Real Alternative DBMS ALTIBASE, Since 1999

ORACLE to ALTIBASE 변환 가이드

ALTIBASE 5.3.3

2010. 03

ALTIBASE
PERFORMANCE SOLUTIONS

Document Control

Change Record

Date	Author	Change Reference
2010-03-26	snkim	Created

Reviews

Date	Name (Position)
2010-04-27	leekmo(SP), lim272(SP), okseop7(TS), wlgml337(TC)

Distribution

Name	Location

목차

개요

본 문서는 ORACLE에서 ALTIBASE로 변환할 때 고려할 사항과 변환 방법에 대해 설명한다.

ORACLE 10g 와 ALTIBASE 5.3 버전을 대상으로 한다.

아래 문서를 사전에 참고할 것을 권장한다.

- ALTIBASE, ORACLE 비교 자료

OBJECT 변환

ORACLE의 OBJECT를 ALTIBASE로 변환할 때 고려할 사항에 대해 기술한다.

DATATYPE

ORACLE의 TABLE을 ALTIBASE로 변환할 때 각각의 DATATYPE을 어떻게 변환하는지에 대해 설명한다.

분류	ORACLE	ALTIBASE	비고
문자 타입	CHAR	CHAR	최대 32K
	VARCHAR2, VARCHAR	VARCHAR2, VARCHAR	최대 32K. DESC로 조회 시 VARCHAR로 조회
	NCHAR	NCHAR	문자 길이 최대 16000(UTF16), 문자 길이 최대 10666(UTF8)
	NVARCHAR2	NVARCHAR	문자 길이 최대 16000(UTF16), 문자 길이 최대 10666(UTF8)
	LONG	CLOB	최대 2G
LOB 타입	BLOB	BLOB	최대 2G
	CLOB	CLOB	최대 2G
	NCLOB	CLOB	최대 2G
숫자 타입	NUMERIC(p, s)	NUMERIC(p, s)	데이터가 SMALLINT, INTEGER, BIGINT, REAL, DOUBLE등 native type으로 지정 가능 하다면 native type으로 지정하는 것이 좋다. 데이터 처리시 변환 비용에 따른 overhead를 줄일 수 있고, 저장 공간의 효율성이 좋아지기 때문이다. FLOAT 지정 시 precision을 지정하지 않을 경우 23byte로 지정되므로, 적절한 값을 지정 해주는 것이 좋다.
	NUMBER (*p*, *s*)	NUMBER(p, s)	
	DECIMAL(p, s)	DECIMAL(p, s)	
	FLOAT(p), BINARY_FLOAT	FLOAT(p)	
	SMALLINT	SMALLINT	2 byte 정수형 타입
	INT	INTEGER	4 byte 정수형 타입
	REAL	REAL	4 byte 실수형 타입
	BINARY_DOUBLE	DOUBLE	8 byte 실수형 타입
날짜 타입	DATE	DATE	ALTIBASE DATE 타입은 ORACLE DATE 타입의 표현 범위를 포함

분류	ORACLE	ALTIBASE	비고
	INTERVAL YEAR TO MONTH	-	지원 안됨
	INTERVAL DAY TO SECOND	-	
	TIMESTAMP WITH TIME ZONE	-	
	TIMESTAMP WITH LOCAL TIME ZONE	-	
	TIMESTAMP	DATE	ORACLE TIMESTAMP(fractional second precision) 타입의 fractional second precision의 값은 0~9 까지 표현 가능하지만, ALTIBASE DATE 타입은 microsecond까지 표현이 가능하다. 즉, ALTIBASE의 DATE 타입은 ORACLE의 TIMESTAMP(6)과 같다.
이진 타입	BFILE	BLOB	최대 2G
	RAW (*size*)	BLOB	
	LONG RAW	BLOB	

OBJECT 비교

ORACLE	ALTIBASE
CLUSTER	지원하지 않음
CONSTRAINT	CHECK는 지원하지 않음. 그 외 모두 지원
DATABASE LINK	SELECT만 가능
DATABASE TRIGGER	지원. BEFORE UPDATE, DDL TRIGGER는 지원하지 않음
DIMENSTION	지원하지 않음
EXTERNAL PROCEDURE LIBRARY	지원하지 않음
INDEX-ORGANIZED TABLE	지원하지 않음
INDEX	BTREE, RTREE만 지원. BITMAP, CLUSTER, Function based, Global partitioned INDEX는 지원하지 않음
INDEXTYPE	지원하지 않음

JAVA 관련 객체	지원하지 않음
MATERIALIZED VIEW/ MATERIALIZED VIEW LOG	지원하지 않음
OBJECT TABLE	지원하지 않음
OBJECT TYPE	지원하지 않음
OBJECT VIEW	지원하지 않음
OPERATOR	지원하지 않음
PACKAGE	지원하지 않음
SEQUENCE	지원
STORED FUNCTION/PROCEDURE	지원
SYNONYM	지원
TABLE	지원
VIEW	지원. view를 통해 DML 불가
CONTEXT	지원하지 않음
DIRECTORY	지원
PARAMETER FILE	객체로서 지원하지 않음. altibase.properties 파일로 지원
PROFILE	지원하지 않음
ROLE	지원하지 않음
TABLESPACE	DATA TABLESPACE, TEMPORARY TABLESPACE 지원. UNDO TABLESPACE는 ALTIBASE에서 자동으로 1 개만 생성하며 사용자가 생성할 수 없다.
USER	지원. USER 생성 시 CREATE SESSION 권한과 객체를 생성할 수 있는 권한이 자동으로 부여

CREATE TABLESPACE

ORACLE의 TABLESPACE는 모두 DISK TABLESPACE이다. 따라서 ALTIBASE로 변환할 때 CREATE DISK DATA TABLESPACE 구문을 이용해야 한다. 이 때, DISK DATA를 생략해도 default로 DISK TABLESPACE가 생성되므로 CREATE TABLESPACE 구문으로 변환이 가능하다. 다음은 ORACLE의 TABLESPACE를 ALTIBASE로 변환할 때 CREATE TABLESPACE 구문에서 지정하는 여러 옵션들의 변환 가이드를 설명한 것이다.

1. DATA TABLESPACE

ORACLE의 DATA TABLESPACE는 모두 DISK TABLESPACE이다. ALTIBASE는 CREATE TABLESPACE 구문을 이용하여 TABLESPACE를 생성하면 DISK

TABLESPACE로 생성되므로 ORACLE과 동일한 CREATE TABLESPACE 구문으로 변환이 가능하다. 다음은 ORACLE의 TABLESPACE를 ALTIBASE로 변환할 때 CREATE TABLESPACE 구문에서 지정하는 여러 옵션들의 변환 가이드를 설명한 것이다.

