', 'ORIG_TABLE', 'INT_TABLE');
END; /
```

30.3.8. UNREGISTER_DEPENDENT_OBJECT

원본 테이블과 새 테이블 사이에 매핑되어 있는 관련 객체들의 정보를 해제한다.

UNREGISTER_DEPENDENT_OBJECT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE UNREGISTER_DEPENDENT_OBJECT
(
    uname IN VARCHAR2,
    orig_table IN VARCHAR2,
    int_table IN VARCHAR2,
    dep_type IN PLS_INTEGER DEFAULT 0,
    dep_owner IN VARCHAR2,
    dep_orig_name IN VARCHAR2,
    dep_int_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
uname	원본 테이블과 새 테이블이 속한 스키마의 이름이다.
orig_table	원본 테이블의 이름이다.
int_table	새 테이블의 이름이다.
dep_type	관련 객체의 타입이다. 현재 2(인덱스), 3(제약조건), 4(트리거) 외에는 미지원 기능이다.
dep_owner	관련 객체를 소유한 사용자 이름이다.
dep_orig_name	원본 테이블의 관련 객체 이름이다.
dep_int_name	새 테이블의 관련 객체 이름이다.

- 예제

```
CREATE TABLE T1 (A NUMBER PRIMARY KEY, B NUMBER);
CREATE INDEX I1 ON T1(B);
CREATE TABLE INT_T1 (A NUMBER, B NUMBER);
CREATE INDEX INT_I1 ON INT_T1(B);

BEGIN
   DBMS_REDEFINITION.START_REDEF_TABLE('TIBERO', 'T1', 'INT_T1',
     'A A, B B');
END;
/

BEGIN
   DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT('TIBERO', 'T1', 'INT_T1',
      DBMS_REDEFINITION.CONS_INDEX,
      'TIBERO', 'I1', 'INT_I1');
END;
/

BEGIN
   DBMS_REDEFINITION.UNREGISTER_DEPENDENT_OBJECT('TIBERO', 'T1', 'INT_T1',
   DBMS_REDEFINITION.CONS_INDEX,
     'TIBERO', 'I1', 'INT_I1');
END;
/
```

제31장 DBMS_REPAIR

본 장에서는 DBMS_REPAIR 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

31.1. 개요

DBMS_REPAIR는 테이블과 인덱스 내의 깨진 블록을 검사하고 복구하는 기능을 제공하는 패키지이다.

다음은 DBMS_REPAIR 패키지 내에 정의된 상수이다.

- ALL_INDEX_ID

  인덱스의 스키마 객체 ID를 지정할 때 사용하며, 모든 인덱스에 대해 클린업을 수행하도록 지정한다.

```
ALL_INDEX_ID CONSTANT BINARY_INTEGER := 0
```

- LOCK_NOWAIT

  관련된 테이블, 파티션 또는 서브파티션에 대한 DML LOCK을 NOWAIT 모드로 요청한다.

```
LOCK_NOWAIT CONSTANT BOOLEAN := true
```

- LOCK_WAIT

  관련된 테이블, 파티션 또는 서브파티션에 대한 DML LOCK을 WAIT 모드로 요청한다.

```
LOCK_NOWAIT CONSTANT BOOLEAN := false
```

31.2. 함수

본 절에서는 DBMS_REPAIR 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

31.2.1. ONLINE_INDEX_CLEAN

실패하거나 중단된 인덱스 online-build 또는 online-rebuild에 대한 클린업을 수행한다.

ONLINE_INDEX_CLEAN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_REPAIR.ONLINE_INDEX_CLEAN
(
```

```
    object_id       IN  BINARY_INTEGER  DEFAULT ALL_INDEX_ID,
    wait_for_lock   IN  BOOLEAN         DEFAULT LOCK_WAIT
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
object_id	클린업을 수행할 인덱스의 스키마 객체 ID이다. ALL_INDEX_ID 상수로 지정할 경우 모든 인덱스에 대해 클린업을 수행한다.
wait_for_lock	인덱스와 관련된 테이블, 파티션 또는 서브파티션에 대한 DML LOCK을 요청할 때의 LOCK 모드이다. – LOCK_NOWAIT 상수로 설정할 경우 LOCK을 얻는 데 실패하면 클린업이 바로 중단된다. – LOCK_WAIT 상수로 설정할 경우 LOCK을 얻을 때까지 계속 기다린다.

- 반환값

반환값	설명
TRUE	지정된 모든 인덱스에 대해 클린업이 수행된 경우에 반환된다.
FALSE	지정된 인덱스 중 하나 이상의 인덱스에 대한 클린업이 실패한 경우에 반환된다.

- 예제

```
DECLARE
    cleaned BOOLEAN;
BEGIN
    cleaned := false;
    WHILE cleaned = false
    LOOP
        cleaned := DBMS_REPAIR.ONLINE_INDEX_CLEAN(DBMS_REPAIR.ALL_INDEX_ID,
                                                  DBMS_REPAIR.LOCK_WAIT);
        DBMS_LOCK.SLEEP(10);
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
    RAISE;
END;
/

PSM completed
SQL>
```

제32장 DBMS_RESOURCE_MANAGER

본 장에서는 DBMS_RESOURCE_MANAGER 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

32.1. 개요

Tibero에서는 DBMS_RESOURCE_MANAGER를 이용해서 업무의 성격에 따라 그룹을 나누어 자원을 분배할 수 있다. 예를 들어 자원이 제한된 상황에서 OLTP 업무와 batch 업무를 동시에 진행하고 싶을 때, batch 업무에 쓰이는 자원을 OLTP 업무로 몰아주어서 OLTP 업무의 효율성을 높일 수 있다.

DBMS_RESOURCE_MANAGER은 CPU 자원과 같이 업무를 수행하는 데에 필요한 자원을 업무 성격에 따라 원하는대로 효율적으로 재분배하는 기능을 제공하는 패키지이다. DBMS_RESOURCE_MANAGER 패키지 내의 프러시저를 이용하여, RESOURCE PLAN을 데이터베이스에 추가하고 자원 재분배 방식을 원하는대로 설정해서 적용할 수 있다.

다음은 DBMS_RESOURCE_MANAGER 패키지의 특징이다.

- USE_RESOURCE_MANAGER 파라미터가 Y로 설정되어 있어야 패키지가 제대로 동작한다.
- DBMS_SCHEDULER 패키지의 사용할 때 DBA 권한이 필요하며, 추가된 RSRC PLAN은 오직 PLAN의 소유자만 실행하거나 변경할 수 있다.
- 자원 재분배 방식을 결정하는 CONSUMER GROUP, CONSUMER GROUP MAPPING, RSRC PLAN DIRECTIVE를 추가할 수 있다.
- 현재 데이터베이스에 추가된 RSRC PLAN, CONSUMER GROUP, RSRC PLAN DIRECTIVE 등의 정보는 다음 뷰를 통해 확인이 가능하다.
- RSRC PLAN 변경 히스토리가 PLAN 이름 및 TIME 정보와 함께 기록되어 AUDIT 뷰를 통해 확인이 가능하다.

```
DBA_RSRC_PLANS
DBA_RSRC_CONSUMER_GROUPS
DBA_RSRC_PLAN_DIRECTIVES
DBA_RSRC_AUDIT
```

- 적용되어 있는 ACTIVE PLAN 관련 RSRC PLAN, CONSUMER GROUP, SESSION 등의 정보는 다음 뷰를 통해 확인이 가능하다.

```
V$SESSION
V$RSRC_PLAN
```

```
V$RSRC_CONSUMER_GROUPS
V$RSRC_SESSION_INFO
```

- 여러 개의 RSRC PLAN, CONSUMER GROUP, RSRC PLAN DIRECTIVE를 추가할 수 있으며 그 중 하나의 PLAN을 적용할 수 있다.
- 아무런 RSRC PLAN이 적용되지 않았을 때는 하나의 DEFAULT CONSUMER GROUP을 가진 DEFAULT RESOURCE PLAN이 적용되어 있다.
- PLAN 적용 시 어떤 CONSUMER GROUP에도 속하지 않는 세션들은 전부 DEFAULT CONSUMER GROUP에 속하게 된다.
- DEFAULT CONSUMER GROUP의 최소 자원 보장을 위해 모든 CONSUMER GROUP에 할당된 CPU의 합이 100 미만이어야 한다.

32.2. 프러시저

본 절에서는 DBMS_RESOURCE_MANAGER 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

32.2.1. CREATE_CONSUMER_GROUP

새로운 CONSUMER GROUP을 생성하는 프러시저이다.

CREATE_CONSUMER_GROUP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP
(
    consumer_group      IN VARCHAR2,
    category            IN VARCHAR2,
    comments            IN VARCHAR2
);
```

- 파라미터

파라미터	설명
consumer_group	생성할 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
category	CONSUMER GROUP이 속하는 CATEGORY를 지정하기 위한 파라미터이다.
comments	CONSUMER GROUP의 목적 등을 부가적으로 서술하기 위한 파라미터이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
        consumer_group      => 'group s',
        category            => 'default',
        comments            => 'consumer group for sys users');
END;
/
```

32.2.2. CREATE_PLAN

새로운 RESOURCE PLAN을 생성하는 프러시저이다.

CREATE_PLAN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN
(
    plan                IN VARCHAR2,
    comments            IN VARCHAR2
);
```

- 파라미터

파라미터	설명
plan	생성할 RESOURCE PLAN의 이름이다. 이미 생성된 다른 PLAN 이름과 겹치지 않아야 한다.
comments	RESOURCE PLAN의 목적 등을 부가적으로 서술하기 위한 파라미터이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.CREATE_PLAN(
        plan                => 'plan t',
        comments            => 'resource plan for tibero');
END;
/
```

32.2.3. CREATE_PLAN_DIRECTIVE

새로운 RESOURCE PLAN DIRECTIVE를 생성하는 프러시저이다.

CREATE_PLAN_DIRECTIVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
(
    plan                IN VARCHAR2,
    group_or_subplan    IN VARCHAR2,
    comments            IN VARCHAR2,
    mgmt_p1             IN NUMBER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
plan	PLAN DIRECTIVE가 적용되어야 하는 대상 RESOURCE PLAN의 이름이다.
group_or_subplan	PLAN DIRECTIVE가 적용되어야 하는 대상 CONSUMER GROUP 혹은 SUBPLAN의 이름이다.
comments	PLAN DIRECTIVE의 목적 등을 부가적으로 서술하기 위한 파라미터이다.
mgmt_p1	PLAN DIRECTIVE에서 목표로 하는 CPU 자원의 사용률을 백분율로 나타낸 값이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
        plan                => 'plan t',
        group_or_subplan    => 'group s',
        comments            => 'plan directive for sys users',
        mgmt_p1             => 60);
END;
/
```

32.2.4. CREATE_SIMPLE_PLAN

8개 이하의 CONSUMER GROUP과 목표 CPU 사용량을 주어서 바로 간단한 RESOURCE PLAN을 생성하는 프러시저이다.

CREATE_SIMPLE_PLAN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN
(
    simple_plan         IN VARCHAR2 DEFAULT NULL,
    consumer_group1     IN VARCHAR DEFAULT NULL,
    consumer_group2     IN VARCHAR DEFAULT NULL,
    consumer_group3     IN VARCHAR DEFAULT NULL,
    consumer_group4     IN VARCHAR DEFAULT NULL,
    consumer_group5     IN VARCHAR DEFAULT NULL,
    consumer_group6     IN VARCHAR DEFAULT NULL,
    consumer_group7     IN VARCHAR DEFAULT NULL,
    consumer_group8     IN VARCHAR DEFAULT NULL,
    group1_percent      IN NUMBER DEFAULT NULL,
    group2_percent      IN NUMBER DEFAULT NULL,
    group3_percent      IN NUMBER DEFAULT NULL,
    group4_percent      IN NUMBER DEFAULT NULL,
    group5_percent      IN NUMBER DEFAULT NULL,
    group6_percent      IN NUMBER DEFAULT NULL,
    group7_percent      IN NUMBER DEFAULT NULL,
    group8_percent      IN NUMBER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
simple_plan	생성할 RESOURCE PLAN의 이름이다. 이미 생성된 다른 PLAN 이름과 겹치지 않아야 한다.
consumer_group1	생성할 1번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group2	생성할 2번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group3	생성할 3번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group4	생성할 4번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group5	생성할 5번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group6	생성할 6번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group7	생성할 7번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.
consumer_group8	생성할 8번째 CONSUMER GROUP의 이름이다. 이미 생성된 다른 CONSUMER GROUP 이름과 겹치지 않아야 한다.

파라미터	설명
group1_percent	1번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group2_percent	2번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group3_percent	3번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group4_percent	4번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group5_percent	5번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group6_percent	6번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group7_percent	7번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.
group8_percent	8번째 CONSUMER GROUP에게 지정할 목표 CPU 자원 사용량을 백분율로 나타낸 값이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(
        simple_plan         => 'plan s',
        consumer_group 1    => 'group 1',
        consumer_group 2    => 'group 2',
        consumer_group 3    => 'group 3',
        consumer_group 4    => 'group 4',
        consumer_group 5    => 'group 5',
        consumer_group 6    => 'group 6',
        consumer_group 7    => 'group 7',
        consumer_group 8    => 'group 8',
        group1_percent      => '20',
        group1_percent      => '20',
        group1_percent      => '10',
        group1_percent      => '10',
        group1_percent      => '10',
        group1_percent      => '10',
        group1_percent      => '10',
        group1_percent      => '10');
END;
/
```

32.2.5. DELETE_CONSUMER_GROUP

기존에 생성된 CONSUMER GROUP을 삭제할 때 사용하는 프러시저이다.

DELETE_CONSUMER_GROUP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP
(
    consumer_group      IN VARCHAR2
);
```

- 파라미터

파라미터	설명
consumer_group	삭제할 CONSUMER GROUP의 이름이다. 대상 CONSUMER GROUP이 존재해야 한다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP(
        consumer_group    => 'group s');
END;
/
```

32.2.6. DELETE_PLAN

기존에 생성된 RESOURCE PLAN을 삭제할 때 사용하는 프러시저이다.

DELETE_PLAN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN
(
    plan                IN VARCHAR2
);
```

- 파라미터

파라미터	설명
plan	삭제할 RESOURCE PLAN의 이름이다. 대상 RESOURCE PLAN이 존재해야 한다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.DELETE_PLAN(
        plan                => 'plan t');
END;
/
```

32.2.7. DELETE_PLAN_DIRECTIVE

기존에 생성된 PLAN DIRECTIVE를 삭제할 때 사용하는 프러시저이다.

DELETE_PLAN_DIRECTIVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE
(
    plan                IN VARCHAR2,
    group_or_subplan    IN VARCHAR2
);
```

- 파라미터

파라미터	설명
plan	삭제할 PLAN DIRECTIVE가 대상으로 하는 RESOURCE PLAN의 이름이다.
group_or_subplan	삭제할 PLAN DIRECTIVE가 대상으로 하는 CONSUMER GROUP 혹은 SUBPLAN의 이름이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE(
        plan                => 'plan t',
        group_or_subplan    => 'group s');
END;
/
```

32.2.8. SET_CONSUMER_GROUP_MAPPING

기존에 생성된 CONSUMER GROUP에 세션을 지정하는 MAPPING RULE을 만들 때 사용하는 프러시저이다.

SET_CONSUMER_GROUP_MAPPING 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
(
    attribute           IN VARCHAR2,
    value               IN VARCHAR2,
    consumer_group      IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
attribute	어떤 항목을 기준으로 세션을 CONSUMER GROUP으로 MAPPING 할 것인지를 지정한다.
value	어떤 값을 기준으로 세션을 CONSUMER GROUP으로 MAPPING 할 것인지를 지정한다.
consumer_group	생성할 MAPPING RULE이 대상으로 하는 CONSUMER_GROUP의 이름이다.

- 지원되는 attribute

attribute	설명
MODULE_NAME_ACTION	세션에 지정된 module name과 action 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.
MODULE_NAME	세션에 지정된 module name 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.
DATABASE_USER	세션에 지정된 database user 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.
CLIENT_PROGRAM	세션에 지정된 program 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.
CLIENT_OS_USER	세션에 지정된 os user 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.
CLIENT_MACHINE	세션에 지정된 machine 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.
CLIENT_ID	세션에 지정된 client identifier 항목이 같은 session을 지정된 CONSUMER GROUP으로 매핑한다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING(
        attribute           => 'database_user',
        value               => 'sys',
        consumer_group      => 'group s');
END;
/
```

32.2.9. SWITCH_PLAN

기존에 생성된 RESOURCE PLAN을 데이터베이스에 적용할 때 사용하는 프러시저이다.

SWITCH_PLAN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.SWITCH_PLAN
(
    switch_plan         IN VARCAHR2
);
```

- 파라미터

파라미터	설명
switch_plan	적용할 RESOURCE PLAN의 이름이다. 대상 RESOURCE PLAN이 존재해야 한다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.SWITCH_PLAN(
        switch_plan         => 'plan t');
END;
/
```

32.2.10. UPDATE_CONSUMER_GROUP

기존에 생성된 CONSUMER GROUP의 정보를 변경할 때 사용하는 프러시저이다.

UPDATE_CONSUMER_GROUP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP
(
    consumer_group      IN VARCHAR2,
    new_category        IN VARCHAR2 DEFAULT NULL,
    new_comments        IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
consumer_group	변경할 CONSUMER GROUP의 이름이다. 대상 CONSUMER GROUP이 존재해야 한다.
new_category	CONSUMER GROUP이 속하는 CATEGORY를 지정하기 위한 파라미터이다.
new_comments	CONSUMER GROUP의 목적 등을 부가적으로 서술하기 위한 파라미터이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP(
        consumer_group      => 'group s',
        new_cateogry        => 'default',
        new_comments        => 'consumer group for sys users');
END;
/
```

32.2.11. UPDATE_PLAN

기존에 생성된 RESOURCE _PLAN의 정보를 변경할 때 사용하는 프러시저이다.

UPDATE_PLAN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN
(
    plan                IN VARCHAR2,
    new_comments        IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
plan	변경할 RESOURCE PLAN의 이름이다. 대상 RESOURCE PLAN이 존재해야 한다.
new_comments	RESOURCE PLAN의 목적 등을 부가적으로 서술하기 위한 파라미터이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.UPDATE_PLAN(
        plan                => 'plan t',
        new_comments        => 'resource plan for tibero');
END;
/
```

32.2.12. UPDATE_PLAN_DIRECTIVE

기존에 생성된 PLAN_DIRECTIVE의 정보를 변경할 때 사용하는 프러시저이다.

UPDATE_PLAN_DIRECTIVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE
(
    plan                IN VARCHAR2,
    group_or_subplan    IN VARCHAR2,
    new_comments        IN VARCHAR2 DEFAULT NULL,
    mgmt_p1             IN NUMBER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
plan	변경할 PLAN DIRECTIVE가 대상으로 하는 RESOURCE PLAN의 이름이다.
group_or_subplan	변경할 PLAN DIRECTIVE가 대상으로 하는 CONSUMER_GROUP 또는 SUBPLAN의 이름이다.
new_comments	PLAN DIRECTIVE의 목적 등을 부가적으로 서술하기 위한 파라미터이다.
mgmt_p1	PLAN DIRECTIVE에서 목표로 하는 CPU 자원의 사용률을 백분율로 나타낸 값이다.

- 예제

```
BEGIN
    DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
        plan                => 'plan t',
        group_or_subplan    => 'group s',
        new_comments        => 'plan directive for sys users',
        mgmt_p1             => 60);
END;
/
```

제33장 DBMS_RESULT_CACHE

본 장에서는 DBMS_RESULT_CACHE 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

33.1. 개요

DBMS_RESULT_CACHE는 DBA가 데이터베이스의 공유 메모리에 존재하는 Result Cache를 제어할 수 있도록 인터페이스를 제공한다.

다음은 DBMS_RESULT_CACHE 패키지 내에 정의된 상수이다.

- STATUS_BYPS

```
STATUS_BYPS CONSTANT VARCHAR(10) := 'BYPASS'
```

- STATUS_DISA

```
STATUS_DISA CONSTANT VARCHAR(10) := 'DISABLED'
```

- STATUS_ENAB

```
STATUS_ENAB CONSTANT VARCHAR(10) := 'ENABLED'
```

33.2. 프러시저와 함수

본 절에서는 DBMS_RESULT_CACHE 패키지에서 제공하는 프러시저와 함수를 알파벳순으로 설명한다.

33.2.1. BYPASS

Result Cache의 BYPASS 모드를 설정하는 프러시저이다.

BYPASS 모드가 켜지면 저장된 결과를 더 이상 사용할 수 없고, 새로운 결과도 저장되지 않는다. 반대로 BYPASS 모드가 꺼지면 Result Cache를 정상적으로 사용할 수 있다.

BYPASS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESULT_CACHE.BYPASS
(
```

```
    bypass_mode     IN      BOOLEAN
);
```

- 파라미터

파라미터	설명
bypass_mode	– TRUE : Result Cache를 사용할 수 없다. – FALSE : Result Cache를 사용할 수 있다.

- 예제

```
BEGIN
    DBMS_RESULT_CACHE.BYPASS(TRUE);
    DBMS_RESULT_CACHE.FLUSH;
END;
/
```

33.2.2. FLUSH

Result Cache에 저장된 모든 객체들을 지우려고 시도한다. 단, 항상 모든 객체가 지워지지는 않는다.

FLUSH 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - 함수

```
DBMS_RESULT_CACHE.FLUSH
(
    retain_mem      IN      BOOLEAN DEFAULT FALSE,
    retain_stat     IN      BOOLEAN DEFAULT FALSE
)
RETURN BOOLEAN;
```

  - 프러시저

```
DBMS_RESULT_CACHE.FLUSH
(
    retain_mem      IN      BOOLEAN DEFAULT FALSE,
    retain_stat     IN      BOOLEAN DEFAULT FALSE
)
```

- 파라미터

파라미터	설명
retain_mem	– TRUE : Result Cache에 존재하는 유휴 메모리를 유지한다. – FALSE : Result Cache에 존재하는 유휴 메모리를 해제한다.
retain_stat	– TRUE : 통계정보를 유지한다. – FALSE : 통계정보를 지운다.

- 반환값

반환값	설명
BOOLEAN	모든 객체가 지워진 경우 TRUE를 반환한다.

- 예제

```
BEGIN
    DBMS_RESULT_CACHE.FLUSH;
END;
/
```

33.2.3. INVALIDATE

사용자가 지정한 객체에 종속된 모든 결과를 무효화시킨다.

INVALIDATE 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입
  – 객체 이름인 경우

```
DBMS_RESULT_CACHE.INVALIDATE
(
    owner       IN      VARCHAR2,
    name        IN      VARCHAR2
)
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE
(
    owner       IN      VARCHAR2,
    name        IN      VARCHAR2
);
```

  – 데이터 사전 번호인 경우

```
DBMS_RESULT_CACHE.INVALIDATE
(
    obj_id      IN      NUMBER
)
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE
(
    obj_id      IN      NUMBER
);
```

- 파라미터

파라미터	설명
owner	스키마 이름이다.
name	객체 이름이다.
obj_id	데이터 사전 번호이다.

- 반환값

반환값	설명
NUMBER	무효화시킨 객체 수를 반환한다.

- 예제

```
BEGIN
    DBMS_RESULT_CACHE.INVALIDATE('TIBEROPRIGHT', 'EMP');
END;
/
```

33.2.4. INVALIDATE_OBJECT

사용자가 지정한 결과를 무효화시킨다.

INVALIDATE_OBJECT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - 결과 객체의 주소인 경우

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT
(
    id          IN      BINARY_INTEGER
)
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT
(
    id          IN      BINARY_INTEGER
);
```

– 결과 객체의 고유번호인 경우

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT
(
    cache_id    IN      VARCHAR2
)
RETURN NUMBER;
```

```
DBMS_RESULT_CACHE.INVALIDATE_OBJECT
(
    cache_id    IN      VARCHAR2
);
```

- 파라미터

파라미터	설명
id	결과 객체의 주소이다.
cache_id	결과 객체의 고유번호이다.

- 반환값

반환값	설명
NUMBER	무효화시킨 객체수를 반환한다.

- 예제

```
BEGIN
    DBMS_RESULT_CACHE.INVALIDATE_OBJECT(3);
END;
/
```

33.2.5. MEMORY_REPORT

Result Cache의 메모리 사용량 정보를 화면에 출력해 준다. 단, tbSQL의 serveroutput 옵션이 켜져있어야 한다.

MEMORY_REPORT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESULT_CACHE.MEMORY_REPORT
(
    detailed    IN      BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
detailed	– TRUE : 더 자세한 결과가 출력된다. – FALSE : 일반적인 결과가 출력된다.

- 예제

```
BEGIN
    DBMS_RESULT_CACHE.MEMORY_REPORT;
END;
/
```

33.2.6. STATUS

Result Cache의 현재 상태를 반환한다.

STATUS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RESULT_CACHE.STATUS
RETURN VARCHAR2;
```

- 반환값

반환값	설명
VARCHAR2	– STATUS_DISA : Result Cache를 사용할 수 없다. – STATUS_ENAB : Result Cache를 사용할 수 있다. – STATUS_BYPS : Result Cache를 일시적으로 사용할 수 없다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(DBMS_RESULT_CACHE.STATUS);
END;
/
```

제34장 DBMS_RLS

본 장에서는 DBMS_RLS 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

34.1. 개요

DBMS_RLS 패키지는 Tibero의 가상 개인 데이터베이스 기능을 구성하고 관리하는 데 사용할 수 있는 프러시저와 함수를 제공한다.

가상 개인 데이터베이스에서는 테이블, 뷰, 동의어에 대해 동적으로 조건문을 생성하여 해당 스키마 객체에서 반환하는 로우에 대한 접근을 선택적으로 제한한다. 동적 조건문은 문자열을 반환하는 PSM 함수를 통해 만들 수 있으며, 이 함수를 추가되는 보안 정책과 연결시켜 줌으로써 정책이 구현된다. 다음의 예를 보자.

```
DBMS_RLS.ADD_POLICY ('scott', 'emp', 'pol1', 'secadm', 'emp_sec', 'select');
```

pol1이라는 정책을 통해서 scott 스키마의 emp 테이블을 select 할 때마다 Tibero 는 secadm의 emp_sec 함수를 호출하여 조건문을 동적으로 만들게 되고, 그 조건문을 where 절로 추가하여 실제 select를 수행한다.

동적 조건문을 생성하기 위해 현재의 상황(혹은 조건)을 조사할 필요가 생기는데, 이때 보통 애플리케이션 문맥 기능을 이용한다. 해당 내용은 DBMS_SESSION 패키지 및 SYS_CONTEXT 내장 함수를 참조한다.

34.2. 프러시저와 함수

본 절에서는 DBMS_RLS 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

34.2.1. ADD_POLICY

해당 테이블, 뷰, 동의어에 대해 가상 개인 데이터베이스 보안 정책을 추가한다. 이 프러시저는 호출됨과 동시에 현재 트랜잭션을 커밋시킨다.

정책 함수는 다음의 함수 프로토타입에 맞추어 작성해야만 한다.

```
FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2)
    RETURN VARCHAR2;
```

object_schema, object_name 에는 해당 정책을 적용한 테이블, 뷰, 동의어의 스키마 이름과 객체의 이름이 전달된다.

ADD_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RLS.ADD_POLICY
(
    object_schema         IN VARCHAR2 DEFAULT NULL,
    object_name           IN VARCHAR2,
    policy_name           IN VARCHAR2,
    function_schema       IN VARCHAR2 DEFAULT NULL,
    policy_function       IN VARCHAR2,
    statement_types       IN VARCHAR2 DEFAULT NULL,
    update_check          IN BOOLEAN DEFAULT FALSE,
    enable                IN BOOLEAN DEFAULT TRUE,
    static_policy         IN BOOLEAN DEFAULT FALSE,
    policy_type           IN BINARY_INTEGER DEFAULT NULL,
    sec_relevant_cols     IN VARCHAR2 DEFAULT NULL,
    set_relevant_cols_opt IN BINARY_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
object_schema	테이블, 뷰, 동의어를 가지고 있는 스키마이다. 명시하지 않으면 현재 사용자의 스키마이다.
object_name	테이블, 뷰, 동의어의 이름이다.
policy_name	새로 추가하고자 하는 정책의 이름이다. 해당 스키마 객체에 대해 유일해야 한다.
function_schema	조건문 생성 함수의 스키마이다. 명시하지 않으면 현재 사용자의 스키마이다.
policy_function	조건문 생성 함수의 이름이다. 패키지 내에 존재한다면 패키지 이름과 함께 명시한다.
statement_types	정책을 적용할 SQL 문의 유형이다. SELECT, INSERT, UPDATE, DELETE의 조합을 명시할 수 있다. 기본값은 이 4개를 모두 명시한 것이다.
update_check	INSERT, UPDATE일 경우에 적용된다. TRUE일 경우 새로 insert되거나 update된 값이 정책의 조건문을 만족하는지 검사한다. (기본값: FALSE)
enable	정책을 추가함과 동시에 활성화시킬지 명시한다. (기본값: TRUE)
static_policy	정책 유형을 설정한다. policy_type이 지정되지 않을 경우에만 유효하다.

파라미터	설명
	– TRUE : STATIC(기본값) – FALSE : DYNAMIC
policy_type	정책 유형을 설정한다. (기본값: NULL) 가능한 정책 유형 다음과 같다. – STATIC : 반환되는 조건문이 운영 중의 상황과는 무관할 때 사용한다. 정책 함수는 최초 한번만 수행되며 그 결과는 공유 메모리에 저장된다. – SHARED_STATIC : STATIC과 동일하나 대상 스키마 객체와 무관하게 이전에 해당 정책 함수를 수행한 적이 있으면 그 결과를 재사용한다. – CONTEXT_SENSITIVE : SQL 문 수행할 때 애플리케이션 문맥이 바뀌었을 경우 정책 함수를 다시 수행한다. 저장된 조건문은 세션의 종료와 함께 삭제된다. – SHARED_CONTEXT_SENSITIVE : CONTEXT_SENSITIVE와 동일하나 대상 스키마 객체와 무관하게 이전에 해당 정책 함수를 수행한 적이 있으면 그 결과를 재사용한다. – DYNAMIC : 디폴트 정책 유형이다. 정책 함수를 항상 수행한다.
sec_relevant_cols	컬럼 레벨 VPD 기능을 활성화한다. 조건문 함수의 결과에 따라 값을 NULL로 가릴 컬럼들의 이름을 명시한다. 이름은 쉼표 혹은 공백 문자로 구분할 수 있다. 이 인자는 테이블 혹은 뷰에 대해 명시가 가능하며, 동의어에 대해서는 명시가 불가능하다. (기본값: NULL, 컬럼 레벨 VPD 기능을 사용하지 않음)
sec_relevant_cols_opt	가능한 값은 NULL(디폴트) 혹은 dbms_rls.ALL_ROWS 둘 중 하나이다. set_relevant_cols 인자를 통해 컬럼 레벨 VPD 기능을 활성화시켰을 경우에 이 인자의 값은 dbms_rls.ALL_ROWS 이어야 하며, 그렇지 않을 경우에는 NULL이어야 한다.

- 예제

```
BEGIN
    DBMS_RLS.ADD_POLICY (
        object_schema=>'scott',
        object_name=>'emp',
        policy_name=>'pol1',
        function_schema=>'secadm',
        policy_function=>'emp_sec',
```

```
        statement_types=>'insert,update',
        policy_type=>DBMS_RLS.CONTEXT_SENSITIVE);
END;
/
```

34.2.2. DROP_POLICY

해당 테이블, 뷰, 동의어에 대해 걸려 있는 가상 개인 데이터베이스 보안 정책을 제거한다. 이 프러시저는 호출됨과 동시에 현재 트랜잭션을 커밋시킨다.

DROP_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RLS.DROP_POLICY
(
    object_schema   IN VARCHAR2 DEFAULT NULL,
    object_name     IN VARCHAR2,
    policy_name     IN VARCHAR2
);
```

- 파라미터

파라미터	설명
object_schema	테이블, 뷰, 동의어를 가지고 있는 스키마이다. 명시하지 않으면 현재 사용자의 스키마이다.
object_name	테이블, 뷰, 동의어의 이름이다.
policy_name	제거하고자 하는 정책의 이름이다.

34.2.3. ENABLE_POLICY

해당 테이블, 뷰, 동의어에 대해 걸려 있는 가상 개인 데이터베이스 보안 정책을 활성화 혹은 비활성화한다. 이 프러시저는 호출됨과 동시에 현재 트랜잭션을 커밋시킨다.

ENABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RLS.ENABLE_POLICY
(
    object_schema   IN VARCHAR2 DEFAULT NULL,
    object_name     IN VARCHAR2,
```

```
    policy_name     IN VARCHAR2,
    enable          IN BOOLEAN DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
object_schema	테이블, 뷰, 동의어를 가지고 있는 스키마이다. 명시하지 않으면 현재 사용자의 스키마이다.
object_name	테이블, 뷰, 동의어의 이름이다.
policy_name	활성화 혹은 비활성할 정책의 이름이다.
enable	– TRUE : 정책을 활성화한다. – FALSE : 정책을 비활성화한다.

34.2.4. REFRESH_POLICY

이 프러시저를 호출하면 해당 보안 정책으로 인해 영향을 받았던 실행 계획들과 메모리에 저장되어 있던 정책 함수의 결과들을 모두 무효화시킨다. 결과적으로 SQL 문이 다시 수행될 때 문장은 다시 파싱되며, 정책 함수는 다시 수행된다. 비활성화된 정책에 대해 이 프러시저를 호출하면 에러가 반환된다.

REFRESH_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_RLS.REFRESH_POLICY
(
    object_schema   IN VARCHAR2 DEFAULT NULL,
    object_name     IN VARCHAR2,
    policy_name     IN VARCHAR2
);
```

- 파라미터

파라미터	설명
object_schema	테이블, 뷰, 동의어를 가지고 있는 스키마이다. 명시하지 않으면 현재 사용자의 스키마이다.
object_name	테이블, 뷰, 동의어의 이름이다.
policy_name	Refresh하고자 하는 정책의 이름이다.

제35장 DBMS_ROWID

본 장에서는 DBMS_ROWID 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

35.1. 개요

DBMS_ROWID는 SQL 문장이나 PSM을 수행하여 얻은 ROWID에 담긴 정보를 보거나 임의의 ROWID를 만들기 위한 패키지이다. ROWID에는 세그먼트(segment), 블록, 절대 파일(absolute file), 로우의 번호에 대한 정보가 포함되어 있다.

35.2. 프러시저

본 절에서는 DBMS_ROWID 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

35.2.1. ROWID_INFO

ROWID에 담긴 모든 정보를 한 번에 추출하기 위해 사용되는 프러시저이다. 프러시저이기 때문에 SQL 문장 내에서는 사용될 수 없다.

ROWID_INFO 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE ROWID_INFO
(
    rowid_in        IN   ROWID,
    segment_number  OUT  NUMBER,
    absolute_fno    OUT  NUMBER,
    block_number    OUT  NUMBER,
    row_number      OUT  NUMBER
);
```

- 파라미터

파라미터	설명
rowid_in	정보를 얻고자 하는 대상 ROWID이다.
segment_number	ROWID에서 추출한 세그먼트의 번호가 저장된다.

파라미터	설명
absolute_fno	ROWID에서 추출한 절대 파일의 번호가 저장된다.
block_number	ROWID에서 추출한 블록의 번호가 저장된다.
row_number	ROWID에서 추출한 로우의 번호가 저장된다.

- 예제

```
DBMS_ROWID.ROWID_INFO(rowid_1, sgmt_no, fno, blk_no, row_no);
DBMS_OUTPUT.PUT_LINE('segment number : ' || sgmt_no);
DBMS_OUTPUT.PUT_LINE('absolute file number : ' || fno);
DBMS_OUTPUT.PUT_LINE('block number : ' || blk_no);
DBMS_OUTPUT.PUT_LINE('row number : ' || row_no);
```

35.3. 함수

본 절에서는 DBMS_ROWID 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

35.3.1. ROWID_CREATE

입력한 정보에 맞는 ROWID를 생성하는 함수이다. 주로 가져온 ROWID가 올바로 만들어진 것인지 확인해보는 용도로 사용한다. 사용자가 임의의 정보로 ROWID를 생성할 수도 있지만 큰 의미는 없다.

ROWID_CREATE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_CREATE
(
    segment_number  IN   NUMBER,
    absolute_fno    IN   NUMBER,
    block_number    IN   NUMBER,
    row_number      IN   NUMBER
)
RETURN ROWID;
```

- 파라미터

파라미터	설명
segment_number	세그먼트의 번호이다.
absolute_fno	절대 파일의 번호이다.
block_number	파일 내의 블록 번호이다.
row_number	블록 내의 로우 번호이다.

- 예제

```
DBMS_ROWID.ROWID_INFO(rowid_1, sgmt_no, fno, blk_no, row_no);
my_rowid := DBMS_ROWID.ROWID_CREATE(sgmt_no, fno, blk_no, row_no);
DBMS_OUTPUT.PUT_LINE(rowid_1 || ' ' || my_rowid);
```

35.3.2. ROWID_SEGMENT

주어진 ROWID에서 세그먼트의 번호를 추출하여 반환하는 함수이다.

ROWID_SEGMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_SEGMENT
(
    row_id   IN   ROWID
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
row_id	정보를 추출할 ROWID이다.

- 예제

```
SELECT dbms_rowid.rowid_segment(rowid)
FROM table;
```

35.3.3. ROWID_BLOCK_NUMBER

주어진 ROWID에서 파일 내의 블록의 번호를 추출하여 반환하는 함수이다.

ROWID_BLOCK_NUMBER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_BLOCK_NUMBER
(
    row_id   IN   ROWID
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
row_id	정보를 추출할 ROWID이다.

- 예제

```
SELECT dbms_rowid.rowid_block_number(rowid)
FROM table;
```

35.3.4. ROWID_ROW_NUMBER

주어진 ROWID에서 블록 내의 로우의 번호를 추출하여 반환하는 함수이다.

ROWID_ROW_NUMBER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_ROW_NUMBER
(
    row_id   IN   ROWID
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
row_id	정보를 추출할 ROWID이다.

- 예제

```
SELECT dbms_rowid.rowid_row_number(rowid)
FROM table;
```

35.3.5. ROWID_ABSOLUTE_FNO

주어진 ROWID에서 세그먼트 내의 절대 파일의 번호를 추출하여 반환하는 함수이다.

ROWID_ABSOLUTE_FNO 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_ABSOLUTE_FNO
(
    row_id    IN   ROWID
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
row_id	정보를 추출할 ROWID이다.

- 예제

```
SELECT dbms_rowid.rowid_absolute_fno(rowid)
FROM table;
```

35.3.6. ROWID_TO_DBA

주어진 ROWID에서 DBA 주소를 추출하여 반환하는 함수이다.

ROWID_TO_DBA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_TO_DBA
(
    row_id    IN   ROWID
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
row_id	정보를 추출할 ROWID이다.

- 예제

```
SELECT dbms_rowid.rowid_to_dba(rowid)
FROM table;
```

35.3.7. ROWID_TO_RELATIVE_FNO

주어진 ROWID에서 절대 파일의 번호를 추출해 테이블 스페이스의 상대적인 파일 번호로 변환하는 함수이다. 일반적으로 SELECT 문장에서 절대 파일의 번호를 추출했을 때 파일 번호는 이러한 상대적 파일 번호이다.

ROWID_TO_RELATIVE_FNO 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ROWID_TO_RELATIVE_FNO
(
    row_id    IN   ROWID
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
row_id	정보를 추출할 ROWID이다.

- 예제

```
SELECT dbms_rowid.rowid_to_relative_fno(rowid)
FROM table;
```

제36장 DBMS_SCHEDULER

본 장에서는 DBMS_SCHEDULER 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

36.1. 개요

Tibero에서는 주기적으로 DBMS_SCHEDULER에 의해서 데이터베이스에 추가된 JOB을 검사하여, 사용자가 설정한 시각이나 조건이 만족되면 해당하는 JOB을 실행한다.

DBMS_SCHEDULER은 PSM에서 사용 가능한 문장을 JOB으로 등록하고, 이 JOB을 실행할 수 있는 연산을 제공하는 패키지이다. DBMS_SCHEDULER 패키지 내의 프러시저를 이용하여, JOB을 데이터베이스에 추가하고 바로 실행하거나 정해진 시각과 조건에 따라서 실행되도록 설정할 수 있다.

다음은 DBMS_SCHEDULER 패키지의 특징이다.

- DBMS_SCHEDULER 패키지의 사용할 때 DBA 권한은 필요하지 않으며, 추가된 JOB은 오직 JOB의 소유자만 실행하거나 변경할 수 있다.
- JOB을 추가 또는 변경하는 경우 커밋을 실행하지 않아도 자동으로 커밋되며, JOB 내에서 실행한 작업도 자동으로 커밋된다.
- 다음의 뷰를 지원한다.

```
[DBA | ALL | USER]_SCHEDULER_CHAINS
[DBA | ALL | USER]_SCHEDULER_JOBS
[DBA | ALL | USER]_SCHEDULER_JOB_LOG
[DBA | ALL | USER]_SCHEDULER_JOB_RUN_DETAILS
[DBA | ALL | USER]_SCHEDULER_PROGRAMS
[DBA | ALL | USER]_SCHEDULER_RULES
[DBA | ALL | USER]_SCHEDULER_RULE_CTX
[DBA | ALL | USER]_SCHEDULER_RUNNING_JOBS
[DBA | ALL | USER]_SCHEDULER_SCHEDULES
[DBA | ALL | USER]_SCHEDULER_STEPS
[DBA | ALL | USER]_SCHEDULER_STEP_CTX
```

- 등록된 JOB은 내부적으로 DBMS_JOB 패키지를 통해서 수행된다.
- 실행 중인 JOB을 정지할 수 있는 기능은 제공하지 않는다.

시간 기반으로 스케줄링을 설정할 때는 Calendaring Syntax를 사용한다.

파라미터	설명
Calendaring Syntax	Calendaring Syntax의 문법은 다음과 같다. `repeat_interval = FREQ=?` `  [; INTERVAL=?] [; BYSECOND=?] [; BYMINUTE=?] [; BYHOUR=?]` `  [; BYMONTHDAY=?] [; BYDAY=?] [;BYMONTH=?]` FREQ는 반복 유형을 지정하며, 반드시 하나의 단위를 지정해야 한다. 다음은 FREQ에서 지원하는 반복 단위이다. – SECONDLY – MINUTELY – HOURLY – DAILY – WEEKLY – MONTHLY – YEARLY INTERVAL은 FREQ로 지정한 단위를 얼마나 반복적으로 수행할 지 결정하는 파라미터이다. 예를 들어 'FREQ=HOURLY; INTERVAL=3'와 같이 지정할 경우 매 3시간마다 수행한다. 다음은 INTERVAL 외에 사용자 지정 반복 단위이다. – BYSECOND – BYMINUTE – BYHOUR – BYMONTHDAY – BYDAY – BYMONTH

36.2. 프러시저

본 절에서는 DBMS_SCHEDULER 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

36.2.1. CREATE_CHAIN

새로운 JOB 체인을 생성하는 프러시저이다. 체인은 생성할 때 항상 disabled 상태이기 때문에 사용하기 전에 명시적으로 ENABLE 프러시저를 통해 enabled 상태로 변경해야 한다.

CREATE_CHAIN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.CREATE_CHAIN
(
    chain_name          IN    VARCHAR2,
    rule_set_name       IN    VARCHAR2 DEFAULT NULL,
    evaluation_interval IN    INTERVAL DAY TO SECOND DEFAULT NULL,
    comments            IN    VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
chain_name	생성할 체인의 이름이다. 이미 생성된 다른 오브젝트 이름과 겹치지 않아야 한다.
rule_set_name	현재 사용하지 않는 파라미터이다.
evaluation_interval	NULL 값인 경우, JOB이 시작하거나 STEP이 종료한 경우에만 룰에 대한 재평가를 실시한다. 현재 NULL 값만 지원한다.
comments	체인의 목적 등을 부가적으로 서술하기 위한 파라미터이다.

- 예제

```
BEGIN
    /* chain_name이라는 이름을 가진 체인을 생성한다. */
    DBMS_SCHEDULER.CREATE_CHAIN(chain_name          => 'chain_name',
                                rule_set_name       => NULL,
                                evaluation_interval => NULL,
                                comments            => 'my first job chain');
END;
/
```

36.2.2. CREATE_JOB

JOB을 생성하는 프러시저이다. enabled 파라미터를 TRUE로 설정하는 경우 설정한 스케줄에 따라 스케줄러에 의해 자동으로 실행된다. 하지만 disabled 상태인 경우 명시적으로 SET_ATTRIBUTE 프러시저를 통해 enabled 상태로 변경하기 전까지 실행되지 않는다.

CREATE_JOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.CREATE_JOB
(
    job_name            IN VARCHAR2,
    job_type            IN VARCHAR2,
    job_action          IN VARCHAR2,
    program_name        IN VARCHAR2 DEFAULT NULL,
    schedule_name       IN VARCHAR2 DEFAULT NULL,
    number_of_arguments IN BINARY_INTEGER DEFAULT 0,
    start_date          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    repeat_interval     IN VARCHAR2 DEFAULT NULL,
    event_condition     IN VARCHAR2 DEFAULT NULL,
    queue_spec          IN VARCHAR2 DEFAULT NULL,
    end_date            IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    job_class           IN VARCHAR2 DEFAULT 'DEFAULT_JOB_CLASS',
    enabled             IN BOOLEAN DEFAULT FALSE,
    auto_drop           IN BOOLEAN DEFAULT TRUE,
    comments            IN VARCHAR2 DEFAULT NULL,
    credential_name     IN VARCHAR2 DEFAULT NULL,
    destination_name    IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
job_name	생성할 JOB의 이름을 지정한다. 다른 객체와 구분되는 이름으로 지정해야 한다.
job_type	생성할 JOB의 타입을 지정한다. 지원되는 타입은 아래와 같다. – PSM_BLOCK : PSM 코드를 생성하는 JOB이다. JOB에 인자를 전달하는 기능은 지원하지 않으므로 number_of_arguments 파라미터는 0이 되어야 한다. – CHAIN : JOB의 타입이 체인일 경우 지정한다. 체인에 인자를 전달하는 기능은 지원하지 않으므로 number_of_arguments 파라미터는 0이 되어야 한다.
job_action	JOB이 수행할 동작을 지정한다. 타입별로 아래와 같은 동작을 지정할 수 있다.

파라미터	설명
	– PSM_BLOCK : PSM 코드를 지정한다. 반드시 세미콜론으로 끝나야 한다. 예를 들어 다음과 같이 설정한다. `BEGIN my_proc();` `END;` 또는 `DECLARE arg pls_integer:= 10;` `BEGIN my_proc2(arg);` `END;` – CHAIN : JOB 타입이 체인일 경우 수행할 체인 이름을 지정한다.
program_name	JOB과 관련된 프로그램 이름을 지정한다. 현재는 타입이 체인인 경우에만 program_name을 사용한다.
schedule_name	JOB과 관련된 SCHEDULE의 이름을 지정한다.
number_of_arguments	지원되지 않는 기능이므로 값을 무시한다.
start_date	스케줄러에 의해서 JOB을 시작할 시간을 지정한다. NULL인 경우 현재 시간으로 설정된다. 시스템의 상태에 따라서 시작 시간은 오차가 발생할 수 있다. repeat_interval에 지정된 식은 이 값을 참조하여 다음 수행 스케줄을 결정한다.
repeat_interval	JOB 수행 주기를 지정한다. NULL 값으로 지정한 경우 한 번만 수행된다. 주기는 Calendaring Syntax를 이용하여 지정할 수 있다.
event_condition	지원되지 않는 기능이므로 값을 무시한다.
queue_spec	지원되지 않는 기능이므로 값을 무시한다.
end_date	스케줄러에 의해서 JOB이 마지막으로 수행될 시간을 지정한다.
job_class	지원되지 않는 기능이므로 값을 무시한다.
enabled	JOB을 생성할 때 enabled 상태를 지정한다. ENABLE, DISABLE 프러시저를 통해 값을 변경할 수 있다. (TRUE/FALSE, 기본값: FALSE)
auto_drop	수행이 완료된 JOB을 자동으로 삭제할 지 여부를 지정한다.
comments	JOB에 대한 주석을 지정한다.
credential_name	지원되지 않는 기능이므로 값을 무시한다.
destination_name	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
      job_name        => 'job_name',
      job_type        => 'PSM_BLOCK',
      job_action      => 'begin my_proc; end;',
      start_date      => SYSTIMESTAMP,
      repeat_interval => 'FREQ=HOURLY; BYHOUR=1,2,3; BYMINUTE=0; BYSECOND=30',
      enabled         => TRUE,
      comments        => 'Job defined entirely by the CREATE JOB procedure.');
END;
/
```

36.2.3. CREATE_PROGRAM

프로그램을 생성하는 프러시저이다. 현재는 JOB 체인을 생성할 때만 사용된다.

CREATE_PROGRAM 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.CREATE_PROGRAM
(
    program_name             IN VARCHAR2,
    program_type             IN VARCHAR2,
    program_action           IN VARCHAR2,
    number_of_arguments      IN PLS_INTEGER DEFAULT 0,
    enabled                  IN BOOLEAN DEFAULT FALSE,
    comments                 IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
program_name	생성할 프로그램의 이름을 지정한다.
program_type	생성할 프로그램의 타입을 지정한다. 현재는 'PSM_BLOCK' 타입만 지원한다.
program_action	프로그램이 수행할 동작을 지정한다. 타입별로 아래와 같은 동작을 지정할 수 있다. – PSM_BLOCK : PSM 코드를 지정한다. 반드시 세미콜론으로 끝나야 한다. 예)

파라미터	설명
	`` BEGIN my_proc(); END; ``` ``` DECLARE arg pls_integer:= 10; BEGIN my_proc2(arg); END; ```
number_of_arguments	지원되지 않는 기능이므로 값을 무시한다.
enabled	프로그램을 생성할 때 enabled 상태를 지정한다. ENABLE, DISABLE 프러시저를 통해 값을 변경할 수 있다. (TRUE/FALSE, 기본값: FALSE)
comments	프로그램에 대한 주석을 지정한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.CREATE_PROGRAM(program_name   => 'program_name',
                                  program_type   => 'PSM_BLOCK',
                                  program_action => 'my_job;');
END;
/
```

36.2.4. CREATE_SCHEDULE

SCHEDULE을 생성하는 프러시저이다. 현재는 JOB 체인을 생성할 때만 사용된다.

CREATE_SCHEDULE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.CREATE_SCHEDULE
(
    schedule_name            IN VARCHAR2,
    start_date               IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    repeat_interval          IN VARCHAR2,
    end_date                 IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    comments                 IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
schedule_name	생성할 SCHEDULE의 이름을 지정한다.
start_date	스케줄러에 의해서 SCHEDULE을 시작할 시간을 지정한다. NULL인 경우 현재 시간으로 설정된다. 시스템의 상태에 따라서 시작 시간은 오차가 발생할 수 있다. repeat_interval에 지정된 식은 이 값을 참조하여 다음 수행 스케줄을 결정한다.
repeat_interval	SCHEDULE의 수행 주기를 지정한다. NULL 값으로 지정한 경우 한 번만 수행된다. 주기는 Calendaring Syntax를 이용하여 지정할 수 있다.
end_date	스케줄러에 의해서 마지막으로 SCHEDULE이 수행될 시간을 지정한다.
comments	프로그램에 대한 주석을 지정한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.CREATE_SCHEDULE(
        schedule_name   => 'schedule_name',
        repeat_interval => 'FREQ=HOURLY; BYHOUR=1,2,3; BYMINUTE=0; BYSECOND=30'
    );
END;
/
```

36.2.5. DEFINE_CHAIN_RULE

기존에 생성된 체인에서 사용할 룰을 지정할 때 사용하는 프러시저이다. condition과 action의 짝으로 이루어지며, condition은 SQL 구문의 WHERE 절에 해당하는 문법을 사용하며, action은 START, END 명령과 수행할 스텝을 콤마(,)로 구분하여 리스트로 지정한다. JOB 체인 수행 중 condition이 만족되면 해당 action이 수행된다.

DEFINE_CHAIN_RULE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE
(
    chain_name              IN VARCHAR2,
    condition               IN VARCHAR2,
    action                  IN VARCHAR2,
    rule_name               IN VARCHAR2 DEFAULT NULL,
```

```
    comments                IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
chain_name	룰을 적용할 체인 이름을 지정한다.
condition	스텝 속성을 조건문 형식으로 지정하여 JOB 체인을 구성하기 위해 사용한다. Boolean expression으로 TRUE로 평가되는 식일 때만 action이 수행된다. 모든 체인은 체인의 시작을 위해서 반드시 TRUE 평가되는 룰을 가져야 한다. 예를 들어 condition은 SQL WHERE clause syntax를 지원하므로 '1=1' 형태의 식을 지정한다. condition을 SQL의 형태로 사용할려면 select statement와 where 절이 와야 하는데, 이 경우 chain step attribute(체인 스텝 속성)를 bind variable로 사용할 수 있는데, 사용법은 step_name.attribute와 같다. attribute의 종류에는 completed와 state가 있다. state attribute는 'STARTED', 'RUNNING', 'SUCCEEDED', 'FAILED' 값을 가질 수 있다. state attribute가 이 값 중 하나의 값을 갖는다면, completed attribute는 'TRUE'이고, 그렇지 않으면 completed 값은 'FALSE'이다.
action	RULE이 TRUE일 때 수행할 동작을 지정한다. 다음의 문법을 지원한다. – `START step_1[, step_2 ..]` START step_1[, step_2 ..]는 조건이 참일 때 리스트 형태로 지정된 스텝을 수행한다. – `END` END는 조건이 참일 때 체인을 종료한다.
rule_name	생성될 룰의 이름이다.
comments	룰에 대한 comment이다.

- 예제

```
BEGIN
    /* 체인을 시작하기 위한 RULE을 생성한다. */
    DBMS_SCHEDULER.DEFINE_CHAIN_RULE('chain_name',
                                     '1=1',
                                     'start step1',
```

```
                                    'rule1');

    /* step3이 종료되면, 체인을 종료한다. */
    DBMS_SCHEDULER.DEFINE_CHAIN_RULE('chain_name',
                                    'step3.completed =''TRUE''',
                                    'end',
                                    'rule2');

    /* step1이 성공하면 step3을 수행한다. */
    DBMS_SCHEDULER.DEFINE_CHAIN_RULE('chain_name',
                                    'step1.state=''SUCCEEDED''',
                                    'start step3',
                                    'rule3');
END;
/
```

36.2.6. DEFINE_CHAIN_STEP

기존의 생성된 체인에 스텝을 지정할 때 사용하는 프러시저이다.

DEFINE_CHAIN_STEP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DEFINE_CHAIN_STEP
(
    chain_name              IN VARCHAR2,
    step_name               IN VARCHAR2,
    program_name            IN VARCHAR2
);
```

- 파라미터

파라미터	설명
chain_name	스텝을 정의할 체인 이름을 지정한다.
step_name	스텝 이름을 지정한다.
program_name	스텝에서 수행할 프로그램을 지정한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DEFINE_CHAIN_STEP('chain_name', 'step1', 'program_name');
END;
/
```

36.2.7. DISABLE

이 프러시저는 program, job, chain을 disable하기 위해 사용한다. 각 객체의 enabled attribute를 'FALSE'로 설정한다.

DISABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DISABLE
(
    name              IN VARCHAR2,
    force             IN BOOLEAN DEFAULT FALSE,
    commit_semantics  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR'
);
```

- 파라미터

파라미터	설명
name	속성을 변경할 객체의 이름을 지정한다. 현재 JOB만 지원한다.
force	지원되지 않는 기능이므로 값을 무시한다.
commit_semantics	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DISABLE('my_job');
END;
/
```

36.2.8. DROP_CHAIN

데이터베이스에서 기존 체인을 삭제하는 프러시저이다.

DROP_CHAIN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DROP_CHAIN
(
    chain_name              IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
chain_name	삭제할 체인의 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DROP_CHAIN('my_chain');
END;
/
```

36.2.9. DROP_CHAIN_RULE

데이터베이스에서 기존 체인에서 룰을 삭제하는 프러시저이다.

DROP_CHAIN_RULE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DROP_CHAIN_RULE
(
    chain_name              IN VARCHAR2,
    rule_name               IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
chain_name	적용할 체인의 이름을 지정한다.
rule_name	삭제할 룰의 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DROP_CHAIN_RULE('my_chain', 'my_rule1');
END;
/
```

36.2.10. DROP_CHAIN_STEP

데이터베이스에서 기존 체인 스텝을 삭제하는 프러시저이다.

DROP_CHAIN_STEP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DROP_CHAIN_STEP
(
    chain_name              IN VARCHAR2,
    step_name               IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
chain_name	적용할 체인의 이름을 지정한다.
step_name	삭제할 스텝의 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DROP_CHAIN_STEP('my_chain', 'my_step1');
END;
/
```

36.2.11. DROP_JOB

데이터베이스에서 기존 JOB을 삭제하는 프러시저이다.

DROP_JOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DROP_JOB
(
    job_name                IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE,
    defer                   IN BOOLEAN DEFAULT FALSE,
    commit_semantics        IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR'
);
```

- 파라미터

파라미터	설명
job_name	삭제할 JOB의 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.
defer	지원되지 않는 기능이므로 값을 무시한다.
commit_semantics	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DROP_JOB('my_job');
END;
/
```

36.2.12. DROP_PROGRAM

데이터베이스에서 기존 프로그램을 삭제하는 프러시저이다.

DROP_PROGRAM 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DROP_PROGRAM
(
    program_name            IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
program_name	삭제할 프로그램 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DROP_PROGRAM('my_program');
END;
/
```

36.2.13. DROP_SCHEDULE

데이터베이스에서 기존 SCHEDULE을 삭제하는 프러시저이다.

DROP_SCHEDULE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.DROP_SCHEDULE
(
    schedule_name           IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
schedule_name	삭제할 SCHEDULE의 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.DROP_SCHEDULE('my_schedule');
END;
/
```

36.2.14. ENABLE

이 프러시저는 program, job, chain을 enable하기 위해 사용한다. 각 객체의 enabled attribute를 'TRUE'로 설정한다.

ENABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.ENABLE
(
    name               IN VARCHAR2,
    commit_semantics   IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR'
);
```

- 파라미터

파라미터	설명
name	속성을 변경할 객체의 이름을 지정한다. 현재 JOB만 지원한다.
commit_semantics	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.ENABLE('my_job');
END;
/
```

36.2.15. PURGE_LOG

이 프러시저는 기록된 log를 지워준다.

PURGE_LOG 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.PURGE_LOG
(
    log_history       IN PLS_INTEGER DEFAULT 0,
    which_log         IN VARCHAR2 DEFAULT 'JOB_LOG',
    job_name          IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
log_history	log의 내용 중 해당 설정값 이상의 내용들을 삭제한다.
which_log	log를 삭제할 객체의 타입을 지정한다. 현재는 'JOB_LOG'만 지원한다.
job_name	log를 삭제할 객체의 이름을 지정한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.PURGE_LOG();
END;
/
```

36.2.16. RUN_JOB

일반적으로 JOB이 enabled 상태이면 스케줄러에 의해서 자동으로 수행되지만, 이와는 별개로 수동으로 JOB을 수행하기 위한 프러시저이다.

RUN_JOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.RUN_JOB
(
    name                    IN VARCHAR2,
    use_current_session     IN BOOLEAN DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
name	수행할 JOB 이름을 지정한다.
use_current_session	지원되지 않는 기능이므로 값을 무시한다. 항상 현재 세션으로 수행된다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.RUN_JOB('my_job');
END;
/
```

36.2.17. SET_ATTRIBUTE

job, program, chain 객체의 속성을 변경하기 위한 프러시저이다.

SET_ATTRIBUTE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.SET_ATTRIBUTE
(
    name           IN VARCHAR2,
    attribute      IN VARCHAR2,
    value          IN {BINARY_INTEGER|BOOLEAN|VARCHAR2}
);
```

- 파라미터

파라미터	설명
name	attribute를 변경할 객체의 이름을 지정한다.
attribute	변경할 attribute를 지정한다. 다음의 attribute들은 BINARY_INTEGER 속성 변경을 지원한다. – instance_id(instance_id 값은 JOB을 수행할 TAC 노드 인스턴스를 지정하며, 0인 경우 any_instance에 해당한다.) – job_priority – max_runs – max_failures – logging_level – schedule_limit 다음의 attribute들은 BOOLEAN 속성 변경을 지원한다. – auto_drop – restartable 다음의 attribute들은 VARCHAR2 속성 변경을 지원한다. – job_action – repeat_interval – program_action – auto_drop ('TRUE'와 'FALSE'만 지원) – restartable ('TRUE'와 'FALSE'만 지원)
value	변경할 값을 지정한다.

- 예제

```
BEGIN
    /* JOB을 수행할 TAC node instance_id를 0으로(any instance) 설정한다. */
    DBMS_SCHEDULER.SET_ATTRIBUTE('my_job', 'INSTANCE_ID', 0);
END;
/
```

36.2.18. SET_ATTRIBUTE_NULL

job, program, chain 객체의 속성을 NULL로 변경하기 위한 프러시저이다.

SET_ATTRIBUTE_NULL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.SET_ATTRIBUTE_NULL
(
    name           IN VARCHAR2,
    attribute      IN VARCHAR2
);
```

- 파라미터

파라미터	설명
name	attribute를 변경할 객체의 이름을 지정한다.
attribute	변경할 attribute를 지정한다. 다음의 attribute를 지원한다. – max_failures – max_runs – schedule_limit

- 예제

```
BEGIN
    /* MAX_FAILURES를 NULL로 변경한다. */
    DBMS_SCHEDULER.SET_ATTRIBUTE_NULL('my_job', 'MAX_FAILURES');
END;
/
```

36.2.19. STOP_JOB

데이터베이스에서 현재 실행 중인 JOB을 중지하는 프러시저이다.

STOP_JOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SCHEDULER.STOP_JOB
(
    job_name                IN VARCHAR2,
    force                   IN BOOLEAN DEFAULT FALSE,
);
```

- 파라미터

파라미터	설명
job_name	중지할 JOB의 이름을 지정한다.
force	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
BEGIN
    DBMS_SCHEDULER.STOP_JOB('my_job');
END;
/
```

제37장 DBMS_SESSION

본 장에서는 DBMS_SESSION 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

37.1. 개요

DBMS_SESSION 세션의 식별자를 설정하고, 시스템에 고유한 세션의 식별자를 가져오는데 사용되는 패키지이다.

37.2. 프러시저와 함수

본 절에서는 DBMS_SESSION 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

37.2.1. CLEAR_ALL_CONTEXT

문맥의 모든 속성값들을 지운다. 이 프러시저는 CREATE CONTEXT DDL 을 통해 지정된 패키지 내에서만 호출할 수 있다.

CLEAR_ALL_CONTEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.CLEAR_ALL_CONTEXT
(
    namespace IN VARCHAR2
)
```

- 파라미터

파라미터	설명
namespace	지울 문맥의 네임스페이스 이름을 지정한다.

37.2.2. CLEAR_CONTEXT

문맥의 속성값(들)을 지운다. 이 프러시저는 CREATE CONTEXT DDL 을 통해 지정된 패키지 내에서만 호출할 수 있다.

CLEAR_CONTEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.CLEAR_CONTEXT
(
    namespace IN VARCHAR2,
    client_id IN VARCHAR2 DEFAULT NULL,
    attribute IN VARCHAR2 DEFAULT NULL
)
```

- 파라미터

파라미터	설명
namespace	지울 문맥의 네임스페이스 이름을 지정한다.
client_id	ACCESSED GLOBALLY로 생성된 문맥일 경우에만 의미를 가진다. NULL을 명시하면 SET_CONTEXT를 호출할 때에 client_id에 NULL을 주고 지정한 속성을 지울 수 있다.
attribute	특정 속성을 지우고자 할 때 명시한다. 명시하지 않으면 해당 문맥의 (해당 client_id를 가지는) 모든 속성값이 지워진다.

- 예제

```
create or replace package body sec_pkg is
    procedure clear (name varchar2) as
    begin
        dbms_session.clear_context (name);
    end;
end;
/
```

37.2.3. CLEAR_IDENTIFIER

현재 세션의 IDENTIFIER를 NULL로 설정한다.

CLEAR_IDENTIFIER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.CLEAR_IDENTIFIER
```

- 예제

```
BEGIN
   DBMS_SESSION.CLEAR_IDENTIFIER;
END;
/
SELECT CLIENT_IDENTIFIER FROM V$SESSION;
```

37.2.4. LIST_CONTEXT

현재 세션에서 접근이 가능한 문맥들의 내용을 반환하는 함수이다.

LIST_CONTEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE AppCtxRecTyp IS RECORD (namespace VARCHAR2(30),
                             attribute VARCHAR2(30),
                             value     VARCHAR2(256));

TYPE AppCtxTabTyp IS TABLE OF AppCtxRecTyp INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT(list OUT AppCtxTabTyp,
                          size OUT NUMBER);
```

- 파라미터

파라미터	설명
list	문맥의 내용을 저장할 INDEX BY 테이블을 명시한다.
size	buffer에 반환되는 entry의 수이다.

37.2.5. SET_CONTEXT

이 프러시저는 CREATE CONTEXT DDL을 통해 지정된 패키지 내에서만 호출할 수 있다. 설정된 속성 값은 세션이 유지되는 동안 보존된다. 만일 해당 문맥의 속성이 이미 설정되어 있을 경우 SET_CONTEXT 호출은 해당 속성값을 덮어쓴다.

SET_CONTEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.SET_CONTEXT
(
```

```
    namespace IN VARCHAR2,
    attribute IN VARCHAR2,
    value     IN VARCHAR2,
    username  IN VARCHAR2 DEFAULT NULL,
    client_id IN VARCHAR2 DEFAULT NULL
)
```

- 파라미터

파라미터	설명
namespace	속성값을 설정할 문맥의 네임스페이스 이름이다.
attribute	해당 문맥에서의 속성 이름이다.
value	설정할 속성에 대한 속성값이다.
username	해당 속성에 접근할 수 있는 데이터베이스 사용자명이다. GLOBALLY ACCESSED인 문맥에 대한 속성을 설정할 때만 사용한다. 이 값을 설정하게 되면 해당 사용자만 이 속성에 접근할 수 있으며, 설정하지 않으면 모든 사용자가 속성에 접근할 수 있다. (기본값: NULL)
client_id	해당 속성에 대해 애플리케이션에서 사용하는 client_id를 명시할 수 있다. 대소문자를 구별하며 SET_IDENTIFIER 프러시저를 호출했을 때 명시했던 값과 일치해야 한다. GLOBALLY ACCESSED인 문맥에 대한 속성을 설정할 때만 사용한다. (기본값: NULL)

- 예제

```
create or replace package body sec_pkg is
    procedure set_attr (name varchar2, attr varchar2, val varchar2) as
    begin
        dbms_session.set_context (name, attr, val);
    end;
end;
/
```

37.2.6. SET_IDENTIFIER

세션의 CLIENT_IDENTIFIER를 설정한다. 설정된 값은 V$SESSION의 CLIENT_IDENTIFIER 컬럼 또는 SYS_CONTEXT ('USERENV', 'CLIENT_IDENTIFIER') 내장 함수를 호출해서 얻을 수 있다.

SET_IDENTIFIER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.SET_IDENTIFIER
(
    client_id        IN         VARCHAR2
)
```

- 파라미터

파라미터	설명
client_id	클라이언트 식별자이다. 64bytes 이상은 무시된다.

- 예제

```
DECLARE
   client_id VARCHAR2(64);
BEGIN
   client_id := 'MY_CLIENT_ID';
   DBMS_SESSION.SET_IDENTIFIER(client_id);
END;
/
SELECT CLIENT_IDENTIFIER FROM V$SESSION;
```

37.2.7. UNIQUE_SESSION_ID

시스템 유일의 세션 식별자를 반환한다. 세션 식별자 크기는 64bytes이다.

UNIQUE_SESSION_ID 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.UNIQUE_SESSION_ID RETURN VARCHAR2
```

- 예제

```
DECLARE
   u_sess_id VARCHAR2(64);
BEGIN
   u_sess_id := DBMS_SESSION.UNIQUE_SESSION_ID;
   DBMS_OUTPUT.PUT_LINE(u_sess_id);
END;
/
```

37.2.8. CLOSE_DATABASE_LINK

현재 열려 있는 DBLINK를 닫는다.

CLOSE_DATABASE_LINK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.CLOSE_DATABASE_LINK
(
    dblink        IN         VARCHAR2
)
```

- 파라미터

파라미터	설명
dblink	지울 dblink의 이름을 지정한다.

- 예제

```
DECLARE
BEGIN
   DBMS_SESSION.CLOSE_DATABASE_LINK('link1');
END;
/
```

37.2.9. IS_ROLE_ENABLED

이 함수를 호출한 세션에 롤이 적용되어 있는지 확인한다.

IS_ROLE_ENABLED 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.IS_ROLE_ENABLED
(
    rolename        IN         VARCHAR2
)
RETURN BOOLEAN
```

- 파라미터

파라미터	설명
rolename	롤의 적용 여부를 확인할 특권이다.

- 예제

```
DECLARE
   result BOOLEAN;
```

```
BEGIN
   result := DBMS_SESSION.IS_ROLE_ENABLED('CONNECT');
END;
/
```

37.2.10. SET_ROLE

이 프러시저를 호출한 세션에 롤을 변경한다. 이 프러시저는 'SET ROLE'로 시작하는 DDL을 수행하는 것과 동작이 동일하며, 입력값은 'SET ROLE' 다음에 오는 문장이다.