ORACLE	ALTIBASE	비고
BIGFILE|SMALLFILE	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
DATAFILE file specification	DATAFILE file specification	
MINIMUM EXTENT	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
LOGGING|NOLOGGING	-	
FORCE LOGGING	-	
DEFAULT storage 구문	-	
ONLINE|OFFLINE	-	
EXTENT MANAGEMENT LOCAL|DICTIONARY	-	
SEGMENT SPACE MANAGEMENT AUTO|MANUAL	SEGMENT MANAGEMENT AUTO|MANUAL	
FLASHBACK ON|OFF	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제

2. TEMPORARY TABLESPACE

ALTIBASE는 ORACLE과 마찬가지로 CREATE TEMPORATY TABLESPACE ~ TEMPFILE 구문으로 TEMPORARY TABLESPACE를 생성한다. 다만 생성시 다음의 옵션은 알맞게 변환해야 한다.

ORACLE	ALTIBASE	비고
TABLESPACE GROUP	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
EXTENT MANAGEMENT LOCAL|DICTIONARY	-	

3. UNDO TABLESPACE

ALTIBASE는 UNDO TABLESPACE를 사용자가 생성할 수 없다. ALTIBASE가 기본적으로 제공하는 SYS_TBS_DISK_UNDO TABLESPACE만 사용할 수 있다.

CREATE TABLE

ORACLE의 TABLE을 ALTIBASE로 변환할 때 CREATE TABLE시 사용한 여러 옵션을 알맞게 변경해야 한다. ALTIBASE는 TEMPORARY TABLE, OBJECT TABLE, XMLType TABLE을 제공하지 않는다.

ALTIBASE는 메모리 TABLE을 제공한다. 따라서 변환할 TABLE의 특성을 잘 파악하여 메모리 TABLE을 생성한다면 메모리 TABLESPACE를 지정하여 생성해야 한다. 만약 메모리 TABLE을 생성한다면 ORACLE에서 사용했던 CREATE TABLE 구문에 사용한 옵션들을 사용할 수 없다. 메모리 TABLE 생성 구문은 ALTIBASE의 SQL관련 매뉴얼을 참조하면 된다.

ORACLE의 TABLE을 DISK TABLE로 변환하고자 한다면, CREATE TABLE 시 설정할 수 있는 여러 옵션을 다음과 같이 ALTIBASE에 맞게 변환해야 한다.
또한, ALTIBASE TABLE 생성 시 segment 관련 내용 지정할 경우에는 TABLESPACE 지정 -> PCTFREE/PCTUSED 지정 -> INITRANS/MAXTRANS 지정 -> Storage 절 -> logging 절 순으로 지정해 줘야 한다.

1. COLUMN DEFINITION 절

ORACLE	ALTIBASE	비고
SORT	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
DEFAULT	DEFAULT	
ENCRYPT	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
Constraint 구문	Constraint 구문	ALTIBASE는 CHECK를 제외한 Constraint를 제공. 따라서 CHECK는 변환 시 삭제하고 application에서 대체. ALTIBASE는 Constraint 지정 시 ENABLE/DISABLE 옵션, references 절의 ON DELETE SET NULL(ON DELETE CASCADE는 지원) 옵션을 제공하지 않으므로 삭제. ALTIBASE는 PRIMARY KEY, UNIQUE 지정 시 using index 절에는 tablespace 절, parallel 절, logging 절, force 절만 지정 가능하다. 즉 index 이름 및 create index 절은 제공하지 않는다.
Ref Constraint 구문	-	ALTIBASE는 REF컬럼을 지원하지 않으므로 변환 시 해당 옵션 삭제
ORGANIZATION	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
CLUSTER	-	

COMPRESS| NOCOMPRESS	-	
Column의 크기 지정 시 BYTE 키워드 명시 가능	Column의 크기 지정 시 BYTE 키워드 제공하지 않음	ORACLE은 column의 크기 지정 시 BAYE 키워드를 명시할 수 있지만, ALTIBASE는 크기 지정 시 BYTE를 제공하지 않음. ex) ORACLE : c1 VARCHAR2(10 BYTE) => ALTIBASE : c1 VARCHAR2(10)

PRIMARY KEY, UNIQUE Constraint을 지정할 때 USING INDEX 절을 이용하여 INDEX 속성을 지정할 때 ALTIBASE는 TABLESPACE 절, PARALLEL/NOPARALLEL 절, LOGGING/NOLOGGING 절만 지정이 가능하다. 즉 storage 관련 속성은 지정할 수 없다.

2. SEGMENT ATTRIBUTES 절

ORACLE	ALTIBASE	비고
TABLESPACE	TABLESPACE	
PCTFREE	PCTFREE	
PCTUSED	PCTUSED	
INITRANS	INITRANS	
MAXTRANS	MAXTRANS	255 를 120 으로 변경한다. ORACLE의 MAXTRANS는 deprecate 되었고, 항상 그 값은 255 이다. 반면에 ALTIBASE의 MAXTRANS 값을 최대 120 까지 지정할 수 있기 때문에 120 으로 변경한다.
LOGGING| NOLOGGING	LOGGING| NOLOGGING	

3. STORAGE 절

ORACLE	ALTIBASE	비고
INITIAL	INITEXTENTS	bytes -> extent 개수로 변경
NEXT	NEXTEXTENTS	bytes -> extent 개수로 변경
MINEXTENTS	MINEXTENTS	
MAXEXTENTS	MAXEXTENTS	
PCTINCREASE	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
FREELISTS	-	
FREELIST	-	

OPTIMAL	-	
BUFFER POOL	-	

4. LOB STORAGE 절

ORACLE	ALTIBASE	비고
TABLESPACE	TABLESPACE	ALTIBASE의 LOB STORAGE절은 TABLESPACE만 지정 가능
STORAGE	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
CHUNK	-	
PCTVERSION	-	
RETENTION	-	
FREEPOOLS	-	
CACHE	-	
STORAGE IN ROW	-	
LOGGING | NOLOGGING	LOGGING | NOLOGGING	

5. TABLE PARTITIONING 절

ORACLE	ALTIBASE	비고
PARTITION BY RANGE	PARTITION BY RANGE	
PARTITION BY HASH	PARTITION BY HASH	
PARTITION BY LIST	PARTITION BY LIST	
Composite partitioning 구문	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제

6. TABLE PROPERTIES 절

ORACLE	ALTIBASE	비고
ENABLE | DISABLE ROW MOVEMENT	ENABLE | DISABLE ROW MOVEMENT	Partitioned TABLE에만 지원
NOPARALLEL | PARALLEL	NOPARALLEL | PARALLEL	
ENABLE | DISABLE VALIDATE | NOVALIDATE	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제

CREATE INDEX

ALTIBASE는 BTREE와 RTREE INDEX만 제공하고 BITMAP, CLUSTER, Function based, Global partitioned INDEX를 제공하지 않는다.