SET_ROLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SESSION.SET_ROLE
(
    rolecmd        IN         VARCHAR2
)
```

- 파라미터

파라미터	설명
rolecmd	부여할 역할을 지정한다.

- 예제

```
DECLARE
BEGIN
   DBMS_SESSION.SET_ROLE('ALL');
   DBMS_SESSION.SET_ROLE('ALL EXCEPT CONNECT');
END;
/
```

제38장 DBMS_SPACE

본 장에서는 DBMS_SPACE 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

38.1. 개요

DBMS_SPACE는 세그먼트의 크기와 공간 사용에 대한 정보를 제공하는 패키지이다.

38.2. 프러시저

본 절에서는 DBMS_SPACE 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

38.2.1. SPACE_USAGE

세그먼트의 High Water Mark(이하 HWM) 아래의 공간(즉, 현재 사용하는 공간)에 대한 사용 정보를 제공한다. 단, 비트맵 블록(bitmap block), 세그먼트 헤더(segment header), 익스텐트 맵 블록(extent map block)의 정보는 포함되지 않는다.

참고

High Water Mark란 테이블, 인덱스, 클러스터가 생성되고 한 번도 사용되지 않은 공간을 의미한다.

SPACE_USAGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SPACE.SPACE_USAGE
(
    segment_owner       IN  VARCHAR2,
    segment_name        IN  VARCHAR2,
    segment_type        IN  VARCHAR2,
    unformatted_blocks  OUT NUMBER,
    unformatted_bytes   OUT NUMBER,
    fs1_blocks          OUT NUMBER,
    fs1_bytes           OUT NUMBER,
    fs2_blocks          OUT NUMBER,
    fs2_bytes           OUT NUMBER,
    fs3_blocks          OUT NUMBER,
    fs3_bytes           OUT NUMBER,
```

```
    fs4_blocks          OUT NUMBER,
    fs4_bytes           OUT NUMBER,
    full_blocks         OUT NUMBER,
    full_bytes          OUT NUMBER,
    partition_name      IN  VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
segment_owner	스키마 객체를 소유한 사용자의 이름이다.
segment_name	스키마 객체의 이름이다.
segment_type	스키마 객체의 종류로, 다음 중 하나의 값을 갖는다. – TABLE – TABLE PARTITION – TABLE SUBPATITION – INDEX – INDEX PARTITION – INDEX SUBPATITION – CLUSTER – LOB – LOB PARTITION – LOB SUBPATITION
unformatted_blocks	세그먼트의 HWM 아래에서 포맷되지 않은 블록의 개수이다.
unformatted_bytes	세그먼트의 HWM 아래에서 포맷되지 않은 블록의 양(in bytes)이다.
fs1_blocks	0%와 25% 사이에 free space를 갖는 블록의 개수이다.
fs1_bytes	0%와 25% 사이에 free space를 갖는 블록의 양(in bytes)이다.
fs2_blocks	25%와 50% 사이에 free space를 갖는 블록의 개수이다.
fs2_bytes	25%와 50% 사이에 free space를 갖는 블록의 양(in bytes)이다.
fs3_blocks	50%와 75% 사이에 free space를 갖는 블록의 개수이다.
fs3_bytes	50%와 75% 사이에 free space를 갖는 블록의 양(in bytes)이다.
fs4_blocks	75%와 100% 사이에 free space를 갖는 블록의 개수이다.
fs4_bytes	75%와 100% 사이에 free space를 갖는 블록의 양(in bytes)이다.
full_blocks	full로 marking된 블록의 개수이다.

파라미터	설명
full_bytes	full로 marking된 블록의 양(in bytes)이다.
partition_name	다음의 경우에 따라 다르게 정의된다. – 파티션이면 파티션의 이름이다. – 파티션이 compose 방식이라면 서브파티션(subpartition)의 이름이다.

- 예제

```
set serveroutput on;

DECLARE
  unformatted_blocks NUMBER;
  unformatted_bytes  NUMBER;
  fs1_blocks NUMBER;
  fs1_bytes NUMBER;
  fs2_blocks NUMBER;
  fs2_bytes NUMBER;
  fs3_blocks NUMBER;
  fs3_bytes NUMBER;
  fs4_blocks NUMBER;
  fs4_bytes NUMBER;
  full_blocks NUMBER;
  full_bytes NUMBER;

BEGIN
  DBMS_SPACE.SPACE_USAGE('SYS', 'EMP', 'TABLE'
                         ,unformatted_blocks
                         ,unformatted_bytes
                         ,fs1_blocks
                         ,fs1_bytes
                         ,fs2_blocks
                         ,fs2_bytes
                         ,fs3_blocks
                         ,fs3_bytes
                         ,fs4_blocks
                         ,fs4_bytes
                         ,full_blocks
                         ,full_bytes);
  DBMS_OUTPUT.PUT_LINE('Space utilization (TABLES) ');
  DBMS_OUTPUT.PUT_LINE('unformatted_blocks: '|| TO_CHAR(unformatted_blocks) );
  DBMS_OUTPUT.PUT_LINE('unformatted_bytes: '|| TO_CHAR(unformatted_bytes) );
  DBMS_OUTPUT.PUT_LINE('fs1_blocks: '|| TO_CHAR(fs1_blocks) );
  DBMS_OUTPUT.PUT_LINE('fs1_bytes: '|| TO_CHAR(fs1_bytes) );
  DBMS_OUTPUT.PUT_LINE('fs2_blocks: '|| TO_CHAR(fs2_blocks) );
  DBMS_OUTPUT.PUT_LINE('fs2_bytes: '|| TO_CHAR(fs2_bytes) );
```

```
  DBMS_OUTPUT.PUT_LINE('fs3_blocks: '|| TO_CHAR(fs3_blocks) );
  DBMS_OUTPUT.PUT_LINE('fs3_bytes: '|| TO_CHAR(fs3_bytes) );
  DBMS_OUTPUT.PUT_LINE('fs4_blocks: '|| TO_CHAR(fs4_blocks) );
  DBMS_OUTPUT.PUT_LINE('fs4_bytes: '|| TO_CHAR(fs4_bytes) );
  DBMS_OUTPUT.PUT_LINE('full_blocks: '|| TO_CHAR(full_blocks) );
  DBMS_OUTPUT.PUT_LINE('full_bytes: '|| TO_CHAR(full_bytes) );
END;
```

38.2.2. UNUSED_SPACE

세그먼트의 HWM 위의 공간(즉, 현재 사용하지 않는 공간)에 대한 사용 정보를 제공한다.

UNUSED_SPACE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SPACE.UNUSED_SPACE
(
    segment_owner              IN  VARCHAR2,
    segment_name               IN  VARCHAR2,
    segment_type               IN  VARCHAR2,
    total_blocks               OUT NUMBER,
    total_bytes                OUT NUMBER,
    unused_blocks              OUT NUMBER,
    unused_bytes               OUT NUMBER,
    last_used_extent_file_id   OUT NUMBER,
    last_used_extent_block_id  OUT NUMBER,
    last_used_block            OUT NUMBER,
    partition_name             IN  VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
segment_owner	스키마 객체를 소유한 사용자의 이름이다.
segment_name	스키마 객체의 이름이다.
segment_type	스키마 객체의 종류로, 다음 중 하나의 값을 갖는다. – TABLE – TABLE PARTITION – TABLE SUBPATITION – INDEX

파라미터	설명
	– INDEX PARTITION – INDEX SUBPATITION – CLUSTER – LOB – LOB PARTITION – LOB SUBPATITION
total_blocks	세그먼트의 전체 블록의 개수이다.
total_bytes	세그먼트의 전체 블록의 양(in bytes)이다.
unused_blocks	세그먼트의 HWM 위의 블록의 개수이다.
unused_bytes	세그먼트의 HWM 위의 블록의 양(in bytes)이다.
last_used_extent_file_id	맨 마지막에 INSERT를 위해 사용된 익스텐트가 속한 파일 ID이다.
last_used_ex tent_block_id	맨 마지막에 INSERT를 위해 사용된 익스텐트가 속한 시작 블록(starting block)의 ID이다. 파일 ID와 같이 합하면 DBA가 된다.
last_used_extent_block	맨 마지막에 INSERT를 위해 사용된 블록의 숫자(익스텐트 내에서의 오프셋)이다.
partition_name	다음의 경우에 따라 다르게 정의된다. – 파티션이면 파티션의 이름이다. – 파티션이 compose 방식이라면 서브파티션(subpartition)의 이름이다.

- 예제

```
set serveroutput on;

DECLARE
  total_blocks   NUMBER;
  total_bytes    NUMBER;
  unused_blocks  NUMBER;
  unused_bytes   NUMBER;
  last_used_extent_file_id   NUMBER;
  last_used_extent_block_id  NUMBER;
  last_used_block            NUMBER;

BEGIN
  DBMS_SPACE.UNUSED_SPACE('SYS', 'EMP', 'TABLE'
                          ,total_blocks
                          ,total_bytes
                          ,unused_blocks
```

```
                          ,unused_bytes
                          ,last_used_extent_file_id
                          ,last_used_extent_block_id
                          ,last_used_block);

  DBMS_OUTPUT.PUT_LINE('Unused space utilization (TABLES) ');
  DBMS_OUTPUT.PUT_LINE('total_blocks: '|| TO_CHAR(total_blocks) );
  DBMS_OUTPUT.PUT_LINE('total_bytes: '|| TO_CHAR(total_bytes) );
  DBMS_OUTPUT.PUT_LINE('unused_blocks: '|| TO_CHAR(unused_blocks) );
  DBMS_OUTPUT.PUT_LINE('unused_bytes: '|| TO_CHAR(unused_bytes) );
  DBMS_OUTPUT.PUT_LINE('last_used_extent_file_id: '||
                       TO_CHAR(last_used_extent_file_id) );
  DBMS_OUTPUT.PUT_LINE('last_used_extent_block_id: '||
                       TO_CHAR(last_used_extent_block_id) );
  DBMS_OUTPUT.PUT_LINE('last_used_block: '|| TO_CHAR(last_used_block) );
END;
```

제39장 DBMS_SPACE_ADMIN

본 장에서는 DBMS_SPACE_ADMIN 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

39.1. 개요

DBMS_SPACE_ADMIN은 세그먼트를 관리하는 기능을 제공하는 패키지이다.

39.2. 프러시저

본 절에서는 DBMS_SPACE_ADMIN 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

다음은 DBMS_SPACE_ADMIN 패키지에 정의할 수 있는 상수이다.

- SEGMENT_DUMP_EXTENT_MAP
  - 익스텐트 맵의 정보를 같이 덤프(dump)한다.
  - 타입 : POSITIVE
  - 값 : 5
- SEGMENTS_DUMP_BITMAP_SUMMARY
  - 비트맵 정보만 덤프한다.
  - 타입 : POSITIVE
  - 값 : 27

39.2.1. SEGMENT_DUMP

세그먼트 헤더와 비트맵 블록의 정보를 덤프한다. 덤프 정보는 USER_DUMP_DEST 파라미터에 정의된 경로에 저장되며, ALTER SYSTEM DUMP DEST 문으로 확인할 수 있다.

SEGMENT_DUMP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SPACE_ADMIN.SEGMENT_DUMP
(
    tablespace_name       IN VARCHAR2,
    header_relative_file  IN POSITIVE,
    header_block          IN POSITIVE,
    dump_option           IN POSITIVE DEFAULT SEGMENT_DUMP_EXTENT_MAP
);
```

- 파라미터

파라미터	설명
tablespace_name	세그먼트가 있는 테이블 스페이스의 이름이다.
header_relative_file	세그먼트 헤더가 속한 파일의 Relative file number이다.
header_block	세그먼트 헤더 블록(segment header block)의 블록 숫자(block number)이다.
dump_option	다음 중 하나의 옵션을 갖는다. – SEGMENT_DUMP_EXTENT_MAP – SEGMENT_DUMP_BITMAP_SUMMARY

- 예제

```
BEGIN
  DBMS_SPACE_ADMIN.SEGMENT_DUMP('TS', 3, 9,
              DBMS_SPACE_ADMIN.SEGMENT_DUMP_EXTENT_MAP);
END;
```

제40장 DBMS_SPH

본 장에서는 DBMS_SPH 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

40.1. 개요

DBMS_SPH 패키지는 SQL Plan History(이하 SPH) 기능을 사용하는데 필요로 하는 기능을 제공하기 위한 패키지이다. SPH는 서버가 실행한 SQL의 이력을 저장 및 관리해주는 기능이다.

Tibero는 SQL로부터 실행계획을 만들어 실행한다. 이 실행계획은 V$SQL_PLAN 등의 dynamic view를 통해 조회할 수 있다. 그러나 실행계획이 오래동안 사용되지 않거나 서버가 재부팅되면 그 실행계획은 메모리에서 삭제된다. 이 때문에 오랜 시간에 걸친 실행계획의 변화를 감지하고 추적하기가 어렵다.

SPH는 실행계획을 별도의 테이블에 저장함으로써 시간에 따른 실행계획 변화를 사용자가 쉽게 관리할 수 있게 도와준다. 단, 이 패키지는 SYS 사용자만 사용 가능하다.

40.2. 프러시저

본 절에서는 DBMS_SPH 패키지에서 제공하는 프러시저를 설명한다.

40.2.1. REPORT_PLANS

지정된 SQL의 실행계획 변화 이력을 출력한다. REPORT_PLANS_BY_DATE와 동작이 같으나 인자의 타입만 다르다. 결과는 화면에 출력할 수 있고 파일로 저장할 수도 있다. 화면에 출력을 원할 때(인자 TO_FILE이 FALSE일 때)는 serveroutput을 켜고 실행해야 한다.

다음은 결과가 파일로 저장되는 경로이다.

```
$TB_HOME/instance/$TB_SID/sph_report.{mthr_pid}.{current_time}
```

REPORT_PLANS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_PLANS
(
    SQL_HASH_VALUE  IN NUMBER,
    DURATION          IN PLS_INTEGER    DEFAULT 24*365*1000,
    TO_FILE           IN BOOLEAN        DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
SQL_HASH_VALUE	리포트할 SQL의 HASH VALUE이다.
DURATION	SQL의 이력을 출력할 기간이다. 인자는 시작 시점을 의미하며 종료 시점은 항상 현재이다. – 기준 : 실행계획의 마지막 실행시간 – 단위 : 시간(hour) – 범위 : 0~24*365*1000 – 기본값 : 24*365*1000시간(약 1000년)
TO_FILE	리포트 내용을 파일로 저장할지를 설정한다. – TRUE : 내용이 파일에 저장된다. (기본값) – FALSE : 내용이 화면에 출력된다.

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> set serveroutput on

SQL> select hash_value from v$sqlarea where sql_text like 'select hash_value%';

HASH_VALUE
----------
3870695416

1 row selected.

SQL> exec dbms_sph.report_plans(3870695416, 24, FALSE);
            -- 화면에 하루 동안의 실행계획을 모두 출력

                         +--------------------------+
                         | SQL_PLAN_HISTORY REPORT |
                         +--------------------------+

FROM: 1997/05/29 17:29:41
TO:   1997/05/30 17:29:41
```

```
... 생략 ...
```

40.2.2. REPORT_PLANS_BY_DATE

지정된 SQL의 실행계획 변화 이력을 출력한다. REPORT_PLANS와 동작이 같으나 인자의 타입만 다르다. 결과는 화면에 출력할 수 있고 파일로 저장할 수도 있다. 화면에 출력을 원할 때(인자 TO_FILE이 FALSE일 때)는 serveroutput을 켜고 실행해야 한다.

다음은 결과가 파일로 저장되는 경로이다.

```
$TB_HOME/instance/$TB_SID/sph_report.{mthr_pid}.{current_time}
```

REPORT_PLANS_BY_DATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_PLANS_BY_DATE
(
    SQL_HASH_VALUE  IN NUMBER,
    START_DATE        IN DATE       DEFAULT SYSDATE - 365*1000,
    TO_FILE           IN BOOLEAN    DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
SQL_HASH_VALUE	리포트할 SQL의 HASH VALUE이다.
START_DATE	SQL의 이력을 출력할 기간의 시작 시점이다. 종료 시점은 항상 현재이다. – 기준 : 실행계획의 마지막 실행 시간 – 단위 : 일(day) – 범위 : SYSDATE - 365*1000 ~ SYSDATE – 기본값 : SYSDATE - 365*1000(SYSDATE로부터 약 1000년)
TO_FILE	리포트 내용을 파일로 저장할지를 설정한다. – TRUE : 내용이 파일에 저장된다. (기본값) – FALSE : 내용이 화면에 출력된다.

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> set serveroutput on

SQL> select hash_value from v$sqlarea where sql_text like 'select hash_value%';

HASH_VALUE
----------
3870695416

1 row selected.

SQL> exec dbms_sph.report_plans_by_date(3870695416, sysdate - 1, FALSE);
            -- 화면에 하루 동안의 실행계획을 모두 출력

                          +-------------------------+
                          | SQL_PLAN_HISTORY REPORT |
                          +-------------------------+

FROM: 1997/05/29 17:49:11
TO:   1997/05/30 17:49:11

... 생략 ...
```

40.2.3. REPORT_PLAN_HISTORY

조건을 만족하는 모든 SQL의 실행계획 변화 이력을 출력한다. REPORT_PLAN_HISTORY_BY_DATE와 동작이 같으나 인자의 타입만 다르다. 결과는 화면에 출력할 수 있고 파일로 저장할 수도 있다. 화면에 출력을 원할 때(인자 TO_FILE이 FALSE일 때)는 serveroutput을 켜고 실행해야 한다.

다음은 결과가 파일로 저장되는 경로이다.

```
$TB_HOME/instance/$TB_SID/sph_report.{mthr_pid}.{current_time}
```

REPORT_PLAN_HISTORY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_PLAN_HISTORY
(
    DURATION          IN PLS_INTEGER  DEFAULT 24*365*1000,
    MIN_PLAN_COUNT    IN PLS_INTEGER  DEFAULT 1,
```

```
    TO_FILE           IN BOOLEAN      DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
DURATION	SQL의 이력을 출력할 기간이다. 인자는 시작 시점을 의미하며 종료 시점은 항상 현재이다. – 기준 : 실행계획의 마지막 실행 시간 – 단위 : 시간(hour) – 범위 : 0~24*365*1000 – 기본값 : 24*365*1000시간(약 1000년)
MIN_PLAN_COUNT	출력할 SQL의 최소 실행계획 수이다. 예를 들어 2로 지정하면 둘 이상의 실행계획을 가지는 SQL만 출력한다. – 범위 : 1 ~ 1000000 – 기본값 : 1
TO_FILE	리포트 내용을 파일로 저장할지를 설정한다. – TRUE : 내용이 파일에 저장된다. (기본값) – FALSE : 내용이 화면에 출력된다.

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> set serveroutput on

SQL> exec dbms_sph.report_plan_history(24, 1, FALSE);
                -- 화면에 하루 동안 실행된 모든 SQL의 실행계획을 모두 출력

                        +-------------------------+
                        | SQL_PLAN_HISTORY REPORT |
                        +-------------------------+

FROM: 1997/05/29 17:49:11
TO:   1997/05/30 17:49:11
```

```
... 생략 ...
```

40.2.4. REPORT_PLAN_HISTORY_BY_DATE

조건을 만족하는 모든 SQL의 실행계획 변화 이력을 출력한다. REPORT_PLAN_HISTORY와 동작이 같으나 인자의 타입만 다르다. 결과는 화면에 출력할 수 있고 파일로 저장할 수도 있다. 화면에 출력을 원할 때(인자 TO_FILE이 FALSE일 때)는 serveroutput을 켜고 실행해야 한다.

다음은 결과가 파일로 저장되는 경로이다.

```
$TB_HOME/instance/$TB_SID/sph_report.{mthr_pid}.{current_time}
```

REPORT_PLAN_HISTORY_BY_DATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_PLAN_HISTORY_BY_DATE
(
    START_DATE        IN DATE         DEFAULT SYSDATE - 365*1000,
    MIN_PLAN_COUNT    IN PLS_INTEGER  DEFAULT 1,
    TO_FILE           IN BOOLEAN      DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
START_DATE	SQL의 이력을 출력할 기간의 시작 시점이다. 종료 시점은 항상 현재이다. – 기준 : 실행계획의 마지막 실행 시간 – 단위 : 일(day) – 범위 : SYSDATE - 365*1000 ~ SYSDATE – 기본값 : SYSDATE - 365*1000(SYSDATE로부터 약 1000년)
MIN_PLAN_COUNT	출력할 SQL의 최소 실행계획 수이다. 예를 들어 2로 지정하면 둘 이상의 실행계획을 가지는 SQL만 출력한다. – 범위 : 1 ~ 1000000 – 기본값 : 1
TO_FILE	리포트 내용을 파일로 저장할지를 설정한다. – TRUE : 내용이 파일에 저장된다. (기본값)

파라미터	설명
	– FALSE : 내용이 화면에 출력된다.

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> set serveroutput on

SQL> exec dbms_sph.report_plan_history_by_date(sysdate - 1, 1, FALSE);
                -- 화면에 하루 동안 실행된 모든 SQL의 실행계획을 모두 출력

                        +-------------------------+
                        | SQL_PLAN_HISTORY REPORT |
                        +-------------------------+

FROM: 1997/05/29 17:49:11
TO:   1997/05/30 17:49:11

... 생략 ...
```

40.2.5. TRUNCATE_PLAN_HISTORY

SPH에 저장된 실행계획 이력을 삭제한다. TRUNCATE_PLAN_HISTORY_BY_DATE와 동작은 같으며 인자만 다르다.

TRUNCATE_PLAN_HISTORY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE TRUNCATE_PLAN_HISTORY
(
    RETENTION_PERIOD  IN PLS_INTEGER  DEFAULT 24*365*1000
    MAX_COUNT         IN PLS_INTEGER  DEFAULT 1000000
);
```

- 파라미터

파라미터	설명
RETENTION_PERIOD	SQL 이력을 삭제할 기간을 지정한다. 이 기간동안 실행되지 않은 실행계획은 모두 삭제된다.

파라미터	설명
	– 단위 : 시간(hour) – 범위 : 0 ~ 24*365*1000 – 기본값 : 24*365*1000(약 1000년)
MAX_COUNT	실행계획을 삭제하지 않고 유지할 SQL의 수이다. RETENTION_PERIOD와 MAX_COUNT를 모두 지정하면 두 조건을 모두 만족시키는 SQL만이 유지된다. – 범위 : 0 ~ 1000000 – 기본값 : 1000000

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> exec dbms_sph.truncate_plan_history(24);
            -- 하루 동안 실행되지 않은 실행계획을 모두 삭제

PSM completed.
```

40.2.6. TRUNCATE_PLAN_HISTORY_BY_DATE

SPH에 저장된 실행계획 이력을 삭제한다. TRUNCATE_PLAN_HISTORY와 동작은 같으며 인자만 다르다.

TRUNCATE_PLAN_HISTORY_BY_DATE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE TRUNCATE_PLAN_HISTORY_BY_DATE
(
    START_DATE        IN DATE         DEFAULT SYSDATE - 365*1000
    MAX_COUNT         IN PLS_INTEGER  DEFAULT 1000000
);
```

- 파라미터

파라미터	설명
START_DATE	SQL 이력을 삭제할 기준 시점이다. 이 시점 이전의 모든 실행계획 정보가 삭제된다. – 단위 : 시간(hour) – 범위 : SYSDATE - 365*1000 ~ SYSDATE – 기본값 : SYSDATE - 365*1000(SYSDATE로부터 약 1000년)
MAX_COUNT	실행계획을 삭제하지 않고 유지할 SQL의 수이다. START_DATE와 MAX_COUNT를 모두 지정하면 두 조건을 모두 만족시키는 SQL만이 유지된다. – 범위 : 0 ~ 1000000 – 기본값 : 1000000

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> exec dbms_sph.truncate_plan_history_by_date(sysdate - 1);
            -- 하루 동안 실행되지 않은 실행계획을 모두 삭제

PSM completed.
```

40.2.7. UPDATE_PLAN_HISTORY

라이브러리 cache의 실행계획을 SPH에 저장한다.

UPDATE_PLAN_HISTORY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE UPDATE_PLAN_HISTORY
(
    MIN_EXEC_COUNT    IN PLS_INTEGER  DEFAULT 1
);
```

- 파라미터

파라미터	설명
MIN_EXEC_COUNT	업데이트 대상이 될 실행계획의 최소 실행 회수이다. 이 값 미만으로 실행된 실행계획은 SPH에 저장되지 않는다. – 범위 : 1 ~ 1000000 – 기본값 : 1

- 예외 상황

예외 상황	설명
PARAMETER_OUT_OF_RANGE	파라미터가 지정된 범위를 벗어난 경우이다.

- 예제

```
SQL> exec dbms_sph.update_plan_history(2); -- 두 번 이상 실행된 실행계획을 SPH에 저장

PSM completed.
```

제41장 DBMS_SPM

본 장에서는 DBMS_SPM 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

41.1. 개요

현재 **DBMS_SPM** 패키지는 SQL_PLAN_BASELIINE 정보를 수정하는 ALTER_SQL_PLAN_BASELINE, 지우는 DROP_SQL_PLAN_BASELINE, 검증하는 EVOLVE_SQL_PLAN_BASELINE을 제공하는 패키지이다.

41.2. 프러시저와 함수

본 절에서는 DBMS_SPM 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

41.2.1. ALTER_SQL_PLAN_BASELINE

기존 SQL PLAN_BASELINE의 특정 속성을 변경한다.

ALTER_SQL_PLAN_BASELINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SPM.ALTER_SQL_PLAN_BASELINE
(
    sql_handle       IN VARCHAR2 := NULL,
    plan_name        IN VARCHAR2 := NULL,
    attribute_name   IN VARCHAR2,
    attribute_value  IN VARCHAR2
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
sql_handle	변경할 대상의 unique한 SQL 질의문 handle이다. 만약 비어있다면 plan_name이 특정되어야 한다.

파라미터	설명
plan_name	변경할 대상의 plan name이다. sql_handle이 지정되어 있고, plan_name이 비어있다면 sql_handle에 해당하는 모든 plan에 대해서 변경을 적용한다. 만약 비어있다면 sql_handle이 특정되어야 한다.
attribute_name	변경할 특성의 이름이다.
attribute_value	변경할 특성의 값이다. – ENABLED : 사용 가능한 상태를 나타내는 값이다. 'YES' 또는 'NO'로 설정할 수 있다. – FIXED : 고정 여부를 나타내는 값이다. 고정된 SQL_PLAN_BASELIINE은 evolve 되지 않고 사용될 때 우선순위를 갖는다. 'YES' 또는 'NO'로 설정할 수 있다. – PLAN_NAME : 플랜의 이름으로 30자 이내의 문자열이다. – DESCRIPTION : attribute에 대한 설명으로 500자 이내의 문자열이다.

- 반환값

반환값	설명
NUMBER 데이터	변경된 플랜의 수를 반환한다.

- 예제

  – ALTER 전

```
SQL> select signature, sql_handle, plan_name, sql_text, description
from dba_sql_plan_baselines;
SIGNATURE SQL_HANDLE
---------- ------------------------------
PLAN_NAME
----------------------------------------
SQL_TEXT
----------------------------------------
DESCRIPTION
----------------------------------------
2.8525E+18 SYS_BASELINE_d8626e27730e74e4
SQL_PLAN_339798362
select * from t where c1 = 4 and c2 = 4
```

  – ALTER 구문

```
BEGIN :a := DBMS_SPM.ALTER_SQL_PLAN_BASELINE(
   SQL_HANDLE => 'SYS_BASELINE_d8626e27730e74e4',
   PLAN_NAME => 'SQL_PLAN_339798362',
   ATTRIBUTE_NAME => 'DESCRIPTION',
```

```
   ATTRIBUTE_VALUE => 'TEST');
END;
/
```

– ALTER 후

```
SQL> select signature, sql_handle, plan_name, sql_text, description
from dba_sql_plan_baselines;
SIGNATURE SQL_HANDLE
---------- ------------------------------
PLAN_NAME
----------------------------------------
SQL_TEXT
----------------------------------------
DESCRIPTION
----------------------------------------
2.8525E+18 SYS_BASELINE_d8626e27730e74e4
SQL_PLAN_339798362
select * from t where c1 = 4 and c2 = 4
TEST
```

41.2.2. DROP_SQL_PLAN_BASELINE

특정 SQL PLAN_BASELINE을 데이터베이스에서 삭제한다.

DROP_SQL_PLAN_BASELINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SPM.DROP_SQL_PLAN_BASELINE
(
    sql_handle   IN VARCHAR2 := NULL,
    plan_anme    IN VARCHAR2 := NULL
    )
    RETURN NUMBER;
```

- 파라미터

파라미터	설명
sql_handle	변경할 대상의 unique한 SQL 질의문 handle이다. 만약 비어있다면 plan_name이 특정되어야 한다.
plan_name	변경할 대상의 plan name이다. sql_handle이 지정되어 있고, plan_name이 비어있다면 sql_handle에 해당하는 모든 plan에 대해서 변경을 적용한다. 만약 비어있다면 sql_handle이 특정되어야 한다.

- 예제

  – DROP 전

```
SQL> select signature, sql_handle, plan_name, sql_text, description
from dba_sql_plan_baselines;
 SIGNATURE SQL_HANDLE
---------- ------------------------------
PLAN_NAME
----------------------------------------
SQL_TEXT
----------------------------------------
DESCRIPTION
----------------------------------------
2.8525E+18 SYS_BASELINE_d8626e27730e74e4
SQL_PLAN_339798362
select * from t where c1 = 4 and c2 = 4
```

  – DROP 구문

```
BEGIN :a := DBMS_SPM.DROP_SQL_PLAN_BASELINE(
    SQL_HANDLE => 'SYS_BASELINE_d8626e27730e74e4',
    PLAN_NAME => 'SQL_PLAN_339798362');
END;

/
```

  – DROP 후

```
SQL>select signature, sql_handle, plan_name, sql_text, description
from dba_sql_plan_baselines;
0 row selected.
```

41.2.3. EVOLVE_SQL_PLAN_BASELINE

accept 되지 않은 SQL PLAN_BASELINE들을 검증하고 accept로 변경한다.

EVOLVE_SQL_PLAN_BASELINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE
(
    plan_list    IN DBMS_SPM.NAME_LIST,
    time_limit   IN INTEGER  := 0,
    verify       IN VARCHAR2 := 'YES',
```

```
    do_commit    IN VARCHAR2 := 'YES')
)
RETURN CLOB;

DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE
(
    sql_handle   IN VARCHAR2 := NULL,
    plan_name    IN VARCHAR2 := NULL,
    time_limit   IN number := 0,
    verify       IN VARCHAR2 := 'YES',
    do_commit    IN VARCHAR2 := 'YES'
)
RETURN CLOB;
```

- 파라미터

파라미터	설명
sql_handle	변경할 대상의 unique한 SQL 질의문 handle이다. 만약 비어있고 plan_name이 특정되지 않으면 모든 unaccepted plan을 검증한다.
plan_name	변경할 대상의 plan name이다.
plan_list	변경할 대상의 plan name 목록이다.
time_limit	검증하는 시간을 제한하는 값이다. 현재는 미지원 파라미터이다.
verify	검증할 플랜을 실제로 수행할 지 결정하는 파라미터이다. 'YES' 또는 'NO'로 설정할 수 있고, 'NO'인 경우 검증하지 않고 unaccepted plan을 accepted plan으로 변경한다.
do_commit	accepted plan보다 빠른 것으로 검증된 unaccepted plan을 실제로 accepted plan으로 변경할 지 결정하는 파라미터이다. 'YES' 또는 'NO'로 설정할 수 있고, 'NO'인 경우 보고서를 리턴한다.

제42장 DBMS_SQL

본 장에서는 DBMS_SQL 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

42.1. 개요

DBMS_SQL은 PSM으로 데이터 조작어(DML: Data Manipulation Language, 이하 DML), 데이터 정의어(DDL: Data Definition Language) 등 Dynamic SQL 문장을 사용하기 위한 패키지이다.

Dynamic SQL 문장은 PSM의 소스 안에 SQL 문장을 직접 사용하지 않고 나중에 PSM을 실행할 때, 문자열로 전달하여 파싱되는 SQL 문장을 말한다. 따라서 PSM을 생성할 때 존재하지 않았던 테이블에 대해 SELECT를 실행할 수도 있고, CREATE TABLE, DROP TABLE 등의 DDL 문장도 수행할 수 있다.

DBMS_SQL 패키지로 Dynamic SQL 문장을 실행하는 과정은 다음과 같다.

1. OPEN_CURSOR 함수를 이용하여 커서를 열고, 커서 ID를 얻는다.
2. 커서를 구분하기 위해 앞에서 얻은 커서 ID를 다른 함수의 파라미터로 전달한다.
3. Dynamic SQL 문장을 파싱하기 위해 PARSE 함수를 호출한다. 만약 실행할 Dynamic SQL 문장 안에 바인드해야 할 변수가 있는 경우 BIND_VARIABLE 함수를 호출해 변수를 바인드한다.
4. 실행 결과를 가져오기 전에 결과 컬럼의 타입을 정의하기 위해 DEFINE_COLUMN 함수를 호출한다.
5. EXECUTE 함수를 이용하여 파싱된 SQL 문장을 실행한다.
6. 실행한 결과를 fetch하기 위해 FETCH_ROWS 함수를 실행한다. fetch한 결과를 원하는 변수로 가져오기 위해서는 COLUMN_VALUE 함수를 호출한다.
7. 모든 수행을 끝낸 후 커서를 종료하기 위해 CLOSE_CURSOR 함수를 실행한다.

42.2. 타입

본 절에서는 DBMS_SQL 패키지에 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

42.2.1. DESC_REC

DESCRIBE_COLUMNS 프러시저를 통해서 컬럼 정보들을 가져올 때 사용되는 DESC_TAB 타입의 구성요소들의 타입이다.

DESC_REC 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DESC_REC IS RECORD
(
    col_type             binary_integer,
    col_max_len          binary_integer,
    col_name             varchar2(32)  ,
    col_name_len         binary_integer,
    col_schema_name      varchar2(32)  ,
    col_schema_name_len  binary_integer,
    col_precision        binary_integer,
    col_scale            binary_integer,
    col_charsetid        binary_integer,
    col_charsetform      binary_integer,
    col_null_ok          boolean
);
```

42.2.2. DESC_REC2

DESCRIBE_COLUMNS2 프러시저를 통해서 컬럼 정보들을 가져올 때 사용되는 DESC_TAB2 타입의 구성 요소들의 타입이다.

DESC_REC2 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DESC_REC2 IS RECORD
(
    col_type             binary_integer,
    col_max_len          binary_integer,
    col_name             varchar2(32767),
    col_name_len         binary_integer,
    col_schema_name      varchar2(32),
    col_schema_name_len  binary_integer,
    col_precision        binary_integer,
    col_scale            binary_integer,
    col_charsetid        binary_integer,
    col_charsetform      binary_integer,
    col_null_ok          boolean);
);
```

42.2.3. DESC_TAB

DESCRIBE_COLUMNS 프러시저를 통해서 컬럼 정보들을 가져올 때 사용되는 배열 타입이다.

DESC_TAB 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DESC_TAB IS TABLE OF DESC_REC INDEX BY BINARY_INTEGER;
```

42.2.4. DESC_TAB2

DESCRIBE_COLUMNS2 프러시저를 통해서 컬럼 정보들을 가져올 때 사용되는 배열 타입이다.

DESC_TAB2 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DESC_TAB2 IS TABLE OF DESC_REC2 INDEX BY BINARY_INTEGER;
```

42.2.5. VARCHAR2A

VARCHAR2 타입 데이터로 구성된 테이블 타입이다. DBMS_SQL.PARSE 프러시저에서 긴 SQL문장을 넣을때 사용된다.

VARCHAR2A 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE VARCHAR2A IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

42.2.6. DATE_TABLE

DATE 타입 데이터로 구성된 테이블 타입이다.

DATE_TABLE 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DATE_TABLE IS TABLE OF DATE INDEX BY BINARY_INTEGER;
```

42.2.7. NUMBER_TABLE

NUMBER 타입 데이터로 구성된 테이블 타입이다.

NUMBER_TABLE 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE NUMBER_TABLE IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

42.2.8. VARCHAR2_TABLE

VARCHAR2 타입 데이터로 구성된 테이블 타입이다.

VARCHAR2_TABLE 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

42.3. 프러시저

본 절에서는 DBMS_SQL 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

42.3.1. BIND_ARRAY

주어진 커서의 SQL 문장에서 콜론(:)으로 시작하는 변수에 값들의 집합을 설정해주는 프러시저이다. 변수의 값이 모두 설정되지 않은 상태로 EXECUTE 함수를 호출하면 예외가 발생한다. 변수는 이름으로 찾을 수 있으므로, SQL 문장 내의 변수이름과 파라미터로 주어진 이름을 동일하게 호출해야 한다.

BIND_ARRAY 프러시저는 테이블에 동시에 여러 로우를 삽입, 삭제, 변경하는 용도로 활용된다.

BIND_ARRAY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

  BIND_ARRAY 프러시저는 설정할 변수의 타입에 따라 중복으로 선언(overloading)되어 있다.

  해당되는 타입으로는 NUMBER, VARCHAR2, DATE의 인덱스 기반 테이블 타입이 있다.

  – NUMBER_TABLE

```
PROCEDURE BIND_ARRAY
(
    c        IN     INTEGER,
    name     IN     VARCHAR2,
    n_tab    IN     NUMBER_TABLE [,
             index1   IN     PLS_INTEGER,
             index2   IN     PLS_INTEGER]
);
```

– VARCHAR2_TABLE

```
PROCEDURE BIND_ARRAY
(
    c        IN     INTEGER,
    name     IN     VARCHAR2,
    n_tab    IN     VARCHAR2_TABLE [,
             index1   IN     PLS_INTEGER,
             index2   IN     PLS_INTEGER]
);
```

– DATE_TABLE

```
PROCEDURE BIND_ARRAY
(
     c        IN     INTEGER,
     name     IN     VARCHAR2,
     n_tab    IN     DATE_TABLE [,
              index1   IN     PLS_INTEGER,
              index2   IN     PLS_INTEGER]
);
```

- 파라미터

파라미터	설명
c	대상 커서이다.
name	SQL 문장 내의 변수의 이름이다. 이름에서 첫 문자 콜론(:)은 생략해도 된다.
n_tab	SQL 문장 내의 변수에 설정할 값들의 집합 변수이다.
index1	이 파라미터를 사용할 경우 인덱스 기반 테이블 내 바인딩할 영역의 시작점을 지정한다. (≥ 1)
index2	이 파라미터를 사용할 경우 인덱스 기반 테이블 내 바인딩할 영역의 끝점을 지정한다. (≥ index1)

- 예제

```
DECLARE
    stmt VARCHAR2(200);
```

```
    deptno_array DBMS_SQL.NUMBER_TABLE;
    name_array DBMS_SQL.VARCHAR2_TABLE;
    c NUMBER;
    dummy NUMBER;
begin
    deptno_array(1) := 10;
    deptno_array(2) := 20;
    deptno_array(3) := 30;
    deptno_array(4) := 40;
    deptno_array(5) := 50;
    deptno_array(6) := 60;
    name_array(1) := 'Architecture';
    name_array(2) := 'Frontend';
    name_array(3) := 'Core';
    name_array(4) := 'Management';
    name_array(5) := 'QMS';
    name_array(6) := 'Technical Support';

    stmt := 'insert into dept values(:deptno, :name)';
    c := DBMS_SQL.OPEN_CURSOR;

    DBMS_SQL.PARSE(c, stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_ARRAY(c, ':deptno', deptno_array);
    DBMS_SQL.BIND_ARRAY(c, ':name', name_array, 2, 5);
    dummy := DBMS_SQL.EXECUTE(c);

    DBMS_SQL.CLOSE_CURSOR(c);
END;
/
```

42.3.2. BIND_VARIABLE

주어진 커서의 SQL 문장에서 콜론(:)으로 시작하는 변수에 값을 설정해주는 프러시저이다. 변수의 값이 모두 설정되지 않은 상태로 EXECUTE 함수를 호출하면 예외가 발생한다. 변수는 이름으로 찾을 수 있으므로, SQL 문장 내의 변수이름과 파라미터로 주어진 이름을 동일하게 호출해야 한다.

BIND_VARIABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

  BIND_VARIABLE 프러시저는 설정할 변수의 타입에 따라 중복으로 선언(overloading)되어 있다.

  해당되는 타입으로는 NUMBER, VARCHAR2, DATE, TIMESTAMP, INTERVAL, BINARY_FLOAT, BINARY_DOUBLE, CLOB, BLOB이 있다.

– NUMBER

```
PROCEDURE BIND_VARIABLE
(
    c      IN   INTEGER,
    name   IN   VARCHAR2,
    value  IN   NUMBER
);
```

– VARCHAR2

```
PROCEDURE BIND_VARIABLE
(
    c      IN   INTEGER,
    name   IN   VARCHAR2,
    value  IN   VARCHAR2[, OUT_VALUE_SIZE IN INTEGER]
);
```

– DATE

```
PROCEDURE BIND_VARIABLE
(
    c      IN   INTEGER,
    name   IN   VARCHAR2,
    value  IN   DATE
);
```

– TIMESTAMP

```
PROCEDURE BIND_VARIABLE
(
    c       IN   INTEGER,
    name    IN   VARCHAR2,
    value   IN   TIMESTAMP_UNCONSTRAINED
);
```

– INTERVAL

```
PROCEDURE BIND_VARIABLE
(
    C       IN   INTEGER,
    NAME    IN   VARCHAR2,
    VALUE   IN   YMINTERVAL_UNCONSTRAINED
);
```

```
PROCEDURE BIND_VARIABLE
(
    c       IN   INTEGER,
    name    IN   VARCHAR2,
```

```
    value   IN   DSINTERVAL_UNCONSTRAINED
);
```

– BINARY_FLOAT

```
PROCEDURE BIND_VARIABLE
(
    c       IN   INTEGER,
    name    IN   VARCHAR2,
    value   IN   BINARY_FLOAT
);
```

– BINARY_DOUBLE

```
PROCEDURE BIND_VARIABLE
(
    c       IN   INTEGER,
    name    IN   VARCHAR2,
    value   IN   BINARY_DOUBLE
);
```

– CLOB

```
PROCEDURE BIND_VARIABLE
(
    c       IN   INTEGER,
    name    IN   VARCHAR2,
    value   IN   CLOB
);
```

– BLOB

```
PROCEDURE BIND_VARIABLE
(
    c       IN   INTEGER,
    name    IN   VARCHAR2,
    value   IN   BLOB
);
```

CHAR, RAW, ROWID 타입을 위해서는 별도의 프러시저를 사용해야 한다.

– CHAR

```
PROCEDURE BIND_VARIABLE_CHAR
(
    c      IN   INTEGER,
    name   IN   VARCHAR2,
    value  IN   CHAR[, out_value_size IN INTEGER]
);
```

– RAW

```
PROCEDURE BIND_VARIABLE_RAW
(
    c      IN   INTEGER,
    name   IN   VARCHAR2,
    value  IN   RAW[, out_value_size IN INTEGER]
);
```

– ROWID

```
PROCEDURE BIND_VARIABLE_ROWID
(
    c      IN   INTEGER,
    name   IN   VARCHAR2,
    value  IN   ROWID
);
```

- 파라미터

파라미터	설명
c	대상 커서이다.
name	SQL 문장 내의 변수의 이름이다. 이름에서 첫 문자 콜론(:)은 생략해도 된다.
value	SQL 문장 내의 변수에 설정할 변수이다.
out_value_size	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
DECLARE
    csr INTEGER;
    x NUMBER := 1;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'select :var from dual', dbms_sql.native);

    DBMS_SQL.BIND_VARIABLE(csr, 'var', x);

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

42.3.3. CLOSE_CURSOR

주어진 커서를 닫고, NULL로 설정하는 프러시저이다.

CLOSE_CURSOR 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CLOSE_CURSOR
(
    c IN OUT INTEGER
);
```

- 파라미터

파라미터	설명
c	대상 커서를 닫은 후에 NULL로 설정한다.

- 예제

```
DECLARE
    csr INTEGER := DBMS_SQL.OPEN_CURSOR();
BEGIN
    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

42.3.4. COLUMN_VALUE

주어진 커서에서 fetch한 컬럼의 값을 원하는 변수로 가져오는 프러시저이다. 실제 fetch는 FETCH_ROWS 함수에서 일어나고, COLUMN_VALUE 프러시저는 fetch한 후 데이터를 가져오는 데 사용된다.

이때 DEFINE_COLUMN의 column 파라미터와 COLUMN_VALUE의 value 파라미터가 동일한 타입을 가지도록 함수를 호출해야 한다. 예를 들어 column 파라미터가 NUMBER인 DEFINE_COLUMN 프러시저를 호출했다면, COLUMN_VALUE 또한 value 파라미터의 타입이 NUMBER인 프러시저를 호출해야 한다.

동일한 이름으로 타입에 따라 중복으로 선언된 프러시저가 많으므로, 암시적 타입 변환을 고려하여 주의 깊게 사용해야 한다. 이는 변수의 실제 타입이 같아야 하는 것이 아니라, 호출된 함수의 파라미터의 타입이 동일하게 맞춰져야 한다는 의미이다. 예를 들어 DEFINE_COLUMN_CHAR를 사용했다면, COLUMN_VALUE_CHAR를 호출해야 한다.

COLUMN_VALUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

  COLUMN_VALUE 프러시저는 값을 저장할 변수의 타입에 따라 중복으로 선언되어 있다.

  해당되는 타입으로는 NUMBER, VARCHAR2, DATE, TIMESTAMP, INTERVAL, BINARY_FLOAT, BINARY_DOUBLE, CLOB, BLOB이 있다.

– NUMBER

```
PROCEDURE COLUMN_VALUE
(
    c            IN    INTEGER,
    position     IN    INTEGER,
    value        OUT   NUMBER
);
```

– VARCHAR2

```
PROCEDURE COLUMN_VALUE
(
    c            IN    INTEGER,
    position     IN    INTEGER,
    value        OUT   VARCHAR2
);
```

– DATE

```
PROCEDURE COLUMN_VALUE
(
    c            IN    INTEGER,
    position     IN    INTEGER,
    value        OUT   DATE
);
```

– TIMESTAMP

```
PROCEDURE COLUMN_VALUE
(
    c             IN   INTEGER,
    position      IN   INTEGER,
    value         OUT  TIMESTAMP_UNCONSTRAINED
);
```

– INTERVAL

```
PROCEDURE COLUMN_VALUE
(
    c             IN   INTEGER,
    position      IN   INTEGER,
    value         OUT  YMINTERVAL_UNCONSTRAINED
);
```

```
PROCEDURE COLUMN_VALUE
(
    c             IN    INTEGER,
    position      IN    INTEGER,
```

```
    value         OUT   DSINTERVAL_UNCONSTRAINED
);
```

– BINARY_FLOAT

```
PROCEDURE COLUMN_VALUE
(
    c             IN    INTEGER,
    position      IN    INTEGER,
    value         OUT   BINARY_FLOAT
);
```

– BINARY_DOUBLE

```
PROCEDURE COLUMN_VALUE
(
    c             IN    INTEGER,
    position      IN    INTEGER,
    value         OUT   BINARY_DOUBLE
);
```

– BLOB

```
PROCEDURE COLUMN_VALUE
(
    c IN INTEGER,
    position IN INTEGER,
    value  OUT  BLOB
);
```

– CLOB

```
PROCEDURE COLUMN_VALUE
(
    c IN INTEGER,
    position IN INTEGER,
    value  OUT  CLOB
);
```

CHAR, RAW, ROWID 타입을 위해서는 별도의 프러시저를 사용해야 한다.

– CHAR

```
PROCEDURE COLUMN_VALUE_CHAR
(
    c          IN   INTEGER,
    position   IN   INTEGER,
    value      OUT  CHAR[, column_error OUT NUMBER,
```

```
                          actual_length OUT INTEGER]
);
```

– RAW

```
PROCEDURE COLUMN_VALUE_RAW
(
    c          IN   INTEGER,
    position   IN   INTEGER,
    value      OUT  RAW[, column_error OUT NUMBER,
                        actual_length OUT INTEGER]
);
```

– ROWID

```
PROCEDURE COLUMN_VALUE_ROWID
(
    c          IN   INTEGER,
    position   IN   INTEGER,
    value      OUT  ROWID[, column_error OUT NUMBER]
);
```

- 파라미터

파라미터	설명
c	대상 커서이다.
position	fetch할 컬럼 목록에서 상대적 위치이다. (1 이상)
value	가져올 컬럼을 저장할 변수이다.
column_error	– 값이 NULL인 경우 1405이다. – 값이 절삭된 경우 1406이다.
actual_length	실제로 가져온 컬럼의 길이이다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));
INSERT INTO PSM_TABLE VALUES('111');

DECLARE
    csr INTEGER;
    col VARCHAR2(2);
    val VARCHAR2(2);
    ret INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();
```

```
    DBMS_SQL.PARSE(csr, 'select * from psm_table', dbms_sql.native);

    /* result value is truncated */
    DBMS_SQL.DEFINE_COLUMN(csr, 1, col, 2);

    ret := DBMS_SQL.EXECUTE(csr);
    ret := DBMS_SQL.FETCH_ROWS(csr);

    DBMS_SQL.COLUMN_VALUE(csr, 1, val);

    DBMS_OUTPUT.PUT_LINE('val=' || val || '.');

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
val=11.
```

42.3.5. DEFINE_COLUMN

주어진 커서에서 fetch될 컬럼의 타입을 정의하는 프러시저이다. 이 프러시저는 SELECT 문장에만 사용할 수 있다.

DEFINE_COLUMN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

  DEFINE_COLUMN 프러시저는 설정할 변수의 타입에 따라 중복으로 선언되어 있다.

  해당되는 타입으로는 NUMBER, VARCHAR2, DATE, TIMESTAMP, INTERVAL, BINARY_FLOAT, BINARY_DOUBLE, CLOB, BLOB이 있다.

  – NUMBER

  ```
  PROCEDURE DEFINE_COLUMN
  (
      c              IN   INTEGER,
      position       IN   INTEGER,
      column         IN   NUMBER
  );
  ```

  – VARCHAR2

  ```
  PROCEDURE DEFINE_COLUMN
  (
      c              IN   INTEGER,
      position       IN   INTEGER,
  ```

```
    column          IN   VARCHAR2,
    column_size     IN   INTEGER
);
```

– DATE

```
PROCEDURE DEFINE_COLUMN
(
    c               IN   INTEGER,
    position        IN   INTEGER,
    column          IN   DATE
);
```

– TIMESTAMP

```
PROCEDURE DEFINE_COLUMN
(
    c               IN   INTEGER,
    position        IN   INTEGER,
    column          IN   TIMESTAMP_UNCONSTRAINED
);
```

– INTERVAL

```
PROCEDURE DEFINE_COLUMN
(
    c               IN   INTEGER,
    position        IN   INTEGER,
    column          IN   YMINTERVAL_UNCONSTRAINED
);
```

```
PROCEDURE DEFINE_COLUMN
(
    c               IN   INTEGER,
    position        IN   INTEGER,
    column          IN   DSINTERVAL_UNCONSTRAINED
);
```

– BINARY_FLOAT

```
PROCEDURE DEFINE_COLUMN
(
    c               IN   INTEGER,
    position        IN   INTEGER,
    column          IN    BINARY_FLOAT
);
```

– BINARY_DOUBLE

```
PROCEDURE DEFINE_COLUMN
(
    c              IN   INTEGER,
    position       IN   INTEGER,
    column         IN   BINARY_DOUBLE
);
```

– CLOB

```
PROCEDURE DEFINE_COLUMN
(
    c              IN   INTEGER,
    position       IN   INTEGER,
    column         IN   CLOB
);
```

– BLOB

```
PROCEDURE DEFINE_COLUMN
(
    c              IN   INTEGER,
    position       IN   INTEGER,
    column         IN   BLOB
);
```

CHAR, RAW, ROWID 타입을 위해서는 별도의 프러시저를 사용해야 한다.

– CHAR

```
PROCEDURE DEFINE_COLUMN_CHAR
(
    c              IN   INTEGER,
    position       IN   INTEGER,
    column         IN   CHAR,
    column_size    IN   INTEGER
);
```

– RAW

```
PROCEDURE DEFINE_COLUMN_RAW
(
    c              IN   INTEGER,
    position       IN   INTEGER,
    column         IN   RAW,
    column_size    IN   INTEGER
);
```

– ROWID

```
PROCEDURE DEFINE_COLUMN_ROWID
(
    c              IN   INTEGER,
    position       IN   INTEGER,
    column         IN   ROWID
);
```

- 파라미터

파라미터	설명
c	대상 커서이다.
position	SELECT를 실행할 컬럼 목록에서의 상대적 위치이다. 문장의 컬럼에서 첫 번째 위치를 1로 하며, 1 이상의 값을 입력한다.
column	정의할 컬럼의 변수로, 타입만 참조하며 값은 관계없다.
column_size	VARCHAR2, CHAR 타입에 대해 SELECT를 실행할 컬럼 값의 최대 길이이다. 단, 최대 길이를 초과한 결과 컬럼은 절삭하여 가져온다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));
INSERT INTO PSM_TABLE VALUES('abc');

DECLARE
    csr INTEGER;
    v VARCHAR2(1);
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'select * from psm_table', dbms_sql.native);

    DBMS_SQL.DEFINE_COLUMN(csr, 1, v, 1);

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

42.3.6. DESCRIBE_COLUMNS

주어진 열린 커서에서 결과 테이블의 컬럼들의 타입을 알려주는 프러시저이다. 이 프러시저는 SELECT 문장에만 사용할 수 있으며, 반드시 PARSE 프러시저로 SELECT 문장을 명시한 후 사용해야 한다.

DESCRIBE_COLUMNS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE DESCRIBE_COLUMNS
(
    c              IN    INTEGER,
    col_cnt        OUT   INTEGER,
    desc_t         OUT   DESC_TAB
);
```

- 파라미터

파라미터	설명
c	대상 커서이다.
col_cnt	SELECT 결과 테이블의 컬럼 갯수이다.
desc_t	각 컬럼의 정보들(desc_rec)의 리스트(desc_tab)이다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));
INSERT INTO PSM_TABLE VALUES('abc');

DECLARE
    csr INTEGER;
    col_cnt INTEGER;
    rec_tab DBMS_SQL.DESC_TAB;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'select * from psm_table', dbms_sql.native);

    DBMS_SQL.DESCRIBE_COLUMNS(csr, col_cnt, rec_tab);

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

42.3.7. DESCRIBE_COLUMNS2

주어진 열린 커서에서 결과 테이블의 컬럼들의 타입을 알려주는 프러시저이다. 이 프러시저는 SELECT 문장에만 사용할 수 있으며, 반드시 PARSE 프러시저로 SELECT 문장을 명시한 후 사용해야 한다.

DESCRIBE_COLUMNS에서는 정보들 중 컬럼 이름을 최대 32bytes 밖에 수용하지 못하지만, 이 프러시저는 32767bytes까지 수용하므로, 이 함수를 사용할 것을 권장한다.

DESCRIBE_COLUMNS2 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE DESCRIBE_COLUMNS2
(
    c              IN    INTEGER,
    col_cnt        OUT   INTEGER,
    desc_t         OUT   DESC_TAB2
);
```

- 파라미터

파라미터	설명
c	대상 커서이다.
col_cnt	SELECT 결과 테이블의 컬럼 갯수이다.
desc_t	각 컬럼의 정보들(desc_rec2)의 리스트(desc_tab2)이다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));
INSERT INTO PSM_TABLE VALUES('abc');

DECLARE
    csr INTEGER;
    col_cnt INTEGER;
    rec_tab DBMS_SQL.DESC_TAB2;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'select * from psm_table', dbms_sql.native);

    DBMS_SQL.DESCRIBE_COLUMNS2(csr, col_cnt, rec_tab);

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

42.3.8. PARSE

주어진 SQL 문장을 파싱하는 프러시저이다. 이때 DDL 문장인 경우 즉시 수행된다. 기존에 열려 있는 SQL 문장이 있다면 닫고 새로 입력된 SQL 문장을 파싱한다.

일반적으로 VARCHAR2 타입의 문자열로 SQL문장을 실행하지만, 문자열 길이가 길 경우 CLOB 또는 VARCHAR2A 타입으로 작성할 수 있다.

PARSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - VARCHAR2

```
PROCEDURE PARSE
(
    c               IN    INTEGER,
    statement       IN    VARCHAR2,
    language_flag   IN    INTEGER DEFAULT NULL
);
```

  - CLOB

```
PROCEDURE PARSE
(
    c               IN    INTEGER,
    statement       IN    CLOB,
    language_flag   IN    INTEGER DEFAULT NULL
);
```

  - VARCHAR2A

```
PROCEDURE PARSE
(
    c               IN    INTEGER,
    statement       IN    VARCHAR2A,
    lb              IN    INTEGER,
    ub              IN    INTEGER,
    lfflg           IN    BOOLEAN,
    language_flag   IN    INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	**설명**
c	대상 커서이다.
statement	파싱할 대상이 되는 SQL 문장이다. PSM 문장과 달리 SQL 문장에서는 마지막 세미콜론(;)은 입력하지 않아도 된다.
lb	VARCHAR2A 구성 문자열들 중 하계값이다.
ub	VARCHAR2A 구성 문자열들 중 상계값이다.
lfflg	TRUE이면 VARCHAR2A의 각 구성 문자열 끝마다 newline 문자가 추가된다.
language_flag	지원되지 않는 기능이므로 값을 무시한다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));

DECLARE
    csr INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'select * from psm_table', dbms_sql.native);

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));

DECLARE
    csr INTEGER;
    sql_arr DBMS_SQL.VARCHAR2A;
    r number;
    trec PSM_TABLE%ROWTYPE;
    lb number;
    ub number;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    sql_arr(1) := 'insert ';
    sql_arr(2) := 'into ';
    sql_arr(3) := 'PSM_TABLE ';
    sql_arr(4) := 'values';
    sql_arr(5) := '(';
    sql_arr(6) := '''abc''';
    sql_arr(7) := ')';

    lb := 1;
    ub := 7;
    DBMS_SQL.PARSE(csr, sql_arr, lb, ub, true, dbms_sql.native);

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
```

42.4. 함수

본 절에서는 DBMS_SQL 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

42.4.1. EXECUTE

주어진 커서를 실행하는 함수이다. INSERT, UPDATE, DELETE 문장인 경우 처리된 로우의 수가 반환된다.

EXECUTE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION EXECUTE
(
    c    IN    INTEGER
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
c	대상 커서이다.

- 반환값

반환값	설명
INTEGER	INSERT, UPDATE, DELETE 문장인 경우 처리한 로우의 수를 반환한다.
UNDEFINED	INSERT, UPDATE, DELETE 문장 이외의 경우에 반환한다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR(3));

DECLARE
    csr INTEGER;
    ret INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'insert into psm_table values(''abc'')',
                   dbms_sql.native);

    ret := DBMS_SQL.EXECUTE(csr);
```

```
    DBMS_OUTPUT.PUT_LINE('affected row cnt = ' || ret || '.');

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
affected row cnt = 1.
```

42.4.2. EXECUTE_AND_FETCH

주어진 커서에 EXECUTE 와 FETCH_ROWS 함수를 연속해서 호출하는 것과 동일한 함수이다. 반환되는 값은 FETCH_ROWS와 동일하다. 즉, 실제 fetch한 로우의 수를 반환한다.

EXECUTE_AND_FETCH 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION EXECUTE_AND_FETCH
(
    c      IN   INTEGER,
    exact  IN   BOOLEAN DEFAULT FALSE
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
c	대상 커서이다.
exact	– TRUE : 정확히 하나의 로우가 fetch되지 않으면 예외가 발생한다. – FALSE : 여러 개의 ROW가 fetch되어도 예외가 발생하지 않는다.

- 반환값

반환값	설명
0	fetch할 로우가 더 이상 없는 경우에 반환한다.
1	fetch에 성공한 경우에 반환한다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR(3));
INSERT INTO PSM_TABLE VALUES('abc');

DECLARE
    csr INTEGER;
```

```
    ret INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'SELECT * FROM PSM_TABLE', dbms_sql.native);

    ret := DBMS_SQL.EXECUTE_AND_FETCH(csr);
    DBMS_OUTPUT.PUT_LINE('fetched row cnt = ' || ret || '.');

    LOOP
        ret := DBMS_SQL.FETCH_ROWS(csr);
        DBMS_OUTPUT.PUT_LINE('fetched row cnt = ' || ret || '.');
        exit when ret = 0;
    END LOOP;

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
fetched row cnt = 1.
fetched row cnt = 0.
```

42.4.3. FETCH_ROWS

주어진 커서에서 로우를 fetch하는 함수이다. 이때 실제 fetch된 로우의 수가 반환된다. 더 이상 fetch할 로우가 없는데 이 함수를 계속 호출하게 되면 예외가 발생한다.

FETCH_ROWS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION FETCH_ROWS
(
    c   IN   INTEGER
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
c	대상 커서이다.

- 반환값

반환값	설명
0	fetch할 로우가 더 이상 없는 경우에 반환한다.

반환값	설명
1	fetch에 성공한 경우에 반환한다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));
INSERT INTO PSM_TABLE VALUES('abc');

DECLARE
    csr INTEGER;
    ret INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'SELECT * FROM PSM_TABLE', dbms_sql.native);

    ret := DBMS_SQL.EXECUTE(csr);
    LOOP
        ret := DBMS_SQL.FETCH_ROWS(csr);
        DBMS_OUTPUT.PUT_LINE('fetched row cnt = ' || ret || '.');
        exit when ret = 0;
    END LOOP;

    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
fetched row cnt = 1.
fetched row cnt = 0.
```

42.4.4. OPEN_CURSOR

SQL 문장을 수행하기 위한 커서를 새로 하나 여는 함수이다. 열린 커서는 사용 후에 반드시 close_cursor를 호출해야 리소스가 서버에 반납된다.

한번 열린 커서는 동일한 SQL 문장을 반복해서 수행할 수도 있고, 다른 SQL 문장을 실행하는 데 사용할 수도 있다. 커서를 여는 데 성공하면, 커서 ID를 INTEGER 타입으로 반환한다. 이 값을 이후에 DBMS_SQL 패키지의 커서 파라미터로 사용하면 된다.

OPEN_CURSOR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION OPEN_CURSOR()
RETURN INTEGER;
```

- 예제

```
DECLARE
    csr INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();
END;
/
```

42.4.5. IS_OPEN

주어진 커서가 DBMS_SQL.open_cursor 함수로 열릴 수 있는 커서인지 검사하는 함수이다.

IS_OPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION IS_OPEN
(
    c   IN   INTEGER
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
c	대상 커서이다.

- 반환값

반환값	설명
true	open 상태인 경우에 반환한다.
false	open 상태가 아닌 경우에 반환한다.

- 예제

```
DECLARE
    csr INTEGER := DBMS_SQL.OPEN_CURSOR();
    is_open BOOLEAN;
BEGIN
    is_open := DBMS_SQL.IS_OPEN(csr);

    if is_open then
        DBMS_OUTPUT.PUT_LINE('opened.');
    end if;
```

```
END;
/
opened.
```

42.4.6. LAST_ERROR_POSITION

파싱을 할 때 발생한 에러의 위치를 찾아주는 함수이다. 단, 이 함수는 PARSE 프러시저를 호출한 직후에 사용해야 올바른 결과를 얻을 수 있다.

LAST_ERROR_POSITION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION LAST_ERROR_POSITION return INTEGER;
```

- 반환값

반환값	설명
INTEGER	파싱을 할 때 에러가 발생한 오프셋을 반환한다. (0부터 시작)

- 예제

```
DECLARE
    csr INTEGER;
    position INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'select * from psm_not_exist_table', dbms_sql.native);

EXCEPTION
WHEN OTHERS THEN
    position := DBMS_SQL.LAST_ERROR_POSITION;

    DBMS_OUTPUT.PUT_LINE('error position = ' || position);
    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
error position = 14
```

42.4.7. LAST_ROW_COUNT

현재 수행 중인 문장의 fetch된 로우 수의 총합을 반환하는 함수이다. 단, 이 함수는 PARSE 프러시저를 호출한 직후에 사용해야 올바른 결과를 얻을 수 있다.

LAST_ROW_COUNT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION LAST_ROW_COUNT return INTEGER;
```

- 반환값

반환값	설명
INTEGER	현재 수행 중인 문장의 fetch된 로우 수의 총합을 반환한다.

- 예제

```
CREATE TABLE PSM_TABLE (A VARCHAR2(3));
INSERT INTO PSM_TABLE VALUES('111');
INSERT INTO PSM_TABLE VALUES('222');

DECLARE
    csr INTEGER;
    ret INTEGER;
    cnt INTEGER;
BEGIN
    csr := DBMS_SQL.OPEN_CURSOR();

    DBMS_SQL.PARSE(csr, 'SELECT * FROM PSM_TABLE', dbms_sql.native);

    ret := DBMS_SQL.EXECUTE(csr);
    LOOP
        ret := DBMS_SQL.FETCH_ROWS(csr);
        exit when ret = 0;
    END LOOP;

    cnt := DBMS_SQL.LAST_ROW_COUNT;

    DBMS_OUTPUT.PUT_LINE('last row count = ' || cnt);
    DBMS_SQL.CLOSE_CURSOR(csr);
END;
/
last row count = 2
```

제43장 DBMS_SQLTUNE

본 장에서는 DBMS_SQLTUNE 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

43.1. 개요

현재 **DBMS_SQLTUNE** 패키지는 실시간 SQL 모니터링 기능에 대한 보고서를 생성해 주는 REPORT_SQL_MONITOR 함수 하나만을 제공하고 있는 패키지이다.

43.2. 프러시저와 함수

본 절에서는 DBMS_SQLTUNE 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

43.2.1. REPORT_SQL_MONITOR

특정 SQL 수행에 대해서 실시간 SQL 모니터링 기능에 의해 수집된 성능 관련 정보를 보고서 형태로 돌려준다. 보고서 형식은 텍스트 형식이다.

REPORT_SQL_MONITOR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQLTUNE.REPORT_SQL_MONITOR
(
    sql_id         IN VARCHAR DEFAULT NULL,
    session_id     IN NUMBER  DEFAULT NULL,
    session_serial IN NUMBER  DEFAULT NULL,
    sql_exec_start IN DATE    DEFAULT NULL,
    sql_exec_id    IN NUMBER  DEFAULT NULL
)
RETURN CLOB;
```

- 파라미터

파라미터	설명
sql_id	보고서를 생성할 SQL 수행의 SQL 식별자이다.

파라미터	설명
	여기에 NULL을 명시하면 현재 시스템에서 가장 최근에 모니터링된 SQL 수행에 대한 보고서를 생성한다.
session_id	이 값이 NULL이 아니면 이 값이 명시하는 세션에서 수행된 SQL 수행에 한정하여 보고서를 생성할 수 있다. 만약, 이 파라미터에 NULL 이 아닌 값을 명시하고 sql_id 파라미터에 NULL을 명시할 경우에 해당 세션에서 가장 최근에 모니터링된 SQL 수행에 대해 보고서를 생성한다.
session_serial	원하는 특정 세션을 확실하게 한정하고 싶을 때 추가적으로 이 파라미터를 명시할 수 있다. 만약, session_id 파라미터가 NULL일 경우 이 파라미터는 무시된다.
sql_exec_start	sql_id 파라미터를 명시하였을 때만 이 파라미터를 명시할 수 있다. 모니터링된 SQL 수행 중에서 해당 sql_id를 가진 것 중 sql_exec_start 값이 일치하는 모니터링 정보를 찾아 보고서를 생성한다.
sql_exec_id	sql_id 파라미터를 명시하였을 때만 이 파라미터를 명시할 수 있다. 모니터링된 SQL 수행 중에서 해당 sql_id를 가진 것 중 sql_exec_id 값이 일치하는 모니터링 정보를 찾아 보고서를 생성한다.

- 반환값

반환값	설명
CLOB 데이터	보고서 형식으로 생성된 문자열을 CLOB 형태로 반환한다.

- 예제

```
set long 1000000;
select dbms_sqltune.report_sql_monitor from dual;
```

43.2.2. SQLTEXT_TO_SIGNATURE

특정 SQL 텍스트의 SIGNATURE를 반환한다. SIGNATURE는 DBA_SQL_PROFILES에서 SQL 질의문을 식별하는 데 사용할 수 있다.

SQLTEXT_TO_SIGNATURE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQLTUNE.DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE
(
    sql_text       IN CLOB,
    force_match    IN BINARY_INTEGER := 0
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
sql_text	SIGNATURE로 변환할 SQL 질의문이다.
force_match	– TRUE : WHERE 절에 literal로 적힌 constant를 bind variable 로 바꾼 뒤 normalize를 수행한다. – FALSE : WHERE 절에 literal로 적힌 constant를 bind variable로 바꾸지 않고 normalize를 수행한다.

- 반환값

반환값	설명
NUMBER 데이터	Normalize된 SQL test의 signature를 반환한다.

- 예제

```
DECLARE
    ret1 NUMBER;
    ret2 NUMBER;
BEGIN
    ret1 := DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE('SELECT 1 FROM DUAL', 0);
    ret2 := DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE('SELECT 2 FROM DUAL', 0);
    DBMS_OUTPUT.PUT_LINE(ret1);
    DBMS_OUTPUT.PUT_LINE(ret2);
    ret1 := DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE('SELECT 1 FROM DUAL', 1);
    ret2 := DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE('SELECT 2 FROM DUAL', 1);
    DBMS_OUTPUT.PUT_LINE(ret1);
    DBMS_OUTPUT.PUT_LINE(ret2);
END;
/
```

43.2.3. IMPORT_SQL_PROFILE

SQL PROFILE을 생성한다. 반드시 SQL_TEXT와 SQL_PROFILE_ATTRIBUTE는 인자로 넘겨줘야 한다.

IMPORT_SQL_PROFILE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQLTUNE.DBMS_SQLTUNE.IMPORT_SQL_PROFILE
(
    sql_text       IN CLOB,
    profile        IN SQLPROF_ATTR,
    category       IN VARCHAR2 := 'DEFAULT',
    name           IN VARCHAR2 := NULL,
    description    IN VARCHAR2 := NULL,
    replace        IN boolean := FALSE,
    force_match    IN BINARY_INTEGER := 0
)
```

- 파라미터

파라미터	설명
sql_text	SQL 질의문이다.
profile	sql profile attribute의 정보이다.
category	sql profile이 속해 있는 category이다.
name	sql profile 이름이다.
description	sql profile의 설명이다.
replace	이미 존재하는 sql profile이면 현재 정의하는 것으로 대체할 지 여부이다.
force_match	– TRUE : WHERE 절에 literal로 적힌 constant를 bind variable 로 바꾼 뒤 normalize를 수행한다. – FALSE : WHERE 절에 literal로 적힌 constant를 bind variable로 바꾸지 않고 normalize를 수행한다.

- 예제

```
create table t (c1 number, c2 number);

create index idx on t(c1);

alter session set optimizer_log_outline=y;

select * from t where c1 = 1;

select * from table(dbms_xplan.display_cursor(format=>'outline'));

BEGIN
DBMS_SQLTUNE.IMPORT_SQL_PROFILE(
'select /*+full(t)*/ * from t where c1=1',
SQLPROF_ATTR(
```

```
q'$BEGIN_OUTLINE_DATA$'
, q'$IGNORE_OPTIM_EMBEDDED_HINTS$'
, q'$OPT_PARAM(_ENABLE_HASH_JOIN, 'TRUE')$'
, q'$OPT_PARAM(_ENABLE_IDX_JOIN, 'TRUE')$'
, q'$OPT_PARAM(OPTIMIZER_MODE, 'ALL_ROWS')$'
, q'$OPT_PARAM(_OPT_JOIN_MEMORY_LIMIT, 7340032)$'
, q'$OPT_PARAM(_OPT_BUILD_INDEX_FFS_SELF_JOIN, 'TRUE')$'
, q'$OPT_PARAM(_ENABLE_HASH_GROUPBY, 'TRUE')$'
, q'$OPT_PARAM(_ENABLE_SORT_GROUPBY, 'TRUE')$'
, q'$OPT_PARAM(_ENABLE_ISS, 'TRUE')$'
, q'$OPT_PARAM(_OPT_OR_EXP_FOR_JOIN_PRED, 'TRUE')$'
, q'$OPT_PARAM(_TRANS_UNNEST_SUBQUERY_IN_SELECT_LIST, 'FALSE')$'
, q'$OPT_PARAM(_TRANS_NO_TRIM_LPN, 'FALSE')$'
, q'$OPT_PARAM(_OPT_PE_BROADCAST_LIMIT_ROW, 0)$'
, q'$OPT_PARAM(_OPT_PGROUPBY_PUSH_RATIO, 100)$'
, q'$OPT_PARAM(_OPT_BIND_PEEKING, 'TRUE')$'
, q'$ROWID@LPN$3(T)$'
, q'$INDEX_RS@LPN$3(IDX)$'
, q'$END_OUTLINE_DATA$'
));
END;
/                                                    A
```

43.2.4. ALTER_SQL_PROFILE

기존 SQL PROFILE 의 특정 속성을 변경한다.

ALTER_SQL_PROFILE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQLTUNE.ALTER_SQL_PROFILE
(
    name             IN VARCHAR2,
    attribute_name   IN VARCHAR2,
    value            IN VARCHAR2
)
```

- 파라미터

파라미터	설명
name	sql profile의 이름이다.
attribute_name	sql profile의 configuration attribute 이름이다.
value	변경할 sql configuration attribute의 값이다.

파라미터	설명
	– STATUS : 사용 가능한 상태를 나타내는 값이다. ENABLED 또는 DISABLED 로 조절한다. – NAME : attribute의 이름으로 유니크한 문자열로 만들어야 한다. – DESCRIPTION : attribute에 대한 설명으로 500자 이내의 문자열로 적어야 한다. – CATEGORY : SQL 질의문과 조합되었을 때 유니크한 문자열로 만들어야 한다.

- 예제

  – ALTER 전

```
SQL> select * from dba_sql_profiles;
NAME
--------------------------------------------------------------------------------
CATEGORY
--------------------------------------------------------------------------------

 SIGNATURE
----------
SQL_TEXT
--------------------------------------------------------------------------------
CREATED
---------------------------------------------------------------------
LAST_MODIFIED
---------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
TYPE    STATUS   FORCE_MATCHING
------- -------- --------------
SYS_SQLPROF_4250f54b3b640bf9
DEFAULT
6.9222E+18
select /*+full(t)*/ * from t where c1=1
2020/12/14 05:07:17.304011
2020/12/14 05:07:17.304011

MANUAL  ENABLED  NO
```

  – ALTER 구문

```
BEGIN
    DBMS_SQLTUNE.ALTER_SQL_PROFILE(NAME => 'SYS_SQLPROF_4250f54b3b640bf9',
    ATTRIBUTE_NAME => 'DESCRIPTION', VALUE => 'TEST');
```

```
END;

/
```

– ALTER 후

```
SQL> select * from dba_sql_profiles;
NAME
--------------------------------------------------------------------------------
CATEGORY
--------------------------------------------------------------------------------

 SIGNATURE
----------
SQL_TEXT
--------------------------------------------------------------------------------
CREATED
---------------------------------------------------------------------------
LAST_MODIFIED
---------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
TYPE    STATUS   FORCE_MATCHING
------- -------- --------------
SYS_SQLPROF_4250f54b3b640bf9
DEFAULT
6.9222E+18
select /*+full(t)*/ * from t where c1=1
2020/12/14 05:07:17.304011
2020/12/14 05:07:17.304011
TEST
MANUAL  ENABLED  NO
```

43.2.5. DROP_SQL_PROFILE

특정 SQL PROFILE을 데이터베이스에서 삭제한다.

DROP_SQL_PROFILE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQLTUNE.DROP_SQL_PROFILE
(
    name         IN VARCHAR2,
```

```
    ignore       IN BOOLEAN := FALSE
)
```

- 파라미터

파라미터	설명
name	sql profile의 이름이다.
ignore	에러 발생 시 무시 여부이다.

- 예제

  – DROP 전

```
SQL> select * from dba_sql_profiles;

NAME
--------------------------------------------------------------------------------
CATEGORY
--------------------------------------------------------------------------------

 SIGNATURE
----------
SQL_TEXT
--------------------------------------------------------------------------------
CREATED
---------------------------------------------------------------------------
LAST_MODIFIED
---------------------------------------------------------------------------
DESCRIPTION
--------------------------------------------------------------------------------
TYPE    STATUS   FORCE_MATCHING
------- -------- --------------
SYS_SQLPROF_4250f54b3b640bf9
DEFAULT
6.9222E+18
select /*+full(t)*/ * from t where c1=1
2020/12/14 05:07:17.304011
2020/12/14 05:07:17.304011
TEST
MANUAL  ENABLED  NO
```

  – DROP 구문

```
BEGIN

    DBMS_SQLTUNE.DROP_SQL_PROFILE(NAME => 'SYS_SQLPROF_4250f54b3b640bf9',
    IGNORE => FALSE);
```

```
END;

/
```

– DROP 후

```
SQL> select * from dba_sql_profiles;
0 row selected.
```

제44장 DBMS_SQL_TRANSLATOR

본 장에서는 DBMS_SQL_TRANSLATOR 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

44.1. 개요

DBMS_SQL_TRANSLATOR 패키지를 이용해 SQL 번역 프로파일을 생성하고 이용할 수가 있다. DBMS_SQL_TRANSLATOR 패키지는 호출자 권한으로 수행되는 패키지이다. 그러므로 호출자가 이 패키지의 함수를 통해 번역기 패키지 함수를 호출할 경우, 호출자는 번역기 패키지에 대해 EXECUTE 권한을 가지고 있어야 한다.

번역기 패키지는 다음의 2개의 프러시저를 가지고 있는 PSM 패키지여야 한다.

```
PROCEDURE TRANSLATE_SQL
(
    sql_text        IN  CLOB,
    translated_text OUT CLOB
);

PROCEDURE TRANSLATE_ERROR
(
    error_code          IN  PLS_INTEGER,
    translated_code     OUT PLS_INTEGER,
    translated_sqlstate OUT VARCHAR2
);
```

번역기 패키지 함수의 각 인자에 대한 설명은 다음과 같다.

필드 이름	설명
sql_text	번역할 SQL 문이다.
translated_text	번역된 SQL 문이다.
error_code	번역할 Tibero 에러 코드이다.
translated_code	번역된 에러 코드이다.
translated_sqlstate	번역된 SQLSTATE 이다.

다음은 DBMS_SQL_TRANSLATOR 패키지 내에 정의된 상수이다. 이 상수들은 SQL 번역 프로파일의 속성을 지정할 때 이용한다.

- ATTR_TRANSLATOR

  번역기 패키지를 설정할 때 사용한다. 패키지는 '[*스키마.*]*패키지_이름*' 형식으로 설정할 수 있다. 디폴트로 이 속성은 지정되어 있지 않다.

```
ATTR_TRANSLATOR CONSTANT VARCHAR2(30) := 'TRANSLATOR'
```

- ATTR_TRANSLATE_NEW_SQL

  번역기 패키지를 사용해 새로운 SQL 문 (혹은 에러 코드) 을 번역할지를 결정한다. (기본값: TRUE)

설정값	**설명**
TRUE	맞춤 번역이 등록되어 있지 않은 새로운 SQL문 (혹은 에러 코드) 을 번역기 패키지를 사용해 번역하고, 번역 결과를 맞춤 번역으로 등록한다. 맞춤 번역은 translator에 의해 자동으로 번역되거나, register_sql_translation 등의 함수를 통해서 사용자가 직접 등록할 수 있다.
FALSE	맞춤 번역이 등록되어 있지 않으면 (디폴트로) 에러를 발생시킨다.

```
ATTR_TRANSLATE_NEW_SQL CONSTANT VARCHAR2(30) := 'TRANSLATE_NEW_SQL'
```

- ATTR_RAISE_TRANSLATION_ERROR

  맞춤 번역이 존재하지 않을 때 (그리고 번역기 패키지를 사용할 수 없을 때) 에러를 발생시킬지에 대한 여부를 결정한다. 디폴트로 이 속성은 거짓이다.

```
ATTR_RAISE_TRANSLATION_ERROR CONSTANT VARCHAR2(30) := 'RAISE_TRANSLATION_ERROR'
```

- ATTR_VALUE_TRUE

  속성에 참을 설정하려면 이 상수를 사용할 수 있다.

```
ATTR_VALUE_TRUE CONSTANT VARCHAR2(30) := 'TRUE'
```

- ATTR_VALUE_FALSE

  속성에 거짓을 설정하려면 이 상수를 사용할 수 있다.

```
ATTR_VALUE_FALSE CONSTANT VARCHAR2(30) := 'FALSE'
```

44.2. 프러시저와 함수

본 절에서는 DBMS_SQL_TRANSLATOR 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

44.2.1. CREATE_PROFILE

SQL 번역 프로파일을 생성한다. SQL 번역 프로파일은 별도의 네임스페이스를 가지는 스키마 객체이다.

CREATE_PROFILE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.CREATE_PROFILE
(
    profile_name IN VARCHAR
);
```

- 파라미터

파라미터	설명
profile_name	생성할 프로파일의 이름이다. '[*스키마.*]*이름*' 형식으로 설정이 가능하다.

44.2.2. DEREGISTER_ERROR_TRANSLATION

해당 에러 코드의 맞춤 번역 등록 내용을 제거한다.

DEREGISTER_ERROR_TRANSLATION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.DEREGISTER_ERROR_TRANSLATION
(
    profile_name    IN VARCHAR,
    error_code      IN PLS_INTEGER
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
error_code	등록을 제거할 에러 코드이다.

44.2.3. DEREGISTER_SQL_TRANSLATION

해당 SQL 문의 맞춤 번역 등록 내용을 제거한다.

DEREGISTER_SQL_TRANSLATION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.DEREGISTER_SQL_TRANSLATION
(
     profile_name    IN VARCHAR,
     sql_text        IN CLOB
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
sql_text	등록을 제거할 SQL 문이다.

44.2.4. DROP_PROFILE

SQL 번역 프로파일을 제거한다.

DROP_PROFILE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.DROP_PROFILE
(
     profile_name IN VARCHAR
);
```

- 파라미터

파라미터	설명
profile_name	제거할 프로파일의 이름이다.

44.2.5. ENABLE_ERROR_TRANSLATION

기 등록된 에러 코드의 맞춤 번역을 활성화한다(번역기에서 번역하는 경우에 사용할 수 있도록 한다).

ENABLE_ERROR_TRANSLATION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.ENABLE_ERROR_TRANSLATION
(
    profile_name    IN VARCHAR,
    error_code      IN PLS_INTEGER,
    enable          IN BOOLEAN     DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
error_code	에러 코드이다.
enable	활성화시킬지 혹은 비활성화시킬지를 설정한다.

44.2.6. ENABLE_SQL_TRANSLATION

기 등록된 SQL 문의 맞춤 번역을 활성화한다(번역기에서 번역하는 경우에 사용할 수 있도록 한다).

ENABLE_SQL_TRANSLATION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.ENABLE_SQL_TRANSLATION
(
    profile_name    IN VARCHAR,
    sql_text        IN CLOB,
    enable          IN BOOLEAN     DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
sql_text	SQL 문이다.
enable	활성화시킬지 혹은 비활성화시킬지를 설정한다.

44.2.7. REGISTER_ERROR_TRANSLATION

해당 에러 코드에 대한 맞춤 번역을 등록한다. 번역 프로파일을 통해 에러 코드를 번역할 때, 먼저 해당 에러 코드에 대한 맞춤 번역이 등록되어 있는지를 검사해서, 있으면 해당 번역 내용을 돌려주고 그렇지 않을 경우는 번역기 패키지의 함수를 수행하게 된다.

해당 에러 코드에 대한 맞춤 번역이 이미 등록되어 있을 경우는 새로 인자로 전달되는 내용으로 번역 내용이 업데이트된다.

REGISTER_ERROR_TRANSLATION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.REGISTER_ERROR_TRANSLATION
(
    profile_name        IN VARCHAR,
    error_code          IN PLS_INTEGER,
    translated_code     IN PLS_INTEGER DEFAULT NULL,
    translated_sqlstate IN VARCHAR     DEFAULT NULL,
    enable              IN BOOLEAN     DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
error_code	번역할 Tibero 에러 코드이다.
translated_code	번역된 에러 코드이다.
translated_sqlstate	번역된 SQLSTATE 이다.
enable	활성화시킬지 혹은 비활성화시킬지를 설정한다.

44.2.8. REGISTER_SQL_TRANSLATION

해당 SQL 문에 대한 맞춤 번역을 등록한다.번역 프로파일을 통해 SQL 문을 번역할 때, 먼저 해당 SQL 문에 대한 맞춤 번역이 등록되어 있는지를 검사해서, 있으면 해당 번역 내용을 돌려주고 그렇지 않을 경우는 번역기 패키지의 함수를 수행하게 된다.

해당 SQL 문에 대한 맞춤 번역이 이미 등록되어 있을 경우는 새로 인자로 전달되는 내용으로 번역 내용이 업데이트된다.

REGISTER_SQL_TRANSLATION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.REGISTER_SQL_TRANSLATION
(
    profile_name        IN VARCHAR,
    sql_text            IN CLOB,
    translated_text     IN CLOB        DEFAULT NULL,
```

```
    enable              IN BOOLEAN     DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
sql_text	번역할 SQL 문이다.
translated_text	번역된 SQL 문이다.
enable	활성화시킬지 혹은 비활성화시킬지를 설정한다.

44.2.9. SET_ATTRIBUTE

SQL 번역 프로파일의 속성값을 지정한다.

SET_ATTRIBUTE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.SET_ATTRIBUTE
(
    profile_name    IN VARCHAR,
    attribute_name  IN VARCHAR,
    attribute_value IN VARCHAR
);
```

- 파라미터

파라미터	설명
profile_name	프로파일의 이름이다.
attribute_name	설정할 속성의 이름이다.
attribute_value	설정할 속성값이다.

44.2.10. SQL_HASH

인자로 주어진 SQL 문의 해시값을 계산한다.

SQL_HASH 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.SQL_HASH
(
    sql_text IN CLOB
)
RETURN NUMBER DETERMINISTIC;
```

- 파라미터

파라미터	설명
sql_text	해시값을 계산할 SQL 문이다.

44.2.11. SQL_ID

인자로 주어진 SQL 문의 SQL_ID 값을 반환한다.

SQL_ID 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.SQL_ID
(
    sql_text IN CLOB
)  RETURN VARCHAR DETERMINISTIC;
```

- 파라미터

파라미터	설명
sql_text	SQL_ID 를 구할 SQL 문이다.

44.2.12. TRANSLATE_ERROR

해당 에러 코드를 번역한다. 사용할 SQL 번역 프로파일은 ALTER SESSION SET SQL_TRANSLATION_PROFILE 명령을 통해 프러시저를 수행하기에 앞서 미리 지정되어야만 한다.

TRANSLATE_ERROR 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.TRANSLATE_ERROR
(
    error_code          IN         PLS_INTEGER,
    translated_code     OUT        PLS_INTEGER,
```

```
    translated_sqlstate OUT NOCOPY VARCHAR
);
```

- 파라미터

파라미터	설명
error_code	번역할 Tibero 에러 코드이다.
translated_code	번역된 에러 코드이다.
translated_sqlstate	번역된 SQLSTATE 이다.

44.2.13. TRANSLATE_SQL

해당 SQL 문을 번역한다. 사용할 SQL 번역 프로파일은 ALTER SESSION SET SQL_TRANSLATION_PROFILE 명령을 통해 프러시저를 수행하기에 앞서 미리 지정되어야만 한다.

TRANSLATE_SQL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SQL_TRANSLATOR.TRANSLATE_SQL
(
    sql_text            IN         CLOB,
    translated_text     OUT NOCOPY CLOB
);
```

- 파라미터

파라미터	설명
sql_text	번역할 SQL 문이다.
translated_text	번역된 SQL 문이다.

제45장 DBMS_STATS

본 장에서는 DBMS_STATS 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

45.1. 개요

DBMS_STATS는 데이터베이스 객체의 통계 정보를 수집하고 관리하는 패키지이다. 이 패키지를 사용하면 데이터베이스에 있는 컬럼, 테이블, 데이터 사전(DD: Data Dictionary), 인덱스, 스키마, 시스템 등에 대한 통계 정보를 수집하고 관리(예: 삭제 또는 초기화 등)할 수 있다.

테이블 통계 정보 수집시에 사용자 선택에 따라서 컬럼 각각에 대한 데이터 분포도를 저장한다. 이를 히스토그램이라 하고, 수집 가능한 데이터 타입은 NUMBER, CHAR, VARCHAR, DATE, TIME, TIMESTAMP 이다.

사용자는 자신에게 소속된 객체들에 대해서만 통계 정보 수집이 가능하다. ANALYZE ANY 권한을 부여받으면 모든 사용자의 객체에 대한 통계 정보 수집이 가능하다.

45.2. 상수

본 절에서는 DBMS_STATS 패키지에서 제공하는 상수를 알파벳 순으로 설명한다.

45.2.1. AUTO_DEGREE

STAT 정보를 수집할 경우 병렬 쿼리의 처리 개수를 설정하는 DEGREE 값을 CPU 성능에 따라 자동으로 설정해준다.

45.3. 프러시저

본 절에서는 DBMS_STATS 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

45.3.1. ALTER_STATS_HISTORY_RETENTION

통계 정보 히스토리 보유 기간을 변경하는 프러시저이다. 해당 값은 통계 정보 히스토리를 제거하는 시점에 영향을 끼친다. 히스토리를 제거하는 기능은 “45.3.39. PURGE_STATS” 를 참고한다.

ALTER_STATS_HISTORY_RETENTION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.ALTER_STATS_HISTORY_RETENTION
(
   retention    NUMBER
);
```

- 파라미터

파라미터	설명
retention	통계 정보 히스토리를 보유할 최대 기간(일 단위)이다. 1 이상의 값이어야 하지만, 특별한 목적을 위해 아래와 같이 값을 지정할 수 있다. – -1 : 자동 히스토리 제거를 통해 제거되지 않는다 (보유 기간이 무제한이다). – 0 : 이후 수집되는 통계 정보에 대한 히스토리가 저장되지 않는다. 기존 히스토리들은 자동 제거 기능을 통해 제거된다. – NULL : 기본값인 31일로 설정한다.

45.3.2. COPY_TABLE_STATS

테이블 및 로컬 인덱스의 원본 파티션의 통계 정보를 대상 파티션의 통계 정보로 복사하는 프러시저이다.

RANGE 파티션일 경우 파티션 키의 첫 번째 컬럼의 히스토그램이 대상 파티션 바로 이전 파티션까지를 대상으로 수집되었다면 히스토그램도 같이 갱신한다.

COPY_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.COPY_TABLE_STATS
(
    ownname             IN          VARCHAR2,
    tabname             IN          VARCHAR2,
    srcpartname         IN          VARCHAR2,
    dstpartname         IN          VARCHAR2,
    scale_factor        IN          NUMBER      DEFAULT 1,
    flags               IN          NUMBER      DEFAULT NULL,
    force               IN          BOOLEAN     DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	파티션이 속한 스키마의 이름이다.
tabname	파티션이 속한 테이블의 이름이다.
srcpartname	통계 정보를 복사할 원본 파티션의 이름이다.
dstpartname	통계 정보를 복사할 대상 파티션의 이름이다.
scale_factor	원본 파티션의 로우 갯수, 블록 개수 등을 해당 수치만큼 곱하여 대상 파티션에 저장한다.
flags	사용하지 않는 파라미터이다.
force	TRUE로 설정되면, 테이블의 통계 정보가 잠겨 있어도 복사한다.

- 예외 상황

예외 상황	설명
20003	존재하지 않는 테이블 혹은 파티션 이름을 입력한 경우이다.

45.3.3. CREATE_STAT_TABLE

통계 정보를 저장할 통계 테이블을 생성하는 프러시저이다.

CREATE_STAT_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.CREATE_STAT_TABLE
(
    ownname      VARCHAR2,
    stattab      VARCHAR2,
    tblspace     VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
ownname	통계 테이블이 속한 스키마의 이름이다.
stattab	통계 정보를 저장할 통계 테이블의 이름이다.
tblspace	통계 테이블이 속할 테이블 스페이스의 이름이다. (기본값: NULL, NULL일 경우 owner의 default 테이블 스페이스에 저장된다)

- 예외 상황

예외 상황	설명
7068	잘못된 스키마를 입력한 경우이다.

45.3.4. DELETE_COLUMN_STATS

컬럼의 통계 정보를 삭제하는 프러시저이다.

DELETE_COLUMN_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_COLUMN_STATS
(
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    colname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    cascade_parts    BOOLEAN  DEFAULT TRUE,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	컬럼이 속한 테이블의 이름이다.
colname	컬럼의 이름이다.
partname	통계 정보를 삭제할 파티션의 이름이다. 테이블이 파티션되어 있고, partname의 값이 NULL이면 테이블 수준의 컬럼 통계 정보를 삭제한다.
cascade_parts	테이블이 파티션되어 있고, partname이 NULL일 때 TRUE로 설정하면 모든 파티션 수준의 컬럼 통계 정보도 함께 삭제한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 컬럼 통계 정보가 잠겨 있어도 삭제한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.

45.3.5. DELETE_DATABASE_STATS

데이터베이스에 있는 모든 테이블의 통계 정보를 삭제하는 프러시저이다.

DELETE_DATABASE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_DATABASE_STATS
(
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 삭제한다.

45.3.6. DELETE_DICTIONARY_STATS

모든 데이터 사전의 스키마(SYS, SYSCAT)의 통계 정보를 삭제하는 프러시저이다.

DELETE_DICTIONARY_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_DICTIONARY_STATS
(
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 삭제한다.

45.3.7. DELETE_INDEX_STATS

인덱스의 통계 정보를 삭제하는 프러시저이다.

DELETE_INDEX_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_INDEX_STATS
(
    ownname          VARCHAR2,
    idxname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    cascade_parts    BOOLEAN  DEFAULT TRUE,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
idxname	인덱스의 이름이다.
partname	통계 정보를 삭제할 파티션의 이름이다. 인덱스가 파티션으로 나누어져 있고, partname의 값이 NULL이면 인덱스 수준의 통계 정보를 삭제한다.
cascade_parts	인덱스가 파티션으로 나누어져 있고, partname의 값이 NULL일 때 TRUE로 설정하면 모든 파티션 수준의 통계 정보도 함께 삭제한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 인덱스의 통계 정보가 잠겨있어도 삭제한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.

45.3.8. DELETE_SCHEMA_STATS

스키마 전체의 통계 정보를 삭제하는 프러시저이다.

DELETE_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_SCHEMA_STATS
(
    ownname          VARCHAR2,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 삭제한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.

45.3.9. DELETE_SYSTEM_STATS

workload 시스템의 통계 정보를 삭제하고, noworkload 시스템의 통계 정보를 초기화하는 프러시저이다. workload 시스템의 통계 정보는 INTERVAL, START, STOP 옵션으로, noworkload 시스템의 통계 정보는 NOWORKLOAD 옵션으로 수집한다.

DELETE_SYSTEM_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_SYSTEM_STATS ();
```

- 예외 상황

예외 상황	설명
20000	DBA 권한이 없는 경우이다.

45.3.10. DELETE_TABLE_STATS

테이블의 통계 정보를 삭제하는 프러시저이다.

DELETE_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DELETE_TABLE_STATS
(
  ownname           VARCHAR2,
  tabname           VARCHAR2,
  partname          VARCHAR2 DEFAULT NULL,
  cascade_parts     BOOLEAN  DEFAULT TRUE,
  cascade_columns   BOOLEAN  DEFAULT TRUE,
  cascade_indexes   BOOLEAN  DEFAULT TRUE,
  no_invalidate     BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),
  force             BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
partname	통계 정보를 삭제할 대상 파티션의 이름이다. 파티션으로 나뉘어 있고, partname의 값이 NULL이면 테이블 수준의 통계 정보를 삭제한다.
cascade_parts	테이블이 파티션으로 나뉘어 있고, partname이 NULL일 때 TRUE로 설정하면 모든 파티션 수준의 통계 정보도 함께 삭제한다.
cascade_columns	TRUE로 설정하면 컬럼의 통계 정보도 함께 삭제한다.
cascade_indexes	TRUE로 설정하면 테이블에 있는 인덱스의 통계 정보도 함께 삭제한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.

파라미터	설명
force	TRUE로 설정하면 테이블의 통계 정보가 잠겨 있어도 삭제한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.

45.3.11. DROP_EXTENDED_STATS

지정한 표현식에 대한 통계를 삭제하는 프러시저이다.

DROP_EXTENDED_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DROP_EXTENDED_STATS
(
    ownname     VARCHAR2,
    tabname     VARCHAR2,
    exprname    VARCHAR2
);
```

- 파라미터

파라미터	설명
ownname	지정한 표현식에 대한 통계가 있는 스키마의 이름이다.
tabname	지정한 표현식에 대한 통계가 있는 테이블의 이름이다.
exprname	삭제할 표현식이다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블을 입력한 경우이다.
20009	존재하지 않는 표현식을 입력한 경우이다.

45.3.12. DROP_STAT_TABLE

통계 테이블을 삭제하는 프러시저이다.

DROP_STAT_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.DROP_STAT_TABLE
(
    ownname     VARCHAR2,
    stattab     VARCHAR2
);
```

- 파라미터

파라미터	설명
ownname	통계 테이블이 속한 스키마의 이름이다.
stattab	통계 정보를 저장할 통계 테이블의 이름이다.

- 예외 상황

예외 상황	설명
7068	잘못된 스키마를 입력한 경우이다.

45.3.13. EXPORT_COLUMN_STATS

지정한 컬럼의 통계 정보를 통계 테이블에 저장하는 프러시저이다.

EXPORT_COLUMN_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.EXPORT_COLUMN_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    colname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.

파라미터	설명
tabname	테이블의 이름이다.
colname	컬럼의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20003	잘못된 컬럼을 입력한 경우이다.

45.3.14. EXPORT_DATABASE_STATS

데이터베이스에 있는 테이블의 통계 정보를 통계 테이블에 저장하는 프러시저이다.

EXPORT_DATABASE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.EXPORT_DATABASE_STATS
(
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20000	DBA 권한이 없는 경우이다.

45.3.15. EXPORT_INDEX_STATS

지정한 인덱스의 통계 정보를 통계 테이블에 저장하는 프러시저이다.

EXPORT_INDEX_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.EXPORT_INDEX_STATS
(
    ownname     VARCHAR2,
    indname     VARCHAR2,
    partname    VARCHAR2 DEFAULT NULL,
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
indname	인덱스의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20003	잘못된 인덱스를 입력한 경우이다.

45.3.16. EXPORT_SCHEMA_STATS

스키마 전체의 통계 정보를 통계 테이블에 저장하는 프러시저이다.

EXPORT_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.EXPORT_SCHEMA_STATS
(
    ownname     VARCHAR2,
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20003	잘못된 스키마를 입력한 경우이다.

45.3.17. EXPORT_SYSTEM_STATS

시스템의 통계 정보를 통계 테이블에 저장하는 프러시저이다.

EXPORT_SYSTEM_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.EXPORT_SYSTEM_STATS
(
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20000	DBA 권한이 없는 경우이다.

45.3.18. EXPORT_TABLE_STATS

테이블의 통계 정보를 통계 테이블에 저장하는 프러시저이다.

EXPORT_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.EXPORT_TABLE_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    partname     VARCHAR2,
    stattab      VARCHAR2,
    statid       VARCHAR2 DEFAULT NULL,
    cascade      BOOLEAN  DEFAULT TRUE,
    statown      VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
cascade	테이블이 갖는 컬럼, 인덱스, 파티션의 통계 정보를 저장한다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블을 입력한 경우이다.

45.3.19. GATHER_DATABASE_STATS

데이터베이스의 모든 객체의 통계 정보를 수집하는 프러시저이다.

GATHER_DATABASE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GATHER_DATABASE_STATS
(
    estimate_percent NUMBER   DEFAULT
                   to_estimate_percent_type(get_param('STAT_ESTIMATE_PERCENT'),
    block_sample     BOOLEAN  DEFAULT FALSE,
    method_opt       VARCHAR2 DEFAULT get_param('STAT_METHOD_OPT'),
    degree           NUMBER   DEFAULT to_degree_type (get_param('STAT_DEGREE')),
    cascade          BOOLEAN  DEFAULT to_boolean(get_param('STAT_CASCADE')),
    gather_sys       BOOLEAN  DEFAULT FALSE,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    options          VARCHAR2 DEFAULT 'GATHER',
    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
estimate_percent	샘플링할 로우의 퍼센트를 정의한다. 0.000001 이상 100 이하의 값을 설정할 수 있다. 0을 입력한 경우에는 optimizer가 테이블에 크기에 따라 적합한 sampling rate을 결정하여 수행한다.
block_sample	estimate_percent가 100보다 작을 때 로우 샘플링 대신 블록 샘플링을 사용할 수 있다. 블록 샘플링 방식은 로우 샘플링에 비해 표본이 고르게 뽑히지 않기 때문에 estimate_percent가 충분히 크지 않을 때는 기본값인 FALSE로 두어야 한다.
method_opt	히스토그램 생성에 대한 옵션으로 설정 방법은 [method_opt 파라미터 설정 방법]을 참고한다.
degree	병렬 쿼리의 처리 개수를 설정한다.
cascade	인덱스에 대해서도 통계 정보를 수집한다.
gather_sys	SYS 사용자의 테이블에 대해서도 통계 정보를 수집한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
options	수집할 테이블의 기준을 정한다. – GATHER : 모든 테이블에 대하여 수집한다.

파라미터	설명
	– GATHER AUTO : 자체적인 기준에 의하여 수집이 필요한 테이블들만 수집한다. – GATHER EMPTY : 통계 정보가 없는 테이블들만 수집한다. – GATHER STALE : 마지막 통계 정보 수집 이후 전체 대비 10% 이상의 로우들이 변경된 테이블들만 수집한다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 수집한다.

[method_opt 파라미터 설정방법]

다음과 같은 형식으로 method_opt 파라미터를 설정할 수 있다.

– FOR ALL

```
FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]
```

각 옵션에 대한 설명은 다음과 같다.

옵션	설명
INDEXED	인덱스된 컬럼의 통계 정보를 수집한다.
HIDDEN	실제 컬럼이 아닌 functional index로 생성된 컬럼에 대해서 통계 정보를 수집한다.
size_clause	히스토그램을 생성할 때 길이를 설정하는 데 필요한 절이다.

– FOR COLUMNS

```
FOR COLUMNS [size_clause] column [size_clause] [,column [size_clause]...]
```

각 옵션에 대한 설명은 다음과 같다.

옵션	설명
size_clause	히스토그램을 생성할 때 길이를 설정하는 데 필요한 절이다.

– size_clause

```
size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}
```

각 항목에 대한 설명은 다음과 같다.

항목	설명
integer	히스토그램 버킷의 수 이다. 1~256개까지 설정할 수 있다.

항목	설명
REPEAT	기존에 히스토그램이 있는 경우 버킷 수를 그대로 사용하고, 히스토그램이 없다면 AUTO와 같이 버킷 수를 설정한다.
AUTO	시스템에서 자동으로 값을 결정한다.
SKEWONLY	데이터 분포가 한쪽으로 치우친 분포에 맞게 값을 결정한다.

- 예외 상황

예외 상황	설명
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

45.3.20. GATHER_DICTIONARY_STATS

모든 데이터 사전의 스키마(SYS, SYSCAT) 객체의 통계 정보를 수집하는 프러시저이다.

GATHER_DICTIONARY_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GATHER_DICTIONARY_STATS
(
    estimate_percent NUMBER   DEFAULT
                   to_estimate_percent_type(get_param('STAT_ESTIMATE_PERCENT'),
    block_sample     BOOLEAN  DEFAULT FALSE,
    method_opt       VARCHAR2 DEFAULT get_param('STAT_METHOD_OPT'),
    degree           NUMBER   DEFAULT to_degree_type (get_param('STAT_DEGREE')),
    cascade          BOOLEAN  DEFAULT to_boolean(get_param('STAT_CASCADE')),
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
estimate_percent	샘플링할 로우의 퍼센트를 정의한다. 0.000001 이상 100 이하의 값을 설정할 수 있다. 0을 입력한 경우에는 optimizer가 테이블에 크기에 따라 적합한 sampling rate을 결정하여 수행한다.
block_sample	estimate_percent가 100보다 작을 때 로우 샘플링 대신 블록 샘플링을 사용할 수 있다. 블록 샘플링 방식은 로우 샘플링에 비해 표본이 고르게 뽑히지 않

파라미터	설명
	기 때문에 estimate_percent가 충분히 크지 않을 때는 기본값인 FALSE로 두어야 한다.
method_opt	히스토그램 생성에 대한 옵션이다. 파라미터를 설정하는 방법은 [method_opt 파라미터 설정방법]을 참고한다.
degree	병렬 쿼리의 처리 개수를 설정한다.
cascade	인덱스에 대해서도 통계 정보를 수집한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 수집한다.

- 예외 상황

예외 상황	설명
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

45.3.21. GATHER_INDEX_STATS

인덱스의 통계 정보를 수집하는 프러시저이다.

GATHER_INDEX_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GATHER_INDEX_STATS
(
    ownname          VARCHAR2,
    idxname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    estimate_percent NUMBER DEFAULT to_estimate_percent_type (
                     get_param ('STAT_ESTIMATE_PERCENT')),
    degree           NUMBER DEFAULT to_degree_type(get_param('STAT_DEGREE')),
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.

파라미터	설명
idxname	인덱스의 이름이다.
partname	파티션의 이름이다.
estimate_percent	샘플링할 로우의 퍼센트를 정의한다. 0.000001 이상 100 이하의 값을 설정할 수 있다. 0을 입력한 경우에는 optimizer가 테이블에 크기에 따라 적합한 sampling rate을 결정하여 수행한다.
degree	병렬 쿼리의 처리 개수를 설정한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 인덱스의 통계 정보가 잠겨 있어도 수집한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.

45.3.22. GATHER_SCHEMA_STATS

스키마의 모든 객체의 통계 정보를 수집하는 프러시저이다.

GATHER_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GATHER_SCHEMA_STATS
(
    ownname          VARCHAR2,
    estimate_percent NUMBER   DEFAULT
                   to_estimate_percent_type(get_param('STAT_ESTIMATE_PERCENT'),
    block_sample     BOOLEAN  DEFAULT FALSE,
    method_opt       VARCHAR2 DEFAULT get_param('STAT_METHOD_OPT'),
    degree           NUMBER   DEFAULT to_degree_type (get_param('STAT_DEGREE')),
    cascade          BOOLEAN  DEFAULT to_boolean(get_param('STAT_CASCADE')),
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.

파라미터	설명
estimate_percent	샘플링할 로우의 퍼센트를 정의한다. 0.000001 이상 100 이하의 값을 설정할 수 있다. 0을 입력한 경우에는 optimizer가 테이블에 크기에 따라 적합한 sampling rate을 결정하여 수행한다.
block_sample	estimate_percent가 100보다 작을 때 로우 샘플링 대신 블록 샘플링을 사용할 수 있다. 블록 샘플링 방식은 로우 샘플링에 비해 표본이 고르게 뽑히지 않기 때문에 estimate_percent가 충분히 크지 않을 때는 기본값인 FALSE로 두어야 한다.
method_opt	히스토그램 생성에 대한 옵션이다. 파라미터를 설정하는 방법은 [method_opt 파라미터 설정방법]을 참고한다.
degree	병렬 쿼리의 처리 개수를 설정한다.
cascade	인덱스에 대해서도 통계 정보를 수집한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 수집한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

45.3.23. GATHER_SYSTEM_STATS

시스템의 통계 정보를 수집하는 프러시저이다. 이 프러시저를 사용하려면 DBA 권한이 필요하다.

GATHER_SYSTEM_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GATHER_SYSTEM_STATS
(
    gathering_mode     VARCHAR2 DEFAULT 'NOWORKLOAD',
    interval           NUMBER   DEFAULT 60,
    limit              VARCHAR2 DEFAULT '3G'
);
```

- 파라미터

파라미터	설명
gathering_mode	NOWORKLOAD, INTERVAL, START/STOP 옵션을 지원한다. 각 옵션에 대한 설명은 다음의 [gathering_mode 파라미터 설정 옵션]을 참조한다.
interval	gathering_mode가 INTERVAL일 때 시스템의 성능을 측정하는 시간을 분 단위로 나타낸다. (기본값: 60분)
limit	gathering_mode가 NOWORKLOAD일 때 임의의 입출력을 수행해 볼 최대 데이터양을 Byte 단위로 나타낸다. 이때 단위는 K(Kilo Byte), M(Mega Byte), G(Giga Byte), T(Tera Byte) 등을 사용할 수 있다. 캐시 버퍼(데이터베이스의 캐시, 운영체제의 캐시 등)의 크기보다 큰 값을 사용해야 좀 더 정확한 정보를 얻을 수 있다. (기본값: 3G)

[gathering_mode 파라미터 설정 옵션]

gathering_mode 파라미터에 설정할 수 있는 각 옵션에 대한 설명은 다음과 같다.

옵션	설명
NOWORKLOAD	시스템의 CPU와 입출력 성능을 측정한다. 즉 시스템의 통계 정보 중 CPUSPEED, SEEKTM, TRFSPEED를 수집한다. – 시스템의 통계 정보는 데이터 파일을 통해 수집되기 때문에 데이터베이스와 테이블 스페이스를 모두 생성하고서, 해당 정보를 수집할 것을 권장한다. – 데이터베이스와 테이블 스페이스를 모두 생성한 후 시스템의 통계 정보를 수집하기를 강력히 권장한다. – 서버 부하에 따라 보다 자세한 시스템의 통계 정보를 설정하려면, gathering_mode 파라미터의 옵션 중 START 또는 STOP 또는 INTERVAL 파라미터를 사용해야 한다. – noworkload와 workload 시스템의 통계 정보를 모두 수집하는 경우에는 workload 시스템의 통계 정보가 최적화기(Optimizer)에서 사용된다. 이 모드에서 시스템의 통계 정보를 수집하면 데이터베이스 크기에 따라 몇 분이 걸릴 수 있다. 이를 해결하기 위해 limit 파라미터를 사용하여 수집 시간을 조절할 수 있다.
INTERVAL	interval 파라미터에 설정된 시간 동안 시스템의 성능을 측정한다. workload 시스템의 통계 정보인 SBLKRDTM, MBLKRDTM, MBLKRDCNT를 수집한다.

옵션	설명
	필요하다면 gathering_mode 파라미터의 옵션을 STOP으로 설정하여 interval 파라미터에 설정한 시간이 모두 지나가기 전에 수집을 일찍 마칠 수 있다.
START, STOP	START와 STOP 옵션 간의 시스템 성능을 측정한다. workload 시스템의 통계 정보인 SBLKRDTM, MBLKRDTM, MBLKRDCNT를 수집한다.

- 예외 상황

예외 상황	설명
20000	권한이 부족한 경우이다.
20023	잘못된 gathering_mode 파라미터(NOWORKLOAD, INTERVAL, START, STOP만 가능)를 사용한 경우이다.
20026	잘못된 파라미터 값을 입력한 경우이다.
20027	_DD_AUX_STATS 통계 테이블의 내용이 이상한 경우이다.

45.3.24. GATHER_TABLE_STATS

테이블의 통계 정보를 수집하는 프러시저이다.

GATHER_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GATHER_TABLE_STATS
(
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    partname         VARCHAR2 DEFAULT NULL,
    estimate_percent NUMBER   DEFAULT
                  to_estimate_percent_type(get_param('STAT_ESTIMATE_PERCENT'),
    block_sample     BOOLEAN  DEFAULT FALSE,
    method_opt       VARCHAR2 DEFAULT get_param('STAT_METHOD_OPT'),
    degree           NUMBER   DEFAULT to_degree_type (get_param('STAT_DEGREE')),
    cascade_indexes  BOOLEAN  DEFAULT to_boolean(get_param('STAT_CASCADE')),
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    granularity      VARCHAR2 DEFAULT 'ALL',
    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 현재 스키마의 이름이다.
tabname	테이블의 이름이다.
partname	파티션의 이름이다.
estimate_percent	샘플링할 로우의 퍼센트를 정의한다. 0.000001 이상 100 이하의 값을 설정할 수 있다. 0을 입력한 경우에는 optimizer가 테이블에 크기에 따라 적합한 sampling rate을 결정하여 수행한다.
block_sample	estimate_percent가 100보다 작을 때 로우 샘플링 대신 블록 샘플링을 사용할 수 있다. 블록 샘플링 방식은 로우 샘플링에 비해 표본이 고르게 뽑히지 않기 때문에 estimate_percent가 충분히 크지 않을 때는 기본값인 FALSE로 두어야 한다.
method_opt	히스토그램 생성에 대한 옵션이다. 파라미터를 설정하는 방법은 [method_opt 파라미터 설정방법]을 참고한다.
degree	병렬 쿼리의 처리 개수를 설정한다.
cascade_indexes	인덱스에 대해서도 통계 정보를 수집한다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
granularity	통계 정보를 수집하는 대상을 가리킨다. – ALL : 테이블 및 파티션의 통계 정보를 수집한다. – GLOBAL : 파티션을 제외한 테이블의 통계 정보만 수집한다. – PARTITION : 파티션 수준의 통계 정보만 수집한다.
force	TRUE로 설정하면 테이블의 통계 정보가 잠겨 있어도 수집한다.

- 예외 상황

예외 상황	설명
OBJECT_NOT_EXISTS	객체가 존재하지 않는 경우이다.
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

45.3.25. GET_COLUMN_STATS

데이터 사전(DD: Data Dictionary) 또는 통계 테이블에 있는 지정한 컬럼의 통계 정보를 가져오는 프러시저이다.

GET_COLUMN_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GET_COLUMN_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    colname      VARCHAR2,
    stattab      VARCHAR2 DEFAULT NULL,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL,
    distcnt      OUT      NUMBER,
    density      OUT      NUMBER,
    nullcnt      OUT      NUMBER,
    avgclen      OUT      NUMBER
);
```

- 파라미터

파라미터	**설명**
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
colname	컬럼의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
distcnt	컬럼 내 서로 다른 값들의 개수이다.
density	컬럼 값들의 밀도이다.
nullcnt	컬럼 내 NULL 개수이다.
avgclen	byte 단위로 나타낸 컬럼의 평균길이이다.

- 예외 상황

예외 상황	**설명**
20003	잘못된 컬럼을 입력한 경우이다.
20025	잘못된 통계 테이블을 입력한 경우이다.

45.3.26. GET_INDEX_STATS

데이터 사전(DD: Data Dictionary) 또는 통계 테이블에 있는 지정한 인덱스의 통계 정보를 가져오는 프러시저이다.

GET_INDEX_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GET_INDEX_STATS
(
    ownname      VARCHAR2,
    idxname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2 DEFAULT NULL,
    statid       VARCHAR2 DEFAULT NULL,
    statown      GET_TABLE_STATSVARCHAR2 DEFAULT NULL,
    numrows      OUT   NUMBER,
    numlblks     OUT   NUMBER,
    numdist      OUT   NUMBER,
    clstfct      OUT   NUMBER,
    idxlevel     OUT   NUMBER
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
idxname	인덱스의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
numrows	인덱스의 로우 개수이다.
numblks	인덱스의 블록 개수이다.
numlblks	인덱스의 리프블록 개수이다.
numdist	인덱스 내 서로 다른 값들의 개수이다.
clstfct	인덱스의 클러스터링 팩터이다. 0에서 1 사이의 값으로 정규화되어 있다.
idxlevel	인덱스의 높이이다.

- 예외 상황

예외 상황	설명
20007	잘못된 인덱스를 입력한 경우이다.
20003	잘못된 파티션을 입력한 경우이다.
20004	파티션 인덱스가 아닌데 파티션 이름을 지정한 경우이다.
20025	잘못된 통계 테이블을 입력한 경우이다.

45.3.27. GET_TABLE_STATS

데이터 사전(DD: Data Dictionary) 또는 통계 테이블에 있는 지정된 테이블의 통계 정보를 가져오는 프러시저이다.

GET_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GET_TABLE_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2 DEFAULT NULL,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL,
    numrows      OUT NUMBER,
    numblks      OUT NUMBER,
    avgrlen      OUT NUMBER
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
numrows	테이블의 로우 개수이다.
numblks	테이블의 블록 개수이다.
avgrlen	테이블의 평균 로우 길이이다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블 혹은 파티션을 입력한 경우이다.
20004	파티션 테이블이 아닌데 파티션을 입력한 경우이다.
20025	잘못된 통계 테이블을 입력한 경우이다.

45.3.28. IMPORT_COLUMN_STATS

통계 테이블에 있는 지정한 컬럼의 통계 정보를 데이터 사전에 저장하는 프러시저이다.

IMPORT_COLUMN_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.IMPORT_COLUMN_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    colname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force             BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
colname	컬럼의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 컬럼 통계 정보가 잠겨있어도 저장한다.

- 예외 상황

예외 상황	설명
20003	잘못된 컬럼을 입력한 경우이다.

45.3.29. IMPORT_DATABASE_STATS

통계 테이블에 있는 모든 테이블의 통계 정보를 데이터 사전에 저장하는 프러시저이다.

IMPORT_DATABASE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.IMPORT_DATABASE_STATS
(
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 저장한다.

- 예외 상황

예외 상황	설명
20000	DBA 권한이 없는 경우이다.

45.3.30. IMPORT_INDEX_STATS

통계 테이블에 있는 지정한 인덱스의 통계 정보를 데이터 사전에 저장하는 프러시저이다.

IMPORT_INDEX_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.IMPORT_INDEX_STATS
(
    ownname     VARCHAR2,
    indname     VARCHAR2,
    partname    VARCHAR2 DEFAULT NULL,
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
indname	인덱스의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 인덱스의 통계 정보가 잠겨있어도 저장한다.

- 예외 상황

예외 상황	설명
20003	잘못된 인덱스를 입력한 경우이다.

45.3.31. IMPORT_SCHEMA_STATS

통계 테이블에 있는 스키마 전체의 통계 정보를 데이터 사전에 저장하는 프러시저이다.

IMPORT_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.IMPORT_SCHEMA_STATS
(
    ownname     VARCHAR2,
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 저장한다.

- 예외 상황

예외 상황	설명
20003	잘못된 스키마를 입력한 경우이다.

45.3.32. IMPORT_SYSTEM_STATS

통계 테이블에 있는 시스템의 통계 정보를 데이터 사전에 저장하는 프러시저이다.

IMPORT_SYSTEM_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.IMPORT_SYSTEM_STATS
(
    stattab     VARCHAR2,
    statid      VARCHAR2 DEFAULT NULL,
    statown     VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.

- 예외 상황

예외 상황	설명
20000	DBA 권한이 없는 경우이다.

45.3.33. IMPORT_TABLE_STATS

통계 테이블에 있는 테이블의 통계 정보를 데이터 사전에 저장하는 프러시저이다.

IMPORT_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.IMPORT_TABLE_STATS
(
    ownname       VARCHAR2,
    tabname       VARCHAR2,
    partname      VARCHAR2,
    stattab       VARCHAR2,
    statid        VARCHAR2 DEFAULT NULL,
    cascade       BOOLEAN  DEFAULT TRUE,
    statown       VARCHAR2 DEFAULT NULL,
    no_invalidate    BOOLEAN  DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),

    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
cascade	테이블이 갖는 컬럼, 인덱스, 파티션의 통계 정보를 저장한다.

파라미터	설명
statown	통계 테이블이 속한 스키마의 이름이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 테이블의 통계 정보가 잠겨있어도 저장한다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블을 입력한 경우이다.

45.3.34. LOCK_TABLE_STATS

해당 테이블과 소속된 인덱스, 파티션, 컬럼의 통계 정보들을 잠그는 프러시저이다. 잠겨진 통계 정보는 수집 혹은 삭제, 복사가 불가능하다.

단일 테이블에 대한 수집, 삭제, 복사 (GATHER_TABLE_STATS, DELETE_TABLE_STATS 등)에 대해서는 에러가 발생하고 , 여러 객체에 대한 수집, 삭제, 복사 (GATHER_SCHEMA_STATS, DELETE_DATABASE_STATS 등)에 대해서는 건너뛰어 진행한다.

LOCK_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.LOCK_TABLE_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블을 입력한 경우이다.

45.3.35. LOCK_SCHEMA_STATS

해당 스키마에 소속된 테이블, 인덱스, 파티션, 컬럼의 통계 정보들을 잠그는 프러시저이다. 잠겨진 통계 정보는 수집 혹은 삭제, 복사가 불가능하다.

단일 테이블에 대한 수집, 삭제, 복사 (GATHER_TABLE_STATS, DELETE_TABLE_STATS 등)에 대해서는 에러가 발생하고 , 여러 객체에 대한 수집, 삭제, 복사 (GATHER_SCHEMA_STATS, DELETE_DATABASE_STATS 등)에 대해서는 건너뛰어 진행한다.

LOCK_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.LOCK_SCHEMA_STATS
(
    ownname      VARCHAR2
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다.

- 예외 상황

예외 상황	설명
20002	잘못된 스키마를 입력한 경우이다.

45.3.36. SET_COLUMN_STATS

데이터 사전(DD: Data Dictionary) 또는 통계 테이블에 사용자가 입력한 지정된 컬럼에 대한 임의의 통계 정보를 기록하는 프러시저이다.

SET_COLUMN_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.SET_COLUMN_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    colname      VARCHAR2,
    stattab      VARCHAR2 DEFAULT NULL,
    statid       VARCHAR2 DEFAULT NULL,
```

```
    statown      VARCHAR2 DEFAULT NULL,
    distcnt      NUMBER,
    density      NUMBER,
    nullcnt      NUMBER,
    avgclen      NUMBER,
    no_invalidate  BOOLEAN DEFAULT to_boolean(SET_param('STAT_NO_INVALIDATE')),
    force          BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
colname	컬럼의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
distcnt	컬럼 내 서로 다른 값들의 개수이다.
density	컬럼 값들의 밀도이다.
nullcnt	컬럼 내 NULL 개수이다.
avgclen	byte 단위로 나타낸 컬럼의 평균길이이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 컬럼 통계 정보가 잠겨있어도 저장한다.

- 예외 상황

예외 상황	설명
20003	잘못된 컬럼을 입력한 경우이다.
20025	잘못된 통계 테이블을 입력한 경우이다.

45.3.37. SET_INDEX_STATS

데이터 사전(DD: Data Dictionary) 또는 통계 테이블에 사용자가 입력한 지정된 인덱스에 대한 임의의 통계 정보를 기록하는 프러시저이다.

SET_INDEX_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.SET_INDEX_STATS
(
    ownname      VARCHAR2,
    idxname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2 DEFAULT NULL,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL,
    numrows      NUMBER,
    numlblks     NUMBER,
    numdist      NUMBER,
    clstfct      NUMBER,
    idxlevel     NUMBER,
    no_invalidate   BOOLEAN  DEFAULT to_boolean(SET_param('STAT_NO_INVALIDATE')),
    force           BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
idxname	인덱스의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
numrows	인덱스의 로우 개수이다.
numblks	인덱스의 블록 개수이다.
numlblks	인덱스의 리프블록 개수이다.
numdist	인덱스 내 서로 다른 값들의 개수이다.
clstfct	인덱스의 클러스터링 팩터이다. 0에서 1 사이의 값으로 정규화되어 있다.
idxlevel	인덱스의 높이이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 인덱스의 통계 정보가 잠겨있어도 저장한다.

- 예외 상황

예외 상황	설명
20003	잘못된 파티션을 입력한 경우이다.
20004	파티션 인덱스가 아닌데 파티션 이름을 입력한 경우이다.

예외 상황	설명
20007	해당 인덱스가 비활성화되어 있거나 통계 정보를 저장할 수 없는 형식의 인덱스이다.
20025	잘못된 통계 테이블을 입력한 경우이다.

45.3.38. SET_TABLE_STATS

데이터 사전(DD: Data Dictionary) 또는 통계 테이블에 사용자가 입력한 지정된 테이블에 대한 임의의 통계 정보를 기록하는 프러시저이다.

SET_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.SET_TABLE_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2,
    partname     VARCHAR2 DEFAULT NULL,
    stattab      VARCHAR2 DEFAULT NULL,
    statid       VARCHAR2 DEFAULT NULL,
    statown      VARCHAR2 DEFAULT NULL,
    numrows      NUMBER,
    numblks      NUMBER,
    avgrlen      NUMBER,
    no_invalidate  BOOLEAN  DEFAULT to_boolean(SET_param('STAT_NO_INVALIDATE')),
    force          BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.
partname	파티션의 이름이다.
stattab	통계 테이블의 이름이다.
statid	통계 테이블 내에서 구분할 통계 정보 ID이다.
statown	통계 테이블이 속한 스키마의 이름이다.
numrows	테이블의 로우 개수이다.
numblks	테이블의 블록 개수이다.

파라미터	설명
avgrlen	테이블의 평균 로우 길이이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 테이블의 통계 정보가 잠겨있어도 수정한다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블 혹은 파티션을 입력한 경우이다.
20004	해당 테이블은 파티션된 테이블이 아니다.
20025	잘못된 통계 테이블을 입력한 경우이다.

45.3.39. PURGE_STATS

지정된 시점보다 오래된 통계 정보 히스토리들을 제거하는 프러시저이다. 이 프러시저 수행을 하려면 SYSDBA 혹은 ANALYZE ANY 권한이 필요하다.

PURGE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.PURGE_STATS
(
    before_timestamp    IN          TIMESTAMP WITH TIME ZONE
);
```

- 파라미터

파라미터	설명
before_timestamp	– NULL이 아닌 값을 지정할 경우 지정된 시점보다 이전의 히스토리들은 삭제된다. – NULL일 경우 GET_STATS_HISTORY_RETENTION 함수에서 조회되는 보유 기간값에 맞춰서 삭제된다.

45.3.40. RESTORE_SCHEMA_STATS

스키마의 모든 객체의 통계 정보를 지정된 시간대(as_of_timestamp)로 복원하는 프러시저이다.

RESTORE_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.RESTORE_SCHEMA_STATS
(
    ownname          VARCHAR2,
    as_of_timestamp  TIMESTAMP WITH TIME ZONE,
    no_invalidate    BOOLEAN DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
as_of_timestamp	복원할 통계 정보의 시간대이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 잠겨 있는 테이블 및 관련 객체들의 통계 정보들도 복원한다.

45.3.41. RESTORE_TABLE_STATS

테이블의 통계 정보를 지정된 시간대(as_of_timestamp)로 복원하는 프러시저이다. 이 프러시저는 연관된 인덱스와 컬럼들의 통계 정보들까지 복원한다.

RESTORE_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.RESTORE_TABLE_STATS
(
    ownname          VARCHAR2,
    tabname          VARCHAR2,
    as_of_timestamp TIMESTAMP WITH TIME ZONE,
    no_invalidate    BOOLEAN DEFAULT to_boolean(get_param('STAT_NO_INVALIDATE')),
    force            BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.

파라미터	설명
tabname	테이블의 이름이다.
as_of_timestamp	복원할 통계 정보의 시간대이다.
no_invalidate	TRUE로 설정하면 관련된 Physical Plan을 삭제하지 않는다.
force	TRUE로 설정하면 테이블의 통계 정보가 잠겨 있어도 복원한다.

45.3.42. SET_PARAM

DBMS_STATS 패키지의 디폴트 파라미터의 값을 변경하는 프러시저이다.

SET_PARAM 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.SET_PARAM
(
   pname      VARCHAR2,
   pval       VARCHAR2
);
```

- 파라미터

파라미터	설명
pname	파라미터의 이름이다.
pval	파라미터의 설정 값이다. – NULL로 설정한 경우 VT_PARAMETER의 값을 참조하여 Tibero 를 설치할 때 기본값으로 변경된다. – NULL로 설정하지 않은 경우 다음과 같은 옵션을 설정한다. • STAT_CASCADE : DBMS_STATS 패키지에서 사용하는 cascade 파라미터의 기본값이다. (설치 기본값: Y) • STAT_METHOD_OPT : DBMS_STATS 패키지에서 사용하는 method_opt 파라미터의 기본값이다. (설치 기본값: FOR ALL COLUMNS SIZE 32) • STAT_NO_INVALIDATE : DBMS_STATS 패키지에서 사용하는 no_invalidate 파라미터의 기본값이다. (설치 기본값: N) • STAT_ESTIMATE_PERCENT : DBMS_STATS 패키지에서 사용하는 estimate_percent 파라미터의 기본값이다. (설치 기본값: 0)

파라미터	설명
	• STAT_DEGREE : DBMS_STATS 패키지에서 사용하는 degree 파라미터의 기본값이다. (설치 기본값: 1)

- 예외 상황

예외 상황	설명
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

45.3.43. SET_SYSTEM_STATS

시스템의 통계 값을 수동으로 설정하는 프러시저이다.

SET_SYSTEM_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.SET_SYSTEM_STATS
(
    pname      VARCHAR2,
    pvalue     VARCHAR2
);
```

- 파라미터

파라미터	설명
pname	파라미터의 이름이다.
pvalue	파라미터의 설정 값이다. 다음은 파라미터에 대한 설명이다. – CPUSPEED : 1μs 당 평균 CPU 사이클 수(Mhz)이다. – SEEKTM : 평균 입출력 탐색 시간(I/O seek time)이다. (단위: ms) – TRFSPEED : 평균 입출력 전송 속도(I/O transfer speed)이다. (단위: byte/ms) – SBLKRDTM : 임의의 단일 블록을 읽는 데 걸리는 평균 시간이다. (단위: ms) – MBLKRDTM : 연속된 BLKRDCNT 블록을 읽는 데 걸리는 평균 시간이다. (단위: ms)

파라미터	설명
	– BLKRDCNT : 연속된 여러 블록을 읽을 때의 평균 블록 수이다.

- 예외 상황

예외 상황	설명
20000	DBA 권한이 없는 경우이다.
20021	잘못된 파라미터 값을 입력한 경우이다.
20022	잘못된 파라미터 이름을 입력한 경우이다.

45.3.44. UNLOCK_TABLE_STATS

해당 테이블과 소속된 인덱스, 파티션, 컬럼의 통계 정보들에 대한 잠금을 해제하는 프러시저이다. 잠금이 해지된 통계 정보는 수집 혹은 삭제, 복사가 다시 가능해진다.

UNLOCK_TABLE_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.UNLOCK_TABLE_STATS
(
    ownname      VARCHAR2,
    tabname      VARCHAR2
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다. NULL일 경우 이 프러시저를 호출한 스키마의 이름이다.
tabname	테이블의 이름이다.

- 예외 상황

예외 상황	설명
20003	잘못된 테이블을 입력한 경우이다.

45.3.45. UNLOCK_SCHEMA_STATS

해당 스키마에 소속된 테이블, 인덱스, 파티션, 컬럼의 통계 정보들에 대한 잠금을 해제하는 프러시저이다. 잠금이 해지된 통계 정보는 수집 혹은 삭제, 복사가 다시 가능해진다.

UNLOCK_SCHEMA_STATS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.UNLOCK_SCHEMA_STATS
(
    ownname      VARCHAR2
);
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다.

- 예외 상황

예외 상황	설명
20002	잘못된 스키마를 입력한 경우이다.

45.4. 함수

본 절에서는 DBMS_STATS 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

45.4.1. CREATE_EXTENDED_STATS

지정한 표현식에 대한 통계를 생성하는 함수이다.

CREATE_EXTENDED_STATS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.CREATE_EXTENDED_STATS
(
    ownname     VARCHAR2,
    tabname     VARCHAR2,
    exprname    VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ownname	스키마의 이름이다.

파라미터	설명
tabname	테이블의 이름이다.
exprname	통계정보를 생성할 표현식이다.

- 예외 상황

예외 상황	설명
8026	잘못된 표현식을 입력한 경우이다.
20003	잘못된 테이블을 입력한 경우이다.
20009	이미 존재하는 표현식을 입력한 경우이다.

45.4.2. GET_PARAM

DBMS_STATS 패키지의 디폴트 파라미터의 값을 반환하는 함수이다.

GET_PARAM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GET_PARAM
(
    pname     VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
pname	파라미터의 이름이다.

- 예외 상황

예외 상황	설명
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

45.4.3. GET_STATS_HISTORY_AVAILABILITY

사용 가능한 가장 오래된 통계 정보 히스토리의 저장 시점을 반환한다. 사용자는 통계 정보를 이 시점보다 예전으로 복원할 수 없다.

GET_STATS_HISTORY_AVAILABILITY 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY
RETURN TIMESTAMP WITH TIME ZONE;
```

45.4.4. GET_STATS_HISTORY_RETENTION

현재 설정된 통계 정보 히스토리의 최대 보유 기간을 반환한다.

GET_STATS_HISTORY_RETENTION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.GET_STATS_HISTORY_RETENTION
RETURN NUMBER;
```

45.4.5. TO_BOOLEAN

TRUE, FALSE 문자열을 BOOLEAN 타입의 TRUE, FALSE 값으로 변경하여 반환하는 함수이다.

TO_BOOLEAN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_STATS.TO_BOOLEAN
(
    strval     VARCHAR2
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
strval	TRUE 또는 FALSE 문자열 값이다.

- 예외 상황

예외 상황	설명
BAD_INPUT_VALUE	잘못된 파라미터 값을 입력한 경우이다.

제46장 DBMS_SYSTEM

본 장에서는 DBMS_SYSTEM 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

46.1. 개요

DBMS_SYSTEM은 특정 경우에 대한 시스템적인 제어에 유용한 프러시저 및 함수(들)을 제공한다.

SYS 권한이 있는 사용자만 이 패키지를 사용할 수 있다.

46.2. 프러시저와 함수

본 절에서는 DBMS_SYSTEM 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

46.2.1. SET_SQL_TRACE_IN_SESSION

특정 세션의 SQL 추적 로그 작성을 시작하거나 중지할 수 있다. 세션의 식별자 및 시리얼 번호는 V$SESSION 뷰를 통해 조회할 수 있다.

SQL 추적 로그는 $TB_HOME/instance/$TB_SID/log/sqltrace 경로에 생성된다.

SET_SQL_TRACE_IN_SESSION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION
(
    sid                 IN      NUMBER,
    serial#             IN      NUMBER,
    sql_trace           IN      BOOLEAN
);
```

- 파라미터

파라미터	설명
sid	세션의 식별자이다.
serial#	세션의 시리얼 번호이다.
sql_trace	SQL 추적 로그를 작성하려면 true, 중지하려면 false를 입력한다.

- 예제

```
begin
    for c in (select sid, serial# from v$session where username = 'TIBERO') loop
        DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(c.sid, c.serial#, true);
    end loop;
end;
/
```

제47장 DBMS_TPR

본 장에서는 DBMS_TPR 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다. 보다 자세한 설명은 "Tibero 관리자 안내서"의 "제17장 Tibero Performance Repository"를 참조한다.

47.1. 개요

DBMS_TPR 패키지는 Tibero Performance Repository(이하 TPR) 기능을 사용하는데 필요로 하는 기능을 제공하기 위한 패키지이다.

47.2. 프러시저

본 절에서는 DBMS_TPR 패키지에서 제공하는 프러시저를 설명한다.

47.2.1. CREATE_SNAPSHOT

TPR을 위한 스냅샷을 바로 남긴다.

CREATE_SNAPSHOT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CREATE_SNAPSHOT();
```

- 예제

```
exec DBMS_TPR.CREATE_SNAPSHOT();
/
```

47.2.2. CREATE_SNAPSHOT_ALL

TAC 구성에서 모든 인스턴스에서 TPR을 위한 스냅샷을 바로 남긴다. 이때, 기존에 각 스냅샷별로 매겨지던 스냅샷 ID에 추가로, 모든 인스턴스에서 함께 떨어진 스냅샷을 지칭하기 위해 글로벌 스냅샷 ID가 추가로 매겨진다.

CREATE_SNAPSHOT_ALL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CREATE_SNAPSHOT_ALL();
```

- 예제

```
exec DBMS_TPR.CREATE_SNAPSHOT_ALL();
/
```

47.2.3. REPORT_TEXT

특정 구간의 성능 분석 리포트(text)를 생성한다.

REPORT_TEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_TEXT
(
    begin_time       IN DATE
    end_time         IN DATE
    instance_no      IN VARCHAR
    file_name        IN VARCHAR
);
```

- 파라미터

파라미터	**설명**
begin_time	성능 분석 리포트 대상 구간 중 시작 시간이다.
end_time	성능 분석 리포트 대상 구간 중 종료 시간이다.
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

- 예제

아래의 예제는 두 개의 인자 모두 DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력해야 한다.

```
exec DBMS_TPR.REPORT_TEXT('2011/07/01 12:00:00', '2011/07/02 11:59:59');
/
```

파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_TEXT('2011/07/01 12:00:00',
'2011/07/02 11:59:59', file_name=>'tpr.txt');
/
```

47.2.4. REPORT_TEXT_ID

특정 스냅샷 ID 혹은 ID 구간의 성능 분석 리포트(text)를 생성한다.

REPORT_TEXT_ID 프러시저는 input이 특정 ID 혹은 ID 구간 두 종류로 나뉘어지며, 그 세부 내용은 다음과 같다.

- 특정 ID
  - 프로토타입

```
PROCEDURE REPORT_TEXT_ID
(
    one_snap_id       IN NUMBER
    file_name         IN VARCHAR
);
```

  - 파라미터

파라미터	설명
one_snap_id	성능 분석 리포트 대상 스냅샷 ID이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

  - 예제

```
exec DBMS_TPR.REPORT_TEXT_ID(5);
/
```

- ID 구간
  - 프로토타입

```
PROCEDURE REPORT_TEXT_ID
(
    begin_snap_id       IN NUMBER
    end_snap_id         IN NUMBER
    instance_no         IN VARCHAR
    file_name           IN VARCHAR
);
```

– 파라미터

파라미터	설명
begin_snap_id	성능 분석 리포트 대상 구간 중 시작 스냅샷 ID이다.
end_snap_id	성능 분석 리포트 대상 구간 중 종료 스냅샷 ID이다.
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

– 예제

파라미터 중 instance_no를 입력하지 않으면 기본적으로 모든 인스턴스를 대상으로 하는 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_TEXT_ID(5, 10);
/
```

파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_TEXT_ID(5, 10, instance_no=>'2');
/
```

47.2.5. REPORT_TEXT_GID

특정 스냅샷 GID 혹은 GID 구간의 성능 분석 리포트(text)를 생성한다.

REPORT_TEXT_GID 프러시저는 input이 특정 GID 혹은 GID 구간 두 종류로 나뉘어지며, 그 세부 내용은 다음과 같다.

- 특정 GID

  – 프로토타입

```
PROCEDURE REPORT_TEXT_GID
(
    one_snap_gid       IN NUMBER
    file_name          IN VARCHAR
);
```

  – 파라미터

파라미터	설명
one_snap_gid	성능 분석 리포트 대상 스냅샷의 GID이다.

파라미터	설명
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

– 예제

```
exec DBMS_TPR.REPORT_TEXT_GID(3);
/
```

- GID 구간

– 프로토타입

```
PROCEDURE REPORT_TEXT_ID
(
    begin_snap_gid       IN NUMBER
    end_snap_gid         IN NUMBER
    file_name            IN VARCHAR
);
```

– 파라미터

파라미터	설명
begin_snap_gid	성능 분석 리포트 대상 구간 중 시작 스냅샷의 GID이다.
end_snap_gid	성능 분석 리포트 대상 구간 중 종료 스냅샷의 GID이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

– 예제

```
exec DBMS_TPR.REPORT_TEXT_GID(3, 4);
/
```

47.2.6. REPORT_TEXT_LAST

마지막으로 남긴 스냅샷에 대한 성능 분석 리포트(text)를 생성한다.

REPORT_TEXT_LAST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_TEXT_LAST
(
    file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

- 예제

```
exec DBMS_TPR.REPORT_TEXT_LAST();
/
```

47.2.7. REPORT_TEXT_SPECIFIC_TIMES

특정 요일, 시간대에 해당되는 스냅샷들에 대한 성능 분석 리포트(text)를 생성한다.

REPORT_TEXT_SPECIFIC_TIMES 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_TEXT_SPECIFIC_TIMES(
   begin_time        IN DATE
   end_time          IN DATE
   begin_hour        IN VARCHAR2
   end_hour          IN VARCHAR2
   days              IN VARCHAR2
   instance_no       IN VARCHAR2
   file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
begin_time	성능 분석 리포트 대상 구간 중 시작 날짜와 시간이다. DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력한다.
end_time	성능 분석 리포트 대상 구간 중 종료 날짜와 시간이다. DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력한다.

파라미터	설명
begin_hour	성능 분석 리포트 대상 구간 중 출력되는 스냅샷 시간대의 시작 시간이다. VARCHAR2 타입으로 'HH24:MM' 포맷을 따라야 한다. (예: 09:00)
end_hour	성능 분석 리포트 대상 구간 중 출력되는 스냅샷 시간대의 종료 시간이다. VARCHAR2 타입으로 'HH24:MM' 포맷을 따라야 한다. (예: 18:00)
days	성능 분석 리포트 대상 구간 중 리포트 출력할 스냅샷의 요일이다. (기본값: MON,TUE,WED,THU,FRI,SAT,SUN)
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

- 예제

  아래의 예제는 1월 1일 00:00부터 2월 1일 23:59:59 사이에 월,수,금 오전 9시부터 오후 6시까지에 해당하는 스냅샷에 대한 리포트 생성을 하는 예제이다.

```
exec DBMS_TPR.REPORT_TEXT_SPECIFIC_TIMES('2015/01/01 00:00:00',
                                         '2015/02/01 23:59:59',
                                         '09:00', '18:00', 'MON,WED,FRI');
/
```

  파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 리포트를 출력한다. 입력하지 않는 경우 기본적으로 모든 인스턴스를 대상으로 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_TEXT_SPECIFIC_TIMES( '2015/01/01 00:00:00',
                                          '2015/02/01 23:59:59',
                                          '09:00', '18:00',
                                          'MON,WED,FRI', instance_no=>'1');
/
```

  파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_TEXT_SPECIFIC_TIMES('2015/01/01 00:00:00',
                                         '2015/02/01 23:59:59',
                                         '09:00', '18:00', 'MON,WED,FRI',
                                         instance_no=>'1', 'workinghours.txt');
/
```

47.2.8. REPORT_TEXT_MARKED

유저가 지정한 스냅샷들에 대한 성능 분석 리포트(text)를 생성한다.

REPORT_TEXT_MARKED 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_TEXT_MARKED
(
   file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

- 예제

지정한 스냅샷에 대한 리포트를 생성해주기 때문에 먼저 출력할 스냅샷을 지정해야 한다.