또한, ALTIBASE INDEX 생성 시 segment 관련 내용 지정할 경우에는 TABLESPACE 지정 -> PARALLEL/NOPARALLEL 지정 -> LOGGING/NOLOGGING 지정 -> storage 절 순으로 지정해야 한다.

다음은 ALTIBASE로 변환할 때 CREATE INDEX 구문에 사용하는 옵션들에 대한 변환 방법이다.

ORACLE	ALTIBASE	비고
TABLESPACE	TABLESPACE	
LOGGING|NOLOGGING	LOGGING|NOLOGGING	
NOPARALLEL| PARALLEL	NOPARALLEL| PARALLEL	
COMPUTE STATISTICS	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
REVERSE	-	
SORT|NOSORT	-	
ONLINE	-	
COMPRESS| NOCOMPRESS	-	
PCTFREE, PCTUSED,	-	
INITRANS	INITRANS	
Storage 구문	TABLE의 Storage 구문과 동일	

CREATE VIEW

ALTIBASE는 Materialized-view와 DML이 가능한 Updatable-view (INSERT, UPDATE, DELETE를 수행할 수 있는 view)를 제공하지 않는다.

ALTIBASE의 VIEW는 ORACLE의 VIEW 생성 구문과 동일하게 CREATE OR REPLACE VIEW 구문으로 생성한다. 따라서 큰 변환 없이 VIEW를 생성할 수 있다. 하지만, 다음의 옵션들은 변환 시 참고해야 한다.

ORACLE	ALTIBASE	비고
WITH READ ONLY	WITH READ ONLY	ALTIBASE는 WITH READ ONLY 옵션의 VIEW만 제공하므로 해당 옵션이 default임
[NO] FORCE	[NO] FORCE	

WITH CHECK OPTION	-	ALTIBASE는 지원하지 않으므로 변환 시 해당 옵션 삭제
XMLType view 구문	-	
Object view 구문	-	

CREATE TRIGGER

ALTIBASE의 TRIGGER는 BEFORE UPDATE TRIGGER 및 DDL TRIGGER를 제공하지 않는다. 또한 TRIGGER의 DML 이벤트를 OR 연산을 이용하여 여러 개를 나열할 수 없다. 따라서 각각의 DML 이벤트에 대한 TRIGGER를 별도로 생성해야 한다.

또한, ALTIBASE는 TRIGGER 대상 TABLE에 LOB 컬럼이 있으면 에러가 발생한다.

다음은 CREATE TRIGGER시 사용하는 구문을 ALTIBASE로 변환할 때 고려해야 할 사항에 대해 설명한 것이다.

ORACLE	ALTIBASE	비고
BEFORE|AFTER|INSTEAD OF	BEFORE|AFTER	ALTIBASE는 BEFORE UPDATE와 INSTEAD OF는 지원하지 않음
DML 이벤트 구문	DML 이벤트 구문	ALTIBASE는 OR를 이용해 여러 DML 이벤트를 지정할 수 없다. DML 이벤트 구문은 ORACLE과 동일
DDL 이벤트 구문	-	ALTIBASE는 DDL TRIGGER를 제공하지 않음
WHEN 조건	WHEN 조건	
FOR EACH ROW	FOR EACH ROW	
REFERENCING	REFERENCING	DELETE 이벤트에서 NEW는 REFERENCING 할 수 없고, INSERT 이벤트에서는 OLD는 REFERENCING 할 수 없다. OLD/NEW에 alias를 줄 경우 OLD/NEW는 키워드이므로 alias로 사용할 수 없다.
Trigger body 구문	Trigger body 구문	ALTIBASE Trigger body 구문은 AS BEGIN 구문으로 시작해야 한다. 또한 ORACLE의 Trigger body 구문은 DECLARE 절로 시작할 수 있지만, ALTIBASE의 Trigger body는 AS 절에 선언 부분을 지정 해야 하고, DECLARE 절을 명시할 수 없다. OLD/NEW row 사용 시 ':' 사용할 수 없다. Ex) :old (x) -> old(o)

CREATE SEQUENCE

ALTIBASE의 CREATE SEQUENCE 구문은 ORACLE과 동일하다. 다만, ORACLE에서 제공하는 ORDER, NOORDER 옵션은 지원하지 않는다. 또한, ORACLE SEQUENCE의 maxvalue 값은 28 자리 정수까지 지정할 수 있지만, ALTIBASE SEQUENCE의 maxvalue는 (-9223372036854775807)부터 9223372036854775806 까지의 범위 내에서 지정할 수 있다.

CREATE SYNONYM

ALTIBASE의 CREATE SYNONYM 구문은 ORACLE과 동일하다. 다만, ORACLE에서 제공하는 OR REPLACE 구문은 지원하지 않는다.

ALTER TABLE

ALTIBASE는 Constraint 추가 시 한번에 1 개의 Constraint만 추가 가능하다. ORACLE에서 "ALTER TABLE ADD (*CONSTRAINT constraint_name constraint 지정,…*);" 문으로 여러 개의 Constraint를 추가하는 문장은 Constraint별로 나눠서 "ALTER TABLE ADD *CONSTRAINT constraint_name constraint 지정*;" 문으로 Constraint 별로 나눠서 실행해야 하고 비록 Constraint를 1 개만 지정한다 하더라도 '('와 ')'로 묶을 수 없다.

ORACLE은 PRIMARY KEY와 UNIQUE를 지정 시 INDEX를 미리 생성한 후 지정이 가능하지만, ALTIBASE는 Constraint를 지정하는 시점에 내부적으로 INDEX를 생성하기 때문에 동일 컬럼에 대해 PK Constraint, UNIQUE Constraint, INDEX 생성 중 하나만 가능하다. 즉, INDEX를 생성한 컬럼에 PK Constraint를 지정할 수 없다

SQL 변환

ORACLE에서 ALTIBASE로 SQL을 변환하는 방법과 고려할 사항에 대해 기술한다.

OUTER JOIN

ALTIBASE는 ANSI 표준의 JOIN 연산만 제공한다. 따라서 ORACLE의 (+) 연산자를 이용한 JOIN 구문은 ANSI표준의 JOIN 구문으로 변경해야 한다.