V$TPR_SNAPSHOT을 조회하면 스냅샷과 관련된 정보 및 marking되었는지 여부를 볼 수 있다.

```
select * from V$TPR_SNAPSHOT;
/
```

```
   SNAP_ID    THREAD# INSTANCE_NUMBER BEGIN_INTERVAL_TIME
   ---------- ---------- --------------- --------------------------------
   END_INTERVAL_TIME                  SNAP_GID MARKED_FOR_REPORT
   ------------------------------- ---------- -----------------
            1          0               0 2015/11/26
            2015/11/26                                N

            2          0               0 2015/11/27
            2015/11/27                                N
/
```

SNAP_ID 2번에 해당하는 리포트를 출력하기 위해 다음과 같은 query를 실행한다.

```
update V$TPR_SNAPSHOT set MARKED_FOR_REPORT='Y' where SNAP_ID=2;
commit;
/
```

V$TPR_SNAPSHOT을 다시 조회하면 원하는 스냅샷이 marking되었음을 확인할 수 있다.

```
   SNAP_ID    THREAD# INSTANCE_NUMBER BEGIN_INTERVAL_TIME
   ---------- ---------- --------------- --------------------------------
   END_INTERVAL_TIME                  SNAP_GID MARKED_FOR_REPORT
```

```
   -------------------------------- ---------- -----------------
            1          0               0 2015/11/26
            2015/11/26                                 N

            2          0               0 2015/11/27
            2015/11/27                                 Y
/
```

원하는 스냅샷을 모두 marking하였으면 리포트 출력을 실행시킨다.

```
exec DBMS_TPR.REPORT_TEXT_MARKED();
/
```

파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_TEXT_MARKED('marked.txt');
/
```

47.2.9. REPORT_HTML

특정 구간의 성능 분석 리포트(html)를 생성한다.

REPORT_HTML 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_HTML
(
    begin_time       IN DATE
    end_time         IN DATE
    instance_no      IN VARCHAR
    file_name        IN VARCHAR
);
```

- 파라미터

파라미터	설명
begin_time	성능 분석 리포트 대상 구간 중 시작 시간이다.
end_time	성능 분석 리포트 대상 구간 중 종료 시간이다.
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.html)

- 예제

  다음의 예제는 두 개의 인자 모두 DATE 타입이므로, 사용 중인 DATE 포맷에 맞게 입력해야 한다.

```
exec DBMS_TPR.REPORT_HTML('2011/07/01 12:00:00', '2011/07/02 11:59:59');
/
```

  파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_HTML('2011/07/01 12:00:00', '2011/07/02 11:59:59', file_name=>'tpr.html');
/
```

47.2.10. REPORT_HTML_ID

특정 스냅샷 ID 혹은 ID 구간의 성능 분석 리포트(html)를 생성한다.

REPORT_HTML_ID 프러시저는 input이 특정 ID 혹은 ID 구간 두 종류로 나뉘어지며, 그 세부 내용은 다음과 같다.

- 특정 ID
  - 프로토타입

```
PROCEDURE REPORT_HTML_ID
(
   one_snap_id       IN NUMBER
   file_name         IN VARCHAR
);
```

  - 파라미터

파라미터	설명
one_snap_id	성능 분석 리포트 대상 스냅샷 ID이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{db_name}.{current_time}.html)

  - 예제

```
exec DBMS_TPR.REPORT_HTML_ID(5);
/
```

- ID 구간
  - 프로토타입

```
PROCEDURE REPORT_HTML_ID
(
   begin_snap_id       IN NUMBER
   end_snap_id         IN NUMBER
   instance_no         IN VARCHAR
   file_name           IN VARCHAR
);
```

– 파라미터

파라미터	설명
begin_snap_id	성능 분석 리포트 대상 구간 중 시작 스냅샷 ID이다.
end_snap_id	성능 분석 리포트 대상 구간 중 종료 스냅샷 ID이다.
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.html)

– 예제

파라미터 중 instance_no를 입력하지 않으면 기본적으로 모든 인스턴스를 대상으로 하는 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_HTML_ID(5, 10);
/
```

파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_HTML_ID(5, 10, instance_no=>'2');
/
```

47.2.11. REPORT_HTML_GID

특정 스냅샷 GID 혹은 GID 구간의 성능 분석 리포트(html)를 생성한다.

REPORT_HTML_GID 프러시저는 input이 특정 GID 혹은 GID 구간 두 종류로 나뉘어지며, 그 세부 내용은 다음과 같다.

- 특정 ID

  – 프로토타입

```
PROCEDURE REPORT_HTML_GID
(
```

```
   one_snap_gid       IN NUMBER
);
```

– 파라미터

파라미터	설명
one_snap_gid	성능 분석 리포트 대상 스냅샷의 GID이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.html)

– 예제

```
exec DBMS_TPR.REPORT_HTML_GID(3);
/
```

- GID 구간

– 프로토타입

```
PROCEDURE REPORT_HTML_ID
(
   begin_snap_gid       IN NUMBER
   end_snap_gid         IN NUMBER
   file_name            IN VARCHAR
);
```

– 파라미터

파라미터	설명
begin_snap_gid	성능 분석 리포트 대상 구간 중 시작 스냅샷의 GID이다.
end_snap_gid	성능 분석 리포트 대상 구간 중 종료 스냅샷의 GID이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.html)

– 예제

```
exec DBMS_TPR.REPORT_HTML_GID(3, 4);
/
```

47.2.12. REPORT_HTML_LAST

마지막으로 남긴 스냅샷에 대한 성능 분석 리포트(html)를 생성한다.

REPORT_HTML_LAST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_HTML_LAST
(
    file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.html)

- 예제

```
exec DBMS_TPR.REPORT_HTML_LAST();
/
```

47.2.13. REPORT_HTML_SPECIFIC_TIMES

특정 요일, 시간대에 해당되는 스냅샷들에 대한 성능 분석 리포트(html)를 생성한다.

REPORT_HTML_SPECIFIC_TIMES 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_HTML_SPECIFIC_TIMES
(
   begin_time        IN DATE
   end_time          IN DATE
   begin_hour        IN VARCHAR2
   end_hour          IN VARCHAR2
   days              IN VARCHAR2
   instance_no       IN VARCHAR2
   file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
begin_time	성능 분석 리포트 대상 구간 중 시작 날짜와 시간이다. DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력해야 한다.
end_time	성능 분석 리포트 대상 구간 중 종료 날짜와 시간이다. DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력해야 한다.
begin_hour	성능 분석 리포트 대상 구간 중 출력되는 스냅샷 시간대의 시작 시간이다. VARCHAR2 타입으로 'HH24:MM' 포맷을 따라야 한다. (예: 09:00)
end_hour	성능 분석 리포트 대상 구간 중 출력되는 스냅샷 시간대의 종료 시간이다. VARCHAR2 타입으로 'HH24:MM' 포맷을 따라야 한다. (예: 18:00)
days	성능 분석 리포트 대상 구간 중 리포트 출력할 스냅샷의 요일이다. (기본값: MON,TUE,WED,THU,FRI,SAT,SUN)
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

- 예제

  아래의 예제는 1월 1일 00:00부터 2월 1일 23:59:59 사이에 월,수,금 오전 9시 부터 오후 6시까지에 해당하는 스냅샷에 대한 리포트 생성을 하는 예제이다.

```
exec DBMS_TPR.REPORT_HTML_SPECIFIC_TIMES( '2015/01/01 00:00:00',
                                          '2015/02/01 23:59:59',
                                          '09:00', '18:00', 'MON,WED,FRI');
/
```

  파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 리포트를 출력한다. 입력하지 않는 경우 기본적으로 모든 인스턴스를 대상으로 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_HTML_SPECIFIC_TIMES( '2015/01/01 00:00:00',
                                          '2015/02/01 23:59:59',
                                          '09:00', '18:00', 'MON,WED,FRI',
                                          instance_no=>'1');
/
```

  파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_HTML_SPECIFIC_TIMES( '2015/01/01 00:00:00',
                                          '2015/02/01 23:59:59',
                                          '09:00', '18:00', 'MON,WED,FRI',
                                          instance_no=>'1', 'workinghours.txt');
/
```

47.2.14. REPORT_HTML_MARKED

유저가 지정한 스냅샷들에 대한 성능 분석 리포트(html)를 생성한다.

REPORT_HTML_MARKED 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE REPORT_HTML_MARKED
(
   file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: tpr_report.{*db_name*}.{*current_time*}.txt)

- 예제

지정한 스냅샷에 대한 리포트를 생성해주기 때문에 먼저 출력할 스냅샷을 지정해야한다.

V$TPR_SNAPSHOT을 조회하면 스냅샷과 관련된 정보 및 marking되었는지 여부를 볼 수 있다.

```
select * from V$TPR_SNAPSHOT;
/
```

```
   SNAP_ID    THREAD# INSTANCE_NUMBER BEGIN_INTERVAL_TIME
   ---------- ---------- --------------- ------------------------------
   END_INTERVAL_TIME                 SNAP_GID MARKED_FOR_REPORT
   ------------------------------- ---------- -----------------
            1          0               0 2015/11/26
            2015/11/26                                 N

            2          0               0 2015/11/27
            2015/11/27                                 N
/
```

SNAP_ID 2번에 해당하는 리포트를 출력하기 위해 다음과 같은 query를 실행한다.

```
update V$TPR_SNAPSHOT set MARKED_FOR_REPORT='Y' where SNAP_ID=2;
commit;
/
```

V$TPR_SNAPSHOT을 다시 조회하면 원하는 스냅샷이 marking되었음을 확인할 수 있다.

```
   SNAP_ID    THREAD# INSTANCE_NUMBER BEGIN_INTERVAL_TIME
   ---------- ---------- --------------- ------------------------------
   END_INTERVAL_TIME                 SNAP_GID MARKED_FOR_REPORT
   ------------------------------ ---------- -----------------
            1          0               0 2015/11/26
            2015/11/26                                 N

            2          0               0 2015/11/27
            2015/11/27                                 Y
/
```

원하는 스냅샷을 모두 marking하였으면 리포트 출력을 실행시킨다.

```
exec DBMS_TPR.REPORT_HTML_MARKED();
/
```

파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.REPORT_HTML_MARKED('marked.txt');
/
```

47.2.15. FLUSH_ASH

메모리상에 담겨있는 ASH 샘플들을 _TPR_ACTIVE_SESSION_HISTORY 테이블에 저장한다.

FLUSH_ASH 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FLUSH_ASH ();
```

- 예제

  FLUSH_ASH 프러시저를 수행함으로서 메모리상에만 존재하던 ASH 샘플들이 _TPR_ACTIVE_SESSION_HISTORY 테이블상에 저장된다.

```
exec DBMS_TPR.FLUSH_ASH();
/
```

47.2.16. ASH_REPORT_TEXT

특정 구간의 ASH 성능 분석 리포트(text)를 생성한다.

ASH 샘플은 IPARAM(ACTIVE_SESSION_HISTORY)을 Y로 설정했을 때만 남게 되며, ASH 리포트는 이 샘플들의 정보를 취합하여 리포트를 작성한다.

ASH_REPORT_TEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE ASH_REPORT_TEXT
(
    begin_time       IN DATE
    end_time         IN DATE
    instance_no      IN VARCHAR
    file_name        IN VARCHAR
);
```

- 파라미터

파라미터	설명
begin_time	성능 분석 리포트 대상 구간 중 시작 시간이다.
end_time	성능 분석 리포트 대상 구간 중 종료 시간이다.
instance_no	성능 분석 리포트 대상 인스턴스 번호이다. (기본값: ALL)
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: ash_report.{*db_name*}.{*current_time*}.txt)

- 예제

  아래의 예제는 두 개의 인자 모두 DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력해야 한다.

```
exec DBMS_TPR.ASH_REPORT_TEXT( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59');
/
```

  파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.ASH_REPORT_TEXT( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59',
                               file_name=>'ash.txt');
/
```

  파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 리포트를 출력한다. 입력하지 않는 경우 기본적으로 모든 인스턴스를 대상으로 리포트를 출력한다.

```
exec DBMS_TPR.ASH_REPORT_TEXT( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59',
```

```
                                instance_no=>'1');
/
```

47.2.17. CREATE_BASELINE

특정 구간에 대한 TPR 스냅샷을 Baseline으로 지정한다. 프러시저 실행시 입력된 시작 시간과 종료 시간 사이에 존재하는 TPR 스냅샷들을 Baseline에 등록한다. Baseline에 등록된 TPR 스냅샷들은 Retention 기간을 넘겨도 파기되지 않고 남아있게 된다. 등록된 Baseline에 관한 내용은 _TPR_BASELINE 테이블을 조회하여 확인할 수 있다.

CREATE_BASELINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CREATE_BASELINE
(
    begin_time       IN DATE
    end_time         IN DATE
    instance_no      IN VARCHAR
    baseline_name    IN VARCHAR
    expiration       IN NUMBER
);
```

- 파라미터

파라미터	**설명**
begin_time	Baseline에 포함될 TPR 스냅샷 범위의 시작시간이다.
end_time	Baseline에 포함될 TPR 스냅샷 범위의 종료시간이다.
instance_no	Baseline 대상 인스턴스 번호이다. (기본값: ALL)
baseline_name	만들어질 Baseline의 이름이다. (기본값: BASELINE{*BASELINE_ID*})
expiration	Baseline 자체의 Expiration 기간이다. (기본값: Null, 단위: Hours)

- 예제

  아래의 예제는 두 개의 인자 모두 DATE 타입이므로 사용 중인 DATE 포맷에 맞게 입력해야 한다.

```
exec DBMS_TPR.CREATE_BASELINE( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59');
/
```

  파라미터 중 baseline_name을 지정하면 입력된 이름으로 된 Baseline을 생성한다.

```
exec DBMS_TPR.CREATE_BASELINE( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59',
                                baseline_name=>'my_baseline');
/
```

파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 Baseline을 생성한다. 입력하지 않는 경우 기본적으로 모든 인스턴스를 대상으로 하는 Baseline을 생성한다.

```
exec DBMS_TPR.CREATE_BASELINE( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59',
                               instance_no=>'1');
/
```

파라미터 중 expiration을 입력하면 expiration이 등록된 Baseline이 생성된다. 이러한 Baseline은 생성 후 지정된 expiration만큼의 시간(hrs)이 흐른 뒤 해당 Baseline이 자동 삭제된다. 명시적으로 expiration을 지정하지 않은 경우 사용자가 직접 Baseline을 삭제하기 전까지 Baseline은 삭제되지 않는다.

```
exec DBMS_TPR.CREATE_BASELINE( '2015/01/29 21:00:00',
                               '2015/01/29 22:59:59',
                                expiration=>168);
/
```

47.2.18. CREATE_BASELINE_ID

특정 스냅샷 ID 혹은 ID 구간을 Baseline으로 지정한다.

CREATE_BASELINE_ID 프러시저는 input이 특정 ID 혹은 ID 구간 두 종류로 나뉘어지며, 그 세부 내용은 다음과 같다.

- 특정 ID
  - 프로토타입

```
PROCEDURE CREATE_BASELINE_ID
(
    one_snap_id    IN NUMBER
    baseline_name    IN VARCHAR
    expiration       IN NUMBER
);
```

  - 파라미터

파라미터	설명
one_snap_id	Baseline에 포함될 TPR 스냅샷 ID이다.
baseline_name	만들어질 Baseline의 이름이다. (기본값: BASELINE{*BASELINE_ID*})

파라미터	설명
expiration	Baseline 자체의 Expiration 기간이다. (기본값: Null, 단위: Hours)

– 예제

```
exec DBMS_TPR.CREATE_BASELINE_ID(30);
/
```

파라미터 중 baseline_name을 지정하면 입력된 이름으로 된 Baseline을 생성한다.

```
exec DBMS_TPR.CREATE_BASELINE_ID(30, baseline_name=>'my_baseline');
/
```

파라미터 중 expiration을 입력하면 expiration이 등록된 Baseline이 생성된다. 이러한 Baseline은 생성 후 지정된 expiration만큼의 시간(hrs)이 흐른 뒤 해당 Baseline이 자동 삭제된다. 명시적으로 expiration을 지정하지 않은 경우 사용자가 직접 Baseline을 삭제하기전까지 Baseline은 삭제되지 않는다.

```
exec DBMS_TPR.CREATE_BASELINE_ID(30, expiration=>168);
/
```

- ID 구간

– 프로토타입

```
PROCEDURE CREATE_BASELINE_ID
(
    begin_snap_id    IN NUMBER
    end_snap_id      IN NUMBER
    instance_no      IN VARCHAR
    baseline_name    IN VARCHAR
    expiration       IN NUMBER
);
```

– 파라미터

파라미터	설명
begin_snap_id	Baseline에 포함될 TPR 스냅샷 범위의 시작 ID이다.
end_snap_id	Baseline에 포함될 TPR 스냅샷 범위의 종료 ID이다.
instance_no	Baseline 대상 인스턴스 번호이다. (기본값: ALL)
baseline_name	만들어질 Baseline의 이름이다. (기본값: BASELINE{*BASELINE_ID*})
expiration	Baseline 자체의 Expiration 기간이다. (기본값: Null, 단위: Hours)

– 예제

```
exec DBMS_TPR.CREATE_BASELINE_ID(30, 50);
/
```

파라미터 중 baseline_name을 지정하면 입력된 이름으로 된 Baseline을 생성한다.

```
exec DBMS_TPR.CREATE_BASELINE_ID(30, 50, baseline_name=>'my_baseline');
/
```

파라미터 중 instance_no를 입력하면 해당 인스턴스를 대상으로 하는 Baseline을 생성한다. 입력하지 않는 경우 기본적으로 모든 인스턴스를 대상으로 하는 Baseline을 생성한다.

```
exec DBMS_TPR.CREATE_BASELINE_ID(30, 50, instance_no=>'1');
/
```

파라미터 중 expiration을 입력하면 expiration이 등록된 Baseline이 생성된다. 이러한 Baseline은 생성 후 지정된 expiration만큼의 시간(hrs)이 흐른 뒤 해당 Baseline이 자동 삭제된다. 명시적으로 expiration을 지정하지 않은 경우 사용자가 직접 Baseline을 삭제하기전까지 Baseline은 삭제되지 않는다.

```
exec DBMS_TPR.CREATE_BASELINE_ID(30, 50, expiration=>168);
/
```

47.2.19. DROP_BASELINE

지정된 Baseline 이름을 지니는 Baseline을 제거한다.

DROP_BASELINE 프러시저의 세부 내용은 다음과 같다

- 프로토타입

```
PROCEDURE DROP_BASELINE
(
    baseline_name_    IN VARCHAR
);
```

- 파라미터

파라미터	설명
baseline_name_	삭제할 Baseline의 이름이다.

- 예제

```
exec DBMS_TPR.DROP_BASELINE('my_baseline');
/
```

47.2.20. DROP_BASELINE_ID

지정된 Baseline ID를 지니는 Baseline을 제거한다.

DROP_BASELINE_ID 프러시저의 세부 내용은 다음과 같다

- 프로토타입

```
PROCEDURE DROP_BASELINE_ID
(
    baseline_id_    IN NUMBER
);
```

- 파라미터

파라미터	설명
baseline_id_	삭제할 Baseline의 Baseline ID이다.

- 예제

```
exec DBMS_TPR.DROP_BASELINE_ID(5);
/
```

47.2.21. BASELINE_REPORT_TEXT

지정된 Baseline 이름을 지니는 Baseline에 포함된 TPR 스냅샷들에 대한 리포트(text)를 작성한다. 이때 출력되는 리포트는 Baseline에 포함된 TPR 스냅샷에 대해 REPORT_TEXT 프러시저를 수행할 때 출력되는 리포트와 동일하다.

BASELINE_REPORT_TEXT 프러시저의 세부 내용은 다음과 같다

- 프로토타입

```
PROCEDURE BASELINE_REPORT_TEXT
(
    baseline_name_    IN VARCHAR
    file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
baseline_name_	리포트를 출력할 Baseline의 이름이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: baseline_report.{*db_name*}.{*current_time*}.txt)

- 예제

```
exec DBMS_TPR.BASELINE_REPORT_TEXT('my_baseline');
/
```

파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.BASELINE_REPORT_TEXT('my_baseline', file_name=>'baseline.txt');
/
```

47.2.22. BASELINE_REPORT_TEXT_ID

지정된 Baseline ID를 지니는 Baseline에 포함된 TPR 스냅샷들에 대한 리포트(text)를 작성한다. 이때 출력되는 리포트는 Baseline에 포함된 TPR 스냅샷에 대해 REPORT_TEXT 프러시저를 수행할 때 출력되는 리포트와 동일하다.

BASELINE_REPORT_TEXT_ID 프러시저의 세부 내용은 다음과 같다

- 프로토타입

```
PROCEDURE BASELINE_REPORT_TEXT_ID
(
    baseline_id_      IN NUMBER
    file_name         IN VARCHAR
);
```

- 파라미터

파라미터	설명
baseline_id_	리포트를 출력할 Baseline의 ID이다.
file_name	만들어질 성능 분석 리포트 파일명이다. (기본값: baseline_report.{*db_name*}.{*current_time*}.txt)

- 예제

```
exec DBMS_TPR.BASELINE_REPORT_TEXT_ID(5);
/
```

파라미터 중 file_name을 지정하면 지정된 파일명의 리포트를 출력한다.

```
exec DBMS_TPR.BASELINE_REPORT_TEXT_ID(5, file_name=>'baseline.txt');
/
```

제48장 DBMS_TRANSACTION

본 장에서는 DBMS_TRANSACTION 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

48.1. 개요

DBMS_TRANSACTION은 tbPSM 프로그램 내에서 트랜잭션을 관리하는 SQL 문장을 실행하는 패키지이다.

참고

트랜잭션을 관리하는 SQL 문장에 대한 자세한 내용은 "Tibero SQL 참조 안내서"를 참고한다.

48.2. 프러시저

본 절에서는 DBMS_TRANSACTION 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

48.2.1. COMMIT

현재 트랜잭션을 커밋하는 프러시저이다.

COMMIT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_TRANSACTION.COMMIT;
```

- 예제

```
BEGIN
    DBMS_TRANSACTION.COMMIT;
END;
```

48.2.2. ROLLBACK, ROLLBACK_SAVEPOINT

현재 진행 중인 트랜잭션 전체를 롤백하거나 일부 저장점(savepoint)까지 롤백을 수행하는 프러시저이다.

ROLLBACK, ROLLBACK_SAVEPOINT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - 트랜잭션 전체를 롤백하는 경우

```
DBMS_TRANSACTION.ROLLBACK;
```

  - 저장점까지 롤백하는 경우

```
DBMS_TRANSACTION.ROLLBACK_SAVEPOINT
(
    savepoint       IN VARCHAR
);
```

- 파라미터

파라미터	설명
savepoint	저장점의 이름이다.

- 예제

```
BEGIN
    INSERT INTO EMP VALUES (1, 'chulsoo');
    DBMS_TRANSACTION.ROLLBACK;
END;
```

48.2.3. SAVEPOINT

현재 트랜잭션 내에 새로운 저장점을 설정하는 프러시저이다.

SAVEPOINT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_TRANSACTION.SAVEPOINT
(
    savepoint     IN VARCHAR
);
```

- 파라미터

파라미터	설명
savepoint	설정할 저장점의 이름이다.

- 예제

```
BEGIN
    DBMS_TRANSACTION.SAVEPOINT('sp1');
    INSERT INTO EMP VALUES (2, 'younghee');
    DBMS_TRANSACTION.ROLLBACK_SAVEPOINT('sp1');
END;
```

제49장 DBMS_TYPES

본 장에서는 DBMS_TYPES 패키지의 기본 개념과 패키지 내의 상수들의 의미를 설명한다.

49.1. 개요

DBMS_TYPES은 Tibero의 데이터 타입들의 번호를 열거하는 패키지이다.

DBMS_SQL.DESCRIBE_COLUMNS 프러시저를 이용한 컬럼의 타입 번호를 추출하는 경우 이 패키지에 정의된 값을 비교함으로써 실제 타입을 알 수 있다.

49.2. 상수 목록

DBMS_TYPES에서 정의된 상수 목록은 아래와 같다.

알고리즘	설명
TYPECODE_NUMBER	NUMBER 타입
TYPECODE_CHAR	CHAR 타입
TYPECODE_VARCHAR	VARCHAR 타입
TYPECODE_RAW	RAW 타입
TYPECODE_DATE	DATE 타입
TYPECODE_TIME	TIME 타입
TYPECODE_TIMESTAMP	TIMESTAMP 타입
TYPECODE_ITV_YTM	INTERVAL YEAR TO MONTH 타입
TYPECODE_ITV_DTS	INTERVAL DAY TO SECOND 타입
TYPECODE_LONG	LONG 타입
TYPECODE_LONG_RAW	LONG RAW 타입
TYPECODE_BLOB	BLOB 타입
TYPECODE_CLOB	CLOB 타입
TYPECODE_BFILE	BFILE 타입
TYPECODE_ROWID	ROWID 타입
TYPECODE_CSR	CSR 타입
TYPECODE_NCHAR	NCHAR 타입
TYPECODE_NVARCHAR	NVARCHAR 타입

알고리즘	설명
TYPECODE_NCLOB	NCLOB 타입
TYPECODE_TS_TZ	TIMESTAMP WITH TIME ZONE 타입
TYPECODE_TS_LTZ	TIMESTAMP WITH LOCAL TIME ZONE 타입
TYPECODE_BIN_FLOAT	BINARY_FLOAT 타입
TYPECODE_BIN_DOUBLE	BINARY_DOUBLE 타입
TYPECODE_OBJECT	OBJECT 타입
TYPECODE_JSON	JSON 타입
TYPECODE_NONE	지정되지 않은 타입 (null 등이 해당)
TYPECODE_UNKNOWN	알 수 없는 타입

제50장 DBMS_UTILITY

본 장에서는 DBMS_UTILITY 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

50.1. 개요

DBMS_UTILITY은 여러 가지 유용한 기능들을 제공한다.

50.2. 타입

본 절에서는 DBMS_UTILITY 패키지에서 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

50.2.1. UNCL_ARRAY

문자열의 배열이다. 문자열의 최대 길이는 227 byte이다.

UNCL_ARRAY 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE uncl_array IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```

50.2.2. LNAME_ARRAY

문자열의 배열이다. 문자열의 최대 길이는 4000byte이다.

LNAME_ARRAY 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE lname_array IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

50.2.3. MAXNAME_ARRAY

문자열의 배열이다. 문자열의 최대길이는 32767byte이다.

MAXNAME_ARRAY 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE maxname_array IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

50.3. 프러시저와 함수

본 절에서는 DBMS_UTILITY 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

50.3.1. COMMA_TO_TABLE

테이블 이름들이 쉼표로 구분되어 나열된 문자열을 분석하여 테이블 이름들을 구성 요소로 하는 인덱스 테이블로 변환하는 프러시저이다.

COMMA_TO_TABLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.COMMA_TO_TABLE
(
    list             IN              VARCHAR2,
    tablen           OUT             BINARY_INTEGER,
    tab              OUT             UNCL_ARRAY
);
```

```
DBMS_UTILITY.COMMA_TO_TABLE
(
    list             IN              VARCHAR2,
    tablen           OUT             BINARY_INTEGER,
    tab              OUT             LNAME_ARRAY
);
```

- 파라미터

파라미터	설명
list	쉼표로 구분되어 있는 테이블 이름들의 목록 문자열이다. 테이블 이름들은 a [. b [. c]][@ d]로 설정한다. (a, b, c, d는 따옴표로 묶이거나 묶이지 않은 식별자 형태의 문자열) 이때 입력값이 NULL값일 경우에도 콤마(,)로 구분한다. 예를 들어 입력값이 ',a'거나 'a,'인 경우 각각 앞, 뒤에 NULL 값이 들어온 것으로 간주한다.
tablen	생성되는 이름들의 갯수이다.
tab	테이블 이름들을 구성 요소로 갖는 인덱스 테이블이다. uncl_array와 lname_array를 입력 값으로 받을 수 있다.

- 예제

```
declare
    tab_len binary_integer;
```

```
    tab dbms_utility.uncl_array;
begin
    DBMS_UTILITY.COMMA_TO_TABLE('a,"b".c,d.e.f,g.h@i', tab_len, tab);

    for i in 1..tab_len loop
        dbms_output.put_line(tab(i));
    end loop;
end;
/
```

50.3.2. COMPLIE_SCHEMA

지정된 스키마 안의 모든 프러시저, 함수, 패키지, 뷰, 트리거를 컴파일해 주는 프러시저이다. 이 프러시저를 호출하는 유저가 지정된 스키마에 대한 recompile 권한이 없으면 원하는 결과를 얻지 못할 수 있다.

COMPILE_SCHEMA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.COMPILE_SCHEMA
(
   schema          IN VARCHAR2,
   compile_all     IN BOOLEAN DEFAULT TRUE,
   reuse_setting   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
schema	컴파일할 스키마의 이름이다.
compile_all	– TRUE : 스키마의 모든 오브젝트들을 컴파일한다. (기본값) – FALSE : 스키마 내 INVALID한 상태의 오브젝트들만 컴파일한다.
reuse_setting	– TRUE : 오브젝트의 세션 설정을 재사용한다. – FALSE : 현재 세션의 설정이 적용된다. (기본값)

- 예제

```
CREATE TABLE RECOMPILE_TAB (ID NUMBER);
CREATE OR REPLACE PROCEDURE RECOMPILE_PROC IS
    a number;
BEGIN
    select count(*) into a from RECOMPILE_TAB;
```

```
END;
/
DROP TABLE RECOMPILE_TAB;
CREATE TABLE RECOMPILE_TAB (ID NUMBER);

DECLARE
    stat VARCHAR2(10);
BEGIN
   select status into stat FROM USER_OBJECTS WHERE OBJECT_NAME = 'RECOMPILE_PROC';

   dbms_output.put_line('status of RECOMPILE_PROC before COMPILE_SCHEMA
call: ' || stat);

   DBMS_UTILITY.COMPILE_SCHEMA(USER,FALSE);

   select status into stat FROM USER_OBJECTS WHERE OBJECT_NAME = 'RECOMPILE_PROC';

   dbms_output.put_line('status of RECOMPILE_PROC after COMPILE_SCHEMA
call: ' || stat);
END;
/

-- status of RECOMPILE_PROC before COMPILE_SCHEMA call: INVALID
-- status of RECOMPILE_PROC after COMPILE_SCHEMA call: VALID
```

50.3.3. EXEC_DDL_STATEMENT

DDL 명령을 실행하는 기능을 제공하는 함수이다.

EXEC_DDL_STATEMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.EXEC_DDL_STATEMENT
(
    parse_string          IN VARCHAR2
);
```

- 파라미터

파라미터	설명
parse_string	실행할 DDL 명령을 지정하는 VARCHAR2 유형의 입력 인수이다.

- 예제

```
DECLARE
 BEGIN
 DBMS_UTILITY.EXEC_DDL_STATEMENT('CREATE TABLE TEST (ID NUMBER)');
END;
/
```

50.3.4. FORMAT_CALL_STACK

콜 스택의 내용을 문자열로 반환하는 함수이다. 반환 문자열이 32767bytes를 초과하는 경우 원하는 결과를 얻지 못할 수 있다.

FORMAT_CALL_STACK 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.FORMAT_CALL_STACK
RETURN VARCHAR2;
```

- 예제

```
create or replace procedure proc3
as
begin
        dbms_output.put_line (dbms_utility.format_call_stack);
end;
/
create or replace procedure proc2
as
begin
        proc3;
end;
/
create or replace procedure proc1
as
begin
        proc2;
end;
/
begin
        proc1;
end;
/
-- ----- PSM Call Stack -----
--  object     line  object
--  handle   number  name
```

```
-- 0x7f50fa3943a8      4 procedure SYS.PROC3
-- 0x7f50fa3942d0      4 procedure SYS.PROC2
-- 0x7f50fa3941f8      4 procedure SYS.PROC1
-- 0x7f50fa394120      2 anonymous block
```

50.3.5. FORMAT_ERROR_BACKTRACE

BACKTRACE 에러 메시지 문자열을 반환하는 함수이다. 반환 문자열이 712bytes를 초과하는 경우 원하는 결과를 얻지 못할 수 있다.

FORMAT_ERROR_BACKTRACE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
RETURN VARCHAR2;
```

- 예제

```
CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR0
IS
    variable positive;
BEGIN
    variable := -1;
END;
/

CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR1 IS
BEGIN
  BACKTRACE_EXAMPLE_PR0 ();
END;
/

CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR2 IS
BEGIN
  BACKTRACE_EXAMPLE_PR1 ();
END;
/

CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR3 IS
BEGIN
  BACKTRACE_EXAMPLE_PR2 ();
END;
/
```

```
CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR4 IS
  BEGIN BACKTRACE_EXAMPLE_PR3 ();
END;
/

CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR5 IS
  BEGIN BACKTRACE_EXAMPLE_PR4 ();
END;
/

CREATE OR REPLACE PROCEDURE BACKTRACE_EXAMPLE_PR6 IS
BEGIN
  BACKTRACE_EXAMPLE_PR5 ();
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE (DBMS_UTILITY.FORMAT_ERROR_BACKTRACE() );
END;
/

begin
    BACKTRACE_EXAMPLE_PR6 ();
end;
/

-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR0,  line 5.
-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR1,  line 3.
-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR2,  line 3.
-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR3,  line 3.
-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR4,  line 2.
-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR5,  line 2.
-- TBR-15163: Unhandled exception at SYS.BACKTRACE_EXAMPLE_PR6,  line 3.
```

50.3.6. FORMAT_ERROR_STACK

에러 스택의 내용을 문자열로 반환하는 함수이다. 반환 문자열이 712bytes를 초과하는 경우 원하는 결과를 얻지 못할 수 있다.

FORMAT_ERROR_STACK 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.FORMAT_ERROR_STACK
RETURN VARCHAR2;
```

- 예제

```
CREATE OR REPLACE PROCEDURE CALLSTACK_EXAMPLE_PR0
IS
    variable positive;
BEGIN
    variable := -1;
END;
/

CREATE OR REPLACE PROCEDURE CALLSTACK_EXAMPLE_PR1 IS
BEGIN
  CALLSTACK_EXAMPLE_PR0 ();
END;
/

CREATE OR REPLACE PROCEDURE CALLSTACK_EXAMPLE_PR2 IS
BEGIN
  CALLSTACK_EXAMPLE_PR1 ();
END;
/

begin
    CALLSTACK_EXAMPLE_PR2 ();
exception when others then
    dbms_output.put_line (DBMS_UTILITY.FORMAT_ERROR_STACK);

    dbms_output.put_line ('exception handled.');
end;
/

--TBR-15035: Number is out of range.
--exception handled.
```

50.3.7. GET_HASH_VALUE

주어진 문자열의 해시 값을 구하는 함수이다.

GET_HASH_VALUE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.GET_HASH_VALUE
(
        name            IN            VARCHAR2,
```

```
        base            IN            NUMBER,
        hash_size       IN            NUMBER
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
name	해시 값을 얻고자 하는 문자열이다.
base	해시 값의 시작점이다.
hash_size	해시 테이블의 크기이다. 크기는 1 이상이어야 한다.

- 예제

```
declare
    hash_value number;
begin
    hash_value := DBMS_UTILITY.GET_HASH_VALUE('Oleg', 1000, 100);

    dbms_output.put_line(hash_value);
end;
/
```

50.3.8. GET_TIME

현재 시간을 1/100초 단위의 숫자로 표현하는 함수이다. 일반적으로 시작과 끝에 호출하여 소요 시간을 측정하는 데 사용된다.

GET_TIME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.GET_TIME
RETURN NUMBER;
```

- 예제

```
declare
    s number;
    e number;
    c sys_refcursor;
begin
    s := DBMS_UTILITY.GET_TIME;
    open c for select * from all_objects;
    close c;
```

```
    e := DBMS_UTILITY.GET_TIME;

    dbms_output.put_line(e - s);
end;
/
```

50.3.9. TABLE_TO_COMMA

테이블 이름들을 구성 요소로 하는 인덱스 테이블을 분석하여 테이블 이름들이 쉼표로 구분되어 나열된 문자열로 변환하는 프러시저이다.

TABLE_TO_ COMMA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_UTILITY.TABLE_TO_COMMA
(
    tab              IN              UNCL_ARRAY,
    tablen           OUT             BINARY_INTEGER,
    list             OUT             VARCHAR2
);
```

```
DBMS_UTILITY.TABLE_TO_COMMA
(
    tab              IN              LNAME_ARRAY,
    tablen           OUT             BINARY_INTEGER,
    list             OUT             VARCHAR2
);
```

- 파라미터

파라미터	설명
tab	테이블 이름들을 구성 요소로 갖는 인덱스 테이블이다. uncl_array와 lname_array를 입력 값으로 받을 수 있다.
tablen	생성되는 이름들의 갯수이다. 이때 tab에 변수를 자체적으로 할당할 경우 변수를 할당한 부분의 앞 부분이 비어있더라도 NULL값이 할당된것으로 간주하여 tablen에 값을 부여한다. 예를 들어 tab(3) := 'a';를 통해 테이블의 3번째 값에 a를 할당한 경우 tab(1)과 tab(2)가 비어있어도 NULL 값이 할당된 것으로 간주하여 tablen의 결과는 3이 된다.
list	콤마로 구분되어 있는 테이블 이름들의 목록 문자열이다.

파라미터	설명
	이때 tab에 변수를 자체적으로 할당할 경우, 변수를 할당한 부분의 앞 부분이 비어있더라도 NULL값이 할당된것으로 간주하여 list에 값을 부여한다. 예를 들어 tab(3) := 'a';를 통해 테이블의 3번째 값에 a를 할당한 경우 tab(1)과 tab(2)가 비어있어도 NULL 값이 할당된 것으로 간주하여 list의 결과는 ',,a'가 된다.

- 예제

```
declare
    tab dbms_utility.uncl_array;
    tablen binary_integer;
    str varchar2(1000);
begin
    tab(3) := 'a';
    dbms_utility.table_to_comma(tab, tablen, str);

   dbms_output.put_line(str);
   dbms_output.put_line(tablen);
end;
/
```

제51장 DBMS_VERIFY

본 장에서는 DBMS_VERIFY 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

51.1. 개요

DBMS_VERIFY는 여러 요소들에 대하여 정합성을 검사하는 기능을 제공한다.

51.2. 프러시저

본 절에서는 DBMS_VERIFY 패키지에서 제공하는 프러시저에 대해서 설명한다.

51.2.1. ALL_INDEX

DBMS에 존재하는 모든 인덱스의 정합성을 검사하는 함수이다.

ALL_INDEX 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_VERIFY.ALL_INDEX
(
    DOP              IN              NUMBER DEFAULT 0,
    UNUSABLE         IN              BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
DOP	Degree Of Parallelism으로 병렬화할 정도를 의미한다.
UNUSABLE	값이 TRUE면 정합성이 훼손된 인덱스를 UNUSABLE 상태로 만든다.

- 예제

```
declare
begin
    DBMS_VERIFY.ALL_INDEX();
```

```
end;
/
```

51.2.2. SCHEMA_INDEX

해당 스키마에 존재하는 모든 인덱스의 정합성을 검사하는 함수이다.

SCHEMA_INDEX 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_VERIFY.SCHEMA_INDEX
(
    SCHEMA           IN              VARCHAR2,
    DOP              IN              NUMBER DEFAULT 0,
    UNUSABLE         IN              BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
SCHEMA	인덱스의 정합성을 검사할 스키마의 이름이다.
DOP	Degree Of Parallelism으로 병렬화할 정도를 의미한다.
UNUSABLE	값이 TRUE면 정합성이 훼손된 인덱스를 UNUSABLE 상태로 만든다.

- 예제

```
declare
    schema varchar2(1024);
begin
    schema := 'TIBERO';
    DBMS_VERIFY.SCHEMA_INDEX(schema);
end;
/
```

51.2.3. TABLE_INDEX

해당 테이블에 존재하는 모든 인덱스의 정합성을 검사하는 함수이다.

TABLE_INDEX 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_VERIFY.TABLE_INDEX
(
    SCHEMA          IN          VARCHAR2,
    TBLNAME         IN          VARCHAR2,
    DOP             IN          NUMBER DEFAULT 0,
    UNUSABLE        IN          BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
SCHEMA	인덱스의 정합성을 검사할 스키마의 이름이다.
TBLNAME	인덱스의 정합성을 검사할 테이블의 이름이다.
DOP	Degree Of Parallelism으로 병렬화할 정도를 의미한다.
UNUSABLE	값이 TRUE면 정합성이 훼손된 인덱스를 UNUSABLE 상태로 만든다.

- 예제

```
declare
    schema varchar2(1024);
    tblname varchar2(1024);
begin
    schema := 'TIBERO';
    tblname := 'TT';
    DBMS_VERIFY.TABLE_INDEX(schema, tblname);
end;
/
```

51.2.4. INDEX

인덱스의 정합성을 검사하기 위해서 인덱스로 찾은 행의 수와 테이블을 스캔하여 찾은 행의 수를 비교하는 함수이다.

INDEX 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_VERIFY.INDEX
(
    SCHEMA          IN          VARCHAR2,
    TBLNAME         IN          VARCHAR2,
    IDXNAME         IN          VARCHAR2,
    DOP             IN          NUMBER DEFAULT 0,
    UNUSABLE        IN          BOOLEAN DEFAULT FALSE,
```

```
    PARTNAME        IN              VARCHAR2 DEFAULT NULL,
    SUBPARTNAME     IN              VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
SCHEMA	인덱스의 정합성을 검사할 스키마의 이름이다.
TBLNAME	인덱스의 정합성을 검사할 테이블의 이름이다.
IDXNAME	정합성을 검사할 인덱스의 이름이다.
DOP	Degree Of Parallelism으로 병렬화할 정도를 의미한다.
UNUSABLE	값이 TRUE면 정합성이 훼손된 인덱스를 UNUSABLE 상태로 만든다.
PARTNAME	인덱스 정합성을 검사할 파티션의 이름이다. NULL일 경우 모든 Local Index Partition에 대해 검사한다.
SUBPARTNAME	인덱스 정합성을 검사할 서브파티션의 이름이다. NULL일 경우 모든 Local Index Subpartition에 대해 검사한다.

- 예제

```
declare
    schema varchar2(1024);
    tblname varchar2(1024);
    idxname varchar2(1024);
    partname varchar2(1024);
    subpartname varchar(1024);
begin
    schema := 'TIBERO';
    tblname := 'TT';
    idxname := 'TT_IDX';
    partname := 'TT_PART';
    subpartname := 'TT_SUBPART';

    DBMS_VERIFY.INDEX(schema, tblname, idxname, 4, TRUE, partname, subpartname);
end;
/
```

51.2.5. TABLE_INDEX_SORT

특정 테이블에 속한 모든 인덱스에 대하여 정렬 정합성 체크를 수행하는 함수이다.

TABLE_INDEX_SORT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_VERIFY.TABLE_INDEX_SORT
(
    SCHEMA          IN              VARCHAR2,
    TBLNAME         IN              VARCHAR2,
    DESCRIPTION     IN              BOOLEAN DEFAULT FALSE,
    UNUSABLE        IN              BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
SCHEMA	체크하려는 테이블이 소속된 스키마의 이름이다.
TBLNAME	체크하려는 인덱스들이 포함된 테이블의 이름이다.
DESCRIPTION	값이 TRUE면 체크 도중 상세 진행 설명을 출력한다. SET SERVEROUTPUT ON 상태에서만 출력된다.
UNUSABLE	값이 TRUE면 정합성이 훼손된 인덱스를 UNUSABLE 상태로 만든다. 만약 파티션이나 서브 파티션이 설정되어 있다면, 해당 파티션 혹은 서브파티션만 UNUSABLE 상태로 만든다.

- 예제

```
declare
    schema varchar2(1024);
    tblname varchar2(1024);
begin
    schema := 'TIBERO';
    tblname := 'TT';

    DBMS_VERIFY.TABLE_INDEX_SORT(schema, tblname);
end;
/
```

51.2.6. INDEX_SORT

특정 인덱스에 대하여 정렬 정합성 체크를 수행하는 함수이다.

INDEX_SORT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_VERIFY.INDEX_SORT
(
    SCHEMA          IN              VARCHAR2,
    IDXNAME         IN              VARCHAR2,
```

```
    PARTNAME        IN              VARCHAR2 DEFAULT NULL,
    DESCRIPTION     IN              BOOLEAN DEFAULT FALSE,
    UNUSABLE        IN              BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
SCHEMA	체크하려는 테이블이 소속된 스키마의 이름이다.
IDXNAME	체크하려는 인덱스의 이름이다.
PARTNAME	인덱스에서 체크할 특정 파티션 혹은 서브 파티션의 이름이다.
DESCRIPTION	값이 TRUE면 체크 도중 상세 진행 설명을 출력한다. SET SERVEROUTPUT ON 상태에서만 출력된다.
UNUSABLE	값이 TRUE면 정합성이 훼손된 인덱스를 UNUSABLE 상태로 만든다. 만약 파티션이나 서브 파티션이 설정되어 있다면, 해당 파티션 혹은 서브파티션만 UNUSABLE 상태로 만든다.

- 예제

```
declare
    schema varchar2(1024);
    idxname varchar2(1024);
begin
    schema := 'TIBERO';
    idxname := 'IDX';

    DBMS_VERIFY.INDEX_SORT(schema, idxname);
end;
/
```

제52장 DBMS_XMLDOM

본 장에서는 DBMS_XMLDOM 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다. DBMS_XMLDOM 패키지는 Solaris x86에서는 지원되지 않는다.

52.1. 개요

DBMS_XMLDOM은 XML 문서 처리를 위한 DOM API를 제공한다.

다음은 DBMS_XMLDOM 패키지 내에 정의된 상수이다.

```
ELEMENT_NODE CONSTANT PLS_INTEGER := 1;
ATTRIBUTE_NODE CONSTANT PLS_INTEGER := 2;
TEXT_NODE CONSTANT PLS_INTEGER := 3;
CDATA_SECTION_NODE CONSTANT PLS_INTEGER := 4;
ENTITY_REFERENCE_NODE CONSTANT PLS_INTEGER := 5;
ENTITY_NODE CONSTANT PLS_INTEGER := 6;
PROCESSING_INSTRUCTION_NODE CONSTANT PLS_INTEGER := 7;
COMMENT_NODE CONSTANT PLS_INTEGER := 8;
DOCUMENT_NODE CONSTANT PLS_INTEGER := 9;
DOCUMENT_TYPE_NODE CONSTANT PLS_INTEGER := 10;
DOCUMENT_FRAGMENT_NODE CONSTANT PLS_INTEGER := 11;
NOTATION_NODE CONSTANT PLS_INTEGER := 12;
```

52.2. 타입

본 절에서는 DBMS_XMLDOM 패키지에서 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

52.2.1. DOMAttr

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Attribute를 의미한다.

DOMAttr 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMAttr IS RECORD(id RAW(16));
```

52.2.2. DOMCDataSection

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 CDataSection을 의미한다.

DOMCDataSection 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMCDataSection IS RECORD(id RAW(16));
```

52.2.3. DOMCharacterData

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 CharacterData를 의미한다.

DOMCharacterData 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMCharacterData IS RECORD(id RAW(16));
```

52.2.4. DOMComment

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Comment를 의미한다.

DOMComment 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMComment IS RECORD(id RAW(16));
```

52.2.5. DOMDocument

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Document를 의미한다.

DOMDocument 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMDocument IS RECORD(id RAW(16));
```

52.2.6. DOMDocumentFragment

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 DocumentFragment를 의미한다.

DOMDocumentFragment 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMDocumentFragment IS RECORD(id RAW(16));
```

52.2.7. DOMDocumentType

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 DocumentType을 의미한다.

DOMDocumentType 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMDocumentType IS RECORD(id RAW(16));
```

52.2.8. DOMElement

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Element를 의미한다.

DOMElement 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMElement IS RECORD(id RAW(16));
```

52.2.9. DOMEntity

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Entity를 의미한다.

DOMEntity 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMEntity IS RECORD(id RAW(16));
```

52.2.10. DOMEntityReference

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 EntityReference를 의미한다.

DOMEntityReference 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMEntityReference IS RECORD(id RAW(16));
```

52.2.11. DOMImplementation

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Implementation을 의미한다.

DOMImplementation 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMImplementation IS RECORD(id RAW(16));
```

52.2.12. DOMNamedNodeMap

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 NamedNodeMap을 의미한다.

DOMNamedNodeMap 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMNamedNodeMap IS RECORD(id RAW(16));
```

52.2.13. DOMNode

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 노드를 의미한다.

DOMNode 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMNode IS RECORD(id RAW(16));
```

52.2.14. DOMNodeList

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 NodeList를 의미한다.

DOMNodeList 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMNodeList IS RECORD(id RAW(16));
```

52.2.15. DOMNotation

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Notation을 의미한다.

DOMNotation 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMNotation IS RECORD(id RAW(16));
```

52.2.16. DOMProcessingInstruction

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 ProcessingInstruction을 의미한다.

DOMProcessingInstruction 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMProcessingInstruction IS RECORD(id RAW(16));
```

52.2.17. DOMText

RAW 타입으로 이루어진 ID를 가진 RECORD 타입이다. DOM에서 Text를 의미한다.

DOMText 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE DOMText IS RECORD(id RAW(16));
```

52.3. 프러시저와 함수

본 절에서는 DBMS_XMLDOM 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

52.3.1. ADOPTNODE

선택된 노드를 Document에 추가한다. 본래 있던 Document에서는 해당 노드가 제거되고, 새로운 노드를 출력한다.

ADOPTNODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.ADOPTNODE
(
    doc               IN           DOMDocument,
    importedNode      IN           DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
doc	노드가 추가될 Document이다.
importedNode	Document에 추가될 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    new_doc   dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    n_ele     dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    new_doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    n_ele := dbms_xmldom.makeNode(ele);
    n := dbms_xmldom.appendChild(docNode, n_ele);
    n := dbms_xmldom.adoptNode(new_doc, n_ele);
    dbms_xmldom.freeDocument(doc);
    dbms_xmldom.freeDocument(new_doc);
```

```
end;
/
```

52.3.2. APPENDCHILD

선택된 노드에 자식 노드를 추가하는 함수로 추가된 자식 노드를 출력한다.

APPENDCHILD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.APPENDCHILD
(
    n             IN              DOMNode,
    newchild      IN              DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	자식 노드가 추가될 노드이다.
newchild	노드에 추가될 자식 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    n := dbms_xmldom.makeNode(ele);
    n := dbms_xmldom.appendChild(docNode, n);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.3. APPENDDATA

DOMCharacterData 끝에 문자열을 연장한다.

APPENDDATA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.APPENDDATA
(
    cd            IN              DOMCharacterData,
    arg           IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
cd	DOMCharacterData이다.
arg	DOMCharacterData에 추가할 값이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    txt       dbms_xmldom.DOMText;
    cd        dbms_xmldom.DOMCharacterData;
begin
    doc := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'textnode');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makeNode(txt));
    dbms_xmldom.appendData(cd, '_appending');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.4. CLONENODE

DOMNode를 복사한다. 복사한 DOMNode는 부모 노드가 존재하지 않는다.

CLONENODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CLONENODE
(
```

```
    n             IN             DOMNode,
    deep          IN             BOOLEAN
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	DOMNode이다.
deep	자식 노드를 복사할지 결정한다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    ele        dbms_xmldom.DOMElement;
    n          dbms_xmldom.DOMNode;
    cloneNode  dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc,'test');
    n := dbms_xmldom.makeNode(ele);
    cloneNode := dbms_xmldom.cloneNode(n, false);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.5. CREATEATTRIBUTE

해당 문서 안에 DOMAttr을 생성하는 함수이다.

CREATEATTRIBUTE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATEATTRIBUTE
(
    doc           IN             DOMDocument,
    name          IN             VARCHAR2
)
RETURN DOMAttr;
```

```
DBMS_XMLDOM.CREATEATTRIBUTE
(
    doc           IN             DOMDocument,
    name          IN             VARCHAR2,
```

```
    ns              IN              VARCHAR2
)
RETURN DOMAttr;
```

- 파라미터

파라미터	설명
doc	DOMAttr이 생성될 문서이다.
name	새로운 DOMAttr의 qualified name이다.
ns	새로운 DOMAttr의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    attr      dbms_xmldom.DOMAttr;
begin
    doc := dbms_xmldom.newDomDocument;
    attr := dbms_xmldom.createAttribute(doc, 'test');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.6. CREATECDATASECTION

해당 문서 안에 CDataSection을 생성하는 함수이다.

CREATECDATASECTION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATECDATASECTION
(
    doc           IN              DOMDocument,
    data          IN              VARCHAR2
)
RETURN DOMCDataSection;
```

- 파라미터

파라미터	설명
doc	CDataSection이 생성될 문서이다.
data	CDataSection에 들어갈 내용이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    cd        dbms_xmldom.DOMCDataSection;
begin
    doc := dbms_xmldom.newDomDocument;
    cd  := dbms_xmldom.createCDataSection(doc, 'contents of cdatasection');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.7. CREATECOMMENT

해당 문서 안에 COMMENT를 생성하는 함수이다.

CREATECOMMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATECOMMENT
(
    doc           IN              DOMDocument,
    data          IN              VARCHAR2
)
RETURN DOMComment;
```

- 파라미터

파라미터	설명
doc	COMMENT가 생성될 문서이다.
data	COMMENT에 들어갈 내용이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    cm        dbms_xmldom.DOMComment;
begin
    doc := dbms_xmldom.newDomDocument;
    cm  := dbms_xmldom.createComment(doc, 'comment!!');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.8. CREATEDOCUMENT

해당 문서 안에 DOMDocument를 생성하는 함수이다.

CREATEDOCUMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATEDOCUMENT
(
    namespaceURI      IN          VARCHAR2,
    qualifiedName     IN          VARCHAR2,
    doctype           IN          DOMTYPE := NULL
)
RETURN DOMDocument;
```

- 파라미터

파라미터	설명
namespaceURI	DOMDocument의 네임스페이스 URI이다.
qualifiedName	루트 엘리먼트의 이름이다.
doctype	문서의 타입이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
begin
    doc := dbms_xmldom.createDocument('xmlns="http://www.w3.org/1999"', 'QName');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.9. CREATEDOCUMENTFRAGMENT

해당 문서 안에 DOMDocumentFragment를 생성하는 함수이다.

CREATEDOCUMENTFRAGMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATEDOCUMENTFRAGMENT
(
    doc           IN              DOMDocument,
```

```
)
RETURN DOMDocumentFragment;
```

- 파라미터

파라미터	설명
doc	Element가 생성될 문서이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    docFrg    dbms_xmldom.DOMDocumentFragment;
begin
    doc := dbms_xmldom.newDomDocument;
    docFrg := dbms_xmldom.createDocumentFragment(doc);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.10. CREATEELEMENT

해당 문서 안에 Element를 생성하는 함수이다.

CREATEELEMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATEELEMENT
(
    doc           IN              DOMDocument,
    tagname       IN              VARCHAR2
)
RETURN DOMElement;
```

```
DBMS_XMLDOM.CREATEELEMENT
(
    doc           IN              DOMDocument,
    tagname       IN              VARCHAR2,
    ns            IN              VARCHAR2
)
RETURN DOMElement;
```

- 파라미터

파라미터	설명
doc	Element가 생성될 문서이다.
tagname	Element의 태그 이름이다.
ns	Element의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'root');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.11. CREATEENTITYREFERENCE

해당 문서 안에 DOMEntityReference를 생성하는 함수이다.

CREATEENTITYREFERENCE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATEENTITYREFERENCE
(
    doc           IN              DOMDocument,
    name          IN              VARCHAR2
)
RETURN DOMEntityReference;
```

- 파라미터

파라미터	설명
doc	DOMEntityReference가 생성될 문서이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    etr       dbms_xmldom.DOMEntityReference;
begin
    doc := dbms_xmldom.newDomDocument;
```

```
    etr := dbms_xmldom.createEntityReference(doc, 'entity');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.12. CREATEPROCESSINGINSTRUCTION

해당 문서 안에 DOMProcessingInstruction를 생성하는 함수이다.

CREATEPROCESSINGINSTRUCTION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATEPROCESSINGINSTRUCTION
(
    doc           IN              DOMDocument,
    target        IN              VARCHAR2,
    data          IN              VARCHAR2
)
RETURN DOMEntityReference;
```

- 파라미터

파라미터	설명
doc	DOMProcessingInstruction이 생성될 문서이다.
target	ProcessingInstruction의 타깃이다.
data	ProcessingInstruction의 데이터이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    pi        dbms_xmldom.DOMProcessingInstruction;
begin
    doc := dbms_xmldom.newDomDocument;
    pi := dbms_xmldom.createProcessingInstruction(doc, 'target', 'data');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.13. CREATETEXTNODE

해당 문서 안에 Text를 생성하는 함수이다.

CREATETEXTNODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.CREATETEXTNODE
(
    doc           IN              DOMDocument,
    data          IN              VARCHAR2
)
RETURN DOMText;
```

- 파라미터

파라미터	설명
doc	Text가 생성될 문서이다.
data	Text에 들어갈 내용이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    txt       dbms_xmldom.DOMText;
begin
    doc := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'text');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.14. DELETEDATA

DOMCharacterData의 특정 오프셋에 데이터를 지운다.

DELETEDATA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.DELETEDATA
(
    cd            IN              DOMCharacterData,
    offset        IN              NUMBER,
    cnt           IN              NUMBER
);
```

- 파라미터

파라미터	설명
cd	DOMCharacterData이다.
offset	지워질 데이터의 오프셋이다.
cnt	지워질 문자 데이터의 개수이다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    txt        dbms_xmldom.DOMText;
    cd         dbms_xmldom.DOMCharacterData;
begin
    doc  := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'textnode');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makenode(txt));
    dbms_xmldom.deleteData(cd, 4, 4);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.15. FREEDOCFRAG

DOMDocumentFragment Type을 초기화해 주는 프러시저다.

FREEDOCFRAG 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.FREEDOCFRAG
(
    doc          IN              DOMDocumentFragment
);
```

- 파라미터

파라미터	설명
doc	초기화해 줄 DocumentFragment이다.

52.3.16. FREEDOCUMENT

Document에 관련된 리소스들을 해제하는 프러시저다.

FREEDOCUMENT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.FREEDOCUMENT
(
    doc          IN              DOMDocument
);
```

- 파라미터

파라미터	설명
doc	Resource를 해제할 문서이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
begin
    doc := dbms_xmldom.newDomDocument;
    dbms_xmldom.freedocument(doc);
end;
/
```

52.3.17. FREENODE

DOMNode에 관련된 리소스들을 해제하는 프러시저다. DOMNode가 문서인 경우에 DOMNode 타입을 초기화하고 해당 문서와 관련된 리소스를 해제한다. 문서가 아닌 경우에는 해당 DOMNode 타입만 초기화해준다.

FREENODE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.FREENODE
(
    n          IN              DOMNode
);
```

- 파라미터

파라미터	설명
n	리소스를 해제할 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
begin
    doc := dbms_xmldom.newDomDocument;
    dbms_xmldom.freenode(dbms_xmldom.makeNode(doc));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.18. GETATTRIBUTE

특정 Element로부터 Attribute의 값을 가져오는 함수이다.

GETATTRIBUTE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2
)
RETURN VARCHAR2;
```

```
DBMS_XMLDOM.GETATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    ns            IN              VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
elem	해당 Attribute를 찾을 Element이다.
name	Attribute의 이름이다.
ns	Attribute의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    attr_val  VARCHAR2(20);
```

```
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr_val := dbms_xmldom.getAttribute(ele, 'attr1');
    dbms_output.put_line(attr_val);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.19. GETATTRIBUTENODE

Element로부터 특정 DOMAttr을 가져오는 함수이다.

GETATTRIBUTENODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETATTRIBUTENODE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2
)
RETURN DOMAttr;
```

```
DBMS_XMLDOM.GETATTRIBUTENODE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    ns            IN              VARCHAR2
)
RETURN DOMAttr;
```

- 파라미터

파라미터	설명
elem	해당 Attribute를 찾을 Element이다.
name	Attribute의 이름이다.
ns	Attribute의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
```

```
    attr      dbms_xmldom.DOMAttr;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr := dbms_xmldom.getAttributeNode(ele, 'attr1');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.20. GETATTRIBUTES

Element에 있는 모든 Attribute들을 출력하는 함수이다.

GETATTRIBUTES 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETATTRIBUTES
(
    n          IN              DOMNode
)
RETURN DOMNamedNodeMap;
```

- 파라미터

파라미터	설명
n	DOMNode 형태의 Element가 와야한다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
    attr      dbms_xmldom.DOMAttr;
    nm        dbms_xmldom.DOMNamedNodeMap;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'first_attribute');
    dbms_xmldom.setAttribute(ele, 'attr2', 'second_attribute');
    nm := dbms_xmldom.getAttributes(dbms_xmldom.makeNode(ele));
    dbms_xmldom.freeDocument(doc);
```

```
end;
/
```

52.3.21. GETCHILDNODES

해당 노드의 자식 노드들을 출력한다.

GETCHILDNODES 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETCHILDNODES
(
    n          IN              DOMNode
)
RETURN DOMNodeList;
```

- 파라미터

파라미터	설명
n	자식 노드들을 출력할 부모 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    nl        dbms_xmldom.DOMNodeList;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child2');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    nl := dbms_xmldom.getChildNodes(p);
    dbms_xmldom.freeDocument(doc);
```

```
end;
/
```

52.3.22. GETCHILDRENBYTAGNAME

특정 태그명을 갖는 Element들을 자식 노드들에서 찾아서 출력해주는 함수이다.

GETCHILDRENBYTAGNAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETCHILDRENBYTAGNAME
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2
)
RETURN DOMNodeList;
```

```
DBMS_XMLDOM.GETCHILDRENBYTAGNAME
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    ns            IN              VARCHAR2
)
RETURN DOMNodeList;
```

- 파라미터

파라미터	설명
elem	자식 노드들을 출력할 Parent Element 노드이다.
name	찾을 노도들의 태그명이다.
ns	찾을 노드들의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    nl        dbms_xmldom.DOMNodeList;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
```

```
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    nl := dbms_xmldom.getChildrenByTagName(ele, 'child1');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.23. GETDATA

DOMChatateрData 데이터를 가져온다.

GETDATA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETDATA
(
    cd          IN              DOMCharacterData
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cd	데이터를 가져올 DOMCharacterData이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    txt       dbms_xmldom.DOMText;
    cd        dbms_xmldom.DOMCharacterData;
begin
    doc := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'textnode');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makeNode(txt));
    dbms_output.put_line(dbms_xmldom.getData(cd));
    dbms_xmldom.freeDocument(doc);
```

```
end;
/
```

52.3.24. GETDOCUMENTELEMENT

문서의 root element를 가져온다.

GETDOCUMENTELEMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETDOCUMENTELEMENT
(
    doc           IN              DOMDocument
)
RETURN DOMElement;
```

- 파라미터

파라미터	설명
doc	root element를 가져올 문서이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument(
           '<first><second></second></first>');
    ele := dbms_xmldom.getDocumentElement(doc);
    n := dbms_xmldom.makeNode(ele);
    dbms_output.put_line(dbms_xmldom.getNodeName(n));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.25. GETELEMENTSBYTAGNAME

특정 태그명을 갖는 Element들을 찾아서 출력해주는 함수이다.

GETELEMENTSBYTAGNAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME
(
    doc           IN              DOMDocument,
    name          IN              VARCHAR2
)
RETURN DOMNodeList;
```

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2
)
RETURN DOMNodeList;
```

```
DBMS_XMLDOM.GETELEMENTSBYTAGNAME
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    ns            IN              VARCHAR2
)
RETURN DOMNodeList;
```

- 파라미터

파라미터	설명
doc	Element들을 찾을 Document 노드이다.
elem	Element들을 찾을 Parent Element 노드이다.
name	찾을 노도들의 태그명이다.
ns	찾을 노드들의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    nl        dbms_xmldom.DOMNodeList;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
```

```
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    nl := dbms_xmldom.getElementsByTagname(doc, 'child1');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.26. GETEXPANDEDNAME

확장된 이름을 가져오는 프러시저와 함수이다.

GETEXPANDEDNAME 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETEXPANDEDNAME
(
    n             IN              DOMNode,
    data          IN              VARCHAR2
);
```

```
DBMS_XMLDOM.GETEXPANDEDNAME
(
    a          IN              DOMAttr
)
RETURN VARCHAR2;
```

```
DBMS_XMLDOM.GETEXPANDEDNAME
(
    elem          IN              DOMElement,
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
n	DOMNode이다.
data	확장된 이름을 가져올 노드의 이름이다.
a	DOMAttr이다.
elem	DOMElement이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'name', 'ns');
    dbms_output.put_line(dbms_xmldom.getExpandedName(ele));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.27. GETFIRSTCHILD

첫 번째에 있는 자식 노드를 출력하는 함수이다.

GETFIRSTCHILD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETFIRSTCHILD
(
    n           IN              DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	자식을 찾을 부모 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
```

```
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child2');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    n := dbms_xmldom.getFirstChild(p);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.28. GETIMPLEMENTATION

문서의 DOMImplementation을 가져온다.

GETIMPLEMENTATION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETIMPLEMENTATION
(
    doc         IN              DOMDocument
)
RETURN DOMImplementation;
```

- 파라미터

파라미터	설명
doc	DOMImplementation을 가져올 문서이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    impl      dbms_xmldom.DOMImplementation;
begin
    doc := dbms_xmldom.newDomDocument;
    impl := dbms_xmldom.getImplementation(doc);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.29. GETLASTCHILD

마지막에 있는 자식 노드를 출력하는 함수이다.

GETLASTCHILD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETLASTCHILD
(
    n           IN              DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	자식을 찾을 부모 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child2');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    n := dbms_xmldom.getLastChild(p);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.30. GETLENGTH

길이를 출력하는 함수이다. CharacterData의 경우에는 문자의 길이를 출력하고, DOMNamedNodeMap 또는 DOMNodeList인 경우에는 개수를 출력한다.

GETLENGTH 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETLENGTH
(
    cd           IN              DOMCharacterData
)
RETURN NUMBER;
```

```
DBMS_XMLDOM.GETLENGTH
(
    nnm           IN              DOMNamedNodeMap
)
RETURN NUMBER;
```

```
DBMS_XMLDOM.GETLENGTH
(
    nl           IN              DOMNodeList
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
cd	CharacterData를 가지고 있는 노드이다.
nnm	여러 노드 정보를 맵 형태로 가지고 있는 타입이다.
nl	여러 노드 정보를 List 형태로 가지고 있는 타입이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    nl        dbms_xmldom.DOMNodeList;
    len       Number;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child2');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
```

```
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    nl := dbms_xmldom.getElementsByTagname(doc, 'child1');
    len := dbms_xmldom.getLength(nl);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.31. GETLOCALNAME

DOMAttr와 DOMElement에서 로컬이름을 가져온다. DOMNode는 qualified 이름에서 로컬 이름을 가져온다.

GETLOCALNAME 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETLOCALNAME
(
    n           IN              DOMNode,
    data        OUT             VARCHAR2
);
```

```
DBMS_XMLDOM.GETLOCALNAME
(
    a           IN              DOMAttr
)
RETURN VARCHAR2;
```

```
DBMS_XMLDOM.GETLOCALNAME
(
    elem        IN              DOMElement
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
n	DOMNode이다.
data	local 이름이 리턴된다.
a	DOMAttr이다.
elem	DOMElement이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'name', 'ns');
    dbms_output.put_line(dbms_xmldom.getlocalname(ele));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.32. GETNAME

해당 노드의 이름을 출력해주는 함수이다.

GETNAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETNAME
(
    a           IN              DOMAttr
)
RETURN VARCHAR2;
```

```
DBMS_XMLDOM.GETNAME
(
    dt           IN              DOMDocumentType
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
a	이름을 가져올 Attribute 노드이다.
dt	이름을 가져올 DOMDocumentType 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    attr      dbms_xmldom.DOMAttr;
begin
    doc := dbms_xmldom.newDomDocument;
```

```
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr := dbms_xmldom.getAttributeNode(ele, 'attr1');
    dbms_output.put_line(dbms_xmldom.getName(attr));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.33. GETNAMEDITEM

DOMNamedNodeMap에서 특정 이름을 갖는 노드를 가져오는 함수이다.

GETNAMEDITEM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETNAMEDITEM
(
    nnm         IN              DOMNamedNodeMap,
    name        IN              VARCHAR2
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.GETNAMEDITEM
(
    nnm         IN              DOMNamedNodeMap,
    name        IN              VARCHAR2,
    ns          IN              VARCHAR2
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
nnm	노드들을 맵 형태로 가지고 있는 DOMNamedNodeMap 타입이다.
name	DOMNamedNodeMap에서 가져올 노드의 이름이다.
ns	DOMNamedNodeMap에서 가져올 노드의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
    nm        dbms_xmldom.DOMNamedNodeMap;
```

```
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'first_attribute');
    dbms_xmldom.setAttribute(ele, 'attr2', 'second_attribute');
    nm := dbms_xmldom.getAttributes(dbms_xmldom.makeNode(ele));
    n := dbms_xmldom.getNamedItem(nm, 'attr1');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.34. GETNEXTSIBLING

다음 형제 노드를 가져오는 함수이다.

GETNEXTSIBLING 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETNEXTSIBLING
(
    n         IN              DOMNode,
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	DOMNode이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    n         dbms_xmldom.DOMNode;
    docNode   dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument('<root><tag1></tag1><tag2></tag2></root>');
    docNode := dbms_xmldom.makeNode(doc);
    n := dbms_xmldom.getFirstChild(docNode);
    n := dbms_xmldom.getNextSibling(n);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.35. GETNODENAME

노드의 이름을 출력하는 함수이다.

GETNODENAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETNODENAME
(
    n         IN              DOMNode
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
n	이름을 출력할 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
    nm        dbms_xmldom.DOMNamedNodeMap;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'first_attribute');
    dbms_xmldom.setAttribute(ele, 'attr2', 'second_attribute');
    nm := dbms_xmldom.getAttributes(dbms_xmldom.makeNode(ele));
    n := dbms_xmldom.getNamedItem(nm, 'attr1');
    dbms_output.put_line(dbms_xmldom.getNodeName(n));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.36. GETNODETYPE

노드의 타입 정보를 DBMS_XMLDOM의 상수형태로 출력해주는 함수이다.

GETNODETYPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETNODETYPE
(
    n         IN              DOMNode
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
n	타입 정보를 출력할 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'root');
    dbms_output.put_line(dbms_xmldom.getNodeType(dbms_xmldom.makeNode(doc)));
    dbms_output.put_line(dbms_xmldom.getNodeType(dbms_xmldom.makeNode(ele)));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.37. GETNODEVALUE

노드의 값을 출력해주는 함수이다.

GETNODEVALUE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETNODEVALUE
(
    n         IN              DOMNode
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
n	값을 출력할 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    txt       dbms_xmldom.DOMText;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'text');
    n := dbms_xmldom.makeNode(txt);
    dbms_output.put_line(dbms_xmldom.getNodeValue(n));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.38. GETOWNERDOCUMENT

해당 노드가 속해 있는 Document를 출력해주는 함수이다.

GETOWNERDOCUMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETOWNERDOCUMENT
(
    n         IN              DOMNode
)
RETURN DOMDocument;
```

- 파라미터

파라미터	설명
n	문서를 찾을 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'root');
    doc := dbms_xmldom.getOwnerDocument(dbms_xmldom.makeNode(ele));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.39. GETPARENTNODE

해당 노드의 부모 노드를 출력해주는 함수이다.

GETPARENTNODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETPARENTNODE
(
    n           IN              DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	부모를 찾을 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    n1        dbms_xmldom.DOMNode;
    n2        dbms_xmldom.DOMNode;
    n3        dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n1 := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child2');
    n2 := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n3 := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    n := dbms_xmldom.getParentNode(n1);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.40. GETQUALIFIEDNAME

Element 또는 Attribute의 qualified name을 반환하는 함수이다.

GETQUALIFIEDNAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETQUALIFIEDNAME
(
    n         IN              DOMAttr
)
RETURN VARCHAR2;
```

```
DBMS_XMLDOM.GETQUALIFIEDNAME
(
    n         IN              DOMElement
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
n	Qualified name을 출력할 Attribute 노드 또는 Element 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    docNode   dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    ele       dbms_xmldom.DOMElement;
    attr      dbms_xmldom.DOMAttr;
    attr_val  VARCHAR2(20);
begin
    doc := dbms_xmldom.newDomDocument(
             '<ns:test xmlns:ns="URI">content</ns:test>');
    docNode := dbms_xmldom.makeNode(doc);
    n := dbms_xmldom.getFirstChild(docNode);
    ele := dbms_xmldom.makeElement(n);
    dbms_output.put('QName : ');
    dbms_output.put_line(dbms_xmldom.getqualifiedname(ele));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.41. GETVALUE

Attribute노드의 값을 출력해주는 함수이다.

GETVALUE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETVALUE
(
    a         IN              DOMAttr
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
a	값을 출력할 Attribute 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    attr      dbms_xmldom.DOMAttr;
    attr_val  VARCHAR2(20);
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr := dbms_xmldom.getAttributeNode(ele, 'attr1');
    attr_val := dbms_xmldom.getValue(attr);
    dbms_output.put_line(attr_val);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.42. GETXMLTYPE

문서를 XMLType 형태로 출력한다.

GETXMLTYPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.GETXMLTYPE
(
    doc          IN              DOMDocument
)
RETURN XMLType;
```

- 파라미터

파라미터	설명
doc	XMLType 형태로 출력된 DOM 형태의 문서이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    x         XMLType;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    n := dbms_xmldom.makeNode(ele);
    n := dbms_xmldom.appendChild(docNode, n);
    x := dbms_xmldom.getXMLType(doc);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.43. HASATTRIBUTE

해당 Element 노드가 특정 ATTRIBUTE를 가지고 있는지 확인하는 함수이다.

HASATTRIBUTE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.HASATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.HASATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    ns            IN              VARCHAR2
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
elem	해당 Attribute를 찾을 Element이다.
name	Attribute의 이름이다.
ns	Attribute의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    has_attr  BOOLEAN;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    has_attr := dbms_xmldom.HasAttribute(ele, 'attr1');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.44. HASFEATURE

DOMImplementation에 특정 기능이 있는지 확인하는 함수이다.

HASFEATURE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.HASFEATURE
(
    di            IN              DOMImplementation,
    feature       IN              VARCHAR2,
    version       IN              VARCHAR2
```

```
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
di	DOMImplementation이다.
feature	확인할 기능이다.
version	DOM 버전이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    impl      dbms_xmldom.DOMImplementation;
    feature   VARCHAR2(30);
    version   VARCHAR2(30);
begin
    doc := dbms_xmldom.newDomDocument;
    impl := dbms_xmldom.getImplementation(doc);
    feature := 'XML';
    version := '1.0';
    if dbms_xmldom.hasfeature(impl, feature, version) then
        dbms_output.put_line(feature || version);
    end if;
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.45. IMPORTNODE

문서 안에 해당 노드를 복사하여 추가하는 함수이다.

IMPORTNODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.IMPORTNODE
(
    doc               IN              DOMDocument,
    importedNode      IN              DOMNode,
    deep              IN              BOOLEAN
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
doc	노드를 추가할 문서이다.
importedNode	추가될 노드이다.
deep	– True : 해당 노드의 밑에 있는 모든 노드들을 추가한다. – False : 해당 노드만 추가한다.

- 예제

```
declare
    doc1       dbms_xmldom.DOMDocument;
    doc2       dbms_xmldom.DOMDocument;
    ele1       dbms_xmldom.DOMElement;
    ele2       dbms_xmldom.DOMElement;
    doc1Node   dbms_xmldom.DOMNode;
    n0         dbms_xmldom.DOMNode;
    n1         dbms_xmldom.DOMNode;
    n2         dbms_xmldom.DOMNode;
begin
    doc1 := dbms_xmldom.newDomDocument;
    doc2 := dbms_xmldom.newDomDocument;
    doc1Node := dbms_xmldom.makeNode(doc1);
    ele1 := dbms_xmldom.createElement(doc1, 'rootTag');
    ele2 := dbms_xmldom.createElement(doc1, 'ele');
    n1 := dbms_xmldom.appendChild(doc1Node, dbms_xmldom.makeNode(ele1));
    n2 := dbms_xmldom.appendChild(n1, dbms_xmldom.makeNode(ele2));
    n0 := dbms_xmldom.importNode(doc2, n1, FALSE);
    dbms_xmldom.freeDocument(doc1);
    dbms_xmldom.freeDocument(doc2);
end;
/
```

52.3.46. INSERTBEFORE

선택된 노드 앞에 해당노드를 추가하는 함수이다. 선택된 노드가 Null일 경우, 해당 노드를 가장 끝에 추가한다.

INSERTBEFORE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.INSERTBEFORE
(
    n             IN              DOMNode,
    newchild      IN              DOMNode,
    refchild      IN              DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	추가될 노드의 부모 노드이다.
newchild	노드에 추가될 자식 노드이다.
refchild	refchild 앞에 newchild가 추가된다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    ele        dbms_xmldom.DOMElement;
    ele1       dbms_xmldom.DOMElement;
    ele2       dbms_xmldom.DOMElement;
    ele3       dbms_xmldom.DOMElement;
    docNode    dbms_xmldom.DOMNode;
    nele       dbms_xmldom.DOMNode;
    n          dbms_xmldom.DOMNode;
    n1         dbms_xmldom.DOMNode;
    n2         dbms_xmldom.DOMNode;
begin
    doc  := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele  := dbms_xmldom.createElement(doc, 'rootTag');
    ele1 := dbms_xmldom.createElement(doc, 'ele1');
    ele2 := dbms_xmldom.createElement(doc, 'ele2');
    ele3 := dbms_xmldom.createElement(doc, 'ele3');
    nele := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    n1   := dbms_xmldom.appendChild(nele, dbms_xmldom.makeNode(ele1));
    n2   := dbms_xmldom.appendChild(nele, dbms_xmldom.makeNode(ele2));
    n := dbms_xmldom.insertBefore(nele, dbms_xmldom.makeNode(ele3), n1);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.47. INSERTDATA

DOMCharacterData의 특정 오프셋에 데이터를 추가한다.

INSERTDATA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.INSERTDATA
(
    cd            IN              DOMCharacterData,
    offset        IN              NUMBER,
    arg           IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
cd	DOMCharacterData이다.
offset	데이터를 추가 할 위치이다.
arg	추가할 데이터이다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    txt        dbms_xmldom.DOMText;
    cd         dbms_xmldom.DOMCharacterData;
begin
    doc  := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'node');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makenode(txt));
    dbms_xmldom.insertData(cd, 0, 'text');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.48. ISNULL

NULL 여부를 확인해주는 함수이다.

ISNULL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.ISNULL
(
    a             IN              DOMAttr
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    cds           IN              DOMCDataSection
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    cd            IN              DOMCharacterData
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    com           IN              DOMComment
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    doc           IN              DOMDocument
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    df            IN              DOMDocumentFragment
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    dt            IN              DOMDocumentType
```

```
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    elem          IN              DOMElement
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    ent           IN              DOMEntity
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    eref          IN              DOMEntityReference
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    di            IN              DOMImplementation
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    nnm           IN              DOMNamedNodeMap
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    n             IN              DOMNode
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    nl            IN              DOMNodeList
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    nt            IN              DOMNotation
```

```
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    pi            IN              DOMProcessingInstruction
)
RETURN BOOLEAN;
```

```
DBMS_XMLDOM.ISNULL
(
    t             IN              DOMText
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
a	DOMAttr이 NULL인지 확인한다.
cds	DOMCDataSection이 NULL인지 확인한다.
cd	DOMCharacterData가 NULL인지 확인한다.
com	DOMComment가 NULL인지 확인한다.
doc	DOMDocument가 NULL인지 확인한다.
df	DOMDocumentFragment가 NULL인지 확인한다.
dt	DOMDocumentType이 NULL인지 확인한다.
elem	DOMElement가 NULL인지 확인한다.
ent	DOMEntity가 NULL인지 확인한다.
eref	DOMEntityReference가 NULL인지 확인한다.
di	DOMImplementation이 NULL인지 확인한다.
nnm	DOMNamedNodeMap이 NULL인지 확인한다.
n	DOMNode가 NULL인지 확인한다.
nl	DOMNodeList가 NULL인지 확인한다.
nt	DOMNotation이 NULL인지 확인한다.
pi	DOMProcessingInstruction이 NULL인지 확인한다.
t	DOMText가 NULL인지 확인한다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
begin
```

```
    dbms_xmldom.isNULL(doc);
    doc  := dbms_xmldom.newDomDocument;
    dbms_xmldom.isNULL(doc);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.49. ITEM

NamedNodeMap과 NodeList에서 해당 Item을 출력해주는 함수이다.

ITEM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.ITEM
(
    nl          IN              DOMNodeList,
    index       IN              Number
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.ITEM
(
    nnm         IN              DOMNamedNodeMap,
    index       IN              Number
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
nl	Item을 가져올 NodeList이다.
nnm	Item을 가져올 NamedNodeMap이다.
index	가져올 Item의 Index이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    p         dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    nl        dbms_xmldom.DOMNodeList;
```

```
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    p := dbms_xmldom.appendChild(docNode, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child1');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child2');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    ele := dbms_xmldom.createElement(doc, 'child3');
    n := dbms_xmldom.appendChild(p, dbms_xmldom.makeNode(ele));
    nl := dbms_xmldom.getChildNodes(p);
    n := dbms_xmldom.item(nl, 1);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.50. MAKECHARACTERDATA

노드를 DOMCharacterData로 캐스팅한다.

MAKECHARACTERDATA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.MAKECHARACTERDATA
(
    n            IN              DOMNode
)
RETURN DOMAttr;
```

- 파라미터

파라미터	설명
n	DOMCharacterData로 캐스팅할 DOMNode이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    txt       dbms_xmldom.DOMText;
    n         dbms_xmldom.DOMNode;
    cd        dbms_xmldom.DOMCharacterData;
begin
    doc := dbms_xmldom.newDomDocument;
```

```
    txt := dbms_xmldom.createTextNode(doc, 'DomText');
    n := dbms_xmldom.makeNode(txt);
    cd := dbms_xmldom.makeCharacterData(n);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.51. MAKEATTR

노드를 Attribute로 캐스팅한다.

MAKEATTR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.MAKEATTR
(
    n            IN              DOMNode
)
RETURN DOMAttr;
```

- 파라미터

파라미터	설명
n	DOMAttr로 캐스팅할 DOMNode이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
    attr      dbms_xmldom.DOMAttr;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr := dbms_xmldom.getAttributeNode(ele, 'attr1');
    n := dbms_xmldom.makeNode(attr);
    attr := dbms_xmldom.makeAttr(n);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.52. MAKEDOCUMENT

노드를 Document로 캐스팅한다.

MAKEDOCUMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.MAKEDOCUMENT
(
    n            IN              DOMNode
)
RETURN DOMDocument;
```

- 파라미터

파라미터	설명
n	DOMDocument로 캐스팅할 DOMNode이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    root      dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    root := dbms_xmldom.makeNode(doc);
    doc := dbms_xmldom.makeDocument(root);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.53. MAKEELEMENT

노드를 Element로 캐스팅한다.

MAKEELEMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.MAKEELEMENT
(
    n            IN              DOMNode
)
RETURN DOMElement;
```

- 파라미터

파라미터	설명
n	DOMElement로 캐스팅할 DOMNode이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    n := dbms_xmldom.makeNode(ele);
    ele := dbms_xmldom.makeElement(n);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.54. MAKENODE

Document, DOMAttr 등을 DOMNode로 캐스팅한다.

MAKENODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.MAKENODE
(
    a            IN             DOMAttr
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    cds          IN             DOMCDataSection
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    cd           IN             DOMCharacterData
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    com          IN             DOMComment
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    doc          IN             DOMDocument
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    df           IN             DOMDocumentFragment
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    dt           IN             DOMDocumentType
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    elem         IN             DOMElement
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    ent          IN             DOMEntity
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    eref         IN             DOMEntityReference
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    n            IN             DOMNotation
)

RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    pi           IN             DOMProcessingInstruction
)
RETURN DOMNode;
```

```
DBMS_XMLDOM.MAKENODE
(
    t            IN             DOMText
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
a	DOMNode로 캐스팅할 DOMAttr이다.
cds	DOMNode로 캐스팅할 DOMCDataSection이다.
cd	DOMNode로 캐스팅할 DOMCharacterData이다.
com	DOMNode로 캐스팅할 DOMComment이다.
doc	DOMNode로 캐스팅할 DOMDocument이다.
df	DOMNode로 캐스팅할 DOMDocumentFragment이다.
dt	DOMNode로 캐스팅할 DOMDocumentType이다.
elem	DOMNode로 캐스팅할 DOMElement이다.
ent	DOMNode로 캐스팅할 DOMEntity이다.
eref	DOMNode로 캐스팅할 DOMEntityReference이다.
n	DOMNode로 캐스팅할 DOMNotation이다.
pi	DOMNode로 캐스팅할 DOMProcessingInstruction이다.
t	DOMNode로 캐스팅할 DOMText이다.

- 예제

```
declare
    doc          dbms_xmldom.DOMDocument;
    docNode      dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.55. NEWDOMDOCUMENT

DOM 형태의 문서를 생성한다.

NEWDOMDOCUMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.NEWDOMDOCUMENT
RETURN DOMDocument;
```

```
DBMS_XMLDOM.NEWDOMDOCUMENT
(
    xmldoc       IN             SYS.XMLTYPE
)
RETURN DOMDocument;
```

```
DBMS_XMLDOM.NEWDOMDOCUMENT
(
    cl           IN             CLOB
)
RETURN DOMDocument;
```

- 파라미터

파라미터	설명
xmldoc	DOMDocument를 생성할 입력값이다.
cl	DOMDocument를 생성할 입력값이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
begin
    doc := dbms_xmldom.newDomDocument;
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.56. REMOVEATTRIBUTE

해당 Element노드의 특정 ATTRIBUTE를 제거하는 프러시저다.

REMOVEATTRIBUTE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.REMOVEATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2
);
```

```
DBMS_XMLDOM.REMOVEATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    ns            IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
elem	해당 Attribute를 찾을 Element이다.
name	Attribute의 이름이다.
ns	Attribute의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    dbms_xmldom.removeAttribute(ele, 'attr1');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.57. REMOVECHILD

해당 노드의 자식 노드를 제거하는 함수이다.

REMOVECHILD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.REMOVECHILD
(
    n             IN              DOMNode,
    oldchild      IN              DOMNode
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	자식 노드를 가지고 있는 노드이다.
oldchild	지워질 자식 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    nele      dbms_xmldom.DOMNode;
    deleted   dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    nele := dbms_xmldom.makeNode(ele);
    nele := dbms_xmldom.appendChild(docNode, nele);
    deleted := dbms_xmldom.removeChild(docNode, nele);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.58. REPLACECHILD

해당 노드의 자식 노드를 교체하는 함수이다.

REPLACECHILD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.REPLACECHILD
(
    n             IN              DOMNode,
    newchild      IN              DOMNode,
    oldchild      IN              DOMNode
```

```
)
RETURN DOMNode;
```

- 파라미터

파라미터	설명
n	자식 노드를 가지고 있는 노드이다.
newchild	새로 넣을 자식 노드이다.
oldchild	바꿔질 자식 노드이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele1      dbms_xmldom.DOMElement;
    ele2      dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    nele1     dbms_xmldom.DOMNode;
    nele2     dbms_xmldom.DOMNode;
    replaced  dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele1 := dbms_xmldom.createElement(doc, 'rootTag');
    ele2 := dbms_xmldom.createElement(doc, 'newroot');
    nele1 := dbms_xmldom.makeNode(ele1);
    nele2 := dbms_xmldom.makeNode(ele2);
    nele1 := dbms_xmldom.appendChild(docNode, nele1);
    replaced := dbms_xmldom.replaceChild(docNode, nele2, nele1);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.59. REPLACEDATA

DOMCharacterData의 특정 범위의 데이터를 변경한다.

REPLACEDATA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.REPLACEDATA
(
    cd            IN              DOMCharacterData,
```

```
    offset        IN              NUMBER,
    cnt           IN              NUMBER,
    arg           IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
cd	DOMCharacterData이다.
offset	변경할 데이터의 오프셋이다.
cnt	교체될 문자의 개수이다.
arg	교체할 값이다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    txt        dbms_xmldom.DOMText;
    cd         dbms_xmldom.DOMCharacterData;
begin
    doc  := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'node');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makenode(txt));
    dbms_xmldom.setData(cd, 'text');
    dbms_xmldom.replacedata(cd, 2, 1, 's');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.60. SETATTRIBUTE

특정 Element에 Attribute를 추가하는 프러시저다.

SETATTRIBUTE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SETATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    newvalue      IN              VARCHAR2
);
```

```
DBMS_XMLDOM.SETATTRIBUTE
(
    elem          IN              DOMElement,
    name          IN              VARCHAR2,
    newvalue      IN              VARCHAR2,
    ns            IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
elem	해당 Attribute를 넣을 Element이다.
name	Attribute의 이름이다.
newvalue	Attribute의 값이다.
ns	Attribute의 네임스페이스이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    n         dbms_xmldom.DOMNode;
    attr_val  VARCHAR2(20);
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr_val := dbms_xmldom.getAttribute(ele, 'attr1');
    dbms_output.put_line(attr_val);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.61. SETATTRIBUTENODE

특정 Element에 Attribute를 추가하는 함수이다.

SETATTRIBUTENODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SETATTRIBUTENODE
(
    elem          IN              DOMElement,
```

```
    attr       IN                VARCHAR2
)
RETURN DOMAttr;
```

- 파라미터

파라미터	설명
elem	해당 Attribute를 넣을 Element이다.
attr	추가할 DOMAttr이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    attr      dbms_xmldom.DOMAttr;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    attr := dbms_xmldom.createAttribute(doc, 'mytag');
    attr := dbms_xmldom.setAttributeNode(ele, attr);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.62. SETDATA

DOMCharacterData를 세팅한다.

SETDATA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SETDATA
(
    cd            IN             DOMCharacterData,
    data          IN             VARCHAR2
);
```

- 파라미터

파라미터	설명
cd	DOMCharacterData이다.
data	세팅할 데이터이다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    txt        dbms_xmldom.DOMText;
    cd         dbms_xmldom.DOMCharacterData;
begin
    doc  := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'node');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makenode(txt));
    dbms_xmldom.setData(cd, 'text');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.63. SETNODEVALUE

노드의 값을 변경하는 프러시저다.

SETNODEVALUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SETNODEVALUE
(
    n         IN              DOMNode,
    value     IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
n	값을 변경할 노드이다.
value	노드에 적용할 값이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    attr      dbms_xmldom.DOMAttr;
    n         dbms_xmldom.DOMNode;
begin
    doc := dbms_xmldom.newDomDocument;
    attr := dbms_xmldom.createAttribute(doc, 'attr');
    n := dbms_xmldom.makeNode(attr);
```

```
    dbms_xmldom.setNodeValue(n, 'value');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.64. SETVALUE

Attribute 노드의 값을 변경하는 프러시저다.

SETVALUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SETVALUE
(
    a         IN              DOMAttr,
    value     IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
a	값을 변경할 Attribute 노드이다.
value	Attribute에 적용할 값이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    attr      dbms_xmldom.DOMAttr;
begin
    doc := dbms_xmldom.newDomDocument;
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    dbms_xmldom.setAttribute(ele, 'attr1', 'new_attribute');
    attr := dbms_xmldom.getAttributeNode(ele, 'attr1');
    dbms_xmldom.setValue(attr, 'new_attr');
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.65. SPLITTEXT

DOMText를 두개로 분리한다.

SPLITTEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SPLITTEXT
(
    t            IN              DOMText,
    offset       IN              NUMBER
)
RETURN DOMText;
```

- 파라미터

파라미터	설명
t	DOMText이다.
offset	분리할 데이터의 오프셋이다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    txt        dbms_xmldom.DOMText;
    st         dbms_xmldom.DOMText;
begin
    doc  := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'textnode');
    st := dbms_xmldom.splitText(txt, 4);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.66. SUBSTRINGDATA

DOMCharacterData에서 특정 범위의 데이터를 출력한다.

SUBSTRINGDATA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.SUBSTRINGDATA
(
```

```
    cd          IN              DOMCharacterData,
    offset      IN              NUMBER,
    cnt         IN              NUMBER
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cd	DOMCharacterData이다.
offset	DOMCharacterData에서 출력할 데이터의 시작 위치이다.
cnt	DOMCharacterData에서 출력할 데이터의 개수이다.

- 예제

```
declare
    doc        dbms_xmldom.DOMDocument;
    txt        dbms_xmldom.DOMText;
    cd         dbms_xmldom.DOMCharacterData;
begin
    doc  := dbms_xmldom.newDomDocument;
    txt := dbms_xmldom.createTextNode(doc, 'textnode');
    cd := dbms_xmldom.makeCharacterData(dbms_xmldom.makenode(txt));
    dbms_output.put_line(dbms_xmldom.substringData(cd, 0, 4));
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.67. WRITETOBUFFER

DOM 문서를 VARCHAR2 형태로 출력한다.

WRITETOBUFFER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.WRITETOBUFFER
(
    n           IN              DOMNode,
    buffer      IN OUT          VARCHAR2
);
```

```
DBMS_XMLDOM.WRITETOBUFFER
(
    doc         IN              DOMDocument,
```

```
    buffer       IN OUT          VARCHAR2
);
```

- 파라미터

파라미터	설명
n	DOMNode이다.
buffer	문서가 출력될 공간이다.
doc	DOMDocument이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    buf       VARCHAR2(100);
begin
    doc := dbms_xmldom.newDomDocument('<root></root>');
    dbms_xmldom.writeToBuffer(doc, buf);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.68. WRITETOCLOB

DOM 문서를 CLOB 형태로 출력한다.

WRITETOCLOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.WRITETOCLOB
(
    doc          IN              DOMDocument,
    cl           OUT             CLOB
);
```

```
DBMS_XMLDOM.WRITETOCLOB
(
    n            IN              DOMNode,
    cl           OUT             CLOB
);
```

- 파라미터

파라미터	설명
doc	DOM 형태의 문서이다.
n	DOM 문서의 노드이다.
cl	결과가 출력될 CLOB이다.

- 예제

```
declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    cl        CLOB;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    n := dbms_xmldom.makeNode(ele);
    n := dbms_xmldom.appendChild(docNode, n);
    dbms_lob.createTemporary(cl, false);
    dbms_xmldom.writeToClob(doc, cl);
    dbms_xmldom.freeDocument(doc);
end;
/
```

52.3.69. WRITETOFILE

DOM 문서를 파일로 출력한다.

WRITETOFILE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLDOM.WRITETOFILE
(
    doc          IN              DOMDocument,
    filename     IN              VARCHAR2
);
```

```
DBMS_XMLDOM.WRITETOFILE
(
    n            IN              DOMNode,
    filename     IN              VARCHAR2
);
```

```
DBMS_XMLDOM.WRITETOFILE
(
    doc          IN              DOMDocument,
    filename     IN              VARCHAR2,
    charset      IN              VARCHAR2
);
```

```
DBMS_XMLDOM.WRITETOFILE
(
    n            IN              DOMNode,
    filename     IN              VARCHAR2,
    charset      IN              VARCHAR2
);
```

- 파라미터

파라미터	설명
doc	DOM 형태의 문서이다.
n	DOM 문서의 노드이다.
filename	결과를 출력할 파일 경로이다.
charset	결과 문자열의 charset이다.

- 예제

```
create or replace directory xmlfolder as '/tmp';

declare
    doc       dbms_xmldom.DOMDocument;
    ele       dbms_xmldom.DOMElement;
    docNode   dbms_xmldom.DOMNode;
    n         dbms_xmldom.DOMNode;
    cl        CLOB;
begin
    doc := dbms_xmldom.newDomDocument;
    docNode := dbms_xmldom.makeNode(doc);
    ele := dbms_xmldom.createElement(doc, 'rootTag');
    n := dbms_xmldom.makeNode(ele);
    n := dbms_xmldom.appendChild(docNode, n);
    dbms_lob.createTemporary(cl, false);
    dbms_xmldom.writeToFile(doc, 'XMLFOLDER/test.xml');
    dbms_xmldom.freeDocument(doc);
end;
/
```

제53장 DBMS_XMLGEN

본 장에서는 DBMS_XMLGEN 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다. 예제는 XMLTYPE이 CLOB일 때를 기준으로 작성되었으며 OBJECT인 경우엔 사용에 제약이 있을 수 있다.

53.1. 개요

DBMS_XMLGEN은 쿼리를 입력으로 받고 해당 쿼리의 결과 집합을 XML 문서로 생성해주는 패키지이다.

다음은 DBMS_XMLGEN 패키지 내에 정의된 상수이다.

- Conversion Flag
  - 프로토타입

```
ENTITY_ENCODE             CONSTANT NUMBER := 0;
ENTITY_DECODE             CONSTANT NUMBER := 1;
```

  - 인자

    DBMS_XMLGEN.CONVERT 함수의 인자로 사용하는 flag에 대한 설명이다.

인자	설명
ENTITY_ENCODE	데이터를 인코딩하여 결과를 출력한다.
ENTITY_DECODE	데이터를 디코딩하여 결과를 출력한다.

- Null Handling Flag
  - 프로토타입

```
DROP_NULLS            CONSTANT NUMBER := 0;
NULL_ATTR             CONSTANT NUMBER := 1;
EMPTY_TAG             CONSTANT NUMBER := 2;
```

  - 인자

    DBMS_XMLGEN.SETNULLHANDLING 프러시저의 인자로 사용하는 flag에 대한 설명이다.

인자	설명
DROP_NULLS	NULL 값을 출력하지 않는다.
NULL_ATTR	NULL ATTRIBUTE로 출력한다.

인자	설명
EMPTY_TAG	EMPTY ELEMENT로 출력한다.

53.2. 타입

본 절에서는 DBMS_XMLGEN 패키지에서 제공하는 별도 정의된 타입을 설명한다.

53.2.1. ctxHandle

ctxHandle은 NUMBER의 서브타입으로 특정 컨텍스트를 의미하는 값이다.

ctxHandle 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
SUBTYPE ctxHandle IS NUMBER;
```

53.3. 프러시저와 함수

본 절에서는 DBMS_XMLGEN 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

53.3.1. CLOSECONTEXT

컨텍스트 핸들로부터 컨텍스트를 종료한다.

CLOSECONTEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.CLOSECONTEXT
(
    ctx IN ctxHandle
);
```

- 파라미터

파라미터	설명
ctx	CLOSECONTEXT에 의해 닫힐 컨텍스트 핸들이다.

- 예제

```
DECLARE
    ctx dbms_xmlgen.ctxHandle;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select 1 from dual');
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.2. CONVERT

XML 데이터를 escaped 또는 unescaped XML 데이터로 변환하는 함수이다.

CLOB 타입을 입력으로 받으면 CLOB 타입을, VARCHAR 타입을 입력으로 받으면 VARCHR 타입을 반환한다.

CONVERT 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - CLOB인 경우

```
DBMS_XMLGEN.CONVERT
(
    xmlData IN CLOB,
    flag    IN NUMBER := ENTITY_ENCODE
 )
RETURN CLOB;
```

  - VARCHR인 경우

```
DBMS_XMLGEN.CONVERT
(
    xmlData IN VARCHAR2,
    flag    IN NUMBER := ENTITY_ENCODE
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
xmlData	CLOB이나 VARCHAR 타입의 인코딩이나 디코딩할 데이터이다.
flag	인코딩인지 디코딩인지 여부를 결정하는 flag 값이다. – ENTITY_ENCODE : 인코딩을 하는 경우 설정한다. (기본값) – ENTITY_DECODE : 디코딩을 하는 경우 설정한다.

- 예제

```
DECLARE
    escape_str VARCHAR2(100);
BEGIN
    escape_str := DBMS_XMLGEN.CONVERT('Hi escaped_str! < & >',
                                      DBMS_XMLGEN.ENTITY_ENCODE);
    DBMS_OUTPUT.PUT_LINE(escape_str);
END;
/
```

53.3.3. GETNUMROWSPROCESSED

GETXML 또는 GETXMLTYPE에 의해 처리된 로우의 수를 출력한다.

GETNUMROWSPROCESSED 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.GETNUMROWSPROCESSED
(
    ctx          IN   ctxHandle
)
RETURN NUMBER;
```

- 파라미터

파라미터	설명
ctx	NEWCONTEXT 함수 호출에 의해 생성되는 컨텍스트 핸들이다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(DBMS_XMLGEN.GETNUMROWSPROCESSED(ctx));
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.4. GETXML

주어진 ctxHandle로부터 CLOB 형태의 XML 문서를 생성한다.

GETXML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.GETXML
(
    ctx          IN   ctxHandle,
    dtdOrSchema  IN   NUMBER
)
RETURN CLOB;
```

- 파라미터

파라미터	설명
ctx	NEWCONTEXT 함수 호출에 의해 생성되는 컨텍스트 핸들이다.
dtdOrSchema	DTD 또는 Schema의 출력 여부를 결정하는 값이다. NONE 설정만 지원 가능하다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.5. GETXMLTYPE

주어진 ctxHandle로부터 XMLType 형태의 XML 문서를 생성한다.

GETXMLTYPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.GETXMLTYPE
(
    ctx          IN   ctxHandle,
    dtdOrSchema  IN   NUMBER
)
RETURN XMLType;
```

- 파라미터

파라미터	설명
ctx	NEWCONTEXT 함수 호출에 의해 생성되는 컨텍스트 핸들이다.
dtdOrSchema	DTD 또는 Schema의 출력 여부를 결정하는 값이다. NONE 설정만 지원 가능하다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result XMLType;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    result := DBMS_XMLGEN.GETXMLTYPE(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.6. NEWCONTEXT

쿼리문으로부터 컨텍스트 핸들을 생성한다. 컨텍스트 핸들은 GETXML 함수에서 XML 문서를 얻기 위해 사용된다.

NEWCONTEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.NEWCONTEXT
(
    query IN VARCHAR2
)
RETURN ctxHandle;
```

```
DBMS_XMLGEN.NEWCONTEXT
(
    query IN SYS_REFCURSOR
)
RETURN ctxHandle;
```

- 파라미터

파라미터	설명
query	XML 문서로 변경될 데이터를 출력하는 쿼리문 또는 커서다.

- 예제

```
DECLARE
    ctx dbms_xmlgen.ctxHandle;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select 1 from dual');
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.7. SETBINDVALUE

바인드 파라미터의 값을 설정한다.

SETBINDVALUE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.SETBINDVALUE
(
    ctxHdl      IN    ctxHandle,
    bindName    IN    VARCHAR2,
    bindValue   IN    VARCHAR2
);
```

- 파라미터

파라미터	설명
ctxHdl	실행한 쿼리의 컨텍스트 핸들이다.
bindName	바인드 파라미터의 이름이다.
bindValue	바인드 파라미터의 값이다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T where emp_name = :NAME');
    DBMS_XMLGEN.SETBINDVALUE(ctx, 'NAME', 'TOM');
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.8. SETMAXROWS

GETXML 또는 GETXMLType에서 처리할 최대 로우의 수를 설정한다.

SETMAXROWS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.SETMAXROWS
(
    ctx      IN    ctxHandle,
    maxRows  IN    NUMBER
);
```

- 파라미터

파라미터	설명
ctx	실행한 쿼리의 컨텍스트 핸들이다.
maxRows	처리할 최대 로우의 수이다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
```

```
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    DBMS_XMLGEN.SETMAXROWS(ctx, 1);
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.9. SETNULLHANDLING

NULL 데이터 처리 방식을 설정한다.

SETNULLHANDLING 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.SETNULLHANDLING
(
    ctx   IN    ctxHandle,
    flag  IN    NUMBER
);
```

- 파라미터

파라미터	설명
ctx	NULL 데이터 처리 방식을 설정할 컨텍스트 핸들이다.
flag	NULL 데이터 처리 방식을 결정하는 flag 값이다. – DROP_NULLS : NULL 데이터를 표시하지 않는다. (기본값) – NULL_ATTR : NULL ATTRIBUTE로 표현한다. – EMPTY_TAG : EMPTY ELEMENT로 표현한다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, NULL);
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx    dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
```

```
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    DBMS_XMLGEN.SETNULLHANDLING(ctx, DBMS_XMLGEN.EMPTY_TAG);
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.10. SETROWTAG

GETXML 또는 GETXMLType의 XML Output의 Row Tag 이름을 설정한다.

SETROWTAG 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.SETROWTAG
(
    ctx         IN    ctxHandle,
    rowTagName  IN    VARCHAR2
);
```

- 파라미터

파라미터	설명
ctx	NEWCONTEXT 함수 호출에 의해 실행된 컨텍스트 핸들이다.
rowTagName	설정할 Row Tag 이름이다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    DBMS_XMLGEN.SETROWTAG(ctx, 'MYROWTAG');
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.11. SETROWSETTAG

GETXML 또는 GETXMLType의 XML Output의 Row Set Tag 이름을 설정한다.

SETROWSETTAG 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.SETROWSETTAG
(
    ctx            IN    ctxHandle,
    rowSetTagName  IN    VARCHAR2
);
```

- 파라미터

파라미터	설명
ctx	NEWCONTEXT 함수 호출에 의해 생선된 컨텍스트 핸들이다.
rowSetTagName	설정할 Row Set Tag 이름이다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    DBMS_XMLGEN.SETROWSETTAG(ctx, 'MYROWSETTAG');
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

53.3.12. SETSKIPROWS

GETXML 또는 GETXMLType에서 처리하지 않고 무시할 로우의 수를 설정한다.

SETSKIPROWS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XMLGEN.SETSKIPROWS
(
    ctx       IN    ctxHandle,
    skipRows  IN    NUMBER
);
```

- 파라미터

파라미터	설명
ctx	실행한 쿼리의 컨텍스트 핸들이다.
skipRows	무시할 로우의 수이다.

- 예제

```
CREATE TABLE EMP_T (EMP_ID NUMBER, EMP_NAME VARCHAR2(16));
INSERT INTO EMP_T VALUES(2015001, 'TOM');
INSERT INTO EMP_T VALUES(2015002, 'DAVID');
INSERT INTO EMP_T VALUES(2015003, 'GEORGE');
DECLARE
    ctx dbms_xmlgen.ctxHandle;
    result CLOB;
BEGIN
    ctx := DBMS_XMLGEN.NEWCONTEXT('select * from EMP_T');
    DBMS_XMLGEN.SETSKIPROWS(ctx, 1);
    result := DBMS_XMLGEN.GETXML(ctx);
    DBMS_OUTPUT.PUT_LINE(result);
    DBMS_XMLGEN.CLOSECONTEXT(ctx);
END;
/
```

제54장 DBMS_XMLPARSER

본 장에서는 DBMS_XMLPARSER 패키지의 기본 개념과 패키지 내의 프러시저와 함수들의 의미를 설명한다.

54.1. 개요

DBMS_XMLPARSER는 XML 문서를 파싱하고 XML 문서의 구조 및 내용에 접근할 수 있는 API를 제공하는 패키지이다.

54.2. 프러시저와 함수

본 절에서는 DBMS_XMLPARSER 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

54.2.1. FREEPARSER

Parser 개체를 free 한다.

FREEPARSER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FREEPARSER
(
    p Parser
);
```

- 파라미터

파라미터	IN / OUT	설명
p	(IN)	Parser 인스턴스이다.

54.2.2. GETDOCUMENT

Parser에 의해 만들어진 DOM tree document의 document node를 리턴한다. 이 함수는 문서가 파싱된 후에 불려야 한다.

GETDOCUMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION GETDOCUMENT
(
  p Parser
)
RETURN DOMDocument;
```

- 파라미터

파라미터	IN / OUT	설명
p	(IN)	Parser 인스턴스이다.

54.2.3. GETVALIDATIONMODE

설정된 Validation mode를 가져온다. 리턴값이 TRUE인 경우 validation mode임을 의미한다.

GETVALIDATIONMODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION GETVALIDATIONMODE
(
  p Parser
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	IN / OUT	설명
p	(IN)	Parser 인스턴스이다.

54.2.4. NEWPARSER

새로운 parser 인스턴스를 리턴한다.

NEWPARSER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION newParser
RETURN Parser;
```

54.2.5. PARSE

주어진 URL 또는 파일에 저장된 XML을 파싱한다. 파싱이 실패하면 application error가 발생한다. 함수와 프러시저 형태 둘 다 가지고 있다.

PARSE 함수와 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - 함수

```
FUNCTION parse
(
url VARCHAR2
)
RETURN DOMDocument;
```

  - 프러시저

```
PROCEDURE parse
(
  p Parser,
  url VARCHAR2
);
```

- 파라미터

파라미터	IN / OUT	설명
url	(IN)	파싱해야 할 URL 또는 파일의 전체 경로이다.
p	(IN)	Parser 인스턴스이다.

54.2.6. PARSEBUFFER

주어진 버퍼에 저장되어 있는 XML을 파싱한다.

PARSEBUFFER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE PARSEBUFFER
(
  p   Parser,
  doc VARCHAR2
);
```

- 파라미터

파라미터	IN / OUT	설명
p	(IN)	Parser 인스턴스이다.
doc	(IN)	파싱해야 할 XML 문서 버퍼이다.

54.2.7. PARSECLOB

CLOB에 저장되어 있는 XML을 파싱한다.

PARSECLOB 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE PARSECLOB
(
  p   Parser,
  doc CLOB
);
```

- 파라미터

파라미터	IN / OUT	설명
p	(IN)	Parser 인스턴스이다.
doc	(IN)	파싱해야 할 XML 문서 버퍼

54.2.8. SETVALIDATIONMODE

Validation mode를 설정한다.

SETVALIDATIONMODE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE setValidationMode
(
```

```
  p   Parser,
  yes BOOLEAN
);
```

- 파라미터

파라미터	IN / OUT	설명
p	(IN)	Parser 인스턴스이다.
yes	(IN)	설정할 mode로, TRUE면 validation한다.

제55장 DBMS_XMLSAVE

본 장에서는 DBMS_XMLSAVE 패키지의 기본 개념과 패키지 내의 상수 및 프러시저와 함수들의 의미를 설명한다.

55.1. 개요

DBMS_XMLSAVE는 XML-to-database-type 기능을 제공한다.

55.2. 상수

DBMS_XMLSAVE에서는 패키지 내의 함수나 프러시저의 파라미터를 설정하기 위한 상수들을 포함하고 있다.

지원하는 상수들의 목록은 다음 표에 있다.

상수	설명
DEFAULT_ROWTAG	데이터베이스 레코드들에 대응하는 XML 요소에 대한 태그 이름의 기본값을 지칭하는 상수로, 상수값은 'ROW'이다.
DEFAULT_DATE_FORMAT	기본 날짜 형식은 'MM/dd/yyyy HH:mm:ss'이다.
MATCH_CASE	XML 요소들을 데이터베이스 개체에 대응시킬 때 XML SQL Utility는 case-sensitive 해야 함을 명시한다.
IGNORE_CASE	XML 요소들을 데이터베이스 개체에 대응시킬 때 XML SQL Utility는 case-insensitive 해야 함을 명시한다.

55.3. 타입

DBMS_XMLSAVE의 함수와 프러시저 에서는 ctxType 타입을 사용한다.

타입	설명
ctxType	처리할 쿼리 context 타입으로, “55.4.6. NEWCONTEXT”의 리턴값이다.

55.4. 프러시저와 함수

본 절에서는 DBMS_XMLSAVE 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

55.4.1. CLEARKEYCOLUMNLIST

Key column 리스트를 지운다.

CLEARKEYCOLUMNLIST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CLEARKEYCOLUMNLIST
(
   ctxHdl IN ctxType
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 Context이다.

55.4.2. CLEARUPDATECOLUMNLIST

Update column 리스트를 지운다.

CLEARUPDATECOLUMNLIST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CLEARUPDATECOLUMNLIST
(
   ctxHdl IN ctxType
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 Context이다.

55.4.3. CLOSECONTEXT

특정 save context를 닫는다.

CLOSECONTEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE CLOSECONTEXT
(
   ctxHdl IN ctxType
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 Context이다.

55.4.4. DELETEXML

XML 문서의 데이터에 명시된 레코드들을 context를 생성할 때 명시된 테이블에서 지운다. 리턴값은 지워진 행의 개수이다.

DELETEXML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION DELETEXML
(
  ctxHdl IN ctxPType,
    xDoc IN VARCHAR2
)
RETURN NUMBER;
```

```
FUNCTION DELETEXML
(
  ctxHdl IN ctxType,
  xDoc IN CLOB
)
RETURN NUMBER;
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 Context이다.
xDoc	(IN)	XML 문서를 포함하고 있는 문자열이다.

55.4.5. INSERTXML

Context 형성 시 명시된 테이블에 XML 문서의 데이터를 삽입한다. 리턴값은 삽입된 행의 개수이다.

INSERTXML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION INSERTXML
(
  ctxHdl IN ctxType,
  xDoc IN VARCHAR2
)
RETURN NUMBER;
```

```
FUNCTION INSERTXML
(
  ctxHdl IN ctxType,
  xDoc IN CLOB
)
RETURN NUMBER;
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
xDoc	(IN)	XML 문서를 포함하고 있는 문자열이다.

55.4.6. NEWCONTEXT

새로운 save context를 만들고 처리할 context를 리턴한다.

NEWCONTEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION NEWCONTEXT
(
   targetTable IN VARCHAR2
)
RETURN ctxType;
```

- 파라미터

파라미터	IN / OUT	설명
targetTable	(IN)	XML 문서를 올릴 표적 테이블이다.

55.4.7. SETDATEFORMAT

XML 문서로부터 만들어진 date 타입의 형식을 정한다.

SETDATEFORMAT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SETDATEFORMAT
(
   ctxHdl IN ctxType,
   mask IN VARCHAR2
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
mask	(IN)	Date 타입의 형식에 대한 syntax이다.

55.4.8. SETIGNORECASE

Database의 column 또는 attribute에 XML 구성 요소를 대응시킬 때 대소문자를 무시할 지 말지 XXML SQL Utility에게 말해준다. 이러한 대응은 XML 태그 이름을 기반으로 한다.

SETIGNORECASE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SETIGNORECASE
(
   ctxHdl IN ctxType,
   flag IN NUMBER
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
flag	(IN)	XML 문서의 대소문자를 무시할 지 말지 결정한다. 1이면 무시하고 0이면 무시하지 않는다.

55.4.9. SETKEYCOLUMN

Column을 key column 리스트에 추가한다.

SETKEYCOLUMN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SETKEYCOLUMN
(
   ctxHdl IN ctxType,
   colName IN VARCHAR2
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
colName	(IN)	Key column 리스트에 추가할 column의 이름으로 NULL이면 안된다.

55.4.10. SETROWTAG

데이터베이스의 레코드에 대응되는 XML 구성 요소에 대해 XML 문서 상에서 사용되는 태그 이름을 지어준다.

SETROWTAG 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SETROWTAG
(
   ctxHdl IN ctxType,
   tag IN VARCHAR2
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
tag	(IN)	태그 이름이다.

55.4.11. SETUPDATECOLUMN

Column을 update column 리스트에 추가한다.

SETUPDATECOLUMN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SETUPDATECOLUMN
(
   ctxHdl IN ctxType,
   colName IN VARCHAR2
);
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
colName	(IN)	Update column 리스트에 추가할 column의 이름으로 NULL이면 안된다.

55.4.12. UPDATEXML

Context 형성 시 명시된 테이블에 XML 문서 데이터를 업데이트한다. 리턴값은 업데이트 된 행의 개수이다.

UPDATEXML 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION UPDATEXML
(
  ctxHdl IN ctxType,
  xDoc IN VARCHAR2
)
RETURN NUMBER;
```

```
FUNCTION UPDATEXML
(
  ctxHdl IN ctxType,
  xDoc IN CLOB
)
RETURN NUMBER;
```

- 파라미터

파라미터	IN / OUT	설명
ctxHdl	(IN)	처리할 context이다.
xDoc	(IN)	XML 문서를 포함하고 있는 문자열이다.

제56장 DBMS_XPLAN

본 장에서는 DBMS_XPLAN 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

56.1. 개요

DBMS_XPLAN은 플랜 정보와 플랜의 수행 정보들에 대해 사용자가 원하는 항목을 선택해서 다양한 포맷으로 출력할 수 있는 기능을 제공한다.

출력 방식은 pipelined function 기능을 이용하여 TABLE() 구문을 통해 쿼리의 결과로 해당 정보들을 출력한다. 또한 출력 정보는 VSQL_PLAN, VSQL_PLAN_STATISTICS 뷰를 통해서 구하기 때문에 GATHER_SQL_PLAN_STAT 파라미터가 켜져 있어야 수행 정보를 구할 수 있다.

56.2. 함수

본 절에서는 DBMS_XPLAN 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

56.2.1. DISPLAY_CURSOR

Physical Plan Cache에 등록되어 있는 플랜에 대해 SQL_ID 값을 통해 조회하는 함수이다. pipelined function이기 때문에 TABLE() 함수를 이용해서 사용한다.

DISPLAY_CURSOR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XPLAN.DISPLAY_CURSOR
(
    in_sql_id       VARCHAR2 default NULL,
    in_child_no     NUMBER default NULL,
    format          VARCHAR2 default 'BASIC LAST SQL'
)
RETURN dbms_xplan_type_table pipelined;
```

- 파라미터

파라미터	설명
in_sql_id	조회하려는 플랜의 SQL_ID 값이다. 입력을 생략한 경우에는 해당 세션의 마지막 수행 쿼리의 SQL_ID 값을 사용하게 된다.
in_child_no	조회하려는 플랜의 CHILD_NUMBER 값이다. 이 값이 생략된 경우 입력된 SQL_ID 값과 일치하는 모든 플랜을 조회한다. CHILD_NUMBER 값은 SQL_ID가 입력된 경우에만 지정될 수 있다.
format	출력하고자 하는 항목을 명시할 수 있다. 항목의 종류에는 개별 항목과 개별 항목 여러 개를 설정하는 그룹 항목이 있다. 항목 이름 앞에 하이픈(-)를 붙이면 해당 항목을 제외할 수 있다.

- 개별 출력 항목

항목	설명
CARDS	optimizer에서 예측한 해당 플랜 노드의 row 수이다.
COST	optimizer에서 예측한 해당 플랜 노드의 cost이다.
PARTITION	파티션 관련 정보이다.
PARALLEL	parallel execution 관련 정보이다.
PREDICATE	플랜 노드별 predicate 정보이다.
REMOTE	플랜 노드별 데이터베이스 링크에 수행한 쿼리 내용이다.
ROWS	해당 플랜 노드에서 실제 수행된 row 수이다.
ELAPTIME	해당 플랜 노드에서 실제 수행된 시간이다.
USEDMEM	해당 플랜 노드에서 실제 사용된 메모리 양이다.
TEMPREAD	해당 플랜 노드에서 실제 사용된 temp read 횟수이다.
TEMPWRITE	해당 플랜 노드에서 실제 사용된 temp write 횟수이다.
BUFGETS	해당 플랜 노드에서 실제 요청한 buffer get 횟수이다.
STARTS	해당 플랜 노드가 실제 재시작된 횟수이다.
LAST	수행 정보 값을 마지막 수행 정보 값만 출력한다. 지정하지 않은 경우 수행 정보 값은 모든 수행에 대한 누적 값을 보여준다.
PRECISE	CARDS, ROWS에 대해 반올림 없이 실제 값을 보여준다.
HEADER	플랜의 기본 정보(sql id, hash value, 총 수행 횟수, 총 패치 횟수, 플랜 수행 시간)를 보여준다.
SQL	플랜 생성에 사용된 쿼리문을 보여준다.
READS	해당 플랜 노드에서 실제 수행된 디스크 I/O 횟수이다.

- 그룹 출력 항목

항목	설명
IOSTATS	수행 정보 중 IO와 관련된 모든 항목을 보여준다. `TEMPREAD + TEMPWRITE + BUFGETS + READS`
MEMSTATS	수행 정보 중 메모리와 관련된 모든 항목을 보여준다. `USEDMEM`
ALLSTATS	수행 정보 중 IO, 메모리와 관련된 모든 항목을 보여준다. `IOSTATS + MEMSTATS`
BASIC	출력 기본 포맷으로 optimizer에서 예측한 cardinality, cost와 마지막 수행에 대한 노드별 수행 시간을 출력한다. `CARDS + COST + PART + ELAPTIME + LAST`
TYPICAL	BASIC 포맷에 추가로 마지막 수행에 대한 노드별 처리 row 수와 predicate 정보, remote sql 정보를 출력한다. `BASIC + PE + ROWS + STARTS + PRED + REMOTE + PRECISE`
ALL	TYPICAL 포맷에 ALLSTATS 항목을 보여준다. `TYPICAL + ALLSTATS`

- 예제

다음은 현재 세션에서 마지막으로 수행한 쿼리의 플랜 정보를 보는 예이다.

```
SQL> select * from table(dbms_xplan.display_cursor);
```

다음은 기본 포맷에서 파티션 정보를 제외하고 '9xg1f36cgn19q' SQL_ID의 플랜 정보를 보는 예이다.

```
SQL> set pagesize 0
SQL> set linesize 100
SQL> select * from table (
   2 dbms_xplan.display_cursor('9xg1f36cgn19q', 53, 'BASIC LAST SQL -PARTITION'));
SQL ID          : 9xg1f36cgn19q
HASH VALUE      : 2566522166
PLAN HASH VALUE: 3469845920
EXECUTIONS      : 1
FETCHES         : 43
LOADED AT       : 2019/04/03 14:09:01
TOT ELAPSED TIME: 00:00:00.5774
AVG ELAPSED TIME: 00:00:00.5774
```

```
SQL            : select p_name, p_mfgr, ps_supplycost, p_type,
       min (ps_supplycost) over (partition by ps_partkey) min_supplycost,
       p_comment, ps_comment
from part, psup
where p_partkey = ps_partkey



-------------------------------------------------------------------------------
| ID  | Operation                | Name | Cost (%CPU)  | Cards    | Elaps. Time |
-------------------------------------------------------------------------------
|   1 | COLUMN PROJECTION        |      |  261   (8.81)|   95475  |00:00:00.0299|
|   2 |  WINDOW (SIMPLE)         |      |  260   (8.46)|   95475  |00:00:00.0907|
|   3 |   GROUP BY               |      |  244   (2.46)|   95475  |00:00:00.0063|
|   4 |    ORDER BY (SORT)       |      |  242   (1.65)|   95475  |00:00:00.0824|
|   5 |     HASH JOIN            |      |  240    (.83)|   95475  |00:00:00.0874|
|   6 |      TABLE ACCESS (FULL) | PSUP |  105      (0)|   87372  |00:00:00.0072|
|   7 |      TABLE ACCESS (FULL) | PART |  132      (0)|   89536  |00:00:00.0097|
-------------------------------------------------------------------------------


Note
-------------------------------------------------------------------------------

    6 - dynamic sampling used for this table (96 blocks)
    7 - dynamic sampling used for this table (96 blocks)
```

다음은 마지막 수행 시간, 수행 결과 row, 요청 buffer gets 값을 보는 예이다.

```
SQL> select * from table (
   2 dbms_xplan.display_cursor('9xg1f36cgn19q', 53, 'ROWS ELAPTIME +BUFGETS'));
-------------------------------------------------------------------------------
| ID  | Operation                | Name | Rows     | Elaps. Time  | CR Gets  |
-------------------------------------------------------------------------------
|   1 | COLUMN PROJECTION        |      |   99999  |00:00:00.0299 |       0  |
|   2 |  WINDOW (SIMPLE)         |      |   99999  |00:00:00.0907 |       0  |
|   3 |   GROUP BY               |      |   99999  |00:00:00.0063 |       0  |
|   4 |    ORDER BY (SORT)       |      |   99999  |00:00:00.0824 |       0  |
|   5 |     HASH JOIN            |      |   99999  |00:00:00.0874 |       0  |
|   6 |      TABLE ACCESS (FULL) | PSUP |   99999  |00:00:00.0072 |     245  |
|   7 |      TABLE ACCESS (FULL) | PART |   99999  |00:00:00.0097 |     307  |
-------------------------------------------------------------------------------

Note
-------------------------------------------------------------------------------
    6 - dynamic sampling used for this table (96 blocks)
    7 - dynamic sampling used for this table (96 blocks)
```

다음은 수행 중 메모리 사용량과 디스크 I/O 값을 보는 예이다.

```
SQL> select * from table (dbms_xplan.display_cursor(
   2 '9xg1f36cgn19q', 53, 'ALLSTATS -TEMPREAD -TEMPWRITE'));
-------------------------------------------------------------------------------
| ID  | Operation                | Name | CR Gets  | Used Mem |  Reads   |
-------------------------------------------------------------------------------
|   1 | COLUMN PROJECTION        |      |       0  |       0K |       0  |
|   2 |  WINDOW (SIMPLE)         |      |       0  |     198K |       0  |
|   3 |   GROUP BY               |      |       0  |       0K |       0  |
|   4 |    ORDER BY (SORT)       |      |       0  |    7698K |       0  |
|   5 |     HASH JOIN            |      |       0  |    5795K |       0  |
|   6 |      TABLE ACCESS (FULL) | PSUP |     245  |       0K |     244  |
|   7 |      TABLE ACCESS (FULL) | PART |     307  |       0K |     306  |
-------------------------------------------------------------------------------

Note
-------------------------------------------------------------------------------
    6 - dynamic sampling used for this table (96 blocks)
    7 - dynamic sampling used for this table (96 blocks)
```

56.2.2. DISPLAY_TPR

TPR에 등록되어 있는 플랜에 대해 SQL_HASH_VALUE 값을 통해 조회하는 함수이다. pipelined function 이기 때문에 TABLE() 함수를 이용해서 사용한다.

DISPLAY_TPR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XPLAN.DISPLAY_TPR
(
    sql_hash_value      NUMBER,
    plan_hash_value     NUMBER default NULL,
    instance#           NUMBER default NULL,
    format              VARCHAR2 default 'TYPICAL'
)
RETURN dbms_xplan_type_table pipelined;
```

- 파라미터

파라미터	설명
sql_hash_value	조회하려는 플랜의 SQL_HASH_VALUE 값이다.

파라미터	설명
plan_hash_value	조회하려는 플랜의 PLAN_HASH_VALUE 값이다. 이 값이 생략된 경우 입력된 SQL_HASH_VALUE 값과 일치하는 모든 플랜을 조회한다.
instance#	조회하려는 플랜이 생성된 인스턴스 번호 이다. 이 값이 생략된 경우 V$DATABASE 에 나타난 로컬 instance_id를 사용한다.
format	아래 3가지 포맷을 지원한다. – BASIC : OPERATION ID, NAME 만 표시하며 추가적으로 옵션 사용 가능하다. – TYPICAL : OPERATION ID, NAME, CARDS, COST, BUFGETS, ROWS, ELAPTIME, PARTITION, PREDICATE을 표시한다. 기본적으로 OPERATION ID, NAME, CARDS, COST를 출력하며 해당 내용이 있을 경우 BUFGETS, ROWS, ELAPTIME, PARTITION, PREDICATE 정보를 출력한다. (기본값) – ALL : TYPICAL, REMOTE을 표시한다.

- 개별 출력 항목

항목	설명
CARDS	optimizer에서 예측한 해당 플랜 노드의 row 수이다.
COST	optimizer에서 예측한 해당 플랜 노드의 cost이다.
BUFGETS	해당 플랜 노드에서 실제 요청한 buffer get 횟수이다.
ROWS	해당 플랜 노드에서 실제 수행된 row 수이다.
ELAPTIME	해당 플랜 노드에서 실제 수행된 시간이다.
PARTITION	파티션 관련 정보이다.
PREDICATE	플랜 노드별 predicate 정보이다.
REMOTE	플랜 노드별 데이터베이스 링크에 수행한 쿼리 내용이다.

- 예제

다음은 기본 포맷에서 '12345678' SQL_HASH_VALUE의 플랜 정보를 보는 예이다.

```
SQL> set pagesize 300
SQL> set linesize 400
SQL> select * from table(dbms_xplan.display_tpr('12345678'));

COLUMN_VALUE
--------------------------------------------------------------------------------
SQL ID          : cp2tt6ptgqws0
PLAN HASH VALUE : 968455237
SQL TEXT        :
select t1.a, t2.a from t1,t2 where t1.a = t2.a union select t2.a, t3.a from t2,t3 where t2.a=t3.a
```

```
-------------------------------------------------------------------------------------------------------
| ID  | Operation               | Name | Cards   | Cost   (%CPU)| CR Gets  | Rows    | Elaps. Time  |
-------------------------------------------------------------------------------------------------------
|   1 | UNION                   |      |    534K |  815   (4.66)|       0  |    300K |00:00:00.0000 |
|   2 |  DISTINCT (SORT)        |      |    267K |  399   (2.76)|       0  |    300K |00:00:00.0002 |
|   3 |   HASH JOIN             |      |    267K |  393   (1.27)|       0  |    300K |00:00:00.0002 |
|   4 |    TABLE ACCESS (FULL)  | T1   |    248K |  194      (0)|     549  |    300K |00:00:00.0000 |
|   5 |    TABLE ACCESS (FULL)  | T2   |    248K |  194      (0)|     549  |    300K |00:00:00.0000 |
|   6 |  DISTINCT (SORT)        |      |    267K |  399   (2.76)|       0  |    300K |00:00:00.0002 |
|   7 |   HASH JOIN             |      |    267K |  393   (1.27)|       0  |    300K |00:00:00.0002 |
|   8 |    TABLE ACCESS (FULL)  | T2   |    248K |  194      (0)|     909  |    300K |00:00:00.0000 |
|   9 |    TABLE ACCESS (FULL)  | T3   |    248K |  194      (0)|     549  |    300K |00:00:00.0000 |
-------------------------------------------------------------------------------------------------------

Predicate Information
-------------------------------------------------------------------------------------------------------
    3 - access: ("T1"."A" = "T2"."A") (0.000)
    7 - access: ("T2"."A" = "T3"."A") (0.000)

Note
-------------------------------------------------------------------------------------------------------
    4 - dynamic sampling used for this table (98 blocks)
    5 - dynamic sampling used for this table (98 blocks)
    9 - dynamic sampling used for this table (98 blocks)


SQL ID          : cp2tt6ptgqws0
PLAN HASH VALUE : 2624411436
SQL TEXT        :
select t1.a, t2.a from t1,t2 where t1.a = t2.a union select t2.a, t3.a from t2,t3 where t2.a=t3.a

-------------------------------------------------------------------------------------------------------
| ID  | Operation               | Name | Cards   | Cost   (%CPU)| CR Gets  | Rows    | Elaps. Time  |
-------------------------------------------------------------------------------------------------------
|   1 | UNION                   |      |    534K |  815   (4.66)|          |         |              |
|   2 |  DISTINCT (SORT)        |      |    267K |  399   (2.76)|          |         |              |
|   3 |   HASH JOIN             |      |    267K |  393   (1.27)|          |         |              |
|   4 |    TABLE ACCESS (FULL)  | T1   |    248K |  194      (0)|          |         |              |
|   5 |    TABLE ACCESS (FULL)  | T2   |    248K |  194      (0)|          |         |              |
|   6 |  DISTINCT (SORT)        |      |    267K |  399   (2.76)|          |         |              |
|   7 |   HASH JOIN             |      |    267K |  393   (1.27)|          |         |              |
|   8 |    TABLE ACCESS (FULL)  | T2   |    248K |  194      (0)|          |         |              |
|   9 |    TABLE ACCESS (FULL)  | T3   |    248K |  194      (0)|          |         |              |
-------------------------------------------------------------------------------------------------------

Predicate Information
-------------------------------------------------------------------------------------------------------
    3 - access: ("T1"."A" = "T2"."A") (0.000)
    7 - access: ("T2"."A" = "T3"."A") (0.000)

Note
-------------------------------------------------------------------------------------------------------
    4 - dynamic sampling used for this table (98 blocks)
    5 - dynamic sampling used for this table (98 blocks)
    9 - dynamic sampling used for this table (98 blocks)
```

제57장 DBMS_XSLPROCESSOR

본 장에서는 DBMS_XSLPROCESSOR 패키지의 기본 개념과 패키지 내의 프러시저와 함수들의 의미를 설명한다.

57.1. 개요

DBMS_XSLPROCESSOR는 XML 문서의 구성과 내용을 관리할 수 있는 인터페이스를 제공하는 패키지이다.

57.2. 프러시저와 함수

본 절에서는 DBMS_XSLPROCESSOR 패키지에서 제공하는 함수와 프러시저를 알파벳 순으로 설명한다.

57.2.1. SELECTNODES

DOM tree로부터 주어진 패턴과 일치하는 노드들을 찾아 리턴해준다.

SELECTNODES 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XSLPROCESSOR.SELECTNODES
(
   n           IN   DBMS_XMLDOM.DOMNODE,
   pattern     IN   VARCHAR2,
   namespace   IN VARCHAR2 := NULL
)
RETURN DBMS_XMLDOM.DOMNODELIST;
```

- 파라미터

파라미터	설명
n	DOM tree의 뿌리 노드이다.
pattern	사용할 패턴이다.
namespace	선언된 namespace이다.

57.2.2. SELECTSINGLENODE

DOM tree로부터 주어진 패턴과 일치하는는 첫 번째 노드를 찾아 리턴해준다.

SELECTSINGLENODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
DBMS_XSLPROCESSOR.SELECTSINGLENODE
(
   n           IN   DBMS_XMLDOM.DOMNODE,
   pattern     IN   VARCHAR2,
   namespace   IN VARCHAR2 := NULL
)
RETURN DBMS_XMLDOM.DOMNODE;
```

- 파라미터

파라미터	설명
n	DOM tree의 뿌리 노드이다.
pattern	사용할 패턴이다.
namespace	선언된 namespace이다.

57.2.3. VALUEOF

주어진 패턴과 일치하는 첫 번째 노드의 값을 리턴한다. 함수와 프러시저 형태 둘 다 가지고 있다.

VALUEOF 함수와 프러시저의 세부 내용은 다음과 같다.

- 프로토타입
  - 함수

```
DBMS_XSLPROCESSOR.VALUEOF
(
  n           IN    DBMS_XMLDOM.DOMNODE,
  pattern     IN    VARCHAR2,
  namespace   IN    VARCHAR2 := NULL
)
RETURN VARCHAR2;
```

  - 프러시저

```
DBMS_XSLPROCESSOR.VALUEOF
(
  n           IN    DBMS_XMLDOM.DOMNODE,
  pattern     IN    VARCHAR2,
  val         OUT   VARCHAR2,
  namespace   IN    VARCHAR2 := NULL
);
```

- 파라미터

파라미터	설명
n	DOM tree의 뿌리 노드이다.
pattern	사용할 패턴이다.
val	노드로부터 추출한 값이다.
namespace	선언된 namespace이다.

제58장 HTF

본 장에서는 HTF 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

58.1. 개요

HTF는 HTML 태그를 생성하는 패키지이다. 예를 들어 HTF.HTMLOPEN은 <HTML> 태그를, HTF.HTMLCLOSE는 </HTML> 태그를 생성한다.

58.2. 함수

본 절에서는 HTF 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

58.2.1. ADDRESS

<ADDRESS>와 </ADDRESS> 태그를 작성하는 함수이다. 이 태그는 저작자나 주소, 서명 등을 정의한다.

ADDRESS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ADDRESS
(
    cvalue          IN      VARCHAR2,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cvalue	시작, 종료 태그 사이에 들어갈 문자열이다.
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<ADDRESS CLEAR="cclear" NOWRAP cattributes>cvalue</ADDRESS>
```

58.2.2. ANCHOR

<A>와 </A> 태그를 작성하는 함수이다. 이 태그는 하이퍼텍스트 연결의 시작점이나 끝점을 정의한다.

ANCHOR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ANCHOR
(
    curl            IN      VARCHAR2,
    ctext           IN      VARCHAR2,
    cname           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
curl	HREF 속성값이다.
ctext	시작, 종료 태그 사이에 들어갈 문자열이다..
cname	NAME 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<A HREF="curl" NAME="cname" cattributes>ctext</A>
```

58.2.3. ANCHOR2

<A>와 </A> 태그를 작성하는 함수이다. 이 태그는 하이퍼텍스트 연결의 시작점이나 끝점을 정의한다.

ANCHOR2 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ANCHOR2
(
    curl            IN      VARCHAR2,
    ctext           IN      VARCHAR2,
    cname           IN      VARCHAR2    DEFAULT NULL,
    ctarget         IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
curl	HREF 속성값이다.
ctext	시작, 종료 태그 사이에 들어갈 문자열이다..
cname	NAME 속성값이다.
ctarget	TARGET 속성의 값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<A HREF="curl" NAME="cname" TARGET="ctarget" cattributes>ctext</A>
```

58.2.4. APPLETCLOSE

</APPLET> 태그를 작성하는 함수이다.

APPLETCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.APPLETCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</APPLET>
```

58.2.5. APPLETOPEN

<APPLET> 태그를 작성하는 함수이다. 이 태그는 Java 애플릿을 정의한다.

APPLETOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.APPLETOPEN
(
    ccode           IN      VARCHAR2,
    cheight         IN      NUMBER,
    cwidth          IN      NUMBER,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ccode	CODE 속성값이다.
cheight	HEIGHT 속성값이다.
cwidth	WIDTH 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<APPLET CODE="code" HEIGHT=cheight WIDTH=cwidth cattributes>
```

58.2.6. AREA

<AREA> 태그를 작성하는 함수이다. 이 태그는 이미지 영역을 정의한다.

AREA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.AREA
(
    ccoords         IN      VARCHAR2,
    cshape          IN      VARCHAR2    DEFAULT NULL,
    chref           IN      VARCHAR2    DEFAULT NULL,
```

```
    cnohref         IN      VARCHAR2    DEFAULT NULL,
    ctarget         IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ccoords	COORDS 속성값이다.
cshape	SHAPE 속성값이다.
chref	HREF 속성값이다.
cnohref	이 값이 NULL이 아닐 경우 NOHREF 속성이 추가된다.
ctarget	TARGET 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<AREA COORDS="ccoords" SHAPE="cshape" HREF="chref" NOHREF TARGET="ctarget"
cattributes>
```

58.2.7. BASEFONT

<BASEFONT> 태그를 작성하는 함수이다. 이 태그는 기본 글꼴 크기를 정의한다.

BASEFONT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BASEFONT
(
    nsize           IN      INTEGER
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
nsize	SIZE 속성값이다.

- HTML 태그

함수는 다음 HTML 태그를 생성한다.

```
<BASEFONT SIZE="nsize">
```

58.2.8. BGSOUND

<BGSOUND> 태그를 작성하는 함수이다. 이 태그는 음악 파일을 정의한다.

BGSOUND 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BGSOUND
(
    csrc            IN      VARCHAR2,
    cloop           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
csrc	SRC 속성값이다.
cloop	LOOP 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<BGSOUND SRC="csrc" LOOP="cloop" cattributes>
```

58.2.9. BIG

<BIG>와 </BIG> 태그를 작성하는 함수이다. 이 태그는 현재 글꼴 크기보다 한 단계 더 큰 글꼴을 정의한다.

BIG 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BIG
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<BIG cattributes>ctext</BIG>
```

58.2.10. BLOCKQUOTECLOSE

</BLOCKQUOTE> 태그를 작성하는 함수이다.

BLOCKQUOTECLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BLOCKQUOTECLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</BLOCKQUOTE>
```

58.2.11. BLOCKQUOTEOPEN

<BLOCKQUOTE> 태그를 작성하는 함수이다. 이 태그는 인용문을 정의한다.

BLOCKQUOTEOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BLOCKQUOTEOPEN
(
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<BLOCKQUOTE CLEAR="cclear" NOWRAP cattributes>
```

58.2.12. BODYCLOSE

</BODY> 태그를 작성하는 함수이다.

BODYCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BODYCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</BODY>
```

58.2.13. BODYOPEN

<BODY> 태그를 작성하는 함수이다. 이 태그는 본문의 시작을 정의한다.

BODYOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BODYOPEN
(
    cbackground     IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cbackground	BACKGROUND 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<BODY BACKGROUND="cbackground" cattributes>
```

58.2.14. BOLD

<B>와 </B> 태그를 작성하는 함수이다. 이 태그는 굵은 글꼴로 표현함을 정의한다.

BOLD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BOLD
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<B cattributes>ctext</B>
```

58.2.15. BR

 태그를 작성하는 함수이다. 이 태그는 줄바꿈을 정의한다.

BR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.BR
(
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<BR CLEAR="cclear" cattributes>
```

58.2.16. CENTER

<CENTER>와 </CENTER> 태그를 작성하는 함수이다. 이 태그는 가운데 정렬함을 정의한다.

CENTER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.CENTER
(
    ctext           IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<CENTER>ctext</CENTER>
```

58.2.17. CENTERCLOSE

</CENTER> 태그를 작성하는 함수이다.

CENTERCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.CENTERCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</CENTER>
```

58.2.18. CENTEROPEN

<CENTER> 태그를 작성하는 함수이다.

CENTEROPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.CENTEROPEN
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<CENTER>
```

58.2.19. CITE

<CITE>와 </CITE> 태그를 작성하는 함수이다. 이 태그는 인용문을 정의한다.

CITE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.CITE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<CITE cattributes>ctext</CITE>
```

58.2.20. CODE

<CODE>와 </CODE> 태그를 작성하는 함수이다. 이 태그는 고정폭 글꼴로 표현함을 정의한다.

CODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.CODE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<CODE cattributes>ctext</CODE>
```

58.2.21. COMMENT

<!--와 --> 태그를 작성하는 함수이다. 이 태그는 주석을 정의한다.

COMMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.COMMENT
(
    ctext           IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	주석으로 사용할 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<!-- ctext -->
```

58.2.22. DFN

<DFN>와 </DFN> 태그를 작성하는 함수이다. 이 태그는 이탤릭체로 표현함을 정의한다.

DFN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DFN
(
    ctext           IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<DFN>ctext</DFN>
```

58.2.23. DIRLISTCLOSE

</DIR> 태그를 작성하는 함수이다.

DIRLISTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DIRLISTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</DIR>
```

58.2.24. DIRLISTOPEN

<DIR> 태그를 작성하는 함수이다. 이 태그는 디렉토리 목록을 정의한다.

DIRLISTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DIRLISTOPEN
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<DIR>
```

58.2.25. DIV

<DIV> 태그를 작성하는 함수이다. 이 태그는 문서의 블럭을 정의한다.

DIV 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DIV
(
    calign          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
calign	ALIGN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<DIV ALIGN="calign" cattributes>
```

58.2.26. DLISTCLOSE

</DL> 태그를 작성하는 함수이다.

DLISTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DLISTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</DL>
```

58.2.27. DLISTDEF

<DD> 태그를 작성하는 함수이다. 이 태그는 용어의 설명을 정의한다.

DLISTDEF 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DLISTDEF
(
    ctext           IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	설명으로 사용할 문자열이다.
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<DD CLEAR="cclear" cattributes>ctext
```

58.2.28. DLISTOPEN

<DL> 태그를 작성하는 함수이다. 이 태그는 용어 목록을 정의한다.

DLISTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DLISTOPEN
(
    cclear          IN      VARCHAR2    OPENAULT NULL,
    cattributes     IN      VARCHAR2    OPENAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<DL CLEAR="cclear" cattributes>
```

58.2.29. DLISTTERM

<DT> 태그를 작성하는 함수이다. 이 태그는 용어를 정의한다.

DLISTTERM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.DLISTTERM
(
    ctext           IN      VARCHAR2    TERMAULT NULL,
    cclear          IN      VARCHAR2    TERMAULT NULL,
    cattributes     IN      VARCHAR2    TERMAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	용어로 사용할 문자열이다.
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<DT CLEAR="cclear" cattributes>ctext
```

58.2.30. EM

<EM>와 </EM> 태그를 작성하는 함수이다. 이 태그는 강조함을 정의한다.

EM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.EM
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<EM cattributes>ctext</EM>
```

58.2.31. ESCAPE_SC

HTML에서 특별한 의미를 가지는 문자들로 변환해주는 함수이다.