ORACLE의 (+)를 이용한 JOIN은 (+)가 없는 쪽의 TABLE이 기준 TABLE이 되고 (+)가 있는 TABLE이 OUTER JOIN 대상이 된다. ANSI 표준의 OUTER JOIN 구문은 기준 TABLE이 왼쪽에 위치하는 경우는 LEFT OUTER JOIN, 오른쪽에 위치하는 경우는 RIGHT OUTER JOIN을 사용한다.

1. 두 TABLE이 한 개의 컬럼에 대해 OUTER JOIN 될 경우

1.1 ORACLE 질의

```
SELECT t1.a, t1.b, t2.a, t2.c
FROM test1 t1, test2 t2
WHERE t1.a = t2.a(+);
```

1.2 ALTIBASE로 변경된 질의

```
SELECT t1.a, t1.b, t2.a, t2.c
FROM test1 t1 LEFT OUTER JOIN test2 t2 ON t1.a = t2.a;

혹은,

SELECT t1.a, t1.b, t2.a, t2.c
FROM test2 t2 RIGHT OUTER JOIN test1 t1 ON t1.a = t2.a;
```

⇨ (+)가 없는 기준TABLE(test)을 LEFT OUTER JOIN 구문 왼쪽에 위치시키고, 대상TABLE(test2)를 오른쪽에 위치시킨 후 JOIN 조건을 ON절에 명시한다. RIGHT OUTER JOIN을 사용할 경우에는 기준TABLE(test1)을 RIGHT OUTER JOIN 구문 오른쪽에 위치시킨 후 대상TABLE(test2)를 왼쪽에 위치시키면 된다.

2. 기준 TABLE 한 개에 다수의 TABLE이 OUTER JOIN 될 경우

2.1 ORACLE 질의

```
SELECT t1.a, t1.b, t3.d, t4.e, t5.f
FROM test1 t1, test3 t3, test4 t4, test5 t5
WHERE t1.a = t3.a(+)
    AND t1.a = t4.a(+)
    AND t1.a = t5.a(+)
ORDER BY t1.a;
```

2.2 ALTIBASE로 변경된 질의

```
SELECT t1.a, t1.b, t3.d, t4.e, t5.f
FROM test1 t1 LEFT OUTER JOIN test3 t3 ON t1.a = t3.a
                LEFT OUTER JOIN test4 t4 ON t1.a = t4.a
```

```
                LEFT OUTER JOIN test5 t5 ON t1.a = t5.a
ORDER BY t1.a;
```

⇨ 기준TABLE(test1)을 LEFT OUTER JOIN 구문 왼쪽에 위치시키고, 첫 번 째 OUTER JOIN 대상TABLE(test3)를 오른쪽에 위치시킨 후 JOIN 조건을 ON절에 명시한다. 이후 다른 대상 TABLE을 차례로 LEFT OUTER JOIN 구문으로 동일한 형식으로 나열한다.

3. 두 개의 TABLE에 대해 JOIN되는 컬럼이 다수인 경우

3.1 ORACLE 질의

```
SELECT t1.*, t2.*
FROM test1 t1, test2 t2
WHERE t1.a= t2.a(+)
    AND t1.b = t2.b(+)
    AND t1.c = t2.c(+)
    AND t1.a = 2
ORDER BY t1.a, t1.b;
```

3.2 ALTIBASE로 변경된 질의

```
SELECT t1.*, t2.*
FROM test1 t1
     LEFT OUTER JOIN test2 t2 ON t1.a = t2.a
         AND t1.b = t2.b
         AND t1.c = t2.c
WHERE t1.a=2
ORDER BY t1.a, t1.b;
```

⇨ 기준TABLE(test1)을 LEFT OUTER JOIN 구문 왼쪽에 위치시키고, 여러 개의 JOIN 조건을 ON절에 AND 연산자를 이용하여 명시한다. 만약 기타 다른 조건이 있다면 WHERE 절에 해당 조건을 추가해 주면 된다.

4. OUTER JOIN과 일반 JOIN이 함께 사용된 경우

4.1 ORACLE 질의

```
SELECT t1.*, t2.*, t3.*
FROM test1 t1, test2 t2, test3 t3
WHERE t1.a= t2.a
    AND t2.c= t3.c(+)
    AND t1.a = 2;
```

4.2 ALTIBASE로 변경된 질의

```
SELECT t1.*, t2.*, t3.*
FROM test1 t1, test2 t2
LEFT OUTER JOIN test3 t3 ON t2.c = t3.c
WHERE t1.a = t2.a
   AND t1.a = 2;
```

⇨ 기준TABLE(test3)를 LEFT OUTER JOIN 구문 왼쪽에 위치시키고, OUTER JOIN 대상TABLE(test1)을 LEFT OUTER JOIN 오른쪽에 위치시킨 후 여러 개의 JOIN 조건을 ON절에 AND 연산자를 이용하여 명시한다. 일반 JOIN 연산은 WHERE 절에서 "=" 연산자를 이용하여 명시하고, 만약 기타 다른

조건이 있다면 WHERE 절에 AND 연산자를 이용하여 해당 조건을 추가해 주면 된다.

RANK 관련 함수

ORACLE에서 제공하는 ROW_NUMER(), RANK(), DENSE_RANK()는 데이터를 그룹으로 묶고 그것에 대한 순위를 정하는 함수이다. 5 명의 학생에 대한 성적 정보가 입력이 되어 있고 1 등이 2 명이라면 위의 함수는 각각 다음의 결과를 return한다.

```
ROW_NUMBER()   : 1, 2, 3, 4, 5
RANK()         : 1, 1, 3, 4, 5
DENSE_RANK()   : 1, 1, 2, 3, 4
```

ALTIBASE에서는 RANK(), ROW_NUMBER(), DENSE_RANK() 함수를 지원하지 않는다. 따라서 다음과 같은 형태로 변환해서 사용해야 한다.