– &를 &로 바꾼다.

– "를 "로 바꾼다.

– <를 <로 바꾼다.

– >를 >로 바꾼다.

ESCAPE_SC 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ESCAPE_SC
(
    ctext           IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	변환할 문자열이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(HTF.ESCAPE_SC('<html>'));
END;

&lt;html&gt;
```

58.2.32. FONTCLOSE

</FONT> 태그를 작성하는 함수이다.

FONTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FONTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</FONT>
```

58.2.33. FONTOPEN

<FONT> 태그를 작성하는 함수이다. 이 태그는 주어진 폰트 특성을 가지는 문자열의 시작을 정의한다.

FONTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FONTOPEN
(
    ccolor          IN      VARCHAR2    DEFAULT NULL,
    cface           IN      VARCHAR2    DEFAULT NULL,
    csize           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ccolor	COLOR 속성값이다.
cface	FACE 속성값이다.
csize	SIZE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<FONT COLOR="ccolor" FACE="cface" SIZE="csize" cattributes>
```

58.2.34. FORMCHECKBOX

<INPUT> 태그를 작성하는 함수이다. 이 태그는 체크박스 버튼을 정의한다.

FORMCHECKBOX 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMCHECKBOX
(
    cname           IN      VARCHAR2,
    cvalue          IN      VARCHAR2    DEFAULT 'ON',
    cchecked        IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cvalue	VALUE 속성값이다.
cchecked	이 값이 NULL이 아닐 경우 CHECKED 속성이 추가된다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="checkbox" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

58.2.35. FORMCLOSE

</FORM> 태그를 작성하는 함수이다.

FORMCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</FORM>
```

58.2.36. FORMFILE

<INPUT> 태그를 작성하는 함수이다. 이 태그는 파일 필드를 정의한다.

FORMFILE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMFILE
(
    cname           IN      VARCHAR2,
    caccept         IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
caccept	파일 타입 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="file" NAME="cname" ACCEPT="caccept" cattributes>
```

58.2.37. FORMHIDDEN

<INPUT> 태그를 작성하는 함수이다. 이 태그는 숨김 필드를 정의한다.

FORMHIDDEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMHIDDEN
(
    cname           IN      VARCHAR2,
    cvalue          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cvalue	VALUE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="hidden" NAME="cname" VALUE="cvalue" cattributes>
```

58.2.38. FORMIMAGE

<FORMIMAGE> 태그를 작성하는 함수이다. 이 태그는 이미지 필드를 정의한다.

FORMIMAGE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMIMAGE
(
```

```
    cname           IN      VARCHAR2,
    csrc            IN      VARCHAR2,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
csrc	SRC 속성값이다.
calign	ALIGN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="image" NAME="cname" SRC="csrc" ALIGN="calign" cattributes>
```

58.2.39. FORMOPEN

<FORM> 태그를 작성하는 함수이다. 이 태그는 형식의 시작을 정의한다.

FORMOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMOPEN
(
    curl            IN      VARCHAR2,
    cmethod         IN      VARCHAR2    DEFAULT 'POST',
    ctarget         IN      VARCHAR2    DEFAULT NULL,
    cenctype        IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
curl	ACTION 속성값이다.
cmethod	METHOD 속성값이다.

파라미터	설명
ctarget	TARGET 속성값이다.
cenctype	ENCTYPE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<FORM ACTION="curl" METHOD="cmehtod" TARGET="ctarget" ENCTYPE="cenctype" cattributes>
```

58.2.40. FORMPASSWORD

<INPUT> 태그를 작성하는 함수이다. 이 태그는 패스워드 필드를 정의한다.

FORMPASSWORD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMPASSWORD
(
    cname           IN      VARCHAR2,
    csize           IN      VARCHAR2,
    cmaxlength      IN      VARCHAR2    DEFAULT NULL,
    cvalue          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
csize	SIZE 속성값이다.
cmaxlength	MAXLENGTH 속성값이다.
cvalue	VALUE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="password" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

58.2.41. FORMRADIO

<INPUT> 태그를 작성하는 함수이다. 이 태그는 라디오 버튼을 정의한다.

FORMRADIO 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMRADIO
(
    cname           IN      VARCHAR2,
    cvalue          IN      VARCHAR2,
    cchecked        IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cvalue	VALUE 속성값이다.
cchecked	이 값이 NULL이 아닐 경우 CHECKED 속성이 추가된다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="radio" NAME="cname" VALUE="cvalue" CHECKED cattributes>
```

58.2.42. FORMRESET

<INPUT> 태그를 작성하는 함수이다. 이 태그는 초기화 버튼을 정의한다.

FORMRESET 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMRESET
(
    cvalue          IN      VARCHAR2    DEFAULT 'Reset',
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cvalue	VALUE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="reset" VALUE="cvalue" cattributes>
```

58.2.43. FORMSELECTCLOSE

</SELECT> 태그를 작성하는 함수이다.

FORMSELECTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMSELECTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</SELECT>
```

58.2.44. FORMSELECTOPEN

<SELECT> 태그를 작성하는 함수이다. 이 태그는 선택목록 필드를 정의한다.

FORMSELECTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMSELECTOPEN
(
    cname           IN      VARCHAR2,
    cprompt         IN      VARCHAR2    DEFAULT NULL,
    nsize           IN      INTEGER     DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cprompt	태그 앞에 나올 문자열이다.
nsize	SIZE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
cprompt <SELECT NAME="cname" SIZE="nsize" cattributes>
```

58.2.45. FORMSELECTOPTION

<OPTION> 태그를 작성하는 함수이다. 이 태그는 단일 선택 옵션을 정의한다.

FORMSELECTOPTION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMSELECTOPTION
(
    cvalue          IN      VARCHAR2,
    cselected       IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cvalue	옵션값으로 사용할 문자열이다.
cselected	이 값이 NULL이 아닐 경우 SELECTED 속성이 추가된다.

파라미터	설명
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<OPTION SELECTED cattributes>cvalue
```

58.2.46. FORMSUBMIT

<INPUT> 태그를 작성하는 함수이다. 이 태그는 제출 버튼을 정의한다.

FORMSUBMIT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMSUBMIT
(
    cname           IN      VARCHAR2    DEFAULT NULL,
    cvalue          IN      VARCHAR2    DEFAULT 'Submit',
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cvalue	VALUE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="submit" NAME="cname" VALUE="cvalue" cattributes>
```

58.2.47. FORMTEXT

<INPUT> 태그를 작성하는 함수이다. 이 태그는 한 줄 문자열 필드를 정의한다.

FORMTEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMTEXT
(
    cname           IN      VARCHAR2,
    csize           IN      VARCHAR2,
    cmaxlength      IN      VARCHAR2    DEFAULT NULL,
    cvalue          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
csize	SIZE 속성값이다.
cmaxlength	MAXLENGTH 속성값이다.
cvalue	VALUE 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<INPUT TYPE="text" NAME="cname" SIZE="csize" MAXLENGTH="cmaxlength"
VALUE="cvalue" cattributes>
```

58.2.48. FORMTEXTAREA

<TEXTAREA>와 </TEXTAREA> 태그를 작성하는 함수이다. 이 태그는 문자열 필드를 정의한다.

FORMTEXTAREA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMTEXTAREA
(
    cname           IN      VARCHAR2,
    nrows           IN      INTEGER,
    ncolumns        IN      INTEGER,
```

```
    calign          IN      VARCHAR2    DEFAULT NULL,
    cwrap           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
nrows	ROWS 속성값이다.
ncolumns	COLS 속성값이다.
calign	ALIGN 속성값이다.
cwrap	WRAP 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
WARP="cwrap" cattributes>
```

58.2.49. TEXTAREACLOSE

</TEXTAREA> 태그를 작성하는 함수이다.

TEXTAREACLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TEXTAREACLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</TEXTAREA>
```

58.2.50. FORMTEXTAREAOPEN

<TEXTAREA> 태그를 작성하는 함수이다. 이 태그는 문자열 필드의 시작을 정의한다.

FORMTEXTAREAOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FORMTEXTAREAOPEN
(
    cname           IN      VARCHAR2,
    nrows           IN      INTEGER,
    ncolumns        IN      INTEGER,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cwrap           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
nrows	ROWS 속성값이다.
ncolumns	COLS 속성값이다.
calign	ALIGN 속성값이다.
cwrap	WRAP 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TEXTAREA NAME="cname" ROWS="nrows" COLS="ncolumns" ALIGN="calign"
WRAP="cwrap" cattributes>
```

58.2.51. FRAME

<FRAME> 태그를 작성하는 함수이다. 이 태그는 프레임의 특성을 정의한다.

FRAME 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FRAME
(
    csrc            IN      VARCHAR2,
```

```
    cname           IN      VARCHAR2    DEFAULT NULL,
    cmarginwidth    IN      VARCHAR2    DEFAULT NULL,
    cmarginheight   IN      VARCHAR2    DEFAULT NULL,
    cscrolling      IN      VARCHAR2    DEFAULT NULL,
    cnoresize       IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
csrc	URL 문자열이다.
cname	NAME 속성값이다.
cmarginwidth	MARGINWIDTH 속성값이다.
cmarginheight	MARGINHEIGHT 속성값이다.
cscrolling	SCROLLING 속성값이다.
cnoresize	이 값이 NULL이 아닐 경우 NORESIZE 속성이 추가된다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<FRAME SRC="csrc" MARGINWIDTH="cmarginwidth" MARGINHEIGHT="cmarginheight"
SCROLLING="cscrolling" NORESIZE cattributes>
```

58.2.52. FRAMESETCLOSE

</FRAMESET> 태그를 작성하는 함수이다.

FRAMESETCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FRAMESETCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</FRAMESET>
```

58.2.53. FRAMESETOPEN

<FRAMESET> 태그를 작성하는 함수이다. 이 태그는 프레임 집합을 정의한다.

FRAMESETOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.FRAMESETOPEN
(
    crows           IN      VARCHAR2    DEFAULT NULL,
    ccols           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
crows	ROWS 속성값이다.
ccols	COLS 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<FRAMESET ROWS="crows" COLS="ccols" cattributes>
```

58.2.54. HEADCLOSE

</HEAD> 태그를 작성하는 함수이다.

HEADCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.HEADCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</HEAD>
```

58.2.55. HEADER

<H1>와 </H1> 부터 <H6>와 </H6> 까지의 태그를 작성하는 함수이다. 이 태그는 제목의 특성을 정의한다.

HEADER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.HEADER
(
    nsize           IN      INTEGER,
    cheader         IN      VARCHAR2,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
nsize	문자열의 크기값으로 1~6 사이의 값을 사용할 수 있다.
cheader	시작, 종료 태그 사이에 들어갈 문자열이다.
align	ALIGN 속성값이다.
cnowrap	NOWRAP 속성값이다.
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<Hnsize ALIGN="calign" CLEAR="cclear" NOWRAP cattributes>cheader</Hnsize>
```

58.2.56. HEADOPEN

<HEAD> 태그를 작성하는 함수이다. 이 태그는 머릿말의 시작을 정의한다.

HEADOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.HEADOPEN
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<HEAD>
```

58.2.57. HR

<HR> 태그를 작성하는 함수이다. 이 태그는 가로줄을 정의한다.

HR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.HR
(
    cclear          IN      VARCHAR2    DEFAULT NULL,
    csrc            IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cclear	CLEAR 속성값이다.
csrc	SRC 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<HR CLEAR="cclear" SRC="csrc" cattributes>
```

58.2.58. HTMLCLOSE

</HTML> 태그를 작성하는 함수이다.

HTMLCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.HTMLCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</HTML>
```

58.2.59. HTMLOPEN

<HTML> 태그를 작성하는 함수이다. 이 태그는 HTML 문서의 시작을 정의한다.

HTMLOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.HTMLOPEN
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<HTML>
```

58.2.60. IMG

<IMG> 태그를 작성하는 함수이다. 이 태그는 그림 파일을 정의한다.

IMG 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.IMG
(
    curl            IN      VARCHAR2    DEFAULT NULL,
    calign          IN      VARCHAR2    DEFAULT NULL,
    calt            IN      VARCHAR2    DEFAULT NULL,
    cismap          IN      VARCHAR2    DEFAULT NULL,
    cusemap         IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
curl	SRC 속성값이다.
calign	ALIGN 속성값이다.
calt	ALT 속성값이다.
cismap	이 값이 NULL이 아닐 경우 ISMAP 속성이 추가된다.
cusemap	USEMAP 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<IMG SRC="curl" ALIGN="calign" ALT="calt" ISMAP USEMAP="cusemap" cattributes>
```

58.2.61. ISINDEX

<ISINDEX> 태그를 작성하는 함수이다. 이 태그는 프롬프트를 정의한다.

ISINDEX 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ISINDEX
(
    cprompt         IN      VARCHAR2    DEFAULT NULL,
    curl            IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cprompt	PROMPT 속성값이다.
curl	HREF 속성값이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<ISINDEX PROMPT="cprompt" HREF="curl">
```

58.2.62. ITALIC

<I>와 </I> 태그를 작성하는 함수이다. 이 태그는 이탤릭체로 표현함을 정의한다.

ITALIC 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ITALIC
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<I cattributes>ctext</I>
```

58.2.63. KBD

<KBD>와 </KBD> 태그를 작성하는 함수이다. 이 태그는 모노타입 글꼴로 표현함을 정의한다.

KBD 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.KBD
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<KBD cattributes>ctext</KBD>
```

58.2.64. LINKREL

<LINK> 태그를 작성하는 함수이다. 이 태그는 외부로의 연결을 정의한다.

LINKREL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.LINKREL
(
    crel            IN      VARCHAR2,
    curl            IN      VARCHAR2,
    ctitle          IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
crel	REL 속성값이다.
curl	URL 속성값이다.
ctitle	TITLE 속성값이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<LINK REL="crel" HREF="curl" TITLE="ctitle">
```

58.2.65. LINKREV

<LINK> 태그를 작성하는 함수이다. 이 태그는 외부에서의 연결을 정의한다.

LINKREV 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.LINKREV
(
    crev            IN      VARCHAR2,
    curl            IN      VARCHAR2,
    ctitle          IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
crev	REV 속성값이다.
curl	URL 속성값이다.
ctitle	TITLE 속성값이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<LINK REV="crev" HREF="curl" TITLE="ctitle">
```

58.2.66. LISTHEADER

<LH>와 </LH> 태그를 작성하는 함수이다. 이 태그는 목록의 제목을 정의한다.

LISTHEADER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.LISTHEADER
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.

파라미터	설명
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<LH cattributes>ctext</LH>
```

58.2.67. LISTINGCLOSE

</LISTING> 태그를 작성하는 함수이다.

LISTINGCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.LISTINGCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</LISTING>
```

58.2.68. LISTINGOPEN

<LISTING> 태그를 작성하는 함수이다. 이 태그는 HTML 영역이 아님을 정의한다.

LISTINGOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.LISTINGOPEN
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<LISTING>
```

58.2.69. LISTITEM

<LI> 태그를 작성하는 함수이다. 이 태그는 단일 목록을 정의한다.

LISTITEM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.LISTITEM
(
    ctext           IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cdingbat        IN      VARCHAR2    DEFAULT NULL,
    csrc            IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	목록의 이름을 나타내는 문자열이다.
cclear	CLEAR 속성값이다.
cdingbat	DINGBAT 속성값이다.
csrc	SRC 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<LI CLEAR="cclear" DINGBAT="cdingbat" SRC="csrc" cattributes>ctext
```

58.2.70. MAILTO

이메일 형식의 <A>와 </A> 태그를 작성하는 함수이다. 이 태그는 이메일 주소를 정의한다.

MAILTO 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.MAILTO
(
    caddress        IN      VARCHAR2,
```

```
    ctext           IN      VARCHAR2,
    cname           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
caddress	받는 사람의 메일 주소 문자열이다.
ctext	사용자에게 보여줄 문자열이다.
cname	NAME 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<A HREF="mailto:caddress" NAME="cname" cattributes>ctext</A>
```

58.2.71. MAPCLOSE

</MAP> 태그를 작성하는 함수이다.

MAPCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.MAPCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</MAP>
```

58.2.72. MAPOPEN

<MAP> 태그를 작성하는 함수이다. 이 태그는 이미지 영역의 시작을 정의한다.

MAPOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.MAPOPEN
(
    cname           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<MAP NAME="cname" cattributes>
```

58.2.73. MENULISTCLOSE

</MENU> 태그를 작성하는 함수이다.

MENULISTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.MENULISTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</MENU>
```

58.2.74. MENULISTOPEN

<MENU> 태그를 작성하는 함수이다. 이 태그는 한 라인에 하나의 아이템이 존재하는 목록을 정의한다.

MENULISTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.MENULISTOPEN
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<MENU>
```

58.2.75. META

<META> 태그를 작성하는 함수이다. 이 태그는 HTML 문서의 메타 정보를 정의한다.

META 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.META
(
    chttp_equiv     IN      VARCHAR2,
    cname           IN      VARCHAR2,
    ccontent        IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
chttp_equiv	HTTP-EQUIV 속성값이다.
cname	NAME 속성값이다.
ccontent	CONTENT 속성값이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<META HTTP-EQUIV="chttp_equiv" NAME="cname" CONTENT="content">
```

58.2.76. NOBR

<NOBR>와 </NOBR> 태그를 작성하는 함수이다. 이 태그는 줄 바꿈 금지를 정의한다.

NOBR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.NOBR
(
    ctext           IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<NOBR>ctext</NOBR>
```

58.2.77. NOFRAMESCLOSE

</NOFRAMES> 태그를 작성하는 함수이다.

NOFRAMESCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.NOFRAMESCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</NOFRAMES>
```

58.2.78. NOFRAMESOPEN

<NOFRAMES> 태그를 작성하는 함수이다. 이 태그는 비 프레임 영역의 시작을 정의한다.

NOFRAMESOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.NOFRAMESOPEN
```

- HTML 태그

함수는 다음 HTML 태그를 생성한다.

```
<NOFRAMES>
```

58.2.79. OLISTCLOSE

</OL> 태그를 작성하는 함수이다.

OLISTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.OLISTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</OL>
```

58.2.80. OLISTOPEN

<OL> 태그를 작성하는 함수이다. 이 태그는 순서있는 목록의 시작을 정의한다.

OLISTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.OLISTOPEN
(
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cwrap           IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cclear	CLEAR 속성값이다.
cwrap	WRAP 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<OL CLEAR="cclear" WRAP="cwrap" cattributes>
```

58.2.81. PARA

<P> 태그를 작성하는 함수이다. 이 태그는 문단의 시작을 정의한다.

PARA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.PARA
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<P>
```

58.2.82. PARAGRAPH

<P> 태그를 작성하는 함수이다. 이 태그는 문단의 시작을 정의한다.

PARAGRAPH 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.PARAGRAPH
(
    calign          IN      VARCHAR2    DEFAULT NULL,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
calign	ALIGN 속성값이다.
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.

파라미터	설명
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<P ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>cvalue</P>
```

58.2.83. PARAM

<PARAM> 태그를 작성하는 함수이다. 이 태그는 Java 애플릿의 변수를 정의한다.

PARAM 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.PARAM
(
    cname           IN      VARCHAR2,
    cvalue          IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cname	NAME 속성값이다.
cvalue	VALUE 속성값이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<PARAM NAME="cname" VALUE="cvalue">
```

58.2.84. PLAINTEXT

<PLAINTEXT>와 </PLAINTEXT> 태그를 작성하는 함수이다. 이 태그는 비 HTML 영역을 정의한다.

PLAINTEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.PLAINTEXT
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<PLAINTEXT cattributes>ctext</PLAINTEXT>
```

58.2.85. PRECLOSE

</PRE> 태그를 작성하는 함수이다.

PRECLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.PRECLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</PRE>
```

58.2.86. PREOPEN

<PRE> 태그를 작성하는 함수이다. 이 태그는 포맷 비적용 영역의 시작을 정의한다.

PREOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.PREOPEN
(
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cwidth          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cclear	CLEAR 속성값이다.
cwidth	WIDTH 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<PRE CLEAR="cclear" WIDTH="cwidth" cattributes>
```

58.2.87. S

<S>와 </S> 태그를 작성하는 함수이다. 이 태그는 취소선으로 표현함을 정의한다.

S 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.S
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<S cattributes>ctext</S>
```

58.2.88. SAMPLE

<SAMP>와 </SAMP> 태그를 작성하는 함수이다. 이 태그는 표본 영역을 정의한다.

SAMPLE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.SAMPLE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<SAMP cattributes>ctext</SAMP>
```

58.2.89. SCRIPT

<SCRIPT>와 </SCRIPT> 태그를 작성하는 함수이다. 이 태그는 Java나 비주얼 베이직으로 작성된 스크립트를 정의한다.

SCRIPT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.SCRIPT
(
    cscript         IN      VARCHAR2,
    clanguage       IN      VARCHAR2    DEFAULT NULL
```

```
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cscript	시작, 종료 태그 사이에 들어갈 문자열이다.
clanguage	LANGUAGE 속성값이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<SCRIPT LANGUAGE="clanguage">cscript</SCRIPT>
```

58.2.90. SMALL

<SMALL>와 </SMALL> 태그를 작성하는 함수이다. 이 태그는 현재 글꼴 크기보다 한 단계 더 작은 글꼴로 표현함을 정의한다.

SMALL 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.SMALL
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<SMALL cattributes>ctext</SMALL>
```

58.2.91. STRIKE

<STRIKE>와 </STRIKE> 태그를 작성하는 함수이다. 이 태그는 취소선으로 표현함을 정의한다.

STRIKE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.STRIKE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<STRIKE cattributes>ctext</STRIKE>
```

58.2.92. STRONG

<STRONG>와 </STRONG> 태그를 작성하는 함수이다. 이 태그는 강조해서 표현함을 정의한다.

STRONG 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.STRONG
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<STRONG cattributes>ctext</STRONG>
```

58.2.93. STYLE

<STYLE>와 </STYLE> 태그를 작성하는 함수이다. 이 태그는 HTML문서의 스타일 시트를 정의한다.

STYLE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.STYLE
(
    cstyle          IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cstyle	시작, 종료 태그 사이에 들어갈 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<STYLE>cstyle</STYLE>
```

58.2.94. SUB

_와 태그를 작성하는 함수이다. 이 태그는 아랫첨자로 표현함을 정의한다.

SUB 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.SUB
(
    ctext           IN      VARCHAR2,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
calign	ALIGN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<SUB ALIGN="calign" cattributes>ctext</SUB>
```

58.2.95. SUP

^와 태그를 작성하는 함수이다. 이 태그는 윗첨자로 표현함을 정의한다.

SUP 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.SUP
(
    ctext           IN      VARCHAR2,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
calign	ALIGN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<SUP ALIGN="calign" cattributes>ctext</SUP>
```

58.2.96. TABLECAPTION

<CAPTION>와 </CAPTION> 태그를 작성하는 함수이다. 이 태그는 테이블의 제목을 정의한다.

TABLECAPTION 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLECAPTION
(
    ccaption        IN      VARCHAR2,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ccaption	시작, 종료 태그 사이에 들어갈 문자열이다.
calign	ALIGN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<CAPTION ALIGN="calign" cattributes>ccaption</CAPTION>
```

58.2.97. TABLECLOSE

</TABLE> 태그를 작성하는 함수이다.

TABLECLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLECLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</TABLE>
```

58.2.98. TABLEDATA

<TD>와 </TD> 태그를 작성하는 함수이다. 이 태그는 테이블의 단일 셀 데이터를 정의한다.

TABLEDATA 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLEDATA
(
    cvalue          IN      VARCHAR2    DEFAULT NULL,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cdp             IN      VARCHAR2    DEFAULT NULL,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    crowspan        IN      VARCHAR2    DEFAULT NULL,
    ccolspan        IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cvalue	시작, 종료 태그 사이에 들어갈 문자열이다.
calign	ALIGN 속성값이다.
cdp	DP 속성값이다.
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.
crowspan	ROWSPAN 속성값이다.
ccolspan	COLSPAN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TD ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan"
NOWRAP cattributes>cvalue</TD>
```

58.2.99. TABLEHEADER

<TH> 태그를 작성하는 함수이다. 이 태그는 테이블의 머릿글 데이터를 정의한다.

TABLEHEADER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLEHEADER
(
    cvalue          IN      VARCHAR2    DEFAULT NULL,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cdp             IN      VARCHAR2    DEFAULT NULL,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    crowspan        IN      VARCHAR2    DEFAULT NULL,
    ccolspan        IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cvalue	시작, 종료 태그 사이에 들어갈 문자열이다.
calign	ALIGN 속성값이다.
cdp	DP 속성값이다.
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.
crowspan	ROWSPAN 속성값이다.
ccolspan	COLSPAN 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TH ALIGN="calign" DP="cdp" ROWSPAN="crowspan" COLSPAN="ccolspan"
NOWRAP cattributes>cvalue</TH>
```

58.2.100. TABLEOPEN

<TABLE> 태그를 작성하는 함수이다. 이 태그는 테이블의 시작을 정의한다.

TABLEOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLEOPEN
(
    cborder         IN      VARCHAR2    DEFAULT NULL,
    calign          IN      VARCHAR2    DEFAULT NULL,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cborder	BORDER 속성값이다.
calign	ALIGN 속성값이다.
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.
cclear	CLEAR 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TABLE BORDER="cborder" ALIGN="calign" NOWRAP CLEAR="cclear" cattributes>
```

58.2.101. TABLEROWCLOSE

</TR> 태그를 작성하는 함수이다.

TABLEROWCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLEROWCLOSE
```

- HTML 태그

함수는 다음 HTML 태그를 생성한다.

```
</TR>
```

58.2.102. TABLEROWOPEN

<TR> 태그를 작성하는 함수이다. 이 태그는 테이블의 행의 시작을 정의한다.

TABLEROWOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TABLEROWOPEN
(
    calign          IN      VARCHAR2    DEFAULT NULL,
    cvalign         IN      VARCHAR2    DEFAULT NULL,
    cdp             IN      VARCHAR2    DEFAULT NULL,
    cnowrap         IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
calign	ALIGN 속성값이다.
cvalign	VALIGN 속성값이다.
cdp	DP 속성값이다.
cnowrap	이 값이 NULL이 아닐 경우 NOWRAP 속성이 추가된다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TR ALIGN="calign" VALIGN="cvalign" DP="cdp" NOWRAP cattributes>
```

58.2.103. TELETYPE

<TT>와 </TT> 태그를 작성하는 함수이다. 이 태그는 모노타입 글꼴로 표현함을 정의한다.

TELETYPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TELETYPE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TT cattributes>ctext</TT>
```

58.2.104. TITLE

<TITLE>와 </TITLE> 태그를 작성하는 함수이다. 이 태그는 HTML 문서의 제목을 정의한다.

TITLE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.TITLE
(
    ctitle          IN      VARCHAR2
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctitle	시작, 종료 태그 사이에 들어갈 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<TITLE>ctitle</TITLE>
```

58.2.105. ULISTCLOSE

</UL> 태그를 작성하는 함수이다.

ULISTCLOSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ULISTCLOSE
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
</UL>
```

58.2.106. ULISTOPEN

<UL> 태그를 작성하는 함수이다. 이 태그는 순서가 없는 목록의 시작을 정의한다.

ULISTOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.ULISTOPEN
(
    cclear          IN      VARCHAR2    DEFAULT NULL,
    cwrap           IN      VARCHAR2    DEFAULT NULL,
    cdingbat        IN      VARCHAR2    DEFAULT NULL,
    csrc            IN      VARCHAR2    DEFAULT NULL,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
cclear	CLEAR 속성값이다.
cwrap	WRAP 속성값이다.
cdingbat	DINGBAT 속성값이다.
csrc	SRC 속성값이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

함수는 다음 HTML 태그를 생성한다.

```
<UL CLEAR="cclear" WRAP="cwrap" DINGBAT="cdingbat" SRC="csrc" cattributes>
```

58.2.107. UNDERLINE

<U>와 </U> 태그를 작성하는 함수이다. 이 태그는 밑줄로 표현함을 정의한다.

UNDERLINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.UNDERLINE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<U cattributes>ctext</U>
```

58.2.108. VARIABLE

<VAR>와 </VAR> 태그를 작성하는 함수이다. 이 태그는 변수를 정의한다.

VARIABLE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.VARIABLE
(
    ctext           IN      VARCHAR2,
    cattributes     IN      VARCHAR2    DEFAULT NULL
```

```
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
ctext	시작, 종료 태그 사이에 들어갈 문자열이다.
cattributes	기타 속성 문자열이다.

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<VAR cattributes>ctext</VAR>
```

58.2.109. WBR

<WBR> 태그를 작성하는 함수이다. 이 태그는 줄바꿈 위치를 정의한다.

WBR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
HTF.WBR
```

- HTML 태그

  함수는 다음 HTML 태그를 생성한다.

```
<WBR>
```

제59장 HTP

본 장에서는 HTP 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

59.1. 개요

HTP은 HTML 소스 코드를 생성하는 패키지이다. 프러시저를 사용하여 HTML 태그를 작성하고, 합쳐진 소스 코드를 내보내는 역할을 한다. 예를 들어 HTP.HTMLOPEN은 <HTML> 태그를, HTP.HTMLCLOSE는 </HTML> 태그를 생성한다.

59.2. Apache와의 연동

Tibero 5.0부터는 mod_tbpsm라는 Apache 모듈을 구현하여 배포한다. Apache HTTP 서버로부터 온 요청에 따라 mod_tbpsm을 이용하여 HTP 패키지로 구현된 프러시저를 호출하고 생성된 HTML 페이지를 내보낸다.

참고

mod_tbpsm 설정 및 사용 방법은 "Tibero 애플리케이션 개발자 안내서"를 참고한다.

59.3. 프러시저와 함수

본 절에서는 HTP 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

59.3.1. BODYCLOSE

</BODY> 태그를 작성하는 프러시저이다. </BODY> 태그는 HTML 문서 중 본 내용의 끝을 의미한다.

BODYCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.BODYCLOSE
```

- 예제

```
BEGIN
    HTP.BODYCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
</BODY>
```

59.3.2. BODYOPEN

<BODY> 태그를 작성하는 프러시저이다. <BODY> 태그는 HTML 문서 중 본 내용의 시작을 의미하고, 속성을 지정하여 배경 등을 변경할 수 있다.

BODYOPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.BODYOPEN
(
    background       IN      VARCHAR2       DEFAULT NULL,
    attributes       IN      VARCHAR2       DEFAULT NULL
)
```

- 파라미터

파라미터	설명
background	<BODY> 태그의 background 속성값이다. HTML문서의 배경 그림 파일 경로를 지정한다.
attributes	<BODY> 태그의 기타 속성 및 속성값들의 문자열이다.

- 예제

```
BEGIN
    HTP.BODYOPEN('/img/tibero.jpg');
END;
```

다음은 생성된 HTML 코드이다.

```
<BODY background="/img/tibero.jpg">
```

59.3.3. BR

HTML 페이지의 새로운 라인을 가리키는
 태그를 작성하는 프러시저이다.

BR 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.BR
(
    clear         IN     VARCHAR2      DEFAULT NULL,
    attributes    IN     VARCHAR2      DEFAULT NULL
)
```

- 파라미터

파라미터	설명
clear	CLEAR 속성의 값이다.
attributes	기타 속성 및 값들의 문자열이다.

- 예제

```
BEGIN
    HTP.BR;
END;
```

다음은 생성된 HTML 코드이다.

```
<BR>
```

59.3.4. CENTER

<CENTER> 태그, 중앙에 배치될 문자열, </CENTER> 태그를 작성하는 프러시저이다.

CENTER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.CENTER
(
    text      IN       VARCHAR2
)
```

- 파라미터

파라미터	설명
text	웹 페이지의 중앙에 배치될 문자열이다.

- 예제

```
BEGIN
    HTP.CENTER('Tibero');
END;
```

다음은 생성된 HTML 코드이다.

```
<CENTER>Tibero</CENTER>
```

59.3.5. CENTERCLOSE

</CENTER> 태그를 작성하는 프러시저이다.

CENTERCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.CENTERCLOSE
```

- 예제

```
BEGIN
    HTP.CENTERCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
</CENTER>
```

59.3.6. CENTEROPEN

<CENTER> 태그를 작성하는 프러시저이다.

CENTEROPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.CENTEROPEN
```

- 예제

```
BEGIN
    HTP.CENTEROPEN;
    HTP.IMG(url => '/img/tibero.gif');
    HTP.CENTERCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
<CENTER>
<IMG src="/img/tibero.glf">
</CENTER>
```

59.3.7. FONTCLOSE

</FONT> 태그를 작성하는 프러시저이다. <FONT> 태그에서 부여된 속성들은 이 태그 앞까지만 적용된다.

FONTCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.FONTCLOSE
```

- 예제

```
BEGIN
    HTP.FONTCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
</FONT>
```

59.3.8. FONTOPEN

<FONT> 태그를 작성하는 프러시저이다. 이후 작성되는 문자열들의 폰트 설정에 반영된다.

FONTOPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.FONTOPEN
(
    color         IN       VARCHAR2    DEFAULT NULL,
    face          IN       VARCHAR2    DEFAULT NULL,
```

```
    size          IN       VARCHAR2    DEFAULT NULL,
    attributes    IN       VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
color	COLOR 속성의 값이다.
face	FACE 속성의 값이다.
size	SIZE 속성의 값이다.
attributes	기타 속성 및 속성값들의 문자열이다.

- 예제

```
BEGIN
    HTP.FONTOPEN(NULL, 'Fixedsys', 3);
END;
```

다음은 생성된 HTML 코드이다.

```
<FONT face="Fixedsys" size="3">
```

59.3.9. FORMSELECTCLOSE

</SELECT> 태그를 작성하는 프러시저이다.

FORMSELECTCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.FORMSELECTCLOSE
```

- 예제

"59.3.10. FORMSELECTOPEN" 예제를 참조한다.

59.3.10. FORMSELECTOPEN

<SELECT> 태그를 작성하는 프러시저이다.

FORMSELECTOPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.FORMSELECTOPEN
(
    cname       IN  VARCHAR2,
    cprompt     IN  VARCHAR2    DEFAULT NULL,
    nsize       IN  VARCHAR2    DEFAULT NULL,
    cattributes IN  VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
cprompt	선택 리스트 앞에 올 문장이다.
cname	NAME 속성의 값이다.
nsize	SIZE 속성의 값이다.
cattributes	기타 속성 및 속성값들의 문자열이다.

- 예제

```
BEGIN
    HTP.FORMSELECTOPEN('favorite_fruit', '좋아하는 과일은?');
    HTP.FORMSELECTOPTION('사과');
    HTP.FORMSELECTOPTION('포도');
    HTP.FORMSELECTOPTION('귤');
    HTP.FORMSELECTOPTION('자두');
    HTP.FORMSELECTCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
좋아하는 과일은?:
<SELECT NAME="favorite_fruit">
<OPTION>사과
<OPTION>포도
<OPTION>귤
<OPTION>자두
</SELECT>
```

59.3.11. FORMSELECTOPTION

<OPTION> 태그를 작성하는 프러시저이다.

FORMSELECTOPTION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.FORMSELECTOPTION
(
    cvalue          IN  VARCHAR2,
    cselected       IN  VARCHAR2    DEFAULT NULL,
    cattributes     IN  VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
cvalue	선택할 문자열이다.
cselected	SELECTED 속성의 값이다.
cattributes	기타 속성 및 속성값들의 문자열이다.

- 예제

  “59.3.10. FORMSELECTOPEN” 예제를 참조한다.

59.3.12. GET_PAGE

태그를 작성하는 프러시저가 아니라, 생성된 웹 페이지를 읽어온다.

GET_PAGE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.GET_PAGE
(
    page OUT NOCOPY  CLOB
)
```

- 파라미터

파라미터	설명
page	생성된 웹 페이지를 읽어올 값이다.

- 예제

```
DECLARE
    page CLOB;
BEGIN
    HTP.HTMLOPEN;
    HTP.HTMLCLOSE;

    htp.get_page(page);
```

```
    dbms_output.put_line(page);
END;
```

59.3.13. HEADCLOSE

</HEAD> 태그를 작성하는 프러시저이다. </HEAD> 태그는 HTML 문서 중 머릿말의 끝을 의미한다.

HEADCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.HEADCLOSE
```

- 예제

```
BEGIN
    HTP.HEADCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
</HEAD>
```

59.3.14. HEADER

헤더 시작 태그(<H1> ~ <H6>), 헤더 문자열 및 헤더 끝 태그(</H1> ~ </H6>)를 작성하는 프러시저이다.

HEADER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.HEADER
(
    level       IN      PLS_INTEGER,
    name        IN      VARCHAR2,
    align       IN      VARCHAR2    DEFAULT NULL,
    nowrap      IN      VARCHAR2    DEFAULT NULL,
    clear       IN      VARCHAR2    DEFAULT NULL,
    attributes  IN      VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
level	헤더의 레벨이다. 값은 1~6의 범위를 지정할 수 있다.
name	헤더에 표시되는 문자열이다.
align	ALIGN 속성의 값이다.
nowrap	NOWRAP 속성의 값이다.
clear	CLEAR 속성의 값이다.
attributes	기타 속성 및 값들의 문자열이다.

- 예제

```
BEGIN
    HTP.HEADER(1, 'Tibero');
END;
```

다음은 생성된 HTML 코드이다.

```
<H1>Tibero</H1>
```

59.3.15. HEADOPEN

<HEAD> 태그를 작성하는 프러시저이다. <HEAD> 태그는 HTML 문서 중 머릿말의 시작을 의미한다.

HEADOPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.HEADOPEN
```

- 예제

```
BEGIN
    HTP.HEADOPEN;
END;
```

다음은 생성된 HTML 코드이다.

```
<HEAD>
```

59.3.16. HR

HTML 페이지에 가로줄을 그리는 <HR> 태그를 작성하는 프러시저이다.

HR 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.HR
(
    clear       IN      VARCHAR2    DEFAULT NULL,
    src         IN      VARCHAR2    DEFAULT NULL,
    attributes  IN      VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
clear	CLEAR 속성의 값이다.
src	가로줄의 바탕 그림 파일을 지정하는 SRC 속성의 값이다.
attributes	기타 속성 및 값들의 문자열이다.

- 예제

```
BEGIN
    HTP.HR(src => '"/img/tibero.jpg"',
           attributes => 'size="10" noshade style="color:gray;"');
END;
```

다음은 생성된 HTML 코드이다.

```
<HR src="/img/tibero.jpg" size="10" noshade style="color:gray;">
```

59.3.17. HTMLCLOSE

</HTML> 태그를 작성하는 프러시저이다. </HTML> 태그는 HTML 문서의 끝을 의미한다.

HTMLCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.HTMLCLOSE
```

- 예제

```
BEGIN
    HTP.HTMLCLOSE;
END;
```

다음은 생성된 HTML 코드이다.

```
</HTML>
```

59.3.18. HTMLOPEN

<HTML> 태그를 작성하는 프러시저이다. <HTML> 태그는 HTML 문서의 시작을 의미한다.

HTMLOPEN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.HTMLOPEN
```

- 예제

```
BEGIN
    HTP.HTMLOPEN;
END;
```

다음은 생성된 HTML 코드이다.

```
<HTML>
```

59.3.19. IMG

HTML 페이지에 그림을 삽입하는 <IMG> 태그를 작성하는 프러시저이다.

IMG 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.IMG
(
    url        IN      VARCHAR2,
    align      IN      VARCHAR2    DEFAULT NULL,
    alt        IN      VARCHAR2    DEFAULT NULL,
    ismap      IN      VARCHAR2    DEFAULT NULL,
    attributes IN      VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
url	그림 파일을 지정하는 SRC 속성의 값이다.
align	ALIGN 속성의 값이다.

파라미터	설명
alt	웹 브라우저가 그림을 지원하지 않을 경우 대체할 문자열을 지정하는 ALT 속성의 값이다.
ismap	NULL이 아닐 경우 이미지맵임을 가리키는 ISMAP 속성이 부여된다.
attributes	기타 속성 및 값들의 문자열이다.

- 예제

```
BEGIN
    HTP.IMG(url => '"/img/tibero.gif"',
            attributes => 'WIDTH="200" HEIGHT="200" BORDER="1"');
END;
```

다음은 생성된 HTML 코드이다.

```
<IMG src="/img/tibero.gif" WIDTH="200" HEIGHT="200" BORDER="1">
```

59.3.20. IMG2

HTML 페이지에 그림을 삽입하는 <IMG> 태그를 작성하는 프러시저이다. IMG 프러시저에서 usemap 파라미터만 추가된 프러시저이다.

IMG2 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.IMG2
(
    url        IN      VARCHAR2,
    align      IN      VARCHAR2    DEFAULT NULL,
    alt        IN      VARCHAR2    DEFAULT NULL,
    ismap      IN      VARCHAR2    DEFAULT NULL,
    usemap     IN      VARCHAR2    DEFAULT NULL,
    attributes IN      VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
url	그림 파일을 지정하는 SRC 속성의 값이다.
align	ALIGN 속성의 값이다.
alt	웹 브라우저가 그림을 지원하지 않을 경우 대체할 문자열을 지정하는 ALT 속성의 값이다.

파라미터	설명
ismap	NULL이 아닐 경우 이미지맵임을 가리키는 ISMAP 속성이 부여된다.
usemap	USEMAP 속성의 값이다.
attributes	기타 속성 및 값들의 문자열이다.

- 예제

```
BEGIN
    HTP.IMG2(url => '"/img/tibero.gif"',
             attributes => 'WIDTH="200" HEIGHT="200" BORDER="1"');
END;
```

다음은 생성된 HTML 코드이다.

```
<IMG src="/img/tibero.gif" WIDTH="200" HEIGHT="200" BORDER="1">
```

59.3.21. LINE

HTML 페이지에 가로줄을 그리는 <HR> 태그를 작성하는 프러시저이다. HR 프러시저와 똑같은 기능을 제공한다.

LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.LINE
(
    clear       IN      VARCHAR2    DEFAULT NULL,
    src         IN      VARCHAR2    DEFAULT NULL,
    attributes  IN      VARCHAR2    DEFAULT NULL
)
```

- 파라미터

파라미터	설명
clear	CLEAR 속성의 값이다.
src	가로줄의 바탕 그림 파일을 지정하는 SRC 속성의 값이다.
attributes	기타 속성 및 값들의 문자열이다.

- 예제

```
BEGIN
    HTP.LINE(src => '"/img/tibero.jpg"',
```

```
            attributes => 'size="10" noshade style="color:gray;"');
END;
```

다음은 생성된 HTML 코드이다.

```
<HR src="/img/tibero.jpg" size="10" noshade style="color:gray;">
```

59.3.22. PRINT

작성 진행 중인 HTML 소스 코드 뒤에 입력 문자열 및 \n(새줄 문자)을 작성하는 프러시저이다. HTML 소스 코드상의 새줄의 의미이므로 실제 웹 페이지에서 새 줄을 추가하려면
 태그를 작성해야 한다.

PRINT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.PRINT
(
    str       IN       VARCHAR2        DEFAULT NULL
)
```

- 파라미터

파라미터	설명
str	HTML 소스 코드에 추가로 생성할 문자열이다.

- 예제

```
BEGIN
    HTP.PRINT('<html>');
    HTP.PRINT('<head>');
    HTP.PRINT('</head>');
    HTP.PRINT('<body>');
    HTP.PRINT('He');
    HTP.PRINT('llo');
    HTP.PRINT('</body>');
    HTP.PRINT('</html>');
END;
```

다음은 생성된 HTML 코드이다.

```
<html>
<head>
</head>
<body>
He
```

```
llo
</body>
</html>
```

59.3.23. PRN

작성 진행 중인 HTML 소스 코드 뒤에 입력 문자열을 작성하는 프러시저이다. PRINT 프러시저와 다르게 \n(새줄 문자)가 붙지 않는다.

PRN 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.PRN
(
    str       IN       VARCHAR2     DEFAULT NULL
)
```

- 파라미터

파라미터	설명
str	HTML 소스 코드에 추가로 생성할 문자열이다.

- 예제

```
BEGIN
    HTP.PRN('<html>');
    HTP.PRN('<head>');
    HTP.PRN('</head>');
    HTP.PRN('<body>');
    HTP.PRN('He');
    HTP.PRN('llo');  -- He 바로 뒤에 붙어 Hello가 된다. (생성되는 HTML 소스 코드상에서)
    HTP.PRN('</body>');
    HTP.PRN('</html>');
END;
```

다음은 생성된 HTML 코드이다.

```
<html><head></head><body>Hello</body></html>
```

59.3.24. TITLE

<TITLE> </TITLE> 태그 및 그 사이에 제목을 작성하는 프러시저이다. <TITLE> 태그는 웹 브라우저 창에서 제목 표시줄에 표시되는 문자열을 지정한다.

TITLE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
HTP.TITLE
(
    title      IN       VARCHAR2
)
```

- 파라미터

파라미터	설명
title	제목 문자열이다.

- 예제

```
BEGIN
    HTP.TITLE('Tibero');
END;
```

다음은 생성된 HTML 코드이다.

```
<TITLE>Tibero</TITLE>
```

제60장 OWA_UTIL

본 장에서는 OWA_UTIL 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

60.1. 개요

OWA_UTIL는 웹 에이전트에서 주로 사용할 유틸리티를 담고 있는 패키지이다.

60.2. 프러시저

본 절에서는 OWA_UTIL 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

60.2.1. WHO_CALLED_ME

어떤 오브젝트로부터 본 오브젝트가 호출되었는지 정보를 가져온다.

WHO_CALLED_ME 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE WHO_CALLED_ME
(
    owner           OUT      VARCHAR2,
    name            OUT      VARCHAR2,
    lineno          OUT      NUMBER,
    caller_t        OUT      VARCHAR2)

);
```

- 파라미터

파라미터	설명
owner	해당 오브젝트를 호출한 오브젝트의 소유자이다.
name	해당 오브젝트를 호출한 오브젝트의 이름이다.
lineno	해당 오브젝트를 호출한 오브젝트의 라인번호이다.
caller_t	해당 오브젝트를 호출한 오브젝트의 타입이다.

- 예제

```
create or replace procedure OWA_PROC1
as
    v1 varchar(128);
    v2 varchar(128);
    v3 number;
    v4 varchar(128);
begin
    owa_util.who_called_me (v1, v2, v3, v4);

    dbms_output.put_line (v1);
    dbms_output.put_line (v2);
    dbms_output.put_line (v3);
    dbms_output.put_line (v4);
end;
/

create or replace procedure OWA_PROC2
as
begin
    OWA_PROC1;
end;
/

begin
 OWA_PROC1;
end;
/
--2
--ANONYMOUS BLOCK

 begin
    OWA_PROC2;
end;
/
--TIBERO
--OWA_PROC2
--4
--PROCEDURE
```

제61장 SA_COMPONENTS

본 장에서는 SA_COMPONENTS 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

61.1. 개요

SA_COMPONENTS 는 Tibero Label Security 기능의 레이블들의 정의를 관리하는 패키지이다.

61.2. 프러시저

본 절에서는 SA_COMPONENTS 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

61.2.1. ALTER_COMPARTMENT

해당 구획의 짧은 이름, 긴 이름을 바꿔준다. 초기 설정 후에는 구획 번호는 바꿀 수 없다. 해당 구획이 사용 중이라면 짧은 이름은 바꿀 수 없지만, 긴 이름은 바꿀 수 있다.

ALTER_COMPARTMENT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.ALTER_COMPARTMENT
(
   policy_name       IN VARCHAR2,
   comp_num          IN NUMBER(38),
   new_short_name    IN VARCHAR2,
   new_long_name     IN VARCHAR2
);
```

```
SA_COMPONENTS.ALTER_COMPARTMENT
(
   policy_name       IN VARCHAR2,
   short_name        IN VARCHAR2 DEFAULT NULL,
   new_long_name     IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
comp_num	바꿔줄 구획의 번호이다. 저장된 구획 번호는 ALL_SA_COMPARTMENTS 뷰의 COMP_NUM 열에서 찾을 수 있다.
short_name	바꿔줄 구획의 짧은 이름이다. 저장된 짧은 이름은 ALL_SA_COMPARTMENTS 뷰의 SHORT_NAME 열에서 찾을 수 있다.
new_short_name	바꿀 새로운 구획의 짧은 이름이다.
new_long_name	바꿀 새로운 구획의 긴 이름이다.

- 예제

```
BEGIN
  SA_COMPONENTS.ALTER_COMPARTMENT (
   policy_name         => 'tls_pol',
   comp_num            => '65',
   new_short_name      => 'CHEM',
   new_long_name       => 'CHEMICAL');
END;
/
```

61.2.2. ALTER_GROUP

해당 그룹의 짧은 이름, 긴 이름을 바꿔준다. 초기 설정 후에는 그룹 번호는 바꿀 수 없다. 해당 그룹이 사용중이라면 짧은 이름은 바꿀 수 없지만, 긴 이름은 바꿀 수 있다.

ALTER_GROUP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.ALTER_GROUP
(
   policy_name    IN VARCHAR2,
   group_num      IN NUMBER(38),
   new_short_name IN VARCHAR2 DEFAULT NULL,
```

```
   new_long_name  IN VARCHAR2 DEFAULT NULL
);
```

```
SA_COMPONENTS.ALTER_GROUP
(
   policy_name    IN VARCHAR2,
   short_name     IN VARCHAR2,
   new_long_name  IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
group_num	바꿔줄 그룹의 번호이다. 저장된 그룹 번호는 ALL_SA_GROUPS 뷰의 GROUP_NUM 열에서 찾을 수 있다.
short_name	바꿔줄 그룹의 짧은 이름이다. 저장된 짧은 이름은 ALL_SA_GROUPS 뷰의 SHORT_NAME 열에서 찾을 수 있다.
new_short_name	바꿀 새로운 그룹의 짧은 이름이다.
new_long_name	바꿀 새로운 그룹의 긴 이름이다.

- 예제

```
BEGIN
  SA_COMPONENTS.ALTER_GROUP (
   policy_name     => 'tls_pol',
   short_name      => 'TM_FIN',
   new_long_name   => 'TM_FINANCES');
END;
/
```

61.2.3. ALTER_GROUP_PARENT

해당 그룹과 연관 되어있는 부모 그룹을 바꿔준다.

ALTER_GROUP_PARENT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.ALTER_GROUP_PARENT
(
   policy_name     IN VARCHAR2,
   group_num       IN NUMBER(38),
   new_parent_num  IN NUMBER(38)
);
```

```
SA_COMPONENTS.ALTER_GROUP_PARENT
(
   policy_name     IN VARCHAR2,
   group_num       IN NUMBER(38),
   new_parent_name IN VARCHAR2
);
```

```
SA_COMPONENTS.ALTER_GROUP_PARENT
(
   policy_name     IN VARCHAR2,
   short_name      IN VARCHAR2,
   new_parent_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
group_num	바꿔줄 그룹의 번호이다. 저장된 그룹 번호는 ALL_SA_GROUPS 뷰의 GROUP_NUM 열에서 찾을 수 있다.
short_name	바꿔줄 그룹의 짧은 이름이다. 저장된 짧은 이름은 ALL_SA_GROUPS 뷰의 SHORT_NAME 열에서 찾을 수 있다.
new_parent_num	바꿀 새로운 그룹의 부모 그룹 번호이다. 부모 그룹 번호는 ALL_SA_GROUPS 뷰의 PARENT_NUM 열에서 찾을 수 있다.
new_parent_name	바꿀 새로운 그룹의 부모 그룹 이름이다. 그룹 이름은 ALL_SA_GROUPS 뷰의 SHORT_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
  SA_COMPONENTS.ALTER_GROUP_PARENT (
   policy_name         => 'tls_pol',
   group_num           => 1000,
   new_parent_name     => 'TM');
END;
/
```

61.2.4. ALTER_LEVEL

해당 레벨의 짧은 이름, 긴 이름을 바꿔준다. 초기 설정한 후에는 레벨 번호는 바꿀 수 없다. 해당 레벨이 사용중이라면 짧은 이름은 바꿀 수 없지만 긴 이름은 바꿀 수 있다.

ALTER_LEVEL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.ALTER_LEVEL
(
   policy_name     IN VARCHAR2,
   level_num       IN NUMBER(38),
   new_short_name  IN VARCHAR2 DEFAULT NULL,
   new_long_name   IN VARCHAR2 DEFAULT NULL
);
```

```
SA_COMPONENTS.ALTER_LEVEL
(
   policy_name     IN VARCHAR2,
   short_name      IN VARCHAR2,
   new_long_name   IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
level_num	바꿔줄 레벨의 번호이다. 저장된 레벨 번호는 ALL_SA_LEVELS 뷰의 LEVEL_NUM 열에서 찾을 수 있다.

파라미터	설명
short_name	바꿔줄 레벨의 짧은 이름이다. 저장된 짧은 번호는 ALL_SA_LEVELS 뷰의 SHORT_NAME 열에서 찾을 수 있다.
new_short_name	바꿀 새로운 레벨의 짧은 이름이다.
new_long_name	바꿀 새로운 레벨의 긴 이름이다.

- 예제

```
BEGIN
 SA_COMPONENTS.ALTER_LEVEL (
   policy_name     => 'tls_pol',
   level_num       => 8,
   new_short_name  => 'S',
   new_long_name   => 'SECRET');
END;
/
```

61.2.5. CREATE_COMPARTMENT

해당 Label Security 정책에서 새로운 구획을 만들고 구획 번호, 짧은 이름, 긴 이름을 지정한다. 구획 번호는 레이블의 문자열 표현의 순서를 결정한다.

CREATE_COMPARTMENT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.CREATE_COMPARTMENT
(
   policy_name IN VARCHAR2,
   comp_num    IN NUMBER(38),
   short_name  IN VARCHAR2,
   long_name   IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.

파라미터	설명
comp_num	만들 구획의 번호이다.
short_name	만들 구획의 짧은 이름이다.
long_name	만들 구획의 긴 이름이다.

- 예제

```
BEGIN
  SA_COMPONENTS.CREATE_COMPARTMENT (
   policy_name     => 'tls_pol',
   comp_num        => '50',
   short_name      => 'CHEM',
   long_name       => 'CHEMICAL');
END;
/
```

61.2.6. CREATE_GROUP

해당 Label Security 정책에서 새로운 그룹을 만들고 그룹 번호, 짧은 이름, 긴 이름, 부모 그룹 이름을 지정한다.

CREATE_GROUP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.CREATE_GROUP
(
   policy_name IN VARCHAR2,
   group_num   IN NUMBER(38),
   short_name  IN VARCHAR2,
   long_name   IN VARCHAR2,
   parent_name IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
group_num	만들 그룹의 번호이다.
short_name	만들 그룹의 짧은 이름이다.

파라미터	설명
long_name	만들 그룹의 긴 이름이다.
parent_name	만들 그룹의 부모 그룹의 짧은 이름이다. NULL이면 그룹은 최상위 그룹이 된다.

- 예제

```
BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'tls_pol',
   group_num       => 100,
   short_name      => 'TM',
   long_name       => 'TMAX');
END;
/

BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'tls_pol',
   group_num       =>  110,
   short_name      => 'TM_FIN',
   long_name       => 'TM_FINANCES',
   parent_name     => 'TM');
END;
/
```

61.2.7. CREATE_LEVEL

해당 Label Security 정책에서 새로운 레벨을 만들고 레벨 번호, 짧은 이름, 긴 이름을 지정한다. 레벨 번호는 민감도 순위를 결정한다(낮은 숫자가 덜 민감한 데이터를 나타냄).

CREATE_LEVEL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.CREATE_LEVEL
(
   policy_name       IN VARCHAR2,
   level_num         IN NUMBER(38),
   short_name        IN VARCHAR2,
   long_name         IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
level_num	만들 레벨의 번호이다.
short_name	만들 레벨의 짧은 이름이다.
long_name	만들 레벨의 긴 이름이다.

- 예제

```
BEGIN
 SA_COMPONENTS.CREATE_LEVEL (
   policy_name   => 'tls_pol',
   level_num     => 10,
   short_name    => 'TS',
   long_name     => 'TOP_SECRET');
END;
/
```

61.2.8. DROP_COMPARTMENT

해당 Label Security 정책에서 구획 번호나 짧은 이름이 일치하는 구획을 제거한다. 해당 구획이 사용 중이라면 제거할 수 없다.

DROP_COMPARTMENT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.DROP_COMPARTMENT
(
   policy_name IN VARCHAR2,
   comp_num    IN INTEGER
);
```

```
SA_COMPONENTS.DROP_COMPARTMENT
(
   policy_name IN VARCHAR2,
   short_name  IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
comp_num	해당 정책에서 제거할 구획의 번호이다. 저장된 구획 번호는 ALL_SA_COMPARTMENTS 뷰의 COMP_NUM 열에서 찾을 수 있다.
short_name	해당 정책에서 제거할 구획의 짧은 이름이다. 저장된 짧은 이름은 ALL_SA_COMPARTMENTS 뷰의 SHORT_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
  SA_COMPONENTS.DROP_COMPARTMENT (
   policy_name     => 'tls_pol',
   short_name      => 'CHEM');
END;
/
```

61.2.9. DROP_GROUP

해당 Label Security 정책에서 그룹 번호나 짧은 이름이 일치하는 그룹을 제거한다. 해당 그룹이 사용 중이라면 제거 할 수 없다.

DROP_GROUP 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.DROP_GROUP
(
   policy_name IN VARCHAR2,
   group_num   IN NUMBER(38)
);
```

```
SA_COMPONENTS.DROP_GROUP
(
   policy_name IN VARCHAR2,
   short_name  IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
group_num	해당 정책에서 제거할 그룹의 번호이다. 저장된 그룹 번호는 ALL_SA_GROUPS 뷰의 GROUP_NUM 열에서 찾을 수 있다.
short_name	해당 정책에서 제거할 그룹의 짧은 이름이다. 저장된 짧은 이름은 ALL_SA_GROUPS 뷰의 SHORT_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
  SA_COMPONENTS.DROP_GROUP (
   policy_name     => 'tls_pol',
   group_num       => 110);
END;
/
```

61.2.10. DROP_LEVEL

해당 Label Security 정책에서 레벨 번호나 짧은 이름이 일치하는 레벨을 제거한다. 해당 레벨이 사용 중이라면 제거할 수 없다.

DROP_LEVEL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_COMPONENTS.DROP_LEVEL
(
   policy_name IN VARCHAR2,
   level_num   IN NUMBER(38)
);
```

```
SA_COMPONENTS.DROP_LEVEL
(
   policy_name IN VARCHAR2,
```

```
   short_name  IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
level_num	해당 정책에서 제거할 레벨의 번호이다. 저장된 레벨 번호는 ALL_SA_LEVELS 뷰의 LEVEL_NUM 열에서 찾을 수 있다.
short_name	해당 정책에서 제거할 레벨의 짧은 이름이다. 저장된 짧은 이름은 ALL_SA_LEVELS 뷰의 SHORT_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
 SA_COMPONENTS.DROP_LEVEL (
   policy_name     => 'tls_pol',
   level_num       => 10);
END;
/
```

제62장 SA_LABEL_ADMIN

본 장에서는 SA_LABEL_ADMIN 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

62.1. 개요

SA_LABEL_ADMIN 는 Tibero Label Security 정책의 레이블들을 관리하는 패키지이다.

62.2. 프러시저

본 절에서는 SA_LABEL_ADMIN 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

62.2.1. ALTER_LABEL

이 프러시저를 수행하면 레이블 태그와 연관된 문자열 레이블 정의를 변경한다.

레이블 태그는 변경할 수 없다. 레이블 태그와 연관된 문자열 레이블을 변경하면 그 레이블을 가지고 있는 데이터의 민감도는 변경된다. 예를 들어 레이블 태그가 3001인 레이블의 문자열을 S:A에서 S:B로 변경하면 해당하는 데이터의 접근성이 변화된다.

레이블 태그 값이 변경되지 않음에도 이 프로시저를 이용하면 모든 행을 변경할 필요없이 데이터의 민감도를 변경할 수 있다. 어떤 레이블을 바꿀지는 레이블 태그나 문자열 값을 입력하면 된다.

ALTER_LABEL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_LABEL_ADMIN.ALTER_LABEL
(
   policy_name       IN VARCHAR2,
   label_tag         IN BINARY_INTEGER,
   new_label_value   IN VARCHAR2 DEFAULT NULL,
   new_data_label    IN BOOLEAN  DEFAULT NULL
);
```

```
SA_LABEL_ADMIN.ALTER_LABEL
(
   policy_name       IN VARCHAR2,
   label_value       IN VARCHAR2,
```

```
   new_label_value   IN VARCHAR2 DEFAULT NULL,
   new_data_label    IN BOOLEAN  DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
label_tag	바꿔줄 레이블의 정수 태그 값이다. 저장된 레이블 태그 값은 ALL_SA_LABELS 뷰의 LABEL_TAG 열에서 찾을 수 있다.
label_value	바꿔줄 레이블의 문자열 표현 값이다. 저장된 레이블 문자열 표현 값은 ALL_SA_LABELS 뷰의 LABEL 열에서 찾을 수 있다.
new_label_value	바꿀 새로운 레이블의 문자열 표현 값이다. NULL일 경우 레이블 문자열 표현 값은 바뀌지 않는다.
new_data_label	TRUE일 경우 열 데이터에 해당 레이블을 부여할 수 있다. NULL일 경우 해당 레이블의 데이터 레이블 값은 바뀌지 않는다.

- 예제

```
BEGIN
  SA_LABEL_ADMIN.ALTER_LABEL (
   policy_name       => 'tls_pol',
   label_tag         => 10,
   new_label_value   => 'HS',
   new_data_label    => TRUE);
END;
/
```

62.2.2. CREATE_LABEL

이 프러시저를 실행하면 새로운 데이터 레이블을 만들 수 있다.

CREATE_LABEL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_LABEL_ADMIN.CREATE_LABEL
(
   policy_name IN VARCHAR2,
   label_tag   IN BINARY_INTEGER,
   label_value IN VARCHAR2,
   data_label  IN BOOLEAN DEFAULT TRUE
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
label_tag	다른 정책 레이블을 기준으로 레이블의 정렬 순서를 나타내는 고유한 정수 값을 지정한다.
label_value	만들 레이블의 문자열 표현을 지정한다. 레벨, 구획, 그룹 레이블의 짧은 이름을 사용한다. 저장된 이름들은 ALL_SA_LEVELS, ALL_SA_COMPARTMENTS, ALL_SA_GROUPS 뷰의 SHORT NAME 열에서 찾을 수 있다.
data_label	TRUE이면 해당 레이블을 열 데이터에 부여 가능하다. 이 값을 이용해서 데이터에 사용 가능한 레이블인지 정의한다.

- 예제

```
BEGIN
  SA_LABEL_ADMIN.CREATE_LABEL (
   policy_name     => 'tls_pol',
   label_tag       => 1010,
   label_value     => 'HS:FIN',
   data_label      => TRUE);
END;
/
```

62.2.3. DROP_LABEL

해당 레이블을 제거한다. 사용 중인 레이블을 제거하는 것은 불가하다. 이 프러시저는 데이터에 레이블을 부여하기 전이나 레이블을 세팅할 때 사용하길 권유한다. 만약 레이블을 사용 중에 제거하려면 해당 정책을 비활성화한 후 문제를 해결하고 다시 정책을 활성화시켜야 한다.

DROP_LABEL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_LABEL_ADMIN.DROP_LABEL
(
   policy_name       IN VARCHAR2,
   label_tag         IN BINARY_INTEGER
);
```

```
SA_LABEL_ADMIN.DROP_LABEL
(
   policy_name       IN VARCHAR2,
   label_value       IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
label_tag	제거할 레이블의 정수 태그 값이다. 저장된 레이블 태그 값은 ALL_SA_LABELS 뷰의 LABEL_TAG 열에서 찾을 수 있다.
label_value	제거할 레이블의 문자열 표현 값이다. 저장된 레이블 문자열 표현 값은 ALL_SA_LABELS 뷰의 LABEL 열에서 찾을 수 있다.

- 예제

```
BEGIN
  SA_LABEL_ADMIN.DROP_LABEL (
   policy_name     => 'tls_pol',
   label_tag       => 1010);
END;
/
```

제63장 SA_POLICY_ADMIN

본 장에서는 SA_POLICY_ADMIN 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

63.1. 개요

SA_POLICY_ADMIN 는 Tibero Label Security의 정책들을 관리하는 패키지이다.

63.2. 프러시저

본 절에서는 SA_POLICY_ADMIN 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

63.2.1. APPLY_TABLE_POLICY

이 프로시저를 수행하면 해당 테이블에 해당 Label Security 정책을 부여한다. Label Security 정책의 레이블 열이 해당 테이블에 존재하지 않다면 레이블 열을 해당 테이블에 추가하고 값은 NULL로 지정한다.

정책이 적용되면 자동으로 테이블에 정책이 부여된다. 테이블 옵션을 변경하려면 해당 정책을 테이블에서 제거하고 다시 적용시켜야 한다.

ALTER_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_POLICY_ADMIN.APPLY_TABLE_POLICY
(
  policy_name       IN VARCHAR2,
  schema_name       IN VARCHAR2,
  table_name        IN VARCHAR2,
  table_options     IN VARCHAR2 DEFAULT NULL,
  label_function    IN VARCHAR2 DEFAULT NULL,
  predicate         IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다.

파라미터	설명
	저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
schema_name	정책이 보호할 테이블을 소유한 스키마 이름이다.
table_name	정책이 보호할 테이블 이름이다.
table_options	테이블에 사용할 정책 시행 옵션의 콤마(,)로 구분된 목록이다. NULL이면 정책의 default_option이 사용된다. 가능한 정책 시행 옵션은 다음과 같다. – CHECK_CONTROL : INSERT, UPDATE 연산에 READ_CONTROL 정책 옵션을 부여한다. 따라서, 새로 생성된 행에 대해서 정책을 적용한다. – READ_CONTROL : 모든 쿼리에 정책을 적용한다. 권한이 부여된 행에 대해서만 SELECT, UPDATE, DELETE 연산이 가능하다. – WRITE_CONTROL : INSERT_CONTROL, DELETE_CONTROL, UPDATE_CONTROL를 합친 옵션이다. – INSERT_CONTROL : INSERT 연산에 정책을 적용한다. – DELETE_CONTROL : DELETE 연산에 정책을 적용한다. – UPDATE_CONTROL : UPDATE 연산에 정책을 적용한다. – ALL_CONTROL : 모든 정책 시행 옵션을 적용한다. – NO_CONTROL : 아무런 정책 시행 옵션을 적용하지 않는다.
label_function	기본 값으로 사용할 레이블 값을 반환하는 함수를 명시하는 문자열이다.
predicate	READ_CONTROL일 때 사용되는 추가 where 조건을 입력하는 문자열이다 (AND 나 OR 앞에 입력해야한다).

- 예제

```
BEGIN
 SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
  policy_name    => 'tls_pol',
  schema_name    => 'HR',
  table_name     => 'EMPLOYEES',
  table_options  => NULL,
  label_function => NULL,
  predicate      => 'read_control');
END;
/
```

63.2.2. REMOVE_TABLE_POLICY

해당 테이블에 지정된 명시된 Label Security 정책을 제거한다. 테이블에 적용된 정책 조건들과 해당하는 DML 트리거들도 제거된다. 정책 레이블 열도 옵션을 통해 제거할 수 있다.

REMOVE_TABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_POLICY_ADMIN.REMOVE_TABLE_POLICY
(
  policy_name        IN VARCHAR2,
  schema_name        IN VARCHAR2,
  table_name         IN VARCHAR2,
  drop_column        IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
schema_name	정책을 제거할 테이블이 속한 스키마 이름이다.
table_name	정책을 제거할 테이블 이름이다.
drop_column	레이블 열을 제거할 지 여부이다. TRUE이면 레이블 열을 테이블에서 제거하고 FALSE이면 제거하지 않는다.

- 예제

```
BEGIN
 SA_POLICY_ADMIN.REMOVE_TABLE_POLICY(
  policy_name    => 'tls_pol',
  schema_name    => 'HR',
  table_name     => 'EMPLOYEES',
  drop_column    => TRUE);
END;
/
```

제64장 SA_SYSDBA

본 장에서는 SA_SYSDBA 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

64.1. 개요

SA_SYSDBA 패키지는 Tibero Label Security 정책들을 생성, 변경, 작동 제어, 제거하는 데 사용할 수 있는 프러시저를 제공한다.

64.2. 프러시저

본 절에서는 SA_SYSDBA 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

64.2.1. ALTER_POLICY

해당 Label Security 정책을 변경한다. 기본 시행 옵션과 레이블 열 이름을 변경할 수 있다. 하지만 레이블 열 이름은 해당 정책이 테이블에 적용되지 않았을 때만 변경 가능하다.

ALTER_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_SYSDBA.ALTER_POLICY
(
   policy_name       IN  VARCHAR2,
   default_options   IN  VARCHAR2 DEFAULT NULL,
   column_name       IN  VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.