질의를 수행하기 위해 다음의 테스트 데이터를 이용한다.

```
iSQL> SELECT * FROM emp;
ENAME           DEPTNO          SAL
------------------------------------------------------------
SMITH           20               800
ALLEN           30              1600
WARD            30              1250
JONES           20              2975
MARTIN          30              1250
BLAKE           30              2850
CLARK           10              2450
KING            10              5000
TURNER          30              1500
JAMES           30              950
FORD            20              3000
MILLER          10              1300
SCOTT           20              3000
ADAMS           20              1100
14 rows selected.
```

1. ROW_NUMER()

1.1 ORACLE 질의

```
SELECT ename, sal, ROW_NUMBER() OVER (ORDER BY sal DESC) row_num
FROM emp;
```

1.2 ALTIBASE로 변경된 질의

```
SELECT ename, sal, rownum row_num
FROM (SELECT ename, sal
        FROM emp
        ORDER BY sal DESC ) x -- sal ASC였다면 ORDER BY sal ASC로 변경
```

2. ROW_NUMBER() PARTITION BY

2.1 ORACLE 질의

```
SELECT ename, deptno, sal,
      ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC)
FROM EMP;
```

2.2 ALTIBASE로 변경된 질의

```
SELECT ename, deptno, (rownum - count_row) dense_rank
FROM (SELECT a.deptno, a.ename, a.sal,
                    (SELECT COUNT(*) count_row
                     FROM emp b
                     WHERE b.deptno < a.deptno) count_row
          FROM emp a
          ORDER BY deptno, sal DESC) x -- sal ASC였다면 ORDER BY deptno, sal로 변경
ORDER BY deptno, dense_rank;
```

성능을 위해 deptno, sal 컬럼에 인덱스를 생성해야 한다.

3. RANK()

3.1 ORACLE 질의

```
SELECT ename, sal, RANK() OVER (ORDER BY sal DESC) rank
FROM emp;
```

3.2 ALTIBASE로 변경된 질의

```
SELECT ename, sal, rank
FROM (SELECT a.ename, a.sal,
                    (SELECT 1+COUNT(*)
                     FROM emp b
                     WHERE b.sal > a.sal) rank -- sal ASC였다면 WHERE b.sal < a.sal로
변경
                     FROM emp a) x
ORDER BY x.rank;
```

성능을 위해 sal 컬럼에 인덱스를 생성해야 한다.

4. RANK() PARTITION BY

4.1 ORACLE 질의

```
SELECT ename, deptno, sal,
      RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) rank
FROM EMP;
```

4.2 ALTIBASE로 변경된 질의

```
SELECT ename, deptno, sal, rank
FROM (SELECT a.ename, a.deptno, a.sal,
                    (SELECT 1+COUNT(*)
                     FROM emp b
                     WHERE b.deptno = a.deptno
                     AND b.sal > a.sal) rank -- sal ASC였다면 WHERE b.sal < a.sal로 변경

          FROM emp a) x
```

```
ORDER BY x.deptno, x.rank;
```

성능을 위해 deptno, sal 컬럼에 인덱스를 생성해야 한다.

5. DENSE_RANK()

5.1 ORACLE 질의

```
SELECT ename, sal, DENSE_RANK() OVER (ORDER BY sal DESC)
FROM emp;
```

5.2 ALTIBASE로 변경된 질의

```
SELECT ename, sal, dense_rank
FROM (SELECT ename, a.sal,
                    (SELECT 1+COUNT(DISTINCT sal)
                     FROM emp b
                     WHERE b.sal > a.sal) dense_rank -- sal ASC였다면 WHERE b.sal
< a.sal로 변경

        FROM emp a) x
ORDER BY x.dense_rank;
```

성능을 위해 sal 컬럼에 인덱스를 생성해야 한다.

6. DENSE_RANK() PARTITION BY

6.1 ORACLE 질의

```
SELECT ename, deptno, sal,
    DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC)
FROM EMP;
```

6.2 ALTIBASE로 변경된 질의

```
SELECT ename, deptno, sal, dense_rank
FROM (SELECT a.ename, a.deptno, a.sal,
                  (SELECT 1+COUNT(DISTINCT sal)
                   FROM emp b
                   WHERE b.deptno = a.deptno
                   AND b.sal > a.sal) dense_rank -- sal ASC였다면 WHERE b.sal <
a.sal로 변경

        FROM emp a) x
ORDER BY x.deptno, x.dense_rank;
```

성능을 위해 deptno, sal 컬럼에 인덱스를 생성해야 한다.

계층 형 질의

하나의 엔터티가 다른 엔터티가 아닌 자기 자신과 관계를 맺는 순환 관계의 모델을 가질 때, 계층적인 순서로 결과를 조회하도록 질의 문을 제공하는데 이때 사용하는 질의가 계층 형 질의이다. ORACLE에서 사용하는 START WITH와 CONNECT BY 절은 ALTIBASE에서도 동일하게 지원하므로 변환 없이 사용할 수 있다.

하지만 다음의 사항을 주의해야 한다.

1. base TABLE에 대한 질의만 가능하다.

   VIEW와 Inline VIEW에서는 계층 형 질의를 사용할 수 없다. 사용하게 되면 다음의 에러가 발생한다.

```
iSQL>CREATE VIEW v1 AS SELECT * FROM pc;

iSQL>SELECT LPAD(' ',2*(LEVEL-1)) | | item_name item_names
       FROM v1
       START WITH parent_id IS NULL
       CONNECT BY PRIOR item_id = parent_id;

ERR-311A1 : A hierarchical query on neither a created view nor inline view is allowed
```

2. NOCYCLE 절은 IGNORE LOOP 절로 변환 한다.

   2.1 ORACLE 질의

```
SELECT LPAD(' ',2*(LEVEL-1)) | | item_name AS item_names
FROM pc
START WITH parent_id IS NULL
CONNECT BY NOCYCLE PRIOR item_id = parent_id;
```

   2.2 ALTIBASE로 변경된 질의

```
SELECT LPAD(' ',2*(LEVEL-1)) | | item_name AS item_names
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id;
IGNORE LOOP;
```

3. ORDER BY SIBILINGS BY절을 지원하지 않는다.

4. CONNECT_BY_ROOT, CONNECT_BY_ISLEAF, SYS_CONNECT_BY_PATH, CONNECT_BY_ISCYCLE는 지원하지 않는다.

ORACLE 10g에서 지원하는 CONNECT_BY_ROOT, CONNECT_BY_ISLEAF, SYS_CONNECT_BY_PATH, CONNECT_BY_ISCYCLE는 ALTIBASE에서 지원하지 않으므로 다음과 같은 형태로 변환해서 사용해야 한다.

질의를 수행하기 위해 다음의 테스트 데이터를 이용한다.

```
iSQL> SELECT * FROM pc;
ITEM_ID     PARENT_ID     ITEM_NAME          ITEM_QTY
------------------------------------------------------------------------------------------
1001                      Computer            1
1002        1001          BODY                1
1003        1001          monitor             1
1004        1001          printer             1
1005        1002          Mother board        1
1006        1002          Lan card            1
```

```
1007    1002    Power Supply    1
1008    1005    RAM             1
1009    1005    CPU             1
1010    1005    Graphic device  1
1011    1005    ETC device      1
11 rows selected.
```

1. CONNECT_BY_ROOT

자신의 최상의 ROOT 노드 즉, LEVEL 1 에 해당하는 값을 나타내는 의사 컬럼이다.

1.1 ORACLE 질의

```
SELECT LPAD(' ',2*(LEVEL-1))||item_name AS item_names,
CONNECT_BY_ROOT item_id, CONNECT_BY_ROOT item_name
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id;
```

1.2 ALTIBASE로 변경된 질의

```
SELECT a.item_names, b.connect_by_root_item_id,
b.connect_by_root_item_name
FROM (SELECT LPAD(' ',2*(LEVEL-1))||item_name item_names
        FROM pc
        START WITH parent_id IS NULL
        CONNECT BY PRIOR item_id = parent_id) a,
        (SELECT item_id AS connect_by_root_item_id, item_name AS
                connect_by_root_item_name
        FROM pc
        WHERE LEVEL =1
        START WITH parent_id IS NULL
        CONNECT BY PRIOR item_id = parent_id ) b;
```

⇨ 최상의 ROOT 노드에 해당하는 값을 inline-view로 만들어 PRODUCT 연산을 수행한다.

2. CONNECT_BY_ISLEAF

CONNECT_BY_ISLEAF는 자신의 LEAF 노드일 경우(자식을 갖지 않을 경우)에 1 을 return 하고 그렇지 않을 경우에 0 을 return 하는 의사 컬럼이다.

2.1 ORACLE 질의

```
SELECT LPAD(' ',2*(LEVEL-1))||item_name AS item_names,
CONNECT_BY_ISLEAF
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id;
```

2.2 ALTIBASE로 변경된 질의

```
SELECT a.item_names, DECODE(b.parent_id,NULL,1,0) connect_by_isleaf
FROM (SELECT LPAD(' ',2*(level-1))||item_name item_names, item_id
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id) a
```

```
LEFT OUTER JOIN (SELECT DISTINCT parent_id FROM pc) b
ON a.item_id = b.parent_id;
```

⇨ parent_id 값들을 inline-view로 질의한 후 이 inline-view와 outer join을 수행하여 item_id값이 parent_id에 존재하지 않는 경우(자식이 존재하지 않는 경우)는 LEAF 노드이므로 1 을 리턴 하고 그렇지 않은 경우는 0 을 리턴 한다.

3. SYS_CONNECT_BY_PATH

ROOT 노드부터 해당 레코드 항목까지의 경로(PATH)를 반환한다.

3.1 ORACLE 질의

```
SELECT LEVEL, SYS_CONNECT_BY_PATH(item_name,'/')
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id;
```

3.2 ALTIBASE로 변경된 질의

3.2.1 SYS_CONNECT_BY_PATH에 대응하는 사용자 정의 함수를 생성한다.

```
CREATE OR REPLACE FUNCTION sys_connect_by_path_pc
(
    pkey     pc.parent_id%TYPE,
    plevel   INTEGER,
    delim    VARCHAR(10)
)
RETURN VARCHAR2(200)
AS
     path VARCHAR(200);
BEGIN
     DECLARE
          CURSOR c1 IS
               SELECT item_name
               FROM     pc
               WHERE    LEVEL <= plevel
               START    WITH item_id = pkey
               CONNECT BY PRIOR parent_id=item_id ;
     BEGIN
          FOR crec IN c1 LOOP
               path := delim || crec.item_name || path;
          END LOOP;
          RETURN path;
     END ;
END;
/
```

3.2.2 sys_connect_by_path_pc 함수를 호출한다.

```
SELECT LEVEL, SYS_CONNECT_BY_PATH_PC ( item_id, LEVEL, '/' )
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id;
```

4. CONNECT_BY_ISCYCLE

해당 레코드의 항목이 자식 노드를 갖는데 동시에 그 자식 노드가 다시 부모 노드인지 판별하는 의사 컬럼이다.

ALTIBASE는 이러한 경우 무한 루프에 빠지기 때문에 다음의 오류를 return한다.

```
[ERR-311A4 : A loop in hierarchical query detected.]
```

이러한 오류가 발생하지 않게 하기 위해서는 IGNORE LOOP절을 명시해야 한다.

```
SELECT LPAD(' ',2*(LEVEL-1)) | | item_name AS item_names
FROM pc
START WITH parent_id IS NULL
CONNECT BY PRIOR item_id = parent_id;
IGNORE LOOP;
```

ORACLE의 CONNECT_BY_ISCYCLE은 ALTIBASE에서 지원하지 않는다.

ROLLUP & CUBE

ROLLUP과 CUBE는 GROUP BY절에 사용하여 추가적인 grouping 정보를 표현하는 함수이다.

- ROLLUP : grouping된 결과에 group 별 합계 정보를 추가한다.
- CUBE : grouping된 컬럼에 대해 가능한 모든 조합의 합계 정보를 추가한다.

ALTIBASE는 ROLLUP과 CUBE를 지원하지 않기 때문에 다음과 같이 변환해야 한다.

질의를 수행하기 위해 다음의 테스트 데이터를 이용한다.

```
iSQL> SELECT * FROM tmp_sales;
GROUP_ID        SALES_EMP       SALES_QTY
------------------------------------------------------------------
A               test1           5
A               test2           10
A               test3           1
B               test4           10
B               test5           5
C               test6           50
A               test1           5
A               test2           10
…
C               test5           5
```

1. ROLLUP

1.1 ORACLE 질의

```
SELECT group_id, sales_emp, SUM(sales_qty)
FROM tmp_sales
```

```
GROUP BY ROLLUP(group_id,sales_emp);
```

1.2 ALTIBASE로 변경된 질의

```
SELECT group_id, sales_emp, SUM(sales_qty)
FROM (
        SELECT DECODE(no, 1, group_id, 2, group_id) group_id,
                  DECODE(no, 1, sales_emp) sales_emp, sales_qty
        FROM tmp_sales,
        (SELECT LEVEL no FROM dual CONNECT BY LEVEL <= 3) copy_t
       )
GROUP BY group_id, sales_emp
ORDER BY 1, 2;
```

2. CUBE

2.1 ORACLE 질의

```
SELECT group_id, sales_emp, SUM(sales_qty)
FROM tmp_sales
GROUP BY CUBE(group_id, sales_emp);
```

2.2 ALTIBASE로 변경된 질의

```
SELECT NVL(group_id,' '), NVL(sales_emp,' '), Sum(sales_qty)
FROM (
        SELECT DECODE(no, 1, group_id, 2, group_id) group_id,
                  no,
                  DECODE(no, 1, sales_emp, 4, sales_emp) sales_emp,
                 sales_qty
        FROM tmp_sales,
        (SELECT LEVEL no FROM dual CONNECT BY LEVEL <= 4) copy_t
)
GROUP BY group_id, sales_emp
ORDER BY 1, 2;
```

⇨ ORALCE의 CUBE 실행 시 grouping하는 컬럼의 값이 null로 나타나지만, 위의 변환 질의는 null 대신 공백 ‘’으로 나타난다.