파라미터	설명
default_options	정책이 적용되고 테이블 옵션이 지정되지 않은 경우 사용할 기본 시행 옵션이다. 각 옵션을 콤마(,)로 구분해야 한다. 가능한 정책 시행 옵션은 다음과 같다. – CHECK_CONTROL : INSERT, UPDATE 연산에 READ_CONTROL 정책 옵션을 부여한다. 따라서, 새로 생성된 행에 대해서 정책을 적용한다. – READ_CONTROL : 모든 쿼리에 정책을 적용한다. 권한이 부여된 행에 대해서만 SELECT, UPDATE, DELETE 연산이 가능하다. – WRITE_CONTROL : INSERT_CONTROL, DELETE_CONTROL, UPDATE_CONTROL를 합친 옵션이다. – INSERT_CONTROL : INSERT 연산에 정책을 적용한다. – DELETE_CONTROL : DELETE 연산에 정책을 적용한다. – UPDATE_CONTROL : UPDATE 연산에 정책을 적용한다. – ALL_CONTROL : 모든 정책 시행 옵션을 적용한다. – NO_CONTROL : 아무런 정책 시행 옵션을 적용하지 않는다.
column_name	정책과 연결된 열 이름을 지정한다. 저장된 열 이름은 ALL_SA_POLICIES 뷰의 COLUMN_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
 SA_SYSDBA.ALTER_POLICY (
  policy_name      => 'tls_pol',
  default_options  => 'read_control, delete_control');
END;
/
```

64.2.2. CREATE_POLICY

새로운 Label Security 정책을 생성한다. 정책과 연결된 열 이름, 기본 정책 옵션들을 설정한다.

CREATE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_SYSDBA.CREATE_POLICY
(
   policy_name       IN VARCHAR2,
   column_name       IN VARCHAR2 DEFAULT NULL,
   default_options   IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 정책 이름은 중복이 존재하면 안된다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
column_name	정책에 의해 보호되는 테이블에 추가할 열 이름을 지정한다. NULL이면 policy_name_COL이 사용된다. 두 개의 정책은 같은 열 이름을 가질 수 없다.
default_options	정책이 적용되고 테이블 옵션이 지정되지 않은 경우 사용할 기본 시행 옵션이다. 각 옵션을 콤마(,)로 구분해야 한다. 가능한 정책 시행 옵션은 다음과 같다. – CHECK_CONTROL : INSERT, UPDATE 연산에 READ_CONTROL 정책 옵션을 부여한다. 따라서, 새로 생성된 label에 대해서 정책을 적용한다. – READ_CONTROL : 모든 쿼리에 정책을 적용한다. 권한이 부여된 row에 대해서만 SELECT, UPDATE, DELETE 연산이 가능하다. – WRITE_CONTROL : INSERT_CONTROL, DELETE_CONTROL, UPDATE_CONTROL를 합친 옵션이다. – INSERT_CONTROL : INSERT 연산에 정책을 적용한다. – DELETE_CONTROL : DELETE 연산에 정책을 적용한다. – UPDATE_CONTROL : UPDATE 연산에 정책을 적용한다. – ALL_CONTROL : 모든 정책 시행 옵션을 적용한다. – NO_CONTROL : 아무런 정책 시행 옵션을 적용하지 않는다.

- 예제

```
BEGIN
 SA_SYSDBA.CREATE_POLICY (
  policy_name      => 'tls_pol',
  column_name      => 'tls_col',
  default_options  => 'read_control, write_control');
```

```
END;
/
```

64.2.3. DISABLE_POLICY

해당 정책을 제거하지 않고 정책의 시행을 중지시킨다. 정책의 시행을 중지시키면 정책에 의해 보호된 모든 테이블의 접근 중재가 해제된다. 정책의 시행을 중지시키더라도 관리자는 Label Security 관리를 계속 진행 할 수 있다. 모든 테이블의 접근 중재가 해제되기 때문에 사용시 각별한 주의가 필요하다. 다시 정책의 시행을 활성화시키면, 정책에 의해 보호된 모든 테이블의 접근 중재가 다시 재작동된다.

DISABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_SYSDBA.DISABLE_POLICY
(
   policy_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.

- 예제

```
EXEC SA_SYSDBA.DISABLE_POLICY ('tls_pol');
```

64.2.4. DROP_POLICY

해당 Label Security 정책과 연결되어 있던 유저 레이블, 데이터 레이블들을 데이터베이스에서 제거된다. 이 프러시저를 호출하면 해당 정책과 연결되어 있는 모든 것을 시스템에서 제거할 수 있다. 옵션을 통해 정책에 의해 보호되는 모든 테이블들의 레이블 열을 제거할 수 있다. 제거 전에 정책을 중지시킬 필요는 없다.

DROP_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_SYSDBA.DROP_POLICY
(
   policy_name IN VARCHAR2,
   drop_column  BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
policy_name	제거할 Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
drop_column	TRUE이면 보호되는 모든 테이블의 레이블 열들을 제거한다.

- 예제

```
EXEC SA_SYSDBA.DROP_POLICY ('tls_pol');
```

64.2.5. ENABLE_POLICY

정책에 의해 보호되는 테이블에 대한 액세스 제어를 시행한다. 정책은 생성될 때 자동으로 활성화된다. 생성 또는 활성화 후에는 정책으로 보호되는 테이블에 대한 모든 후속 액세스에 대해 정책이 적용된다.

ENABLE_POLICY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_SYSDBA.ENABLE_POLICY
(
   policy_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책과 해당 상태는 ALL_SA_POLICIES 뷰의 POLICY_NAME, STATUS 열에서 찾을 수 있다.

- 예제

```
EXEC SA_SYSDBA.ENABLE_POLICY('tls_pol');
```

제65장 SA_USER_ADMIN

본 장에서는 SA_USER_ADMIN 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

65.1. 개요

SA_USER_ADMIN 는 Tibero Label Security 기능의 사용자 레이블들을 관리하는 패키지이다.

65.2. 프러시저

본 절에서는 SA_USER_ADMIN 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

65.2.1. DROP_ALL_COMPARTMENTS

해당 Label Security 정책에서 해당 유저에게 부여된 모든 구획 레이블을 제거한다.

DROP_ALL_COMPARTMENTS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.DROP_ALL_COMPARTMENTS
(
 policy_name  IN VARCHAR2,
 user_name    IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
user_name	구획 레이블을 가진 유저의 이름이다. 저장된 유저는 ALL_SA_USER_COMPARTMENTS 뷰의 USER_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
 SA_USER_ADMIN.DROP_ALL_COMPARTMENTS (
  policy_name    => 'tls_pol',
  user_name      => 'user1');
END;
/
```

65.2.2. DROP_ALL_GROUPS

해당 Label Security 정책에서 해당 유저에게 부여된 모든 그룹 레이블을 제거한다.

DROP_ALL_GROUPS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.DROP_ALL_GROUPS
(
  policy_name IN VARCHAR2,
  user_name   IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
user_name	그룹 레이블을 가진 유저의 이름이다. 저장된 유저는 ALL_SA_USER_GROUPS 뷰의 USER_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
 SA_USER_ADMIN.DROP_ALL_GROUPS (
  policy_name    => 'tls_pol',
  user_name      => 'user1');
END;
/
```

65.2.3. DROP_COMPARTMENTS

해당 Label Security 정책에서 해당 유저에게 부여된 해당 구획 레이블을 제거한다.

DROP_COMPARTMENTS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.DROP_COMPARTMENTS
(
  policy_name     IN VARCHAR2,
  user_name       IN VARCHAR2,
  comps           IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
user_name	해당 구획 레이블을 가진 유저의 이름이다. 저장된 유저는 ALL_SA_USER_COMPARTMENTS 뷰의 USER_NAME 열에서 찾을 수 있다.
comps	제거할 구획 레이블들을 콤마(,)로 구분한 문자열이다. 해당 정책의 모든 comp는 ALL_SA_USER_COMPARTMENTS 뷰의 POLICY_NAME, COMP 열에서 찾을 수 있다.

- 예제

```
BEGIN
 SA_USER_ADMIN.DROP_COMPARTMENTS (
  policy_name    => 'tls_pol',
  user_name      => 'user1',
  comps          => 'HR');
END;
/
```

65.2.4. DROP_GROUPS

해당 Label Security 정책에서 해당 유저에게 부여된 해당 그룹 레이블을 제거한다.

DROP_GROUPS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.DROP_GROUPS
(
  policy_name IN VARCHAR2,
  user_name   IN VARCHAR2,
  groups      IN VARCHAR2
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
drop_column	해당 그룹 레이블을 가진 유저의 이름이다. 저장된 유저는 ALL_SA_USER_GROUPS 뷰의 USER_NAME 열에서 찾을 수 있다.
groups	제거할 그룹 레이블들을 콤마(,)로 구분한 문자열이다. 저장된 그룹은 ALL_SA_GROUPS 뷰의 SHORT_NAME 열에서 찾을 수 있다.

- 예제

```
BEGIN
 SA_USER_ADMIN.DROP_GROUPS (
  policy_name    => 'tls_pol',
  user_name      => 'user1',
  groups         => 'TM');
END;
/
```

65.2.5. SET_COMPARTMENTS

해당 유저에게 구획 레이블을 부여하고 유저 세션 레이블과 행 레이블의 기본 값을 설정한다.

모든 유저는 구획 레이블을 부여하기 전에 레벨 레이블이 설정되어야 한다. 쓰기 구획 레이블은 읽기 구획 레이블의 부분 집합이어야 한다(쓰기 구획 레이블은 유저가 쓰기 액세스 권한을 갖는 구획 레이블을 의미한다).

SET_COMPARTMENTS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.SET_COMPARTMENTS
(
  policy_name   IN VARCHAR2,
  user_name     IN VARCHAR2,
  read_comps    IN VARCHAR2,
  write_comps   IN VARCHAR2 DEFAULT NULL,
  def_comps     IN VARCHAR2 DEFAULT NULL,
  row_comps     IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
user_name	구획 레이블을 부여할 유저 이름이다.
read_comps	읽기 액세스 권한을 명시하는 구획 레이블들의 짧은 이름을 콤마(,)로 구분한 문자열이다. 모든 구획 레이블들은 ALL_SA_COMPARTMENTS 뷰의 SHORT_NAME에서 찾을 수 있다.
write_comps	쓰기 액세스 권한을 명시하는 구획 레이블들의 짧은 이름을 콤마(,)로 구분한 문자열이다. read_comps의 부분 집합이여야 한다. NULL이면 read_comps 값으로 설정된다.
def_comps	기본 구획 레이블의 짧은 이름이다. read_comps의 부분 집합이여야 한다. NULL이면 read_comps 값으로 설정된다.
row_comps	행 구획 레이블의 짧은 이름이다. write_comps와 def_comps의 부분 집합이여야 한다. NULL이면 def_comps 값을 설정된다.

- 예제

```
BEGIN
 SA_USER_ADMIN.SET_COMPARTMENTS (
  policy_name   => 'tls_pol',
  user_name     => 'user1',
  read_comps    => 'FIN',
  write_comps   => 'FIN',
  def_comps     => 'FIN',
  row_comps     => 'FIN');
```

```
END;
/
```

65.2.6. SET_GROUPS

해당 유저에게 그룹 레이블을 부여하고 유저 세션 레이블과 행 레이블의 기본 값을 설정한다.

모든 유저는 그룹 레이블을 부여하기 전에 레벨 레이블이 설정되어야 한다. 유저의 레벨 레이블 설정 정보는 ALL_SA_USER_LEVELS에서 찾을 수 있다.

SET_GROUPS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.SET_GROUPS
(
  policy_name      IN VARCHAR2,
  user_name        IN VARCHAR2,
  read_groups      IN VARCHAR2,
  write_groups     IN VARCHAR2 DEFAULT NULL,
  def_group        IN VARCHAR2 DEFAULT NULL,
  row_groups       IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.
user_name	그룹 레이블을 부여할 유저 이름이다. 그룹 레이블 부여가 처음이여야 하고 레벨 레이블 부여가 끝난 유저여야 한다. 레벨 레이블 부여가 끝난 유저는 ALL_SA_USER_LEVELS 뷰의 USER_NAME 열에서 찾을 수 있다.
read_groups	읽기 액세스 권한을 명시하는 그룹 레이블들의 짧은 이름을 콤마(,)로 구분한 문자열이다. 모든 그룹 레이블들은 ALL_SA_GROUPS 뷰의 SHORT_NAME에서 찾을 수 있다.
write_groups	쓰기 액세스 권한을 명시하는 그룹 레이블들의 짧은 이름을 콤마(,)로 구분한 문자열이다. read_groups의 부분 집합이여야 한다. NULL이면 read_groups 값으로 설정된다.

파라미터	설명
def_groups	기본 그룹 레이블의 짧은 이름이다. read_groups의 부분 집합이여야 한다. NULL이면 read_groups 값으로 설정된다.
row_groups	행 그룹 레이블의 짧은 이름이다. write_groups와 def_groups의 부분 집합이여야 한다. NULL이면 def_groups 값을 설정된다.

- 예제

```
BEGIN
 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'tls_pol',
  user_name      => 'user1',
  read_groups    => 'TM_FIN',
  write_groups   => 'TM_FIN',
  def_groups     => 'TM_FIN',
  row_groups     => 'TM_FIN');
END;
/
```

65.2.7. SET_LEVELS

해당 유저에게 최소, 최고 레벨을 부여하고 유저 세션 레이블과 행 레이블의 기본 값을 설정한다.

SET_LEVELS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SA_USER_ADMIN.SET_LEVELS
(
   policy_name      IN VARCHAR2,
   user_name        IN VARCHAR2,
   max_level        IN VARCHAR2,
   min_level        IN VARCHAR2 DEFAULT NULL,
   def_level        IN VARCHAR2 DEFAULT NULL,
   row_level        IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
policy_name	Label Security 정책의 이름이다. 저장된 정책은 ALL_SA_POLICIES 뷰의 POLICY_NAME 열에서 찾을 수 있다.

파라미터	설명
user_name	레벨 레이블을 부여할 유저 이름이다.
max_level	읽기, 쓰기 액세스 권한의 가능한 최고 레벨의 짧은 이름이다. 저장된 레벨은 ALL_SA_LEVELS 뷰의 SHORT_NAME 열에서 찾을 수 있다.
min_level	쓰기 액세스 권한의 가능한 최소 레벨의 짧은 이름이다. NULL이면 기본 값은 해당 정책의 가장 낮은 레벨이다.
def_level	기본 레벨 값이다. min_level 이상, max_level 이하여야 한다. 짧은 이름만 가능하다. NULL이면 기본 값은 max_level이다.
row_level	행 레벨 값이다. min_level 이상, def_level 이하여야 한다. 짧은 이름만 가능하다. NULL이면 def_level로 설정된다.

- 예제

```
BEGIN
  SA_USER_ADMIN.SET_LEVELS (
   policy_name     => 'tls_pol',
   user_name       => 'user1',
   max_level       => 'HS',
   min_level       => 'S');
END;
/
```

제66장 TEXT_DDL

본 장에서는 TEXT_DDL 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

66.1. 개요

TEXT_DDL은 TEXT INDEX를 사용할 때 TEXT INDEX의 설정들을 변경하기 위한 패키지이다.

참고

TEXT INDEX 사용방법에 대한 자세한 내용은 "Tibero TEXT 참조 안내서"를 참고한다.

66.2. 프러시저

본 절에서는 TEXT_DDL 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

66.2.1. ADD_STOPWORD

STOPWORD 타입에 새로운 Stop word를 추가하기 위한 프러시저이다.

ADD_STOPWORD 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
ADD_STOPWORD
(
    stoplist_name        IN VARCHAR2,
    stopword             IN VARCHAR2,
    lang                 IN VARCHAR2 DEFAULT 'ALL'
)
```

- 파라미터

파라미터	설명
stoplist_name	추가할 대상 STOPLIST 이름이다.
stopword	추가할 STOPWORD이다.
lang	STOPWORD가 적용될 언어이다. (현재 적용되지 않음)

- 예제

```
SQL>EXEC TEXT_DDL.ADD_STOPWORD('DEFAULT_STOPLIST','test_word');
```

66.2.2. CREATE_PREFERENCE

새로운 PREFERENCE를 만들기 위한 프러시저이다.

CREATE_PREFERENCE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
CREATE_PREFERENCE
(
    udef_pref_name       IN  VARCHAR2,
    pdef_pref_name       IN  VARCHAR2
)
```

- 파라미터

파라미터	설명
udef_pref_name	사용자가 정의할 PREFERENCE 이름이다.
pdef_pref_name	Tibero TEXT에서 정의된 PREFERENCE 이름이다.

- 예제

```
SQL>EXEC TEXT_DDL.CREATE_PREFERENCE('TEST1','BASIC_WORDLIST');
```

66.2.3. CREATE_STOPLIST

새로운 STOPLIST을 만들기 위한 프러시저이다.

CREATE_STOPLIST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
CREATE_STOPLIST
(
    stoplist_name        IN  VARCHAR2,
    stoplist_type        IN  VARCHAR2 DEFAULT 'DEFAULT_STOPLIST'
)
```

- 파라미터

파라미터	설명
stoplist_name	생성할 STOPLIST 이름이다.
stoplist_type	생성할 STOPLIST의 타입이다. 현재 DEFAULT_STOPLIST와 KOREAN_STOPLIST가 있다.

- 예제

```
SQL>EXEC TEXT_DDL.CREATE_STOPLIST('new_stoplist', 'DEFAULT_STOPLIST');
```

66.2.4. DROP_PREFERENCE

생성한 PREFERENCE를 삭제하기 위한 프러시저이다.

DROP_PREFERENCE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DROP_PREFERENCE
(
    udef_pref_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
udef_pref_name	삭제할 PREFERENCE 이름이다.

- 예제

```
SQL>EXEC TEXT_DDL.DROP_PREFERENCE('TEST1');
```

66.2.5. DROP_STOPLIST

STOPLIST를 삭제하기 위한 프러시저이다.

DROP_STOPLIST 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
DROP_STOPLIST
(
    stoplist_name IN VARCHAR2
);
```

- 파라미터

파라미터	설명
stoplist_name	삭제할 STOPLIST 이름이다.

- 예제

```
SQL>EXEC TEXT_DDL.DROP_PREFERENCE('new_stoplist');
```

66.2.6. REMOVE_STOPWORD

STOPLIST에서 STOPWORD를 삭제하기 위한 프러시저이다.

REMOVE_STOPWORD 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
REMOVE_STOPWORD
(
    stoplist_name        IN VARCHAR2,
    stopword             IN VARCHAR2,
    lang                 IN VARCHAR2 DEFAULT 'ALL'
)
```

- 파라미터

파라미터	설명
stoplist_name	삭제할 대상 STOPLIST 이름이다.
stopword	삭제할 STOPWORD이다.
lang	STOPWORD가 적용되는 언어이다. (현재 적용되지 않음)

- 예제

```
SQL>EXEC TEXT_DDL.REMOVE_STOPWORD('DEFAULT_STOPLIST','test_word');
```

66.2.7. SET_ATTRIBUTE

생성된 PREFERENCE의 속성값을 설정하기 위한 프러시저이다.

SET_ATTRIBUTE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
SET_ATTRIBUTE
(
    udef_pref_name       IN  VARCHAR2,
    attribute_name       IN  VARCHAR2,
    value                IN  VARCHAR2
)
```

- 파라미터

파라미터	설명
udef_pref_name	속성값을 변경할 사용자가 정의한 PREFERENCE이다.
attribute_name	변경할 대상 속성이다.
value	변경할 값이다.

- 예제

```
SQL>EXEC TEXT_DDL.SET_ATTRIBUTE('TEST1','PREFIX_INDEX','TRUE');
```

제67장 UTL_COMPRESS

본 장에서는 UTL_COMPRESS 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

67.1. 개요

UTL_COMPRESS은 BLOB이나 RAW데이터에 대해서 압축된 결과 혹은 압축 해제된 결과를 동일 타입으로 반환하는 패키지이다.

67.2. 함수

본 절에서는 UTL_COMPRESS 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

67.2.1. LZ_COMPRESS

데이터를 압축하여 반환하는 함수이다. BLOB 타입을 입력으로 받으면 BLOB 타입을, RAW 타입을 입력으로 받으면 RAW 타입을 반환한다.

LZ_COMPRESS 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB인 경우

```
UTL_COMPRESS.LZ_COMPRESS
(
    src     IN BLOB,
    quality IN BINARY_INTEGER DEFAULT 6
)
RETURN BLOB;
```

  - RAW인 경우

```
UTL_COMPRESS.LZ_COMPRESS
(
    src     IN RAW,
    quality IN BINARY_INTEGER DEFAULT 6
)
RETURN RAW;
```

- 파라미터

파라미터	설명
src	CLOB이나 RAW 타입의 압축할 데이터이다.
quality	압축률을 나타낸다. 1부터 9까지 설정할 수 있으며 숫자가 커질수록 압축률이 좋아진다. (기본값: 6)

- 예제

```
DECLARE
    data RAW(500);
BEGIN
    data := UTL_COMPRESS.LZ_COMPRESS('A1B2C3D4E5F6', 6);

    DBMS_OUTPUT.PUT_LINE(data);
END;
/
```

67.2.2. LZ_UNCOMPRESS

압축된 데이터를 압축 해제하여 반환하는 함수이다. BLOB 타입을 입력으로 받으면 BLOB 타입을, RAW 타입을 입력으로 받으면 RAW 타입을 반환한다.

LZ_UNCOMPRESS 함수의 세부 내용은 다음과 같다.

- 프로토타입
  - BLOB인 경우

```
UTL_COMPRESS.LZ_UNCOMPRESS
(
    src     IN BLOB
)
RETURN BLOB;
```

  - RAW인 경우

```
UTL_COMPRESS.LZ_UNCOMPRESS
(
    src     IN RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
src	CLOB이나 RAW 타입의 압축을 풀 데이터이다.

- 예제

```
DECLARE
    comp_data RAW(500);
    real_data RAW(500);
BEGIN
    comp_data := UTL_COMPRESS.LZ_COMPRESS('A1B2C3D4E5F6', 6);
    real_data := UTL_COMPRESS.LZ_UNCOMPRESS(comp_data);

    DBMS_OUTPUT.PUT_LINE(real_data);
END;
/
```

제68장 UTL_ENCODE

본 장에서는 UTL_ENCODE 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

68.1. 개요

UTL_ENCODE는 호스트 간의 데이터를 전송할 수 있도록 표준 인코딩 기술로 인코딩하는 함수를 제공하는 패키지이다.

68.2. 함수

본 절에서는 UTL_ENCODE 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

68.2.1. BASE64_DECODE

BASE64 형태로 인코딩된 RAW 타입의 문자열을 원래의 이진 값으로 변환하는 함수이다.

BASE64_DECODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION BASE64_DECODE
(
    r  IN  RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r	원본 테이블과 새 테이블이 속한 스키마의 이름이다.

- 예제

```
DECLARE
    org raw(100);
    enc raw(100);
BEGIN
    org := 'aaabae64ecde';
    enc := UTL_ENCODE.BASE64_ENCODE(org);
```

```
    ...
    org := UTL_ENCODE.BASE64_DECODE(enc);
    ...
END;
/
```

68.2.2. BASE64_ENCODE

RAW 타입의 이진 값을 BASE64 형태로 인코딩된 RAW 타입의 문자열로 변환하는 함수이다.

BASE64_ENCODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION BASE64_ENCODE
(
    r  IN  RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r	인코딩할 RAW 타입의 문자열이다.

- 예제

```
DECLARE
    org raw(100);
    enc raw(100);
BEGIN
    org := 'aaabae64ecde';
    enc := UTL_ENCODE.BASE64_ENCODE(org);
    ...
END;
/
```

68.2.3. QUOTED_PRINTABLE_DECODE

Quoted-printable 형식으로 인코딩된 문자열을 읽어들여서 디코딩하는 함수이다.

QUOTED_PRINTABLE_DECODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION QUOTED_PRINTABLE_DECODE
(
    r               IN  RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r	디코딩할 RAW 타입의 문자열이다.

- 반환값

반환값	설명
RAW	디코딩된 RAW 타입의 문자열이다.

- 예제

```
DECLARE
    src VARCHAR2(32767);
    tar RAW(32767);
    dec RAW(32767);
BEGIN
    src := 'tib=ero';
    tar := UTL_ENCODE.QUOTED_PRINTABLE_ENCODE(UTL_RAW.CAST_TO_RAW(src));
    dec := UTL_ENCODE.QUOTED_PRINTABLE_DECODE(tar);
    DBMS_OUTPUT.PUT_LINE('ENCODED DATA :' || UTL_RAW.CAST_TO_VARCHAR2(tar));
    DBMS_OUTPUT.PUT_LINE('DECODED DATA :' || UTL_RAW.CAST_TO_VARCHAR2(dec));
END;
/
```

68.2.4. QUOTED_PRINTABLE_ENCODE

RAW 타입의 입력 문자열을 Quoted-printable 형식의 문자열로 인코딩하는 함수이다.

QUOTED_PRINTABLE_ENCODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION QUOTED_PRINTABLE_ENCODE
(
    r               IN  RAW
```

```
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r	인코딩할 RAW 타입의 문자열이다.

- 반환값

반환값	설명
RAW	인코딩된 RAW 타입의 문자열이다.

- 예제

```
DECLARE
    src VARCHAR2(32767);
    tar RAW(32767);
    dec RAW(32767);
BEGIN
    src := 'tib=ero';
    tar := UTL_ENCODE.QUOTED_PRINTABLE_ENCODE(UTL_RAW.CAST_TO_RAW(src));
    dec := UTL_ENCODE.QUOTED_PRINTABLE_DECODE(tar);
    DBMS_OUTPUT.PUT_LINE('ENCODED DATA :' || UTL_RAW.CAST_TO_VARCHAR2(tar));
    DBMS_OUTPUT.PUT_LINE('DECODED DATA :' || UTL_RAW.CAST_TO_VARCHAR2(dec));
END;
/
```

68.2.5. TEXT_DECODE

문자 집합을 갖는 문자열을 디코딩하는 함수이다. 디코딩 후 변경할 문자 집합에 따라 텍스트를 변환한다.

TEXT_DECODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION TEXT_DECODE
(
    buf              IN  VARCHAR2,
    encode_charset   IN  VARCHAR2    DEFAULT NULL,
    encoding         IN  PLS_INTEGER DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
buf	디코딩할 문자열이다.
encode_charset	변경할 문자 집합을 지정한다. 기본값은 NULL이며, NULL인 경우 문자 집합을 변경하지 않는다. 데이터베이스 문자 집합에서 파라미터에 지정한 문자 집합으로 문자 집합을 변환한다.
encoding	디코딩할 타입을 지정한다. – 1 : BASE64 방식을 디코딩한다. – 2 : Quoted-printable 방식을 디코딩한다. 기본값은 NULL이며 Quoted-printable 방식을 디코딩한다.

- 예제

```
DECLARE
    src VARCHAR2(32767);
    tar VARCHAR2(32767);
    dec VARCHAR2(32767);
BEGIN
    src := 'tibero';
    tar := UTL_ENCODE.TEXT_ENCODE(src, NULL, UTL_ENCODE.BASE64);
    dec := UTL_ENCODE.TEXT_DECODE(tar, NULL, UTL_ENCODE.BASE64);
    DBMS_OUTPUT.PUT_LINE('ENCODED DATA :' || tar);
    DBMS_OUTPUT.PUT_LINE('DECODED DATA :' || dec);
END;
/
```

68.2.6. TEXT_ENCODE

문자 집합을 갖는 문자열을 인코딩하는 함수이다. 변경할 문자 집합에 따라 텍스트를 변환하고 인코딩한다.

TEXT_ENCODE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION TEXT_ENCODE
(
    buf              IN  VARCHAR2,
    encode_charset   IN  VARCHAR2    DEFAULT NULL,
    encoding         IN  PLS_INTEGER DEFAULT NULL
```

```
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
buf	인코딩할 문자열이다.
encode_charset	변경할 문자 집합을 지정한다. 단, 문자 집합을 변경하는 기능은 다음에 구현할 예정이므로 기본값인 NULL로 지정한다.
encoding	인코딩할 타입을 지정한다. – 1 : BASE64 방식으로 인코딩한다. – 2 : Quoted-printable 방식으로 인코딩한다. 기본값은 NULL이며 Quoted-printable 방식으로 인코딩한다.

- 예제

```
DECLARE
    src VARCHAR2(32767);
    tar VARCHAR2(32767);
BEGIN
    src := 'tibero';
    tar := UTL_ENCODE.TEXT_ENCODE(src, NULL, 1);
    ...
END;
/
```

제69장 UTL_FILE

본 장에서는 UTL_FILE 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

69.1. 개요

UTL_FILE은 운영체제에서 관리하는 파일에 접근하기 위한 함수와 프러시저를 제공하는 패키지이다. 파일의 경로는 디렉터리명으로 지정할 수 있다.

UTL_FILE 패키지에서 정의된 타입과 예외 상황은 다음과 같다.

- 타입

  FILE_TYPE의 타입은 파일 식별자로 사용된다.

```
TYPE FILE_TYPE IS RECORD
(
    id       BINARY_INTEGER,
    datatype BINARY_INTEGER
);
```

- 예외 상황

예외 상황	설명
INVALID_PATH	파일의 경로가 유효하지 않은 경우이다.
INVALID_MODE	유효하지 않은 모드를 사용한 경우이다.
INVALID_FILEHANDLE	유효하지 않은 파일 핸들에 파일 연산을 하는 경우이다.
INVALID_OPERATION	파일에 대한 연산을 수행할 수 없는 경우이다.
READ_ERROR	읽기 연산 중에 에러가 발생하는 경우이다.
WRITE_ERROR	쓰기 연산 중에 에러가 발생하는 경우이다.
INTERNAL_ERROR	예기치 못한 에러가 발생하는 경우이다.
FILE_OPEN	파일이 열려 있어서 해당 연산을 수행할 수 없는 경우이다.
INVALID_MAXLINESIZE	FOPEN 함수를 사용할 때 max_linesize 값이 유효하지 않은 경우이다. max_linesize 값은 1과 32767bytes 사이의 값이어야 한다.
INVALID_FILENAME	파일 이름이 유효하지 않은 경우이다.
ACCESS_DENIED	해당 파일에 대한 접근 권한이 없는 경우이다.
INVALID_OFFSET	다음과 같이 유효하지 않은 오프셋이 주어진 경우이다.

예외 상황	설명
	– 상대 오프셋과 절대 오프셋이 모두 NULL일 때 유효하지 않다. – 절대 오프셋이 음수일 때 유효하지 않다. – 오프셋이 파일 끝의 위치를 초과했을 때 유효하지 않다.
DELETE_FAILED	DELETE 연산이 실패한 경우이다.
RENAME_FAILED	RENAME 연산이 실패한 경우이다.

69.2. 프러시저

본 절에서는 UTL_FILE 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

69.2.1. FCLOSE

파일을 닫는 프러시저이다.

FCLOSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FCLOSE
(
   file IN OUT FILE_TYPE
);
```

- 파라미터

파라미터	설명
file	파일 핸들이다.

- 예외 상황
  - WRITE_ERROR
  - INVALID_FILEHANDLE
- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/HOME/TIBERO/PATH';

DECLARE
  fname   VARCHAR2(1024);
BEGIN
```

```
    file := UTL_FILE.FOPEN('USER_PATH', 'MYFILE.DAT', 'r');
    UTL_FILE.FCLOSE(file);
END;
/
```

69.2.2. FCLOSE_ALL

현재 세션에 열린 모든 파일을 닫는 프러시저이다.

FCLOSE_ALL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FCLOSE_ALL;
```

- 예외 상황
  - WRITE_ERROR

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/HOME/TIBERO/PATH';

DECLARE
    outfile   UTL_FILE.FILE_TYPE;
    appfile   UTL_FILE.FILE_TYPE;
    path       VARCHAR(10);
BEGIN
    path := 'USER_PATH';
    outfile := UTL_FILE.FOPEN(path, 'output_file.txt', 'w');
    appfile := UTL_FILE.FOPEN(path, 'append_file.txt', 'w');

    UTL_FILE.FCLOSE_ALL;
END;
/
```

69.2.3. FCOPY

파일의 연속되는 부분을 새로 생성된 파일에 복사하는 프러시저이다.

FCOPY 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FCOPY
(
    location   IN VARCHAR2,
    filename   IN VARCHAR2,
    dest_dir   IN VARCHAR2,
    dest_file  IN VARCHAR2,
    start_line IN BINARY_INTEGER DEFAULT 1,
    end_line   IN BINARY_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
location	복사를 진행할 대상 파일이 위치할 경로의 디렉터리명이다.
filename	복사를 진행할 대상 파일의 이름이다.
dest_dir	복사가 진행된 이후 파일이 위치할 경로의 디렉터리명이다.
dest_file	복사가 진행된 이후의 파일의 이름이다.
start_line	복사를 진행할 시작 위치이다. 파일의 라인 번호를 사용해 명시한다. (기본값: 1, 파일의 첫 번째 의미이다.)
end_line	복사를 끝낼 종료 위치이다. start_line과 마찬가지로 라인 번호로 명시한다. (기본값: NULL, 파일의 마지막 라인을 의미한다.)

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/HOME/TIBERO/PATH';

BEGIN
    UTL_FILE.FCOPY('USER_PATH', 'MYFILE.DAT', 'USER_PATH', 'MTFILE.BAK');
END;
/
```

69.2.4. FFLUSH

아직 파일에 쓰지 않고 버퍼에 남아 있는 데이터를 파일에 쓰는 프러시저이다. 데이터는 반드시 EOL 문자로 끝나야 한다.

FFLUSH 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FFLUSH
(
    file IN FILE_TYPE
);
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.

- 예외 상황
  - INVALID_FILEHANDLE
  - INVALID_OPERATION
  - WRITE_ERROR
- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';
DECLARE
  fhandle  UTL_FILE.FILE_TYPE;
  buffer   VARCHAR2(32767);
  path     VARCHAR2(1024);
  fname    VARCHAR2(1024);
BEGIN
  path     := 'USER_PATH';
  fname    :=  'MYFILE.TXT';
  fhandle  := UTL_FILE.FOPEN(path, fname, 'w');
  buffer   := 'This is the message for output file';
  for i in 1..10 loop
    UTL_FILE.PUT(fhandle, buffer);
  end loop;
  UTL_FILE.NEW_LINE(fhandle);
  UTL_FILE.FFLUSH(fhandle);
  UTL_FILE.FCLOSE(fhandle);
END;
/
```

69.2.5. FGETATTR

디스크 파일의 속성을 반환하는 프러시저이다.

FGETATTR 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FGETATTR
(
    location IN VARCHAR2,
    filename IN VARCHAR2,
    fexists  OUT BOOLEAN,
    file_length OUT NUMBER,
    blocksize OUT BINARY_INTEGER
);
```

- 파라미터

파라미터	설명
location	파일의 경로를 나타내는 디렉터리명이다.
filename	파일의 이름이다.
fexists	파일의 존재 여부이다.
file_length	파일의 길이(Byte)이다.
blocksize	파일 시스템의 블록 크기(Byte)이다.

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
  info_exists  BOOLEAN;
  info_flen    NUMBER;
  info_bsz     BINARY_INTEGER;
  etc_path     VARCHAR2(1024);
  fname        VARCHAR2(1024);
BEGIN
  etc_path := 'USER_PATH';
  fname    := 'hostname';
  UTL_FILE.FGETATTR(etc_path, fname, info_exists, info_flen, info_bsz);

  if info_exists = TRUE then
    DBMS_OUTPUT.PUT_LINE('file name         :' || fname);
    DBMS_OUTPUT.PUT_LINE('file size (bytes) :' || TO_CHAR(info_flen));
    DBMS_OUTPUT.PUT_LINE('block size(bytes) :' || TO_CHAR(info_bsz));
  end if;
END;
/
```

69.2.6. FREMOVE

파일을 삭제하는 프러시저이다. 파일을 삭제하는 데 충분한 권한이 없는 경우 DELETE_FAILED 예외 상황이 발생한다.

FREMOVE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FREMOVE
(
    location IN VARCHAR2,
    filename IN VARCHAR2
);
```

- 파라미터

파라미터	설명
location	파일의 위치를 나타내는 디렉터리명이다.
filename	파일의 이름이다.

- 예외 상황
  - DELETE_FAILED

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

BEGIN
    UTL_FILE.FREMOVE('USER_PATH', 'MYFILE.BAK');
END;
/
```

69.2.7. FRENAME

파일의 이름을 변경하는 프러시저이다. OS 셸에서 사용하는 mv 명령어와 동일하게 동작한다.

FRENAME 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FRENAME
(
    location   IN VARCHAR2,
```

```
    filename   IN VARCHAR2,
    dest_dir   IN VARCHAR2,
    dest_file  IN VARVCHAR2,
    overwrite  IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
location	원본 파일의 경로를 나타내는 디렉터리명이다.
filename	원본 파일의 이름이다.
dest_dir	변경된 파일의 경로를 나타내는 디렉터리명이다.
dest_file	변경된 파일의 이름이다.
overwrite	변경된 파일의 이름이 이미 존재할 경우 덮어쓸지를 지정한다.

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';
BEGIN
    UTL_FILE.FRENAME('USER_PATH', 'MTFILE.BAK', 'USER_PATH', 'MYFILE.DAT');
END;
/
```

69.2.8. FSEEK

파일 포인터의 위치를 변경하는 프러시저이다.

FSEEK 프러시저의 세부내용은 다음과 같다.

- 프로토타입

```
PROCEDURE FSEEK
(
    fid IN UTL_FILE.FILE_TYPE,
    absolute_offset IN PLS_INTEGER DEFAULT NULL,
    relative_offset IN PLS_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
fid	UTL_FILE 패키지의 FILE_TYPE 타입의 파일 식별자이다.
absolute_offset	절대 오프셋, 파일의 시작부터의 오프셋 값이다. (기본값: NULL)

파라미터	설명
relative_offset	상대 오프셋, 현재 파일 포인터 위치에서 양수이면 뒤쪽, 음수이면 앞쪽 방향을 나타낸다. 절대 오프셋이 있는 경우 상대 오프셋은 무시 된다. (기본값: NULL)

- 예외 상황
  - INVALID_OFFSET

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
  infile          UTL_FILE.FILE_TYPE;
  absolute_offset PLS_INTEGER:= 10;
  scan_first      RAW(32767);
  scan_second     RAW(32767);
  len CONSTANT    PLS_INTEGER := 32767;
BEGIN
  infile := UTL_FILE.FOPEN('USER_PATH','codd.bcnf', 'r');

  UTL_FILE.FSEEK(infile, absolute_offset, -5);
  UTL_FILE.GET_RAW(infile, scan_first, 1);
  UTL_FILE.FSEEK(infile, 5);
  UTL_FILE.GET_RAW(infile, scan_second, 1);

  if  scan_first != scan_second then
    DBMS_OUTPUT.PUT_LINE('ralative_offset ignored!');
  end if;

  UTL_FILE.FCLOSE(infile);
END;
/
```

69.2.9. GET_LINE

EOL 문자 또는 파일이 끝날 때까지의 내용을 읽어 오는 프러시저이다. FOPEN 함수에서 지정한 max_linesize 이상을 읽어 올 수 없다.

GET_LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE GET_LINE
(
    file   IN  FILE_TYPE,
    buffer OUT VARCHAR2,
    len    IN  PLS_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.
buffer	읽어 온 데이터이다.
len	파일로부터 읽어 올 Byte의 크기이다. (기본값: NULL, NULL인 경우 max_linesize 값을 갖는다.)

- 예외상황
  - INVALID_FILEHANDLE
  - INVALID_OPERATION
- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
    infile       UTL_FILE.FILE_TYPE;
    length       CONSTANT PLS_INTEGER := 1024;
    read_buffer  VARCHAR2(1024) := NULL;
BEGIN
    infile := UTL_FILE.FOPEN('USER_PATH','hosts', 'r');
    loop
        UTL_FILE.GET_LINE(infile, read_buffer, length);
        DBMS_OUTPUT.PUT_LINE(read_buffer);
        if read_buffer is NULL then
             DBMS_OUTPUT.PUT_LINE(read_buffer);
        else
            exit;
        end if;
    end loop;

    UTL_FILE.FCLOSE(infile);
END;
/
```

69.2.10. GET_RAW

파일로부터 RAW 타입의 문자열을 읽어 오는 프러시저이다. EOL 문자를 무시하며, 읽은 Byte의 크기를 반환한다.

GET_RAW 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE GET_RAW
(
    fid  IN  UTL_FILE.FILE_TYPE,
    r    OUT NOCOPY RAW,
    len  IN  PLS_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
fid	파일 식별자이다.
r	읽은 데이터이다.
len	파일로부터 읽어 올 Byte의 크기이다. (기본값: NULL, NULL인 경우 max_linesize 값을 갖는다.)

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
  infile       UTL_FILE.FILE_TYPE;
  length       CONSTANT PLS_INTEGER := 32;
  read_buffer  RAW(32767);
BEGIN
  infile := UTL_FILE.FOPEN('USER_PATH','hosts', 'r');
  UTL_FILE.GET_RAW(infile, read_buffer, length);
  UTL_FILE.FCLOSE(infile);
END;
/
```

69.2.11. NEW_LINE

파일에 하나 이상의 EOL 문자를 넣는 프러시저이다. EOL 문자는 플랫폼에 따라 다르다.

NEW_LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE NEW_LINE
(
    file  IN FILE_TYPE,
    lines IN NATURAL DEFAULT 1
);
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.
lines	파일에 넣을 EOL 문자의 개수이다.

- 예외 상황
  - INVALID_FILEHANDLE
  - INVALID_OPERATION
  - WRITE_ERROR

69.2.12. PUT

파일에 문자열을 넣을 때 사용하는 프러시저이다.

PUT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE PUT
(
    file IN FILE_TYPE,
    buffer IN VARCHAR2
);
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.
buffer	파일에 쓸 내용이다.

파라미터	설명
	파일이 w(쓰기 텍스트)나 a(추가 텍스트) 모드로 열리지 않은 경우 INVALID_OPERATION 예외 상황이 발생한다.

- 예외 상황
  - INVALID_FILEHANDLE
  - INVALID_OPERATION
  - WRITE_ERROR
- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';
DECLARE
  outfile  UTL_FILE.FILE_TYPE;
  buf      VARCHAR(1024) := NULL;
BEGIN
  outfile := UTL_FILE.FOPEN('USER_PATH', 'MYFILE.DAT', 'w');

  for i in 1..10 loop
    buf := buf || '0123456789';
  end loop;

  UTL_FILE.PUT( outfile, buf );
  UTL_FILE.FCLOSE(outfile);
END;
/
```

69.2.13. PUTF

형식화된 PUT 프러시저이다. PUTF의 형식화된 문자열은 C언어에서 표준 입출력(standard I/O)의 fprintf에서의 형식화된 문자열과 동일하게 %s와 이스케이프 문자를 포함한다. 이스케이프 문자로 \n을 사용할 수 있으며 %s는 그 뒤에 나오는 인수 문자열을 대체한다.

예를 들어 다음과 같은 세 개의 인수가 있다고 가정하면,

```
arg1 = 'string1';
arg2 = 'string2';
arg3 = 'string3';
```

위의 세 개의 인수를 사용하는 형식화된 문자열은 다음과 같다.

```
utl_file.putf( ofile, 'This is example of formatted string : %s %s %s \n',
               arg1, arg2, arg3);
```

또한 위의 내용이 실행되면 다음과 같은 내용이 출력된다.

```
This is example of formated string : string1 string2 string3
```

PUTF 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCUDURE PUTF
(
    file   IN FILE_TYPE,
    format IN VARCHAR2,
    arg1   IN VARCHAR2 DEFAULT NULL,
    arg2   IN VARCHAR2 DEFAULT NULL,
    arg3   IN VARCHAR2 DEFAULT NULL,
    arg4   IN VARCHAR2 DEFAULT NULL,
    arg5   IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.
format	%s와 이스케이프 문자(\n)를 포함하는 형식화된 문자열이다.
arg1-5	%s에 대체되는 선택할 수 있는 문자열이다. (기본값: NULL)

- 예외 상황
  - INVALID_FILEHANDLE
  - INVALID_OPERATION
  - WRITE_ERROR
- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
    fname    VARCHAR2(2048);
    path     VARCHAR2(2048);
    outfile  UTL_FILE.FILE_TYPE;
BEGIN
    fname := 'MYFILE.DAT';
    path  := 'USER_PATH';
    outfile := UTL_FILE.FOPEN(path, fnamet /home/posung/path/MYFILE.DAT!cat
               /home/posung/path/MYFILE.DAT, 'w');
    UTL_FILE.PUTF( outfile, '%s %s formated file output example\n', path, fname);
```

```
    UTL_FILE.FCLOSE(outfile);
END;
/
```

69.2.14. PUT_RAW

RAW 타입 데이터의 내용을 파일에 쓰는 프러시저이다.

PUT_RAW 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE PUT_RAW
(
    fid       IN UTL_FILE.FILE_TYPE,
    r         IN RAW,
    autoflush IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
fid	파일 식별자이다.
r	파일에 쓸 데이터이다.
autoflush	쓰기를 한 후 출력 버퍼를 비울지 여부를 결정한다. – TRUE : 버퍼를 비운다. – FALSE : 기본값으로 버퍼를 비우지 않는다.

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
    outfile  UTL_FILE.FILE_TYPE;
    buf RAW(8) := HEXTORAW('6161616100616100');
BEGIN
    outfile := UTL_FILE.FOPEN('USER_PATH', 'MYFILE.DAT', 'wb');
    UTL_FILE.PUT_RAW(outfile, buf);
    UTL_FILE.FCLOSE(outfile);
END;
/
```

69.2.15. PUT_LINE

라인을 파일에 쓰는 프러시저이다. 이때 라인은 플랫폼에 종속적인 EOL 문자로 끝낸다.

PUT_LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE PUT_LINE
(
    file      IN FILE_TYPE,
    buffer    IN VARCHAR2,
    autoflush IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.
buffer	파일에 쓸 라인이다.
autoflush	쓰기를 한 후 출력 버퍼를 비울지 여부를 결정한다. – TRUE : 버퍼를 비운다. – FALSE : 기본값으로 버퍼를 비우지 않는다.

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
    outfile  UTL_FILE.FILE_TYPE;
    out_buf  VARCHAR2(1024);
BEGIN
    outfile := UTL_FILE.FOPEN('USER_PATH', 'MYFILE.DAT', 'w');
    for i in 1..10 loop
        UTL_FILE.PUT_LINE(outfile,  i || ' put_line is complete', TRUE);
    end loop;
    UTL_FILE.FCLOSE(outfile);
END;
/
```

69.3. 함수

본 절에서는 UTL_FILE 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

69.3.1. FGETPOS

파일 내에서 파일 포인터의 상대적인 위치(Byte)를 반환하는 함수이다.

FGETPOS 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION FGETPOS
(
    fid  IN FILE_TYPE
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
fid	파일 핸들러이다.

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
  file   UTL_FILE.FILE_TYPE;
  fpos   PLS_INTEGER := 0;
  path   VARCHAR2(1024);
  fname  VARCHAR2(1024);
BEGIN
    file := UTL_FILE.FOPEN('USER_PATH', 'MYFILE.DAT', 'r');
    UTL_FILE.FSEEK(file, 10, 1);
    fpos := UTL_FILE.FGETPOS(file);
    DBMS_OUTPUT.PUT_LINE('Offest is ' || TO_CHAR(fpos, 'S9999999999'));
    UTL_FILE.FCLOSE(file);
END;
/
```

69.3.2. FOPEN

파일을 여는 함수이다. 최대 50개의 파일을 동시에 열 수 있다.

FOPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION FOPEN
(
    location     IN VARCHAR2,
    filename     IN VARCHAR2,
    open_mode    IN VARCHAR2,
    max_linesize IN BINARY_INTEGER DEFAULT 1024
)
RETURN FILE_TYPE;
```

- 파라미터

파라미터	설명
location	파일의 경로를 나타내는 디렉터리명이다.
filename	파일의 이름이다.
open_mode	파일 모드이다. 파일 모드의 종류는 다음과 같다. – r : 읽기 텍스트 모드이다. – w : 쓰기 텍스트 모드이다. – a : 추가 텍스트 모드이다. – rb : 읽기 바이트 모드이다. – wb : 쓰기 바이트 모드이다. – ab : 추가 바이트 모드로, 추가 모드에서 파일이 존재하지 않으면 쓰기 모드로 파일을 생성한다.
max_linesize	각 라인에 포함되는 byte 수의 최댓값으로 newline 문자를 포함한다. 최소 1byte에서 최대 32767bytes이다. (기본값: 1024bytes)

- 예외 상황
  - INVALID_PATH
  - INVALID_MODE
  - INVALID_OPERATION
  - INVALID_MAXLINESIZE

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
    infile    UTL_FILE.FILE_TYPE;
    outfile   UTL_FILE.FILE_TYPE;
    appfile   UTL_FILE.FILE_TYPE;
    max_lsz   CONSTANT BINARY_INTEGER := 4096;
    buffer    VARCHAR2(1024);
    data1     VARCHAR2(1024);
    data2     VARCHAR2(1024);
    path      VARCHAR2(1024);
BEGIN
    path := 'USER_PATH';
    outfile := UTL_FILE.FOPEN(path, 'output_file.txt', 'w',  max_lsz);
    appfile := UTL_FILE.FOPEN(path, 'append_file.txt', 'w',  max_lsz);

    data1 := '0123456789';
    data2 := 'abcdefghij';
    for  i in 1..10 loop
        UTL_FILE.PUT_LINE(outfile, data1);
        UTL_FILE.PUT_LINE(outfile, data2);
    end loop;

    UTL_FILE.FCLOSE(outfile);
    UTL_FILE.FCLOSE(appfile);

    infile := UTL_FILE.FOPEN(path, 'output_file.txt', 'r',  max_lsz);
    UTL_FILE.GET_LINE(infile, buffer);
    DBMS_OUTPUT.PUT_LINE(buffer);
    UTL_FILE.FCLOSE(infile);
    appfile := UTL_FILE.FOPEN( path, 'append_file.txt', 'a',  max_lsz );
    UTL_FILE.PUT_LINE(appfile, 'appended');
    UTL_FILE.FCLOSE(appfile);
END;
/
```

69.3.3. IS_OPEN

파일 핸들러를 통해 파일이 열려 있는지 검사하는 함수이다. 파일이 열려있는 경우 TRUE를 반환하고, 그렇지 않은 경우 FALSE를 반환한다.

IS_OPEN 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE IS_OPEN
(
    file IN  FILE_TYPE
)
RETURN BOOLEAN;
```

- 파라미터

파라미터	설명
file	파일 핸들러이다.

- 예제

```
CREATE OR REPLACE DIRECTORY USER_PATH AS '/home/user/path';

DECLARE
    openfile    UTL_FILE.FILE_TYPE;
    open_flag   BOOLEAN;
    status      VARCHAR(10);
    fname       VARCHAR(10);
BEGIN
    fname := 'MYFILE.DAT';
    openfile := UTL_FILE.FOPEN('USER_PATH', fname, 'r');
    if UTL_FILE.IS_OPEN(openfile) then
        status := 'opened';
    else
        status := 'not opened';
    end if;

    DBMS_OUTPUT.PUT_LINE(fname || ' file status : ' || status);
    UTL_FILE.FCLOSE(openfile);
    dbms_output.put_line(fname || 'is closed');
    if utl_file.IS_OPEN(openfile) then
        status := 'opened';
    else
        status := 'not opened';
    end if;
    DBMS_OUTPUT.PUT_LINE(fname || ' file status : ' || status);
END;
/
```

제70장 UTL_HTTP

본 장에서는 HTTP/1.1을 기준으로 작성되었으며, UTL_HTTP 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

70.1. 개요

UTL_HTTP는 웹 표준(RFC2616)인 HTTP 프로토콜에 따라 클라이언트 역할을 제공하는 패키지이다.

프로토콜을 세부 조작하거나 간단히 웹 페이지만을 얻어올 수 있는 서브 프로그램을 제공한다.

70.2. 상수

70.2.1. HTTP 버전 상수

HTTP 버전의 종류를 나타낸다.

HTTP 버전 상수의 세부 내용은 다음과 같다.

```
HTTP_VERSION_1      CONSTANT VARCHAR2(10) := 'HTTP/1.1';
DEFAULT_HTTPS_PORT  CONSTANT PLS_INTEGER := 443;
```

70.2.2. HTTP 포트 상수

HTTP에서 사용할 네트워크 포트 번호를 나타낸다.

HTTP 포트 상수의 세부 내용은 다음과 같다.

```
DEFAULT_HTTP_PORT   CONSTANT PLS_INTEGER := 80;
DEFAULT_HTTPS_PORT  CONSTANT PLS_INTEGER := 443;
```

70.2.3. 상태 코드 상수

HTTP 프로토콜에 따라 서버로부터 응답받은 응답의 상태를 나타내는 코드이다.

상태 코드 상수의 세부 내용은 다음과 같다. 이때 상태 코드는 번호 순으로 정렬되어 있다.

```
HTTP_CONTINUE                   CONSTANT PLS_INTEGER := 100;
HTTP_SWITCHING_PROTOCOLS        CONSTANT PLS_INTEGER := 101;
HTTP_OK                         CONSTANT PLS_INTEGER := 200;
HTTP_CREATED                    CONSTANT PLS_INTEGER := 201;
HTTP_ACCEPTED                   CONSTANT PLS_INTEGER := 202;
HTTP_NON_AUTHORITATIVE_INFO     CONSTANT PLS_INTEGER := 203;
HTTP_NO_CONTENT                 CONSTANT PLS_INTEGER := 204;
HTTP_RESET_CONTENT              CONSTANT PLS_INTEGER := 205;
HTTP_PARTIAL_CONTENT            CONSTANT PLS_INTEGER := 206;
HTTP_MULTIPLE_CHOICES           CONSTANT PLS_INTEGER := 300;
HTTP_MOVED_PERMANENTLY          CONSTANT PLS_INTEGER := 301;
HTTP_FOUND                      CONSTANT PLS_INTEGER := 302;
HTTP_SEE_OTHER                  CONSTANT PLS_INTEGER := 303;
HTTP_NOT_MODIFIED               CONSTANT PLS_INTEGER := 304;
HTTP_USE_PROXY                  CONSTANT PLS_INTEGER := 305;
HTTP_TEMPORARY_REDIRECT         CONSTANT PLS_INTEGER := 307;
HTTP_BAD_REQUEST                CONSTANT PLS_INTEGER := 400;
HTTP_UNAUTHORIZED               CONSTANT PLS_INTEGER := 401;
HTTP_PAYMENT_REQUIRED           CONSTANT PLS_INTEGER := 402;
HTTP_FORBIDDEN                  CONSTANT PLS_INTEGER := 403;
HTTP_NOT_FOUND                  CONSTANT PLS_INTEGER := 404;
HTTP_NOT_ACCEPTABLE             CONSTANT PLS_INTEGER := 406;
HTTP_PROXY_AUTH_REQUIRED        CONSTANT PLS_INTEGER := 407;
HTTP_REQUEST_TIME_OUT           CONSTANT PLS_INTEGER := 408;
HTTP_CONFLICT                   CONSTANT PLS_INTEGER := 409;
HTTP_GONE                       CONSTANT PLS_INTEGER := 410;
HTTP_LENGTH_REQUIRED            CONSTANT PLS_INTEGER := 411;
HTTP_PRECONDITION_FAILED        CONSTANT PLS_INTEGER := 412;
HTTP_REQUEST_ENTITY_TOO_LARGE   CONSTANT PLS_INTEGER := 413;
HTTP_REQUEST_URI_TOO_LARGE      CONSTANT PLS_INTEGER := 414;
HTTP_UNSUPPORTED_MEDIA_TYPE     CONSTANT PLS_INTEGER := 415;
HTTP_REQ_RANGE_NOT_SATISFIABLE  CONSTANT PLS_INTEGER := 416;
HTTP_EXPECTATION_FAILED         CONSTANT PLS_INTEGER := 417;
HTTP_NOT_IMPLEMENTED            CONSTANT PLS_INTEGER := 501;
HTTP_BAD_GATEWAY                CONSTANT PLS_INTEGER := 502;
HTTP_SERVICE_UNAVAILABLE        CONSTANT PLS_INTEGER := 503;
HTTP_GATEWAY_TIME_OUT           CONSTANT PLS_INTEGER := 504;
HTTP_VERSION_NOT_SUPPORTED      CONSTANT PLS_INTEGER := 505;
```

70.3. 타입

본 절에서는 UTL_HTTP 패키지에서 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

70.3.1. HTML_PIECES

서버로부터 응답받은 페이지를 문자열의 배열 형태로 받아올 때 사용하는 타입이다.

HTML_PIECES 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE html_pieces IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
```

70.3.2. req

서버로 보낼 요청 정보를 담고 있는 타입이다.

req 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE req IS RECORD
(
    url             VARCHAR2(32767),
    method          VARCHAR2(64),
    http_version    VARCHAR2(64)
);
```

- 필드

필드 이름	설명
url	웹 페이지의 주소이며, 자세한 내용은 “70.7. URL”을 참고한다.
method	서버에 어떠한 종류의 메서드를 요청할지를 나타낸다. 'OPTIONS', 'GET', 'HEAD', 'POST', 'PUT', 'DELETE', 'TRACE', 'CONNECT'의 값만을 허용한다. (이외는 미지원)
http_version	프로토콜 버전을 나타내며, 값의 종류는 “70.2.1. HTTP 버전 상수”를 참고한다.

70.3.3. resp

서버에서 온 응답에 대한 상태 정보를 담고 있는 타입이다.

resp 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE resp IS RECORD
(
    status_code         PLS_INTEGER,
    reason_phrase       VARCHAR2(256),
    http_version        VARCHAR2(64)
);
```

- 필드

필드 이름	설명
status_code	요청에 따른 응답에 대한 상태를 나타내는 3자리의 숫자이며, 값의 종류는 “70.2.3. 상태 코드 상수”를 참고한다.
reason_phrase	상태 코드에 따르는 간단한 설명이다.
http_version	프로토콜 버전을 나타내며, 값의 종류는 “70.2.1. HTTP 버전 상수”를 참고한다.

70.4. 예외

다음은 UTL_HTTP 패키지에서 미리 제공된 예외이다. 자세한 내용은 “70.5. 함수”와 “70.6. 프러시저”의 "예외 사항" 항목을 참고한다.

- BAD_ARGUMENT
- END_OF_BODY
- REQUEST_FAILED

70.5. 함수

본 절에서는 UTL_HTTP 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

70.5.1. BEGIN_REQUEST

웹 서버와 커넥션을 맺고, 어떤 메서드를 통해서 서버에 요청을 할 지 정한다.

BEGIN_REQUEST 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION BEGIN_REQUEST
(
    url           IN VARCHAR2,
    method        IN VARCHAR2 DEFAULT 'GET',
    http_version  IN VARCHAR2 DEFAULT NULL
)
RETURN req;
```

- 파라미터

파라미터	설명
url	웹 페이지의 주소이며, 자세한 내용은 “70.7. URL”을 참고한다.
method	서버에 어떠한 종류의 메서드를 요청할지를 나타낸다. 'OPTIONS', 'GET', 'HEAD', 'POST', 'PUT', 'DELETE', 'TRACE', 'CONNECT'의 값만을 허용한다. (이외는 미지원)
http_version	프로토콜 버전을 나타내며, 값의 종류는 “70.2.1. HTTP 버전 상수”를 참고한다.

- 예외 사항

예외 사항	설명
REQUEST_FAILED	잘못된 요청으로 웹 서버와 통신할 수 없는 경우에 발생한다. URL이 올바른지 또는 웹 서버가 정상적으로 동작하는지 확인한다.
PKG_TRANSFER_TIMEOUT	20초 이상 웹 서버로부터 응답이 없을 때 타임아웃이 발생한 경우이다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.5.2. GET_RESPONSE

웹 서버에 요청을 보내고, 응답의 헤더를 읽고, 바디를 읽을 준비를 한다. 메시지 BODY를 읽는 방법은 2가지이다.

첫 번째 방법은 메시지 헤더에 "Content-Length: 길이"가 있는 경우이며, 동작은 주어진 길이만큼 메시지 바디를 읽어온다. 두 번째 방법은 메시지 헤더에 "Transfer-Encoding: Chunked"가 오는 경우이며, 캐리지 리턴, 라인피드 문자가 나오는 경우까지 메시지 바디를 읽는다. 헤더에 "Content-Length:길이"와 "Transfer-Encoding: Chunked"이 둘다 주어진 경우 전자는 무시된다.

GET_RESPONSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION GET_RESPONSE
(
    r  IN OUT NOCOPY req
)
RETURN resp;
```

- 파라미터

파라미터	설명
r	요청 식별자이다.

- 예외 사항

예외 사항	설명
BAD_ARGUMENT	BEGIN_REQUEST 또는 SET_HEADER를 선행하지 않고 해당 함수를 호출하거나, 파라미터를 잘못 사용하게 되는 경우 발생한다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.5.3. REQUEST

URL을 통해 웹 서버로 요청을 보내 최대 2000bytes의 데이터를 반환하는 함수이다. 웹 서버로 요청을 보낼 경우 HTTP/1.1 표준의 GET 메서드를 이용한다.

REQUEST 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION REQUEST
(
    url              IN  VARCHAR2,
    proxy            IN  VARCHAR2 DEFAULT NULL,
    wallet_path      IN  VARCHAR2 DEFAULT NULL,
    wallet_password  IN  VARCHAR2 DEFAULT NULL
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
url	웹 페이지의 주소이며, 자세한 내용은 "70.7. URL"을 참고한다.
proxy	프록시 서버의 이름이다. (기본값: NULL, 다음에 구현할 예정이다)
wallet_path	클라이언트 쪽에서 HTTPS를 사용하기 위해 필요한 인증서를 저장하는 경로이다. (기본값: NULL, 다음에 구현할 예정이다)
wallet_password	인증서를 보기 위한 패스워드이다. (기본값: NULL, 다음에 구현할 예정이다)

- 예외 사항

예외 사항	설명
REQUEST_FAILED	잘못된 요청으로 웹 서버와 통신할 수 없는 경우에 발생한다. URL이 올바른지 또는 웹 서버가 정상적으로 동작하는지 확인한다.
PKG_TRANSFER_TIMEOUT	20초 이상 웹 서버로부터 응답이 없을 때 타임아웃이 발생한 경우이다.

- 예제

```
DECLARE
    resp VARCHAR2(32767);
BEGIN
    resp := UTL_HTTP.REQUEST('http://www.w3.org/Protocols/');
END;
```

70.6. 프러시저

70.6.1. END_RESPONSE

HTTP 요청을 끝낸다. 이때 자원(버퍼 메모리, 소켓)이 모두 반환되고, 초기화 상태로 변경된다.

END_RESPONSE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE END_RESPONSE
(
    r  IN OUT NOCOPY resp
);
```

- 파라미터

파라미터	설명
r	http의 response이다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.6.2. GET_BODY_CHARSET

세션에 설정된 기본 메시지 바디 문자 집합의 이름을 가져온다.

GET_BODY_CHARSET 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE GET_BODY_CHARSET
(
    charset    OUT NOCOPY VARCHAR2
);
```

- 파라미터

파라미터	설명
charset	세션에 설정된 문자 집합 이름(캐노니컬 문자 집합 이름)이다.

- 예제

```

DECLARE
  charset varchar(64);
BEGIN
    UTL_HTTP.GET_BODY_CHARSET(charset); -- 캐노니컬 문자 집합의 이름
END;
/
```

70.6.3. GET_DETAILED_EXCP_SUPPORT

세션에 설정된 UTL_HTTP 패키지 수행도중 발생하는 에러의 자세한 메시지 출력 여부 정보를 가져온다.

GET_DETAILED_EXCP_SUPPORT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE GET_DETAILED_EXCP_SUPPORT
(
    enable    OUT BOOLEAN
);
```

- 파라미터

파라미터	설명
enable	기능 활성화 여부를 나타낸다.

- 예제

```
DECLARE
  enable BOOLEAN;
BEGIN
    UTL_HTTP.GET_DETAILED_EXCP_SUPPORT(enable); -- 활성화 여부
END;
/
```

70.6.4. GET_TRANSFER_TIMEOUT

현재 세션에 설정되어 있는 타임아웃 시간을 읽어온다. (기본 타임아웃 시간: 50초)

GET_TRANSFER_TIMEOUT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE GET_TRANSFER_TIMEOUT
(
    timeout    OUT PLS_INTEGER
);
```

- 파라미터

파라미터	설명
timeout	세션에 설정된 타임아웃 시간이다.

- 예제

```
DECLARE
      timeout PLS_INTEGER;
BEGIN
      utl_http.get_transfer_timeout(var);
END;
```

70.6.5. READ_RAW

헤더를 제외한 요청 페이지의 바디를 바이너리 형태로 읽어온다.

READ_RAW 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE READ_RAW
(
    r     IN OUT NOCOPY resp,
    data  OUT NOCOPY RAW,
    len   IN PLS_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
r	응답 식별자이다.
data	헤더를 제외한 바디의 바이너리 형태이다.
len	0 이상의 값이어야 하며, 최대 읽어올 문자열의 길이이다. 이때 요청한 길이보다 바디가 짧게 남아 있는 경우에는 해당 길이보다 적게 읽어온다.

- 예외 사항

예외 사항	설명
BAD_ARGUMENT	GET_RESPONSE 함수를 선행하지 않고 해당 함수를 호출하거나, 파라미터를 잘못 사용하게 되는 경우 발생한다.
END_OF_BODY	바디 내용을 모두 읽었는데, 다시 한번 읽기 요청을 하게 되는 경우 발생한다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.6.6. READ_TEXT

헤더를 제외한 요청 페이지의 바디를 읽어온다.

READ_TEXT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE READ_TEXT
(
    r     IN OUT NOCOPY resp,
    data  OUT NOCOPY VARCHAR2,
    len   IN PLS_INTEGER DEFAULT NULL
);
```

- 파라미터

파라미터	설명
r	응답 식별자이다.
data	헤더를 제외한 바디 문자열이다.
len	0 이상의 값이어야 하며, 최대 읽어올 문자열의 길이이다. 이때 요청한 길이보다 바디가 짧게 남아 있는 경우에는 해당 길이보다 적게 읽어온다.

- 예외 사항

예외 사항	설명
BAD_ARGUMENT	GET_RESPONSE 함수를 선행하지 않고 해당 함수를 호출하거나, 파라미터를 잘못 사용하게 되는 경우 발생한다.
END_OF_BODY	바디 내용을 모두 읽었는데, 다시 한번 읽기 요청을 하게 되는 경우 발생한다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.6.7. READ_LINE

헤더를 제외한 요청 페이지의 바디를 한줄 읽어온다.

READ_LINE 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE READ_LINE
(
    r             IN OUT NOCOPY resp,
    data          OUT NOCOPY VARCHAR2,
    remove_crlf   IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
r	응답 식별자이다.
data	헤더를 제외한 바디의 다음 읽어들일 한 라인의 문자열이다.
remove_crlf	TRUE일 경우 newline charecter를 제외한 한 라인을 읽어온다.

- 예외 사항

예외 사항	설명
BAD_ARGUMENT	GET_RESPONSE 함수를 선행하지 않고 해당 함수를 호출하거나, 파라미터를 잘못 사용하게 되는 경우 발생한다.
END_OF_BODY	바디 내용을 모두 읽었는데, 다시 한번 읽기 요청을 하게 되는 경우 발생한다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.6.8. SET_BODY_CHARSET

HTTP 요청이나 HTTP 응답 메시지 바디의 문자 집합을 지정한다.

SET_BODY_CHARSET의 세부 내용은 다음과 같다.

- CASE1

  HTTP 요청이나 HTTP 응답에 문자 집합(Character Set)을 지정하지 않는 경우 그리고 메시지 헤더에 Content-Type이 지정되지 않은 경우, 문자 집합을 지정할수 있다. 메시지 헤더에 Content-Type이 지정된 경우에는 무시된다. 세션 단위로 해당 정보의 설정이 가능하다.

  – 프로토타입

```
PROCEDURE SET_BODY_CHARSET
(
    charset    IN VARCHAR2 DEFAULT NULL
);
```

  – 파라미터

파라미터	설명
charset	문자 집합(캐노니컬 문자 집합)의 이름이다. (기본값: 'ISO-8859-1') (예: ASCII, UTF-8, EUC-KR, CP949 ... 등)

- CASE2

  HTTP 요청의 헤더에 Content-Type이 지정되지 않은 경우, 바디의 문자 집합을 지정한다. 문자열을 생략하는 경우, 'ISO-8859-1'로 간주된다. HTTP 요청 메시지 바디는 DB 문자 집합 문자열에서 지정된 문자열 집합의 문자열로 자동 변환된다. 메시지 헤더에 Content-Type이 지정된 경우에는 무시된다. 한번의 HTTP 요청에 대하여 설정 가능하다.

  – 프로토타입

```
PROCEDURE SET_BODY_CHARSET
(
    r           IN OUT NOCOPY req,
    charset     IN VARCHAR2 DEFAULT NULL
);
```

  – 파라미터

파라미터	설명
req	HTTP 요청이다.
charset	문자 집합(캐노니컬 문자 집합)의 이름이다. (기본값: 'ISO-8859-1') (예: ASCII, UTF-8, EUC-KR, CP949 ... 등)

- CASE3

  HTTP 응답의 헤더에 Content-Type이 지정되지 않은 경우, 바디의 문자열로 간주되는 문자열 집합을 지정한다. 문자열을 생략하는 경우, 'ISO-8859-1'로 간주된다. HTTP 응답 메시지 바디는 지정된 문자열 집합의 문자열에서 DB 문자 집합 문자열로 자동 변환된다. 메시지 헤더에 Content-Type이 지정된 경우에는 무시된다. 한번의 HTTP 응답에 대하여 설정 가능하다.

  – 프로토타입

```
PROCEDURE SET_BODY_CHARSET
(
    r           IN OUT NOCOPY resp,
    charset     IN VARCHAR2 DEFAULT NULL
);
```

  – 파라미터

파라미터	설명
resp	HTTP 응답이다.
charset	문자 집합(캐노니컬 문자 집합)의 이름이다. (기본값: 'ISO-8859-1') (예: ASCII, UTF-8, EUC-KR, CP949 ... 등)

- 예외 사항

예외 사항	설명
BAD_ARGUMENT	올바른 문자 집합 문자열을 넣지 않는 경우 발생한다.

- 예제

```
-- HTTP 요청과 응답의 바디 문자 집합을 UTF-8로 변경함
DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  value varchar(32767);
BEGIN
  req := UTL_HTTP.BEGIN_REQUEST('http://www.naver.com',
                                'GET','HTTP/1.1');
  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
  UTL_HTTP.WRITE_TEXT(req, 'test_value');
  UTL_HTTP.SET_BODY_CHARSET(req, 'UTF-8');
  resp := UTL_HTTP.GET_RESPONSE(req);
  UTL_HTTP.SET_BODY_CHARSET(resp, 'UTF-8');
  FOR  i in 1 ..  200000
  LOOP
    UTL_HTTP.READ_TEXT(resp, value);
    DBMS_OUTPUT.PUT_LINE ( value );
  END LOOP;
    UTL_HTTP.END_RESPONSE(resp);
exception when others then
    UTL_HTTP.END_RESPONSE(resp);
end;
/
-- HTTP 요청과 응답에 대한 세션의 기본 문자 집합을 UTF-8로 변경함
BEGIN
    UTL_HTTP.SET_BODY_CHARSET('UTF-8');
END;
/
```

70.6.9. SET_DETAILED_EXCP_SUPPORT

세션에 설정된 UTL_HTTP 패키지 수행도중 발생하는 에러의 자세한 메시지 출력 여부를 설정한다.

SET_DETAILED_EXCP_SUPPORT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SET_DETAILED_EXCP_SUPPORT
(
```

```
    enable    OUT BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
enable	기능 활성화 여부를 나타낸다. (기본값: 비활성화)

- 예제

```
BEGIN
    UTL_HTTP.SET_DETAILED_EXCP_SUPPORT(false); -- 비활성화
END;
/
```

70.6.10. SET_HEADER

서버로 보낼 요청에 헤더를 설정한다. 여러 번 반복 수행하여 추가적인 헤더 정보의 추가가 가능하다.

SET_HEADER 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SET_HEADER
(
    r       IN OUT NOCOPY req,
    name    IN VARCHAR2,
    value   IN VARCHAR2
);
```

- 파라미터

파라미터	설명
r	요청 식별자이다.
name	설정할 헤더 이름이다.
value	설정할 헤더 값이다.

- 예제

단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.6.11. SET_RESPONSE_ERROR_CHECK

웹 서버로부터 받은 상태 코드가 에러에 대한 코드일 경우, 이를 GET_RESPONSE GET_RESPONSE에서 상태 에러를 예외로 처리할 지를 설정하는 프러시저다. 400부터 599까지의 상태 코드가 상태 에러에 해당한다.

SET_RESPONSE_ERROR_CHECK 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SET_RESPONSE_ERROR_CHECK
(
    enable    IN BOOLEAN DEFAULT FALSE
);
```

- 파라미터

파라미터	설명
enable	상태 에러 확인 기능 활성화 여부를 나타낸다. (기본값: 비활성화)

- 예제

```
DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  value VARCHAR2(1024);
BEGIN
  UTL_HTTP.SET_RESPONSE_ERROR_CHECK(enable => TRUE); -- 활성화
  req := UTL_HTTP.BEGIN_REQUEST('http://getstatuscode.com/400','GET', 'HTTP/1.1');

  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
  resp := UTL_HTTP.GET_RESPONSE(req);
  UTL_HTTP.END_RESPONSE(resp);
END;
/
```

70.6.12. SET_TRANSFER_TIMEOUT

현재 세션에 타임아웃 시간을 설정한다. (기본 타임아웃 시간: 50초)

SET_TRANSFER_TIMEOUT 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE SET_TRANSFER_TIMEOUT
(
    timeout    IN PLS_INTEGER
);
```

- 파라미터

파라미터	설명
timeout	세션에 설정할 타임아웃 시간이다.

- 예제

```
BEGIN
      utl_http.set_transfer_timeout(30);
END;
/
```

70.6.13. WRITE_TEXT

서버로 보낼 요청의 바디를 설정한다.

WRITE_TEXT의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE WRITE_TEXT
(
    r     IN OUT NOCOPY REQ,
    data  IN VARCHAR2
);
```

- 파라미터

파라미터	설명
r	요청 식별자이다.
data	요청 바디의 내용이다.

- 예외 사항

예외 사항	설명
BAD_ARGUMENT	BEGIN_REQUEST 또는 SET_HEADER를 선행하지 않고 해당 함수를 호출하거나, 파라미터를 잘못 사용하게 되는 경우 발생한다.
END_OF_BODY	바디 내용을 모두 읽었는데, 다시 한번 읽기 요청을 하게 되는 경우 발생한다.

- 예제

  단일 함수 사용의 예가 아니므로, “70.8. 예제”를 참고한다.

70.7. URL

URL(Uniform Resource/ Locator)은 서버의 웹 페이지 주소를 나타내는 형태이며, 다음과 같은 형태로 구성된다.

```
scheme://[user[password]@]host[:port]/[...]
```

다음은 각 항목에 대한 설명이다.

항목	설명
scheme	http와 https가 있으며, https는 다음에 구현할 예정이다.
user	proxy를 위해 존재하며, 다음에 구현할 예정이다.
password	proxy를 위해 존재하며, 다음에 구현할 예정이다.
host	HTTP 서버 호스트 이름이다.
port	통신에 사용할 포트 번호이다.

70.8. 예제

다음은 BEGIN_REQUEST, GET_RESPONSE,END_RESPONSE, READ_RAW를 이용하여 www.example.com에 있는 image.jpg를 읽어오는 예시이다.

```
DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  value RAW(1024);
BEGIN
  req := UTL_HTTP.BEGIN_REQUEST('http://www.example.com/image.jpg',
                                'GET', 'HTTP/1.1');
  resp := UTL_HTTP.GET_RESPONSE(req);

  LOOP
    UTL_HTTP.READ_RAW(resp, value);
    DBMS_OUTPUT.PUT_LINE ( value );
  END LOOP;
EXCEPTION
  WHEN UTL_HTTP.END_OF_BODY THEN
  UTL_HTTP.END_RESPONSE(resp);
END;
/
```

다음은 BEGIN_REQUEST, SET_HEADER, WRITE_TEXT, GET_RESPONSE, END_RESPONSE, READ_TEXT를 이용하여 ims.tmaxsoft.com의 로그인 웹 페이지의 html 스크립트를 읽어오는 예이다.

```
DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  value VARCHAR2(1024);
BEGIN
req := UTL_HTTP.BEGIN_REQUEST('http://ims.tmaxsoft.com/tody/auth/login.do',
                              'POST', 'HTTP/1.1');
  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
  UTL_HTTP.WRITE_TEXT(req, 'test_value');
  resp := UTL_HTTP.GET_RESPONSE(req);

  LOOP
    UTL_HTTP.READ_TEXT(resp, value);
    DBMS_OUTPUT.PUT_LINE ( value );
  END LOOP;
EXCEPTION
  WHEN UTL_HTTP.END_OF_BODY THEN
  UTL_HTTP.END_RESPONSE(resp);
END;
/
```

다음은 BEGIN_REQUEST, SET_HEADER, WRITE_TEXT, GET_RESPONSE, END_RESPONSE, READ_LINE을 이용하여 ims.tmaxsoft.com의 로그인 웹 페이지의 html 스크립트를 읽어오는 예이다.

```
DECLARE
  req   UTL_HTTP.REQ;
  resp  UTL_HTTP.RESP;
  value VARCHAR2(1024);
BEGIN
req := UTL_HTTP.BEGIN_REQUEST('http://ims.tmaxsoft.com/tody/auth/login.do',
                              'POST', 'HTTP/1.1');
  UTL_HTTP.SET_HEADER(req, 'User-Agent', 'Mozilla/4.0');
  UTL_HTTP.WRITE_TEXT(req, 'test_value');
  resp := UTL_HTTP.GET_RESPONSE(req);

  LOOP
    UTL_HTTP.READ_LINE(resp, value, true);
    DBMS_OUTPUT.PUT_LINE ( value );
  END LOOP;
EXCEPTION
  WHEN UTL_HTTP.END_OF_BODY THEN
  UTL_HTTP.END_RESPONSE(resp);
END;
/
```

제71장 UTL_I18N

본 장에서는 UTL_I18N 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

71.1. 개요

UTL_I18N은 국가, 언어들 간의 다양한 변환 및 호환 기능을 제공한다.

71.2. 프러시저와 함수

본 절에서는 UTL_I18N 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

71.2.1. RAW_TO_CHAR

입력받은 RAW 데이터를 현재 데이터베이스 문자 집합 문자열로 변환한다.

RAW_TO_CHAR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_I18N.RAW_TO_CHAR
(
    data            IN      RAW,
    src_charset     IN      VARCHAR2
)
RETURN VARCHAR2
```

- 파라미터

파라미터	설명
data	입력 문자열의 바이너리 데이터이다.
src_charset	입력 데이터 문자열의 문자 집합이다.

- 예제

```
DECLARE
    v_in_raw raw(100) := '74696265726F';
    x varchar2(100);
BEGIN
```

```
    x := UTL_I18N.RAW_TO_CHAR (v_in_raw, 'UTF8');
END;
```

71.2.2. STRING_TO_RAW

현재 데이터베이스 문자 집합인 입력문자열을 다른 문자 집합의 문자열 바이너리 데이터로 변환한다.

STRING_TO_RAW 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_I18N.STRING_TO_RAW
(
    data            IN      VARCHAR2,
    dst_charset     IN      VARCHAR2
)
RETURN RAW
```

- 파라미터

파라미터	설명
data	현재 데이터베이스 문자 집합의 입력 문자열이다.
dst_charset	변환하려는 문자 집합이다.

- 예제

```
DECLARE
    v_in_string varchar2(100) := 'sample_string';
    x raw(100);
BEGIN
    x := UTL_I18N.STRING_TO_RAW (v_in_string, 'UTF8');
END;
```

71.2.3. UNESCAPE_REFERENCE

현재 데이터베이스 ESCAPE 문자가 포함된 입력문자열 데이터를 UNESCAPE한 문자열 데이터로 변환한다.

UNESCAPE_REFERENCE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_I18N.UNESCAPE_REFERENCE
(
    str             IN      VARCHAR2
)
RETURN VARCHAR2
```

- 파라미터

파라미터	설명
str	입력 문자열이다.

- 예제

```
DECLARE
    v_in_string varchar2(100) := '&lt;123&#62;';
    x varchar2(100);
BEGIN
    x := UTL_I18N.UNESCAPE_REFERENCE (v_in_string);
END;
```

제72장 UTL_MATCH

본 장에서는 UTL_MATCH 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

72.1. 개요

UTL_MATCH는 두 문자열의 유사도를 계산하는 함수들을 제공하는 패키지이다.

본 패키지는 두 문자열의 유사도를 계산하는 방법들 중, Levenshtein 알고리즘을 사용하여 Edit Distance를 산출하는 방법과 Jaro-Winkler 알고리즘을 사용하여 Match Score를 산출하는 방법을 사용하는 함수들을 제공한다.

72.2. 함수

본 절에서는 UTL_MATCH 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

72.2.1. EDIT_DISTANCE

주어진 두 문자열의 Edit Distance를 Levenshtein 알고리즘을 이용하여 산출하여 반환하는 함수이다.

EDIT_DISTANCE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_MATCH.EDIT_DISTANCE
(
        str1       IN          VARCHAR2,
        str2       IN          VARCHAR2
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
str1	피연산자이다.
str2	피연산자이다.

- 반환값

반환값	설명
PLS_INTEGER	str1과 str2의 Edit Distance 산출 결과를 반환한다. str1과 str2 중 하나라도 NULL인 경우는 -1을 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_MATCH.EDIT_DISTANCE('apple','aaple'));
END;
/
1
```

72.2.2. EDIT_DISTANCE_SIMILARITY

Levenshtein 알고리즘을 이용하여 산출한 주어진 두 문자열의 Edit Distance를 두 문자열 중 긴 문자열의 길이로 Normalize후 백분율 값으로 반환하는 함수이다.

EDIT_DISTANCE_SIMILARITY 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_MATCH.EDIT_DISTANCE_SIMILARITY
(
        str1       IN          VARCHAR2,
        str2       IN          VARCHAR2
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
str1	피연산자이다.
str2	피연산자이다.

- 반환값

반환값	설명
PLS_INTEGER	str1과 str2의 Edit Distance 산출 결과를 Length가 긴 피연산자의 Length로 Normalize한 값을 백분율 값으로 반환한다. str1과 str2가 모두 NULL인 경우는 100을 반환하고, 둘 중 하나만 NULL인 경우는 0을 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_MATCH.EDIT_DISTANCE_SIMILARITY('apple', 'aaple'));
END;
/
81
```

72.2.3. JARO_WINKLER

주어진 두 문자열의 Match Score를 Jaro-Winkler 알고리즘을 사용하여 산출한 후 반환하는 함수이다.

JARO_WINKLER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_MATCH.JARO_WINKLER
(
        str1       IN          VARCHAR2,
        str2       IN          VARCHAR2
)
RETURN BINARY_DOUBLE;
```

- 파라미터

파라미터	설명
str1	피연산자이다.
str2	피연산자이다.

- 반환값

반환값	설명
BINARY_DOUBLE	str1과 str2의 Match Score 산출 결과를 반환한다. str1과 str2 중 하나라도 NULL인 경우는 0을 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_MATCH.JARO_WINKLER('apple','aaple'));
END;
/
.88
```

72.2.4. JARO_WINKLER_SIMILARITY

Jaro-Winkler 알고리즘을 이용하여 산출한 주어진 두 문자열의 Match Score를 백분율 값으로 반환하는 함수이다.

JARO_WINKLER_SIMILARITY 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_MATCH.JARO_WINKLER_SIMILARITY
(
        str1       IN          VARCHAR2,
        str2       IN          VARCHAR2
)
RETURN PLS_INTEGER;
```

- 파라미터

파라미터	설명
str1	피연산자이다.
str2	피연산자이다.

- 반환값

반환값	설명
PLS_INTEGER	str1과 str2의 Match Score 산출 결과를 백분율값으로 반환한다. str1과 str2 중 하나라도 NULL인 경우는 0을 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_MATCH.JARO_DISTANCE_SIMILARITY('apple','aaple'));
END;
/
88
```

제73장 UTL_RAW

본 장에서는 UTL_RAW 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

73.1. 개요

UTL_RAW는 RAW 타입의 데이터와 관련된 여러 가지 함수를 제공하는 패키지이다.

SQL 문장 내에서 CHAR 또는 VARCHAR 타입의 데이터에 사용할 수 있는 함수는 RAW 타입의 데이터에는 사용할 수 없다. 또한, RAW 타입과 CHAR, VARCHAR 타입 간에는 데이터 변환을 할 수 없다. 이러한 문제를 해결하기 위해 UTL_RAW 내에 정의된 함수를 사용한다. 그러면 CHAR, VARCHAR 타입의 데이터에 사용할 수 있는 함수의 기능을 RAW 타입의 데이터에도 사용할 수 있다.

73.2. 함수

본 절에서는 UTL_RAW 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

73.2.1. BIT_AND

주어진 두 바이너리 데이터에 AND 연산을 수행하여 반환하는 함수이다. 이 두 데이터의 길이가 다르면, 짧은 쪽 데이터의 길이만큼 연산을 수행한 결과에 긴 쪽에 남아있는 부분의 데이터를 첨가한다. 따라서 반환되는 최종 결과 데이터는 긴 쪽의 데이터와 길이가 같다.

BIT_AND 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.BIT_AND
(
    r1       IN          RAW,
    r2       IN          RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r1	피연산자이다.
r2	피연산자이다.

- 반환값

반환값	설명
RAW	r1과 r2의 AND 연산 결과를 반환한다.
NULL	r1과 r2 중 하나라도 NULL인 경우에 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.BIT_AND('abc', '012'));
END;
/
0010

PSM completed
SQL>
```

73.2.2. BIT_COMPLEMENT

주어진 바이너리 데이터에 대한 1의 보수를 반환하는 함수이다.

BIT_COMPLEMENT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.BIT_COMPLEMENT
(
    r       IN       RAW
)  RETURN RAW;
```

- 파라미터

파라미터	설명
r	피연산자이다.

- 반환값

반환값	설명
RAW	r의 COMPLEMENT 연산 결과를 반환한다.
NULL	r가 NULL인 경우에 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.BIT_COMPLEMENT('456'));
END;
/
FBA9

PSM completed
SQL>
```

73.2.3. BIT_OR

주어진 두 바이너리 데이터에 대한 OR 연산을 수행하여 반환하는 함수이다. 두 데이터의 길이가 다르면, 짧은 쪽 데이터의 길이만큼 연산을 수행한 결과에 긴 쪽의 남아있는 부분의 데이터를 첨가한다. 반환되는 최종 결과 데이터는 긴 쪽의 데이터와 길이가 같다.

BIT_OR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.BIT_OR
(
    r1       IN       RAW,
    r2       IN       RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r1	피연산자이다.
r2	피연산자이다.

- 반환값

반환값	설명
RAW	r1과 r2의 OR 연산 결과를 반환한다.
NULL	r1과 r2 중 하나라도 NULL인 경우에 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.BIT_OR('123', '456'));
END;
```

```
/
0577

PSM completed
SQL>
```

73.2.4. BIT_XOR

주어진 두 바이너리 데이터에 대한 XOR 연산을 수행하여 반환하는 함수이다. 두 데이터의 길이가 다르면, 짧은 쪽 데이터의 길이만큼 연산을 수행한 결과에 긴 쪽의 남아있는 부분의 데이터를 첨가한다. 따라서 반환되는 최종 결과 데이터는 긴 쪽의 데이터와 길이가 같다.

BIT_XOR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.BIT_XOR
(
    r1 IN RAW,
    r2 IN RAW
)  RETURN RAW;
```

- 파라미터

파라미터	설명
r1	피연산자이다.
r2	피연산자이다.

- 반환값

반환값	설명
RAW	r1과 r2의 XOR 연산 결과를 반환한다.
NULL	r1과 r2 중 하나라도 NULL인 경우에 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.BIT_XOR('123', '456'));
END;
/
0575
```

```
PSM completed
SQL>
```

73.2.5. CAST_FROM_BINARY_DOUBLE

주어진 BINARY_DOUBLE 타입의 값을 RAW 타입의 데이터로 변환하여 반환하는 함수이다. 반환된 데이터는 8bytes의 길이를 가지며, 지정된 endian에 따라 Byte 순서가 결정된다.

파라미터로 주어진 endianess의 값은 RAW 타입의 데이터가 저장되는 endian을 뜻한다. 즉, 시스템이 little-endian이고, 파라미터가 big-endian인 경우 RAW 타입의 데이터에는 입력 값 n의 Byte 순서와 반대되는 순서로 Byte 값이 저장된다. 파라미터가 little-endian이라면 시스템의 Byte 순서와 동일한 순서로 Byte 값이 저장된다.

CAST_FROM_BINARY_DOUBLE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_FROM_BINARY_DOUBLE
(
    n              IN          BINARY_DOUBLE,
    endianess      IN          PLS_INTEGER     DEFAULT 1
)
RETURN RAW;
```

- 파라미터

파라미터	설명
n	변환할 BINARY_DOUBLE 값이다.
endianess	변환할 endian이다. 1(big-endian) 또는 2(little-endian) 중 하나이다.

- 반환값

반환값	설명
RAW	BINARY_DOUBLE 타입의 값의 바이너리 표현을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_FROM_BINARY_DOUBLE(1.5d));
END;
/
3FF8000000000000

PSM completed
SQL>
```

73.2.6. CAST_FROM_BINARY_FLOAT

주어진 BINARY_FLOAT 타입의 값을 RAW 타입의 데이터로 변환하여 반환하는 함수이다. 반환된 데이터는 4bytes의 길이를 가지며, 지정된 endian에 따라 Byte 순서가 결정된다.

파라미터로 주어진 endianess의 값은 RAW 타입의 데이터가 저장되는 endian을 뜻한다. 즉, 시스템이 little-endian이고, 파라미터가 big-endian인 경우 RAW 타입의 데이터에는 입력 값 n의 Byte 순서와 반대되는 순서로 Byte 값이 저장된다. 파라미터가 little-endian이라면 시스템의 Byte 순서와 동일한 순서로 Byte 값이 저장된다.

CAST_FROM_BINARY_FLOAT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_FROM_BINARY_FLOAT
(
    n              IN          BINARY_FLOAT,
    endianess      IN          PLS_INTEGER     DEFAULT 1
)
RETURN RAW;
```

- 파라미터

파라미터	설명
n	변환할 BINARY_FLOAT 값이다.
endianess	변환할 endian이다. 1(big-endian) 또는 2(little-endian) 중 하나이다.

- 반환값

반환값	설명
RAW	BINARY_FLOAT 타입의 값의 바이너리 표현을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_FROM_BINARY_FLOAT(1.5f));
END;
/
3FC00000

PSM completed
SQL>
```

73.2.7. CAST_FROM_BINARY_INTEGER

주어진 BINARY_INTEGER 타입의 값을 RAW 타입의 데이터로 변환하여 반환하는 함수이다. 반환된 데이터는 4bytes의 길이를 가지며, 지정된 endian에 따라 Byte 순서가 결정된다.

파라미터로 주어진 endianess의 값은 RAW 타입의 데이터가 저장되는 endian을 뜻한다. 즉, 시스템이 little-endian이고, 파라미터가 big-endian인 경우 RAW 타입의 데이터에는 입력 값 n의 Byte 순서와 반대되는 순서로 Byte 값이 저장된다. 파라미터가 little-endian이라면 시스템의 Byte 순서와 동일한 순서로 Byte 값이 저장된다.

CAST_FROM_BINARY_INTEGER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_FROM_BINARY_INTEGER
(
    n              IN          BINARY_INTEGER,
    endianess      IN          PLS_INTEGER         DEFAULT 1
)
RETURN RAW;
```

- 파라미터

파라미터	설명
n	변환할 BINARY_INTEGER 값이다.
endianess	변환할 endian이다. 1(big-endian) 또는 2(little-endian) 중 하나이다.

- 반환값

반환값	설명
RAW	BINARY_INTEGER 타입의 값의 바이너리 표현을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_FROM_BINARY_INTEGER(10));
END;
/
0000000A

PSM completed
SQL>
```

73.2.8. CAST_FROM_NUMBER

주어진 NUMBER 타입 값을 RAW 타입의 데이터로 변환하여 반환하는 함수이다. 반환된 바이너리 데이터는 입력된 값에 대한 Tibero의 내부 표현이며, 반환된 데이터의 길이는 입력 값에 따라 가변 길이를 가진다.

파라미터로 주어진 endianess의 값은 RAW 타입의 데이터가 저장되는 endian을 뜻한다. 즉, 시스템이 little-endian이고, 파라미터가 big-endian인 경우 RAW 타입의 데이터에는 입력 값 n의 Byte 순서와 반대되는 순서로 Byte 값이 저장된다. 파라미터가 little-endian이라면 시스템의 Byte 순서와 동일한 순서로 Byte 값이 저장된다.

CAST_FROM_NUMBER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_FROM_NUMBER
(
    n           IN          NUMBER
)  RETURN RAW;
```

- 파라미터

파라미터	설명
n	변환할 숫자형 값이다.

- 반환값

반환값	설명
RAW	숫자형 값의 바이너리 표현을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_FROM_NUMBER(1.5));
END;
/
03C181B2

PSM completed
SQL>
```

73.2.9. CAST_TO_BINARY_DOUBLE

주어진 RAW 타입의 데이터를 BINARY_DOUBLE 타입의 값으로 변환하여 반환하는 함수이다. 입력되는 바이너리 데이터는 8bytes의 길이를 가진다.

CAST_TO_BINARY_DOUBLE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_TO_BINARY_DOUBLE
(
    r               IN          RAW,
    endianess       IN          PLS_INTEGER     DEFAULT 1
)
RETURN BINARY_DOUBLE;
```

- 파라미터

파라미터	설명
r	변환할 바이너리 데이터이다.

파라미터	설명
endianess	바이너리 데이터의 endian이다. 1(big-endian) 또는 2(little-endian) 중 하나이다.

- 반환값

반환값	설명
BINARY_DOUBLE	RAW 타입의 데이터를 BINARY_DOUBLE 타입의 값으로 변환한 값을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
DECLARE
    bin raw(100);
BEGIN
    bin := UTL_RAW.CAST_FROM_BINARY_DOUBLE(1.5d);
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_BINARY_DOUBLE(bin));
END;
/
1.5E+00

PSM completed
SQL>
```

73.2.10. CAST_TO_BINARY_FLOAT

주어진 RAW 타입의 데이터를 BINARY_FLOAT 타입의 값으로 변환하여 반환하는 함수이다. 입력되는 바이너리 데이터는 4bytes의 길이를 가진다.

CAST_TO_BINARY_FLOAT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_TO_BINARY_FLOAT
(
     r              IN          RAW,
     endianess      IN          PLS_INTEGER     DEFAULT 1
```

```
)
RETURN BINARY_FLOAT;
```

- 파라미터

파라미터	설명
r	변환할 바이너리 데이터이다.
endianess	바이너리 데이터의 endian이다. 1(big-endian) 또는 2(little-endian) 중 하나이다.

- 반환값

반환값	설명
BINARY_FLOAT	RAW 타입의 데이터를 BINARY_FLOAT 타입의 값으로 변환한 값을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
DECLARE
    bin raw(100);
BEGIN
    bin := UTL_RAW.CAST_FROM_BINARY_FLOAT(1.5f);
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_BINARY_FLOAT(bin));
END;
/
1.5E+00

PSM completed
SQL>
```

73.2.11. CAST_TO_BINARY_INTEGER

주어진 RAW 타입의 데이터를 BINARY_INTEGER 타입의 값으로 변환하여 반환하는 함수이다. 입력되는 바이너리 데이터는 4bytes의 길이를 가진다.

CAST_TO_BINARY_INTEGER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_TO_BINARY_INTEGER
(
    r              IN          RAW,
    endianess      IN          PLS_INTEGER     DEFAULT 1
)
RETURN BINARY_INTEGER;
```

- 파라미터

파라미터	설명
r	변환할 바이너리 데이터이다.
endianess	바이너리 데이터의 endian이다. 1(big-endian) 또는 2(little-endian) 중 하나이다.

- 반환값

반환값	설명
BINARY_INTEGER	RAW 타입의 데이터를 BINARY_INTEGER 타입의 값으로 변환한 값을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
DECLARE
    bin raw(100);
BEGIN
    bin := UTL_RAW.CAST_FROM_BINARY_INTEGER(777);
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_BINARY_INTEGER(bin));
END;
/
777

PSM completed
SQL>
```

73.2.12. CAST_TO_NUMBER

주어진 RAW 타입의 데이터를 NUMBER 타입 값으로 변환하여 반환하는 함수이다. 입력되는 바이너리 데이터는 NUMBER 타입 값에 대한 Tibero의 내부 표현이며 가변 길이를 갖는다.

CAST_TO_NUMBER 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_TO_NUMBER
(
    r           IN          RAW
)  RETURN NUMBER;
```

- 파라미터

파라미터	설명
r	변환할 바이너리 데이터이다.

- 반환값

반환값	설명
NUMBER	RAW 타입의 데이터를 NUMBER 타입 값으로 변환한 값을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예외 상황

예외 상황	설명
INVALID_ENDIAN	endian의 값이 1이나 2가 아닌 경우이다.

- 예제

```
DECLARE
    bin raw(100);
BEGIN
    bin := UTL_RAW.CAST_FROM_NUMBER(1004);
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_NUMBER(bin));
END;
/
1004

PSM completed
SQL>
```

73.2.13. CAST_TO_RAW

주어진 VARCHAR 타입의 문자열을 RAW 타입의 데이터로 변환하여 반환하는 함수이다. 입력되는 문자열은 RAW 타입의 데이터의 문자열 표현이다.

CAST_TO_RAW 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_TO_RAW
(
    c        IN          VARCHAR2
)  RETURN RAW;
```

- 파라미터

파라미터	설명
c	변환할 문자열이다.

- 반환값

반환값	설명
RAW	VARCHAR2의 바이너리 표현을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_RAW('0a3bcf'));
END;
/
306133626366

PSM completed
SQL>
```

73.2.14. CAST_TO_VARCHAR2

주어진 RAW 타입의 데이터를 VARCHAR2 타입의 문자열로 변환하여 반환하는 함수이다. 출력되는 문자열은 바이너리 데이터의 문자열 표현이다.

CAST_TO_VARCHAR2 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CAST_TO_VARCHAR2
(
    r           IN          RAW
)  RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
r	변환할 바이너리 데이터이다.

- 반환값

반환값	설명
VARCHAR2	RAW 타입의 데이터를 VARCHAR2 타입으로 변환한 값을 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예제

```
DECLARE
    bin RAW(100);
BEGIN
    bin := UTL_RAW.CAST_TO_RAW('Nanobase');
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_VARCHAR2(bin));
END;
/
Nanobase

PSM completed
SQL>
```

73.2.15. COMPARE

주어진 두 개의 RAW 타입의 데이터를 비교하는 함수이다. 두 바이너리 데이터가 서로 다른 길이를 갖는다면, 짧은 쪽의 데이터에 파라미터 패드(pad)에 저장된 Byte를 첨부하여 같은 길이를 갖도록 한다.

COMPARE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.COMPARE
(
    r1          IN          RAW,
    r2          IN          RAW,
    pad         IN          RAW     DEFAULT NULL
```

```
)
RETURN INTEGER;
```

- 파라미터

파라미터	설명
r1	비교할 첫 번째 피연산자이다.
r2	비교할 두 번째 피연산자이다.
pad	짧은 쪽에 첨부할 Byte이다. NULL이면 '00'을 첨부한다.

- 반환값

반환값	설명
0	r1과 r2가 모두 NULL이거나 길이가 완전히 같을 경우에 반환한다.
N >= 1	r1과 r2가 다른 경우 서로 다른 Byte 값을 갖는 위치를 반환한다.

- 예제

```
DECLARE
    x RAW(20) := UTL_RAW.CAST_TO_RAW('ABCDEF');
    y RAW(20) := UTL_RAW.CAST_TO_RAW('ABCDFF');
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.COMPARE(x, y));
END;
/
5

PSM completed
SQL>
```

73.2.16. CONCAT

주어진 RAW 타입의 데이터를 결합하여 반환하는 함수이다. 이 함수는 1~12개까지의 입력 파라미터를 가질 수 있다. 결합된 바이너리 데이터는 최대 32767bytes 길이를 가질 수 있으며, 이 길이를 초과하면 ERROR_EXP_CONCAT_TOO_LONG 에러를 발생한다.

CONCAT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.CONCAT
(
    r1       IN          RAW,
```

```
    r2       IN          RAW,
    r3       IN          RAW,
    r4       IN          RAW,
    r5       IN          RAW,
    r6       IN          RAW,
    r7       IN          RAW,
    r8       IN          RAW,
    r9       IN          RAW,
    r10      IN          RAW,
    r11      IN          RAW,
    r12      IN          RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
r1, ..., r12	결합할 바이너리 데이터이다. 12개까지 설정이 가능하다.

- 반환값

반환값	설명
RAW	입력 값이 차례로 결합된 RAW를 반환한다.
NULL	입력 값이 모두 NULL인 경우에 반환한다.

- 예제

```
DECLARE
    x RAW(100) := UTL_RAW.CAST_TO_RAW('Give me liberty');
    y RAW(100) := UTL_RAW.CAST_TO_RAW(', or give me death');
    z RAW(200);
BEGIN
    z := UTL_RAW.CONCAT(x, y);
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_VARCHAR2(z));
END;
/
Give me liberty, or give me death

PSM completed
SQL>
```

73.2.17. COPIES

주어진 RAW 타입의 데이터를 n번 반복하여 결합한 바이너리 데이터를 반환하는 함수이다. 반환되는 데이터의 길이는 최대 32767bytes 이내이어야 하며, 이 길이를 초과하면 예외 상황이 발생한다.

COPIES 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.COPIES
(
    r               IN          RAW,
    n               IN          NUMBER
)  RETURN RAW;
```

- 파라미터

파라미터	설명
r	반복할 바이너리 데이터이다.
n	반복 횟수이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	– 파라미터 r가 NULL인 경우이다. – 파라미터 n이 1보다 작은 경우이다. – 반환되는 데이터의 길이가 32767bytes를 초과하는 경우이다.

- 예제

```
DECLARE
    x VARCHAR2(100);
    y RAW(200);
    z RAW(200);
BEGIN
    x := 'Books are ships which pass ';
    x := x || 'through the vast seas of time';
    y := UTL_RAW.CAST_TO_RAW(x);
    z := UTL_RAW.COPIES(y, 1);
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.CAST_TO_VARCHAR2(z));
END;
/
Books are ships which pass through the vast seas of time
```

```
PSM completed
SQL>
```

73.2.18. LENGTH

주어진 RAW 타입의 데이터의 Byte 길이를 반환하는 함수이다.

LENGTH 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.LENGTH
(
    r        IN          RAW
)  RETURN INTEGER;
```

- 파라미터

파라미터	설명
r	길이 반환의 대상이 되는 바이너리 데이터이다.

- 반환값

반환값	설명
NUMBER	데이터의 길이를 반환한다.
NULL	입력 값이 NULL인 경우에 반환한다.

- 예제

```
DECLARE
    x RAW(20) := UTL_RAW.CAST_TO_RAW('Out of mind');
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.LENGTH(x));
END;
/
11

PSM completed
SQL>
```

73.2.19. OVERLAY

RAW 타입의 원본 데이터를 RAW 타입의 대상 데이터에 복사하는 함수이다. 대상 데이터 내에 파라미터 pos가 가리키는 위치로부터 파라미터 len으로 지정한 길이만큼 원본 데이터를 복사한다.

원본 데이터의 길이가 len보다 짧다면 파라미터 패드에 저장된 Byte로 채운다. 대상 데이터의 길이가 pos보다 작다면 pos 위치에까지 pad byte로 채운 다음 연속하여 원본 데이터가 복사된다.

이 함수는 대상 데이터에 원본 데이터를 복사한 데이터를 반환하며, 대상 데이터 자체는 갱신되지 않는다. 반환되는 바이너리 데이터의 최대 길이는 32767Byte이다.

OVERLAY 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.OVERLAY
(
    overlay_str     IN          RAW,
    target          IN          RAW,
    pos             IN          BINARY_INTEGER  DEFAULT 1,
    len             IN          BINARY_INTEGER  DEFAULT NULL,
    pad             IN          RAW             DEFAULT NULL
)
RETURN RAW;
```

- 파라미터

파라미터	설명
overlay_str	원본 바이너리 데이터이다.
target	대상 바이너리 데이터이다.
pos	대상 데이터 내의 위치로, 1이상의 값을 가져야 한다.
len	복사할 원본 데이터의 길이로, 0이상의 값을 가져야 한다. NULL이면 원본 데이터 전체를 복사한다.
pad	첨부할 Byte이다. NULL이면 '00'을 첨부한다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	– 원본 데이터가 NULL이거나 길이가 0인 경우이다. – 대상 데이터가 NULL인 경우이다. – 파라미터 pos가 1보다 작거나, 파라미터 len이 0보다 작은 경우이다. – 반환되는 바이너리 데이터 길이가 32767bytes를 초과한 경우이다.

- 예제

```
DECLARE
    bin RAW(100);
BEGIN
    bin := UTL_RAW.OVERLAY('ee', 'fabcdff', 2, 3, '12');
    DBMS_OUTPUT.PUT_LINE(bin);
END;
/
0FEE1212

PSM completed
SQL>
```

73.2.20. REVERSE

주어진 RAW 타입의 데이터의 Byte 순서를 역으로 변환하여 반환하는 함수이다. 반환되는 바이너리 데이터의 길이는 입력된 바이너리 데이터의 길이와 같다.

REVERSE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.REVERSE
(
    data        IN          RAW
)  RETURN RAW;
```

- 파라미터

파라미터	설명
data	역으로 변환할 바이너리 데이터이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	입력된 데이터가 NULL이거나 길이가 0인 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.REVERSE('abcd132'));
END;
/
32D1BC0A
```

```
PSM completed
SQL>
```

73.2.21. SUBSTR

주어진 RAW 타입의 데이터의 일부 또는 전체를 반환하는 함수이다. 입력된 데이터 내에서 파라미터 pos가 가리키는 위치로부터 파라미터 len이 지정한 길이만큼 반환한다. 바이너리 데이터가 NULL이면 결과값으로 NULL을 반환한다.

SUBSTR 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.SUBSTR
(
    data            IN          RAW,
    pos             IN          BINARY_INTEGER,
    len             IN          BINARY_INTEGER      DEFAULT NULL
)
RETURN RAW;
```

- 파라미터

파라미터	설명
data	바이너리 데이터이다.
pos	바이너리 데이터 내의 오프셋 위치이다. – pos가 1 이상이면 바이너리 데이터의 앞에서부터의 위치를 가리킨다. – pos가 –1 이하이면 바이너리 데이터의 뒤에서부터의 위치를 가리킨다. – pos는 0이 될 수 없다.
len	반환할 데이터의 길이이다. NULL이면 pos 위치로부터 바이너리 데이터의 끝까지 반환한다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	파라미터 pos가 0이거나 파라미터 len이 1보다 작은 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.SUBSTR('abc1234', 2, 2));
END;
/
BC12

PSM completed
SQL>
```

73.2.22. TRANSLATE

주어진 RAW 타입의 데이터 내의 지정된 Byte를 변환하는 함수이다. 주어진 바이너리 데이터 내에 from_set에 포함된 Byte가 존재하면 해당 Byte는 to_set 내의 같은 위치에 포함된 대응되는 Byte로 대체된다. to_set의 길이가 작아서 같은 위치에 대응되는 Byte가 없다면 빈 Byte로 대체된다.

이 함수는 변환된 데이터를 반환하며, 입력된 바이너리 데이터 자체는 갱신되지 않는다.

TRANSLATE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.TRANSLATE
(
    data            IN          RAW,
    from_set        IN          RAW,
    to_set          IN          RAW
)
RETURN RAW;
```

- 파라미터

파라미터	설명
data	바이너리 데이터이다.
from_set	변환할 Byte 집합이다.
to_set	변환 후의 Byte 집합이다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	파라미터 from_set 또는 to_set이 NULL이거나 길이가 0인 경우이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.TRANSLATE('abcde', 'bc', '77'));
END;
/
0A77DE

PSM completed
SQL>
```

73.2.23. TRANSLITERATE

주어진 RAW 타입의 데이터 내의 지정된 Byte를 변환하는 함수이다. 주어진 바이너리 데이터 내에 from_set에 포함된 Byte가 존재하면 해당 Byte는 to_set 내의 같은 위치에 포함된 대응되는 Byte로 대체된다. to_set의 길이가 작아서 같은 위치에 대응되는 Byte가 없다면 패드 값으로 대체된다.

이 함수는 변환된 데이터를 반환하며, 입력된 바이너리 데이터 자체는 갱신되지 않는다.

TRANSLITERATE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.TRANSLITERATE
(
    data            IN          RAW,
    to_set          IN          RAW     DEFAULT NULL,
    from_set        IN          RAW     DEFAULT NULL,
    pad             IN          RAW     DEFAULT NULL
)
RETURN RAW;
```

- 파라미터

파라미터	설명
data	바이너리 데이터이다.
to_set	변환 후의 Byte 집합이다.
from_set	변환할 Byte 집합이다.
pad	to_set의 길이가 from_set의 길이보다 짧을 때 사용될 1byte RAW이다. pad로 입력된 값은 길이에 관계없이 1byte만 사용하게 된다.

- 예외 상황

예외 상황	설명
VALUE_ERROR	data가 NULL이거나 길이가 0인 경우

- 예제

```
DECLARE
    bin RAW(100);
BEGIN
    bin := UTL_RAW.TRANSLITERATE('abcde', 'cd', 'bcde');
    DBMS_OUTPUT.PUT_LINE(bin);
END;
/
0ACD00

PSM completed
SQL>
```

73.2.24. XRANGE

주어진 start_byte RAW 데이터부터 end_byte RAW 데이터까지 1byte씩 증가시켜 나열한 결과를 반환하는 함수이다. start_byte가 end_byte보다 더 크다면 start_byte에서 시작해서 0xFF까지 증가시킨 후 0x00부터 다시 증가시켜 end_byte에서 끝내게 된다.

XRANGE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_RAW.XRANGE
(
    start_byte      IN          RAW     DEFAULT NULL,
    end_byte        IN          RAW     DEFAULT NULL
)
RETURN RAW;
```

- 파라미터

파라미터	설명
start_byte	결과 값의 시작 Byte이다.
end_byte	결과 값의 종료 Byte이다.

- 예제

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(UTL_RAW.XRANGE('a', 'd'));
```

```
END;
/
0A0B0C0D

PSM completed
SQL>
```

제74장 UTL_RECOMP

본 장에서는 UTL_RECOMP 패키지의 기본 개념과 패키지 내의 프러시저를 사용하는 방법을 설명한다.

74.1. 개요

UTL_RECOMP는 Invalid 상태의 오브젝트를 다시 컴파일하기 위한 프러시저를 제공한다. 다시 컴파일되는 오브젝트는 function, procedure, view, trigger, package, type 이다. 이 패키지는 SYS 유저만 수행할 수 있다. 또한 이 패키지의 프러시저가 수행되는 동안 tibero에 다른 DDL을 수행하지 말아야 한다. 이를 따르지 않으면 데드락을 야기할 수 있다.

74.2. 프러시저

본 절에서는 UTL_RECOMP 패키지에서 제공하는 프러시저를 알파벳 순으로 설명한다.

74.2.1. RECOMP_SERIAL

Invalid 오브젝트를 순서대로 다시 컴파일하는 프러시저이다.

RECOMP_SERIAL 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
PROCEDURE RECOMP_SERIAL
(
   schema IN VARCHAR2 DEFAULT NULL
);
```

- 파라미터

파라미터	설명
schema	어떤 owner의 오브젝트를을 다시 컴파일할 지 정해 준다. NULL인 경우 모든 Invalid 상태의 오브젝트들을 다시 컴파일한다.

- 예제

```

exec UTL_RECOMP.RECOMP_SERIAL();
/
```

제75장 UTL_SMTP

본 장에서는 UTL_SMTP 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

75.1. 개요

UTL_SMTP는 SMTP(Simple Mail Transfer Protocol)을 통해 전자 메일 (이메일)을 전송하도록 설계되었다. 해당 패키지는 많은 명령들에 대하여 절차 및 기능 인터페이스를 모두 제공한다. 기능적 형식으로 클라이언트에서 처리할 수 있도록 서버에서 응답을 반환한다. 절차적 형식으로 먼저 응답을 확인하고 응답이 일시적 또는 영구 오류를 나타내는 경우 예외를 발생시킨다.

75.2. 타입

본 절에서는 UTL_SMTP 패키지에서 제공하는 타입들을 설명한다.

75.2.1. CONNECTION Record Type

SMTP 연결을 나타내는데 사용되는 PL/SQL 레코드 타입이다.

CONNECTION Record Type 의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE connection IS RECORD (
    host              VARCHAR2(255),
    port              PLS_INTEGER,
    tx_timeout        PLS_INTEGER,
    private_tcp_con   utl_tcp.connection,
    private_state     PLS_INTEGER);
```

- 파라미터

파라미터	설명
host	연결이 설정될 때 원격 호스트로로 사용된다.
port	연결된 원격 SMTP 서버의 포트 번호이다.
tx_timeout	UTL_SMTP 연결에서 읽기 또는 쓰기 작업이 시간 초과되기 전에 패키지가 대기하는 시간이다.

파라미터	설명
private_tcp_con	해당 필드를 수정하면 안된다.
private_state	해당 필드를 수정하면 안된다.

75.2.2. REPLY,REPLIES Record Types

SMTP 응답을 나타내는 데 사용되는 PL/SQL 레코드 타입이다. 각 SMTP 응답은 응답 코드와 텍스트 메시지로 구성된다.

REPLY,REPLIES Record Types의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE reply IS RECORD (
  code    PLS_INTEGER,
  text    VARCHAR2(508));

TYPE replies IS TABLE OF reply INDEX BY BINARY_INTEGER;
```

- 파라미터

파라미터	설명
code	세 자리 응답 코드이다.
text	답장 메시지이다.

75.3. 프러시저와 함수

본 절에서는 UTL_SMTP 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

75.3.1. CLOSE_DATA 프러시저와 함수

해당 서브프로그램은 전자 메일 메시지를 종료한다.

CLOSE_DATA 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.CLOSE_DATA (
   c     IN OUT NOCOPY connection)
RETURN reply;
```

```
UTL_SMTP.CLOSE_DATA (
   c     IN OUT NOCOPY connection);
```

- 파라미터

파라미터	설명
c	SMTP 연결이다.

- 반환값

파라미터	설명
reply	명령에 대한 응답 회신이 여러 개인 경우 마지막 회신이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.2. COMMAND 프러시저와 함수

해당 서브프로그램은 일반 SMTP 명령을 수행한다.의 설명을 참고한다.

COMMAND 프러시저와 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.COMMAND (
   c     IN OUT NOCOPY    connection,
   cmd   IN               VARCHAR2,
   arg   IN               VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.COMMAND (
   c     IN OUT NOCOPY    connection,
   cmd   IN               VARCHAR2,
   arg   IN               VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
c	SMTP를 연결한다.
cmd	서버에 보낼 SMTP 명령이다.
arg	SMTP 인자에 대한 추가적인 인자이다. cmd와 arg사이에는 공백이 들어간다.

- 반환값

파라미터	설명
reply	명령에 대한 응답. 회신이 여러 개인 경우 마지막 회신이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.3. COMMAND_REPLIES 함수

해당 함수는 일반 SMTP 명령을 수행하고 여러 응답을 회수한다. 의 설명을 참고한다.

COMMAND_REPLIES 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.COMMAND_REPLIES (
   c     IN OUT NOCOPY    connection,
   cmd   IN               VARCHAR2,
   arg   IN               VARCHAR2 DEFAULT NULL)
RETURN replies;
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
cmd	서버에 보낼 SMTP 명령이다.
arg	SMTP 인자에 대한 추가적인 인자이다. cmd와 arg사이에는 공백이 들어간다.

- 반환값

파라미터	설명
replies	명령에 대한 응답. 회신이 여러 개인 경우 마지막 회신이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.4. DATA 프러시저와 함수

해당 서브프로그램은 이메일 메시지의 body 부분에 해당한다.

DATA 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.DATA (
   c     IN OUT NOCOPY connection
   body  IN  VARCHAR2 CHARACTER SET ANY_CS)
RETURN reply;

UTL_SMTP.DATA (
   c     IN OUT NOCOPY connection
   body  IN VARCHAR2 CHARACTER SET ANY_CS);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
body	헤더들을 포함한 보낼 메시지이다.

- 반환값

파라미터	설명
reply	명령에 대한 응답이다. 회신이 여러 개인 경우 마지막 회신이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.5. EHLO 프러시저와 함수

해당 서브프로그램은 EHLO명령을 이용하여, SMTP 서버와 초기 핸드쉐이크를 수행한다.

EHLO 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.EHLO (
   c       IN OUT NOCOPY connection,
   domain  IN)
RETURN replies;

UTL_SMTP.EHLO (
   c       IN OUT NOCOPY connection,
   domain  IN);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
domain	로컬(송신) 호스트의 도매인 이름이다. 식별 목적으로 사용된다.

- 반환값

파라미터	설명
replies	명령에 대한 응답이다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.6. HELO 프러시저와 함수

해당 서브프로그램은 HELO명령을 이용하여, SMTP 서버와 초기 핸드쉐이크를 수행한다.

HELO 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.HELO (
   c       IN OUT NOCOPY   connection,
   domain  IN              VARCHAR2)
RETURN reply;

UTL_SMTP.HELO (
   c       IN OUT NOCOPY   connection,
   domain  IN              VARCHAR2);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
domain	로컬(송신) 호스트의 도매인 이름이다. 식별 목적으로 사용된다.

- 반환값

파라미터	설명
reply	명령에 대한 응답이다. 응답이 여러개 있으면 마지막 응답이 반환된다.

- 예제

단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.7. HELP 함수

해당 함수는 HELP 명령을 전송한다.

HELP 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.HELP (
   c          IN OUT NOCOPY   connection,
   command    IN              VARCHAR2 DEFAULT NULL)
RETURN replies;
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
command	help 메시지를 받기 위한 명령이다.

- 반환값

파라미터	설명
replies	명령에 대한 응답이다.

- 예제

단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.8. MAIL 프러시저와 함수

해당 서브프로그램은 서버와의 메일 트랜잭션을 시작한다.

MAIL 프러시저와 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.MAIL (
   c           IN OUT NOCOPY   connection,
   sender      IN              VARCHAR2,
   parameters  IN              VARCHAR2 DEFAULT NULL)
RETURN reply;
```

```
UTL_SMTP.MAIL (
   c           IN OUT NOCOPY   connection,
   sender      IN              VARCHAR2,
   parameters  IN              VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
sender	메시지를 보내는 사용자의 이메일 주소이다.
parameters	명령에 대한 추가 매개변수이다.

- 반환값

파라미터	설명
replies	명령에 대한 응답이다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.9. NOOP 프러시저와 함수

해당 서브프로그램은 NULL 명령을 실행한다. 서버와 연결이 되어있는지 확인하는 용도로 쓰인다.

NOOP 프러시저와 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.NOOP (
   c  IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.NOOP (
   c  IN OUT NOCOPY connection);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.

- 반환값

파라미터	설명
reply	명령에 대한 응답이다. 응답이 여러 개일 경우 마지막 응답이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.10. OPEN_CONNECTION 함수

해당 함수는 SMTP 서버로 연결하는 함수이다.

OPEN_CONNECTION 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.OPEN_CONNECTION (
   host                             IN  VARCHAR2,
   port                             IN  PLS_INTEGER DEFAULT 25,
   c                                OUT connection,
   tx_timeout                       IN  PLS_INTEGER DEFAULT NULL)
 RETURN reply;

UTL_SMTP.OPEN_CONNECTION (
   host                            IN  VARCHAR2,
   port                            IN  PLS_INTEGER DEFAULT 25,
   tx_timeout                      IN  PLS_INTEGER DEFAULT NULL)
RETURN connection;
```

- 파라미터

파라미터	설명
host	SMTP 서버 호스트의 이름이다.
port	SMTP 서버가 듣고 있는 포트 번호이다. (주로 25번)
c	SMTP로 연결한다.
tx_timeout	UTL_SMTP 연결에 대하여 읽기 또는 쓰기 작업에서 시간이 초과되기 전에 패키지가 대기하는 시간이다. 읽는 작업에서 즉시 읽을 수 있는 데이터가 없으면 시간이 초과된다. 쓰는 작업에서 패키지의 출력 버퍼가 가득차고 차단되지 않은 데이터가 네트워크로 전송되지 않으면 시간이 초과된다.

- 반환값

파라미터	설명
reply	명령에 대한 응답. 응답이 여러 개일 경우 마지막 응답이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.11. OPEN_DATA 프러시저와 함수

해당 서브프로그램은 WRITE_DATA 및 WRITE_RAW_DATA를 사용하여 메일 메시지를 쓸 수 있도록 명령을 보낸다.

OPEN_DATA 프러시저와 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.OPEN_DATA (
   c     IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.OPEN_DATA (
   c     IN OUT NOCOPY connection);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
data	헤더를 포함한 메시지이다.

- 반환값

파라미터	설명
reply	명령에 대한 응답. 응답이 여러 개일 경우 마지막 응답이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.12. QUIT 프러시저와 함수

해당 서브프로그램은 SMTP 세션을 종료하고 서버와의 연결을 끊는다.

QUIT 프러시저와 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.QUIT (
   c  IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.QUIT (
   c  IN OUT NOCOPY connection);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.

- 반환값

파라미터	설명
reply	명령에 대한 응답이다. 응답이 여러 개일 경우 마지막 응답이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.13. RCPT 함수

해당 서브프로그램은 전자 메일 메시지의 수신자를 지정한다.

RCPT 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.RCPT (
   c           IN OUT NOCOPY     connection,
   recipient   IN                VARCHAR2,
   parameters  IN                VARCHAR2 DEFAULT NULL)
RETURN reply;

UTL_SMTP.RCPT (
   c           IN OUT NOCOPY     connection,
   recipient   IN                VARCHAR2,
   parameters  IN                VARCHAR2 DEFAULT NULL);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.

파라미터	설명
recipient	메시지가 전송되는 사용자의 이메일 주소이다.
parameters	RCPT 명령에 대한 추가 매개 변수이다.

- 반환값

파라미터	설명
reply	명령에 대한 응답이다. 응답이 여러 개일 경우 마지막 응답이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.14. RSET 프러시저와 함수

해당 서브프로그램은 현재 메일 트랜잭션을 종료한다.

RSET 프러시저와 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.RSET (
   c  IN OUT NOCOPY connection)
RETURN reply;

UTL_SMTP.RSET (
   c  IN OUT NOCOPY connection);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.

- 반환값

파라미터	설명
reply	명령에 대한 응답이다. 응답이 여러 개일 경우 마지막 응답이 반환된다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.15. VRFY 함수

해당 함수는 대상 이메일 주소의 유효성을 확인한다.

VRFY 함수 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.VRFY (
   c          IN OUT NOCOPY connection
   recipient  IN VARCHAR2)
RETURN reply;
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
recipient	확인할 메일 주소이다.

- 반환값

파라미터	설명
reply	SMTP 응답을 반환한다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.16. WRITE_DATA 프러시저

해당 프로시저는 전자 메일 메시지를 작성한다. WRITE_DATA에 대한 반복 호출은 전자 메일 메시지에 데이터를 추가 한다.

WRITE_DATA 프러시저 의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.WRITE_DATA (
   c     IN OUT NOCOPY connection,
   data  IN VARCHAR2);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
data	헤더를 포함한 메시지 텍스트 부분이다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.3.17. WRITE_RAW_DATA 프러시저

해당 프로시저는 전자 메일 메시지를 작성한다. WRITE_RAW_DATA에 대한 반복 호출은 전자 메일 메시지에 데이터를 추가한다.

WRITE_RAW_DATA 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_SMTP.WRITE_RAW_DATA (
   c     IN OUT NOCOPY connection
   data  IN RAW);
```

- 파라미터

파라미터	설명
c	SMTP로 연결한다.
data	헤더를 포함한 메시지 텍스트 부분이다.

- 예제

  단일 함수 사용의 예가 아니므로, “75.4. 예제”를 참고한다.

75.4. 예제

다음은 OPEN_CONNECTION, HELO, MAIL, RCPT, OPEN_DATA, WRITE_DATA, CLOSE_DATA, QUIT를 이용하여 SMTP를 수행하는 예시이다. 25번 포트를 이용하여 SMTP 서버에 연결하고 간단한 문자 메시지를 보낸다.

```
DECLARE

   t_host VARCHAR2(30) := 'localhost';

   t_port NUMBER := 25;
```

```
 t_domain VARCHAR2(30) := 'tmax.co.kr';



 t_from VARCHAR2(50) := 'tibero@tmax.co.kr';

 t_to VARCHAR2(50) := 'tmax@tmax.co.kr';

 t_intro VARCHAR2(50) := 'This is mail from TMAX';

 l_encoded_username VARCHAR2(1000);

 l_encoded_password VARCHAR2(1000);

 c utl_smtp.connection;

 reply_rset utl_smtp.reply;

 reply_vrfy utl_smtp.reply;

 reply_help utl_smtp.replies;

 reply_command utl_smtp.reply;

 reply_command_replies utl_smtp.replies;

BEGIN

  c := UTL_SMTP.OPEN_CONNECTION(t_host,t_port);

  reply_vrfy := utl_smtp.vrfy(c, 'tmax@tmax.co.kr');

  reply_help := utl_smtp.help(c, 'HELP');

  reply_command := utl_smtp.command(c, 'AUTH LOGIN');

  reply_command_replies := utl_smtp.command_replies(c, 'AUTH','LOGIN');

  dbms_output.put_line('vrfy:'||reply_vrfy.code);

  dbms_output.put_line('vrfy:'||reply_vrfy.text);

  dbms_output.put_line('help:'||reply_help(reply_help.count).code);

  dbms_output.put_line('help:'||reply_help(reply_help.count).text);

  dbms_output.put_line('command:'||reply_command.code);
```

```
    dbms_output.put_line('command:'||reply_command.text);

    dbms_output.put_line('command_replies:'||reply_command_replies(
reply_command_replies.count).code);
    dbms_output.put_line('command_replies:'||reply_command_replies(
reply_command_replies.count).text);



    reply_rset := utl_smtp.rset(c);

    dbms_output.put_line('rset:'||reply_rset.code);

    dbms_output.put_line('rset:'||reply_rset.text);



    UTL_SMTP.HELO(c, t_domain); --UTL_SMTP.EHLO(c, t_domain);

    UTL_SMTP.NOOP(c);

    UTL_SMTP.MAIL(c, t_from);

    UTL_SMTP.RCPT(c, t_to);



    --reply_rset := utl_smtp.rset(c);

    --dbms_output.put_line('rset:'||reply_rset.code);

    --dbms_output.put_line('rset:'||reply_rset.text);



    UTL_SMTP.OPEN_DATA(c);

    UTL_SMTP.WRITE_DATA(c,'From:' || '"tibero" <tibero@tmax.co.kr>' || UTL_TCP.CRLF);

    UTL_SMTP.WRITE_DATA(c,'To:' || '"tmax" <tmax@tmax.co.kr>' || UTL_TCP.CRLF);

   UTL_SMTP.WRITE_RAW_DATA( c, UTL_RAW.CAST_TO_RAW(''|| t_intro||''|| UTL_TCP.CRLF));

    UTL_SMTP.WRITE_DATA(c,'Subject: Test' || UTL_TCP.CRLF);

    UTL_SMTP.WRITE_DATA(c, UTL_TCP.CRLF);

    UTL_SMTP.WRITE_DATA(c,'THIS IS SMTP_TEST1' || UTL_TCP.CRLF);
```

```
    UTL_SMTP.CLOSE_DATA(c);

    UTL_SMTP.QUIT(c);

  EXCEPTION

    WHEN utl_smtp.transient_error OR utl_smtp.permanent_error THEN

      BEGIN

        UTL_SMTP.QUIT(c);

      EXCEPTION

        WHEN UTL_SMTP.TRANSIENT_ERROR OR UTL_SMTP.PERMANENT_ERROR THEN

          NULL;

      END;

      raise_application_error(-20000,

        'Failed to send mail due to the following error: ' || sqlerrm);

  END;

  /


CREATE OR REPLACE PROCEDURE send_email
      ( sender    IN VARCHAR2,
        recipient IN VARCHAR2,
        message   IN VARCHAR2)
      AS
        mailhost VARCHAR2(100) := 'localhost';
        c utl_smtp.connection;
      BEGIN
         c :=utl_smtp.open_connection(mailhost,25);
         utl_smtp.helo(c,mailhost);
         utl_smtp.mail(c,sender);
         utl_smtp.rcpt(c,recipient);
         utl_smtp.data(c,message);
         utl_smtp.quit(c);
     END;

SQL> exec send_email('tibero@tmax.co.kr','tmax@tmax.co.kr',
'This sample is education purpose only');
```

제76장 UTL_TCP

본 장에서는 UTL_TCP 패키지의 기본 개념과 패키지 내의 프러시저와 함수를 사용하는 방법을 설명한다.

76.1. 개요

UTL_TCP은 TCP/IP 클라이언트 역할을 수행할 수 있는 프러시저 및 함수들을 제공하는 패키지이다.

많은 인터넷 환경이 TCP/IP 프로토콜에 기반하므로, 이메일이나 인터넷 연결 등에 유용하게 사용될 수 있다.

76.2. 타입

본 절에서는 UTL_TCP 패키지에 제공하는 별도 정의된 타입들을 알파벳 순으로 설명한다.

76.2.1. CONNECTION

UTL_TCP를 이용한 TCP/IP 연결의 객체 및 연결 정보를 담고 있는 레코드 타입이다.

CONNECTION 타입의 세부 내용은 다음과 같다.

- 프로토타입

```
TYPE CONNECTION IS RECORD
(
    pvid            PLS_INTEGER,
    remote_host     VARCHAR2(255),
    remote_port     PLS_INTEGER,
    tx_timeout      PLS_INTEGER
);
```

- 필드

필드 이름	설명
pvid	TCP/IP의 연결 객체를 지칭하는 ID이다. 변경하지 않도록 한다.
remote_host	연결된 TCP 서버의 주소이다.
remote_port	연결된 TCP 서버의 포트 번호이다.

필드 이름	설명
tx_timeout	TCP 통신을 이용한 읽기/쓰기의 시간 제한으로, 제한 시간이 지나면 타임아웃 에러와 함께 반환한다. (단위: 초) – NULL : 응답이 올 때까지 기다린다. – 0 : 전송 후 대기 없이 즉시 빠져나온다.

76.3. 상수

76.3.1. CRLF

캐리지 리턴 문자, 라인 피드 문자를 가지는 문자열이다.

- 정의

```
    CRLF CONSTANT VARCHAR2(2) := unistr('\000D\000A');
```

76.4. 프러시저와 함수

본 절에서는 UTL_TCP 패키지에서 제공하는 프러시저와 함수를 알파벳 순으로 설명한다.

76.4.1. CLOSE_ALL_CONNECTIONS

세션에서 연결 중인 TCP/IP 연결을 모두 끊는 프러시저이다.

CLOSE_ALL_CONNECTIONS 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.CLOSE_ALL_CONNECTIONS
```

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 25);
    UTL_TCP.CLOSE_CONNECTION(c);
EXCEPTION WHEN OTHERS THEN
    UTL_TCP.CLOSE_ALL_CONNECTIONS;
END;
```

76.4.2. CLOSE_CONNECTION

연결 중인 TCP/IP 연결을 끊는 프러시저이다.

CLOSE_CONNECTION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.CLOSE_CONNECTION
(
    c IN OUT NOCOPY connection
)
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 25);
    UTL_TCP.CLOSE_CONNECTION(c);
END;
```

76.4.3. GET_LINE

원격 TCP 서버로부터 받은 문자열 한 줄을 가져오는 함수이다. 자세한 내용은 동일한 기능을 수행하는 READ_LINE 함수의 설명을 참고한다.

GET_LINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.GET_LINE
(
    c               IN OUT NOCOPY   connection,
    remove_crlf     IN              BOOLEAN     DEFAULT FALSE,
    peek            IN              BOOLEAN     DEFAULT FALSE
)
RETURN VARCHAR2
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
remove_crlf	TRUE일 경우 문자열에 포함된 CR/LF 문자열을 제거하여 반환한다.
peek	TRUE일 경우 TCP input queue로부터 읽어온 데이터를 비우지 않고 보존하여 이후 다시 읽어올 경우 같은 데이터를 읽게 된다.

- 반환값

  읽어들인 문자열이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    data VARCHAR2(100);
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    data := UTL_TCP.GET_LINE(c);
END;
```

76.4.4. GET_RAW

원격 TCP 서버로부터 받은 바이너리 데이터를 가져오는 함수이다. 자세한 내용은 동일한 기능을 수행하는 READ_RAW 함수의 설명을 참고한다.

GET_RAW 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.GET_RAW
(
    c       IN OUT NOCOPY   connection,
    len     IN              PLS_INTEGER DEFAULT 1,
    peek    IN              BOOLEAN     DEFAULT FALSE
)
RETURN RAW
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
len	읽어올 바이너리 데이터의 Byte 갯수이다. (기본값: 1)

파라미터	설명
peek	TRUE일 경우 TCP input queue로부터 읽어온 데이터를 비우지 않고 보존하여 이후 다시 읽어올 경우 같은 데이터를 읽게 된다.

- 반환값

  읽어들인 바이너리 데이터이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    data RAW(100);
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    data := UTL_TCP.GET_RAW(c, 10);  -- 10Byte의 문자열을 읽어올 것을 요청
END;
```

76.4.5. GET_TEXT

원격 TCP 서버로부터 받은 문자열을 가져오는 함수이다. 자세한 내용은 동일한 기능을 수행하는 READ_TEXT 함수의 설명을 참고한다.

GET_TEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.GET_TEXT
(
    c      IN OUT NOCOPY   connection,
    len    IN              PLS_INTEGER DEFAULT 1,
    peek   IN              BOOLEAN     DEFAULT FALSE
)
RETURN VARCHAR2
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
len	읽어올 문자열의 Byte 갯수이다. (기본값: 1)
peek	TRUE일 경우 TCP input queue로부터 읽어온 데이터를 비우지 않고 보존하여 이후 다시 읽어올 경우 같은 데이터를 읽게 된다.

- 반환값

읽어들인 문자열이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    data VARCHAR2(100);
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    data := UTL_TCP.GET_TEXT(c, 10);  -- 10Byte의 문자열을 읽어올 것을 요청
END;
```

76.4.6. OPEN_CONNECTION

외부 TCP 서버로의 TCP/IP 연결을 맺는 프러시저이다.

OPEN_CONNECTION 프러시저의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.OPEN_CONNECTION
(
    remote_host     IN  VARCHAR2,
    remote_port     IN  PLS_INTEGER,
    tx_timeout      IN  PLS_INTEGER DEFAULT NULL
)  RETURN UTL_TCP.CONNECTION
```

- 파라미터

파라미터	설명
remote_host	연결할 TCP 서버의 주소 이름이다. NULL이면 localhost로 자동 설정된다.
remote_port	연결할 TCP 서버의 포트 번호이다.
tx_timeout	TCP 통신을 이용한 읽기/쓰기의 시간 제한으로, 제한 시간이 지나면 타임아웃 에러와 함께 반환한다. (단위: 초) – NULL : 응답이 올 때까지 기다린다. – 0 : 전송 후 대기 없이 즉시 빠져나온다.

- 반환값

TCP/IP 연결 정보를 갖는 UTL_TCP.CONNECTION 타입의 객체이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 25);
END;
```

76.4.7. READ_LINE

원격 TCP 서버로부터 받은 문자열 한 줄을 가져오는 함수이다.

타임아웃이 설정된 경우 타임아웃 시간까지 기다렸다가 만료되었을 때 TRANSFER_TIMEOUT 예외를 발생시킨다. 타임아웃이 NULL일 경우에는 무한정 기다리므로 주의하여 사용하도록 한다.

READ_LINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.READ_LINE
(
    c               IN OUT NOCOPY   connection,
    data            IN OUT NOCOPY   VARCHAR2,
    remove_crlf     IN              BOOLEAN     DEFAULT FALSE,
    peek            IN              BOOLEAN     DEFAULT FALSE
)
RETURN PLS_INTEGER
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
data	읽어들인 문자열이다.
remove_crlf	TRUE일 경우 문자열에 포함된 CR/LF 문자열을 제거하여 반환한다.
peek	TRUE일 경우 TCP input queue로부터 읽어온 데이터를 비우지 않고 보존하여 이후 다시 읽어올 경우 같은 데이터를 읽게 된다.

- 반환값

  읽어들인 문자열의 문자 갯수이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    data VARCHAR2(100);
    len PLS_INTEGER;
```

```
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    len := UTL_TCP.READ_LINE(c, data);
END;
```

76.4.8. READ_RAW

원격 TCP 서버로부터 받은 바이너리 데이터를 가져오는 함수이다.

타임아웃이 설정된 경우 타임아웃 시간까지 기다렸다가 만료되었을 때 TRANSFER_TIMEOUT 예외를 발생시킨다. 타임아웃이 NULL일 경우에는 무한정 기다리므로 주의하여 사용하도록 한다.

READ_RAW 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.READ_RAW
(
    c      IN OUT NOCOPY   connection,
    data   IN OUT NOCOPY   RAW,
    len    IN              PLS_INTEGER DEFAULT 1,
    peek   IN              BOOLEAN     DEFAULT FALSE
)
RETURN PLS_INTEGER
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
data	읽어들인 문자열이다.
len	읽어올 바이너리 데이터의 Byte 갯수이다.
peek	TRUE일 경우 TCP input queue로부터 읽어온 데이터를 비우지 않고 보존하여 이후 다시 읽어올 경우 같은 데이터를 읽게 된다.

- 반환값

  읽어들인 데이터의 Byte 갯수이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    data RAW(100);
    len PLS_INTEGER;
BEGIN
```

```
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    len := UTL_TCP.READ_RAW(c, data, 10);
END;
```

76.4.9. READ_TEXT

원격 TCP 서버로부터 받은 문자열을 가져오는 함수이다.

타임아웃이 설정된 경우 타임아웃 시간까지 기다렸다가 만료되었을 때 TRANSFER_TIMEOUT 예외를 발생시킨다. 타임아웃이 NULL일 경우에는 무한정 기다리므로 주의하여 사용하도록 한다.

READ_TEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.READ_TEXT
(
    c     IN OUT NOCOPY   connection,
    data  IN OUT NOCOPY   VARCHAR2,
    len   IN              PLS_INTEGER DEFAULT 1,
    peek  IN              BOOLEAN     DEFAULT FALSE
)
RETURN PLS_INTEGER
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
data	읽어들인 문자열이다.
len	읽어올 문자열의 문자 갯수이다.
peek	TRUE일 경우 TCP input queue로부터 읽어온 데이터를 비우지 않고 보존하여 이후 다시 읽어올 경우 같은 데이터를 읽게 된다.

- 반환값

  읽어들인 데이터의 Byte 갯수이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    data VARCHAR2(100);
    len PLS_INTEGER;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
```

```
    len := UTL_TCP.READ_TEXT(c, data, 10);
END;
```

76.4.10. WRITE_LINE

원격 TCP 서버로 한줄의 문자열을 전송하는 함수이다. 입력문자열 뒤에 새줄 문자열이 붙어서 전송된다.

WRITE_LINE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.WRITE_LINE
(
    c    IN OUT NOCOPY   connection,
    data IN              VARCHAR2    DEFAULT NULL
)
RETURN PLS_INTEGER
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
data	전송할 문자열이다.

- 반환값

  실제 전송된 데이터의 문자열 갯수이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    rc INTEGER;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    rc := UTL_TCP.WRITE_LINE(c, 'GET / HTTP/1.0');
END;
```

76.4.11. WRITE_RAW

원격 TCP 서버로 바이너리 형태의 데이터를 보내는 함수이다.

WRITE_RAW 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.WRITE_RAW
(
    c     IN OUT NOCOPY   connection,
    data  IN              RAW,
    len   IN              PLS_INTEGER DEFAULT NULL
)
RETURN PLS_INTEGER
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
data	전송할 바이너리 데이터이다.
len	전송할 바이너리 데이터의 Byte 수이다. NULL이면 데이터 전체를 전송한다.

- 반환값

  실제 전송된 데이터의 Byte 수이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    rc INTEGER;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 80);
    rc := UTL_TCP.WRITE_RAW(c, '0D0A'); -- 0d 0a의 hex값을 가진 바이너리 데이터 전송
END;
```

76.4.12. WRITE_TEXT

원격 TCP 서버로 문자열 형태의 데이터를 보내는 함수이다.

WRITE_TEXT 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
UTL_TCP.WRITE_TEXT
(
    c    IN OUT NOCOPY   connection,
    data IN              VARCHAR2,
    len  IN              PLS_INTEGER DEFAULT NULL
```

```
)
RETURN PLS_INTEGER
```

- 파라미터

파라미터	설명
c	기존에 연결된 연결 객체이다. 타입은 UTL_TCP.CONNECTION이다.
data	전송할 문자열 데이터이다.
len	전송할 문자열의 문자 갯수이다. NULL이면 문자열 전체를 전송한다.

- 반환값

  실제 전송된 문자열의 문자 갯수이다.

- 예제

```
DECLARE
    c UTL_TCP.CONNECTION;
    rc INTEGER;
BEGIN
    c := UTL_TCP.OPEN_CONNECTION('localhost', 8080);
    rc := UTL_TCP.WRITE_TEXT(c, 'TIBEROTMAX', 6); -- TIBEROTMAX 문자열 중
                                                  TIBERO 문자열만 전송한다.
END;
```

제77장 UTL_URL

본 장에서는 UTL_URL 패키지의 기본 개념과 패키지 내의 함수를 사용하는 방법을 설명한다.

77.1. 개요

UTL_URL는 URL 주소를 ESCAPE 형태로 변환하기 위한 함수를 제공하는 패키지이다. ESCAPE 형태는 Legal Characters와 Reserved characters를 제외한 문자를 %xx{ASCII의 16진수 표현}으로 변경하는 것을 말한다.

문자 구분	정규 표현식
Unreserved Characters	[0-9] | [a-zA-Z] | - | _ | . | ! | ~ |* | ` | (|)
Reserved characters	; | / | ? | ; | @ | & | = | + | $ | ,

77.2. 함수

본 절에서는 UTL_URL 패키지에서 제공하는 함수를 알파벳 순으로 설명한다.

77.2.1. ESCAPE

URL 주소를 ESCAPE 형태로 변환하는 함수이다.

ESCAPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION ESCAPE
(
    url                     IN  VARCHAR2,
    escape_reserved_chars   IN  BOOLEAN  DEFAULT FALSE,
    url_charset             IN  VARCHAR2 DEFAULT utl_http.get_body_charset
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
url	ESCAPE 형태로 변경할 URL 이름이다.

파라미터	설명
escape_reserved_chars	Reserved characters를 ESCAPE 형태로 변환할지의 여부를 결정한다. – TRUE : 변환한다. – FALSE : 변환하지 않는다. (기본값)
url_charset	URL 주소의 문자 집합이다. 기본값은 UTL_HTTP 패키지의 기본값을 사용하게 되며, UTL_HTTP의 BODY 문자집합의 기본값은 'ISO-8859-1' 이다.

- 예제

```
BEGIN
  dbms_output.put_line( UTL_URL.ESCAPE('http://www.w3.org/blah blah.html') );
END;
/
```

77.2.2. UNESCAPE

ESCAPE 형태(%xx)를 포함하는 URL 주소를 ESCAPE 형태가 아닌 일반 문자열로 변환하는 함수이다.

UNESCAPE 함수의 세부 내용은 다음과 같다.

- 프로토타입

```
FUNCTION UNESCAPE
(
    url             IN  VARCHAR2,
    url_charset     IN  VARCHAR2 DEFAULT utl_http.get_body_charset
)
RETURN VARCHAR2;
```

- 파라미터

파라미터	설명
url	ESCAPE 형태의 URL 주소의 문자열이다.
url_charset	URL 주소의 문자 집합이다. 기본값은 UTL_HTTP 패키지의 기본값을 사용하게 되며, UTL_HTTP의 BODY 문자집합의 기본값은 'ISO-8859-1' 이다.

- 예외 상황

예외 상황	설명
URL_BAD_URL	잘못된 ESCAPE 문자열을 포함하는 경우이다.

- 예제

```
DECLARE
    ourl VARCHAR2(32767);
    eurl VARCHAR2(32767);
BEGIN
    eurl := UTL_URL.ESCAPE('http://www.w3.org/Protocols/');
    ourl := UTL_URL.UNESCAPE(eurl);
END;
/
```

색인

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Z

ㅂ

ㅅ

ㅇ

ㅋ

ㅌ

ㅍ

패키지 구현부, 2
패키지 선언부, 1