GRANT 구문

ORACLE은 특정 USER의 OBJECT 권한을 관리자가 부여할 수 있지만, ALTIBASE는 OBJECT를 소유한 USER만이 부여할 수 있다.

Stored PROCEDURE/FUNCTION 변환

ORACLE의 Stored PROCEDURE와 FUNCTION을 ALTIBASE로 변환할 때 고려할 사항에 대해 기술한다.

일반 사항 비교

다음은 Stored PROCEDURE/FUCTION을 작성할 때 일반적으로 고려해야 할 사항에 대해 설명한다.

1. ALTIBASE AUTOCOMMIT 모드에서 PROCEDURE 처리

ALTIBASE가 AUTOCOMMIT 모드일 경우 PROCEDURE/FUNCTION 전체가 하나의 트랜잭션으로 처리되고 PROCEDURE 내에서의 COMMIT/ROLLBACK은 무시된다. 또 PROCEDURE/FUNCTION을 실행한 후에 자동으로 COMMIT된다. 따라서 PROCEDURE/FUNCTION에서 DML 문장을 실행한 후 결과가 ALTIBASE와 ORACLE이 다를 수 있다. 반대로 ALTIBASE를 NON-AUTOCOMMIT 모드로 적용했다면 ORACLE의 NON-AUTOCOMMIT 모드와 동일한 결과가 나타난다.

참고로, ORACLE은 AUTOCOMMIT 모드로 지정해도 PROCEDURE/FUNCTION 내부에서는 NON-AUTOCOMMIT 모드로 동작한다.

1.1 ORACLE

```
CREATE OR REPLACE PROCEDURE t1_test
(
   in_t IN INTEGER, in_v IN VARCHAR
)
IS
BEGIN
   INSERT INTO t1 VALUES(in_t, in_v);
   ROLLBACK;
END;
/

EXEC t1_test(4, '000004');

SQL> SELECT COUNT(*) FROM t1;
   COUNT(*)
----------------
              0
```

1.2 ALTIBASE

```
CREATE OR REPLACE PROCEDURE t1_test(
   in_t IN INTEGER, in_v IN VARCHAR(20)
)
IS
BEGIN
   INSERT INTO t1 VALUES(in_t, in_v);
   ROLLBACK;
END;
```

```
/

EXEC t1_test(4, '000004');

iSQL> SELECT COUNT(*) FROM t1;
COUNT
-----------
          1
```

2. CURSOR를 OPEN한 상태에서 COMMIT/ROLLBACK, DDL문을 실행할 수 없다.

2.1 ORACLE

```
CREATE OR REPLACE PROCEDURE cur_proc1
(in_val IN INTEGER, out_val OUT INTEGER)
IS
   CURSOR cur1 IS SELECT c1 FROM t1 WHERE c1>in_val;
   col1 INTEGER;
   r1 INTEGER;
BEGIN
   r1 := 0;
   OPEN cur1;
   LOOP
      FETCH cur1 into col1;
      EXIT WHEN cur1%NOTFOUND;
      INSERT INTO t2 VALUES(col1);
      COMMIT;   -- 허용됨
       r1 := r1+1;
      DBMS.PUT_LINE(col1);
   END LOOP;
   CLOSE cur1;
   DBMS.PUT_LINE('# of insert: '||r1);
   out_val := r1;
END;
/
```

2.1 ALTIBASE

```
CREATE OR REPLACE PROCEDURE cur_proc1
(in_val IN INTEGER, out_val OUT INTEGER)
IS
   CURSOR cur1 IS SELECT c1 FROM t1 WHERE c1>in_val;
   col1 INTEGER;
   r1 INTEGER;
BEGIN
   r1 := 0;
   OPEN cur1;
   LOOP
      FETCH cur1 into col1;
      EXIT WHEN cur1%NOTFOUND;
      INSERT INTO t2 VALUES(col1);
      COMMIT;              -- (X)
      r1 := r1+1;
```

```
    PRINTLN(col1);
  END LOOP;
  CLOSE cur1;
  COMMIT;                -- (O)
  PRINTLN('# of insert: '||r1);
  out_val := r1;
END;
/
```

3. PARAMETER의 TYPE과 RETURN TYPE의 CHAR, VARCHAR는 크기도 지정

ALTIBASE에서 PROCEDURE나 FUCTION의 PARAMATER 혹은 RETURN TYPE을 ORACLE처럼 CHAR, VARCHAR로 선언하면 CHAR(1), VARCHAR(1)과 동일한 의미이다. 따라서, 하나의 문자가 아니라 문자열을 사용하고자 한다면, 반드시 그 크기를 지정해 줘야 한다. 만약 지정 해주지 않고 문자열이 2 문자 이상이면 다음과 같은 에러가 발생한다.

```
ERR-2100D : Invalid length of the data type
```

파일 및 출력 처리

ALTIBASE의 파일 및 출력에 관련된 PROCEDURE는 SYSTEM_ 유저에 자동으로 생성이 되어 있고, PUBLIC SYNONYM으로 정의되어 있기 때문에 사용자는 PROCEDURE 이름만 호출하여 사용할 수 있다.

다음은 ORACLE과 ALTIBASE의 파일 처리 방법을 비교한 표이다.

구분	ORACLE	ALTIBASE
표준 출력	DBMS_OUTPUT.PUT	PRINT
	DBMS_OUTPUT.PUT_LINE	PRINTLN
파일 처리	UTL_FILE.FOPEN	FOPEN
	UTL_FILE.FCLOSE	FCLOSE
	UTL_FILE.FCLOSE_ALL	FCLOSE_ALL
	UTL_FILE.FCOPY	FCOPY
	UTL_FILE.FFLUSH	FFLUSH
	UTL_FILE.FREMOVE	FREMOVE
	UTL_FILE.FRENAME	FRENAME
	UTL_FILE.GET_LINE	GET_LINE
	UTL_FILE.IS_OPEN	IS_OPEN
	UTL_FILE.NEW_LINE	NEW_LINE

TYPE과 TYPESET

ALTIBASE는 PACKAGE를 지원하지 않기 때문에 사용자 정의 TYPE들을 TYPESET으로 생성하여 PROCEDURE 간의 전송 수단으로 사용할 수 있다.

따라서 ORALCE의 CREATE TYPE문으로 생성한 사용자 정의 TYPE들을 묶어서 CREATE TYPESET문으로 TYPESET을 생성하여 사용할 수 있다.

다음은 ORACLE의 TYPE과 ALTIBASE의 TYPESET을 생성하는 구문이다.

1. ORACLE

```
CREATE TYPE emp_ary
    AS VARRAY(50) OF VARCHAR2(4000);
```

2. ALTIBASE

```
CREATE TYPESET typeset_1
AS
    TYPE emp_rec_type IS RECORD(
        name VARCHAR(20),
        job_id VARCHAR(10),
        salary NUMBER(8));

    TYPE emp_arr_type IS TABLE OF emp_rec_type INDEX BY INTEGER;
END;
/
```

REF CURSOR

ORACLE의 REF CURSOR는 보통 PACKAGE에 선언을 하고, 이를 PROCEDURE의 OUT PARAMATER로 선언하여 사용한다. 하지만, ALTIBASE는 PACKAGE를 제공하지 않기 때문에 TYPESET으로 생성하여 사용해야 한다.

다음은 ORACLE과 ALTIABSE에서 REF CURSOR를 사용하는 예제이다.

1. ORACLE

```
CREATE OR REPLACE PACKAGE ref_cursor_pkg AS
  TYPE ref_type IS REF CURSOR;
  PROCEDURE ref_cursor_pro(v_result OUT ref_type, v_sql IN VARCHAR2);
END;
/

CREATE OR REPLACE PACKAGE BODY ref_cursor_pkg AS
  PROCEDURE ref_cursor_pro(v_result OUT   ref_type, v_sql IN VARCHAR2)
AS
BEGIN
      OPEN v_result FOR v_sql [USING] [Bind Var];
END;
/
```

2. ALTIBASE

```
CREATE OR REPLACE TYPESET my_type
AS
   TYPE my_cur IS REF CURSOR;
END;
/

CREATE OR REPLACE PROCEDURE opencursor
( v_result OUT my_type.my_cur, v_sql IN VARCHAR(200) )
AS
BEGIN
   OPEN y_result FOR v_sql [USING] [Bind Var];
END;
/
```

WHERE CURRENT OF 구문

ALTIBASE는 CURSOR를 이용한 WHERE CURRENT OF 구문을 지원하지 않는다.
다만, 해당 TABLE에 PRIMARY KEY가 있다면 다음과 같이 변경이 가능하다.

1. ORACLE

```
CREATE OR REPLACE PROCEDURE proc1
IS
CURSOR emp_list IS
    SELECT empno
    FROM employee
    WHERE empno = 1
        FOR UPDATE;
BEGIN
    FOR emplst IN emp_list LOOP
        UPDATE employee
        SET empjob = 'SALESMAN'
        WHERE CURRENT OF emp_list;
    END LOOP;
END;
/
```

2. ALTIBASE

```
CREATE OR REPLACE PROCEDURE proc1
AS
BEGIN
    DECLARE
        CURSOR cur1 IS
            SELECT empno
            FROM employee
            WHERE empno = 1;
        v_empjob VARCHAR(10);
        v_empno INTEGER;
    BEGIN
```

```
        OPEN cur1;
        LOOP
            FETCH cur1 INTO v_empno, v_empjob;
            EXIT WHEN cur1%NOTFOUND;
            UPDATE employee SET empjob = 'SALESMAN'
            WHERE emp_no = v_empno; //emp_no가 PK이어야 한다.
        END LOOP;
        CLOSE cur1;
    END;
END;
/
```

EXCEPTION

ORACLE과 ALTIBASE는 Stored PROCEDURE/FUNCTION에서 발생하는 EXCEPTION들을 미리 시스템에서 정의해 놓았다. 다음은 시스템 정의 EXCEPTION에 대해 비교한 표이다.

ORACLE		ALTIBASE	
SQLERRM	**SQLCODE**	**SQLERRM**	**SQLCODE**
CURSOR_ALREADY_OPEN	-6530	CURSOR_ALREADY_OPEN	201062
DUP_VAL_ON_INDEX	-1	DUP_VAL_ON_INDEX	201063
INVALID_CURSOR	-1001	INVALID_CURSOR	201064
INVALID_NUMBER	-1722	INVALID_NUMBER	201065
NO_DATA_FOUND	+100	NO_DATA_FOUND	100
PROGRAM_ERROR	-6501	PROGRAM_ERROR	201067
STROAGE_ERROR	-6500	STROAGE_ERROR	201068
TIMEOUT_ON_RESOURCE	-51	TIMEOUT_ON_RESOURCE	201069
TOO_MANY_ROWS	-1422	TOO_MANY_ROWS	201070
VALUE_ERROR	-6502	VALUE_ERROR	201071
ZERO_DIVIDE	-1476	ZERO_DIVIDE	201072
ACCESS_INTO_NULL	-6530	지원하지 않음	
CASE_NOT_FOUND	-6592		
COLLECTION_IS_NULL	-6531		
LOGIN_DENIED	-1017		
NOT_LOGGED_ON	-1012		
ROWTYPE_MISMATCH	-6504		
SELF_IS_NULL	-30625		
SUBSCRIPT_BEYOND_COUNT	-6533		
SUBSCRIPT_OUTSIDE_LIMIT	-6532		

SYS_INVALID_ROWID	-1410	

DATA MIGRATION

ORACLE 데이터를 ALTIBASE로 이관하는 방법에 대해 기술한다.

DATA MIGRATION 방법

ORACLE DATA를 ALTIBASE로 이관하기 위해서는 ORACLE 관련 유틸리티(ex SQLDeveloper)를 이용하여 데이터를 텍스트 파일로 download하고 이 텍스트 파일을 ALTIBASE의 iloader 유틸리티를 이용하여 ALTIBASE로 upload한다.

ALTIBASE의 iloader 사용법은 iloader 매뉴얼을 참고하면 된다.

ALTIBASE®

알티베이스㈜
서울특별시 구로구 구로 3 동 182-13
대륭포스트 2 차 1008 호
02-2082-1000
http://www.altibase.com

대전사무소
대전광역시 서구 둔산동 921
주은리더스텔 901 호
042-489-0330

기술본부
서울특별시 구로구 구로동
우림e-biz센터 11 층 1101 호
02-2082-1000

기술지원센터
02-2082-1114
support@altibase.com

ATC (ALTIBASE Technical Center)
http://atc.altibase.com