

차 례

제 1 장 관계형 데이터베이스-	9
1. 관계형 데이터베이스의 개념-	11
1.1 관계형 모델의 구성요소-	11
1.2 관계형 데이터베이스 기능-	11
1.3 관계형 데이터베이스의 정리-	12
1.4 관계 데이터베이스에서 사용하는 용어-	12
1.5 다중 테이블 관계-	13
1.6 관계형 데이터베이스의 속성-	14
1.7 SQL 문장-	14
2. ORACLE 관계형 데이터베이스 관리 시스템-	15
2.1 완전한 ORACLE 솔루션-	15
3. SQL, SQL*Plus, PL/SQL-	16
3.1 SQL-	16
3.2 SQL*Plus-	17
3.3 PL/SQL(Procedural Language/SQL)	18
제 2 장 데이터의 검색-	21
1. 기본적인 SELECT 문 사용법-	23
1.1 SQL SELECT 문장의 성능-	23
1.2 Syntax-	23
1.3 SQL 문장 작성법	23
1.4 SQL 문장 실행	24
1.5 모든 열 선택	24
1.6 특정 Column 선택	24
1.7 Column의 출력 형태	25
1.8 산술 표현식-	25
1.9 Null 값의 처리-	26
1.10 NVL 함수-	26
1.11 열에 별칭(Alias) 부여-	27
1.12 연결 연산자	28

1.13 LITERAL 문자 STRING	29
1.14 중복 행의 제거	30
제 3 장 데이터 제한과 정렬	33
1. 특정 행의 검색	35
1.1 Syntax	35
1.2 WHERE 절에 사용되는 연산자	35
2. ORDER BY 절	42
2.1 Syntax	42
2.2 데이터의 정렬	43
2.3 다중 열에 의한 정렬	44
4 장 단일 행 함수	47
1. SQL 함수	47
1.1 SQL 함수의 특징 및 이점	49
1.2 단일 행 함수(Single Row Function)	49
1.3 문자형 함수(Character Function)	50
1.4 숫자형 함수	58
1.5 날짜형 함수	61
1.6 변환 함수	65
1.7 기타 함수	70
1.8 중첩 함수	70
제 5 장 그룹 함수(Multi Row Function)	73
1. 그룹 함수	75
1.1 그룹 함수의 종류	75
1.2 그룹 함수 사용	75
1.3 데이터 그룹 생성	76
제 6 장 다중 테이블로부터 데이터 검색	83
1. Join	85
1.1 Syntax	85
1.2 Join의 종류	85
1.3 Cartesian Product	85
1.4 Equijoin	86
1.5 Non-Equijoin	89

1.6 Outer Join	90
1.7 Self Join	92
1.8 Set Operators	93
제 7 장 서브쿼리(SUBQUERY)	97
1. SUBQUERY	99
1.1 SUBQUERY 의 개념	99
1.2 Syntax	99
1.3 SUBQUERY 를 사용할 수 있는 절	100
1.4 SUBQUERY 의 유형	100
1.5 단일 행 SUBQUERY	100
1.6 다중 행 SUBQUERY	102
1.7 다중 열 SUBQUERY	105
1.8 SUBQUERY 에서의 NULL 값	109
1.9 FROM 절에서의 SUBQUERY	110
제 8 장 SQL*Plus 명령어	113
1 SQL*Plus 명령어	115
1.1 SQL 과 SQL*Plus 의 차이점	115
1.2 SQL 명령 편집 및 실행	116
1.3 SQL*Plus 를 이용하여 보고서 작성	121
1.4 상호작용 리포트	126
제 9 장 테이블(TABLE) 생성	133
1. 테이블 생성	135
1.1 ORACLE 에서 사용하는 객체	135
1.2 제약 조건	135
1.3 테이블 차트에 의한 테이블 생성	138
1.4 SUBQUERY 를 사용한 테이블 생성	144
1.5 데이터 사전(DATA DICTIONARY) 질의	145
2. 테이블을 수정	147
2.1 새로운 열 추가	148
2.2 열 수정	149
2.3 제약 조건 추가	151
2.4 제약 조건 삭제	152

2.5 제약 조건 비활성화	153
2.6 제약 조건 활성화	154
2.7 제약 조건 조회	155
2.8 객체 이름 변경	156
2.9 TRUNCATE TABLE 문장	156
2.10 테이블에 주석문 추가	157
3. 테이블 삭제	157
3.1 Syntax	158
제 10 장 데이터 조작	161
1. 데이터(DML) 조작어	163
1.1 INSERT 문장	163
1.2 UPDATE 문장	167
1.3 DELETE 문장	170
1.4 데이터베이스 TRANSACTION	171
1.5 읽기 일관성	174
1.6 Locking	174
제 11 장 SEQUENCE	177
1. SEQUENCE	179
1.1 SEQUENCE 특징	179
1.2 Syntax	179
1.3 SEQUENCE 확인	180
1.4 SEQUENCE 사용법	180
1.5 SEQUENCE 값 CACHE	181
1.6 SEQUENCE에서 간격의 경계	181
1.7 SEQUENCE 수정	182
1.8 SEQUENCE 제거	182
제 12 장 VIEW	185
1. VIEW의 개념	187
1.1 VIEW의 장점	197
1.2 VIEW의 종류	188
1.3 VIEW의 생성	188
1.4 VIEW의 구조 및 이름 확인	190

1.5 데이터 액세스 VIEW	190
1.6 복합 VIEW 생성	191
1.7 VIEW에서 DML 연산 수행	192
1.8 WITH CHECK OPTION 절 사용	192
1.9 DML 연산 부정	193
1.10 VIEW의 제거	194
제 13 장 인덱스(INDEX)	197
1. 인덱스(INDEX)의 개요	197
1.1 인덱스의 특징	199
1.2 인덱스 생성 방법	199
1.3 인덱스의 종류	200
1.4 사용자가 인덱스 생성	200
1.5 인덱스 생성 확인	201
1.6 인덱스 제거	201
2. 동의어	202
2.1 Syntax	202
2.2 동의어 삭제	203
제 14 장 사용자 접근 제어	205
1. 사용자 접근 제어	207
1.1 데이터베이스 보안의 두 범주	207
1.2 사용자 생성	207
1.3 권한	208
1.4 ROLE의 개념	215
제 15 장 PL/SQL 개요	219
1. PL/SQL 개요	221
1.1 PL/SQL의 장점	222
1.2 PL/SQL Block 구조	223
1.3 SQL*Plus로 하는 일	226
제 16 장 변수 사용	229
1. 변수	231
1.1 변수 사용	231
1.2 PL/SQL에서 변수 처리	231

1.3	변수 유형	232
1.4	PL/SQL 변수 선언	232
1.5	이름 지정 규칙	233
1.6	변수의 값 지정	233
1.7	스칼라 데이터 형	234
1.8	조합 데이터 형(Composite Datatype)	236
1.9	LOB Datatype 변수	241
1.10	바인드 변수	241
1.11	Non-PL/SQL 변수 참조	242
2.	PL/SQL 블록	242
2.1	PL/SQL 블록 구문과 지침	242
2.2	데이터형 변환	244
2.3	중첩 블록과 변수 범위	244
2.4	PL/SQL에서 연산자	245
2.5	프로그래밍 지침 사항	246
제 17 장	PL/SQL에서 사용 가능한 SQL 문장	249
1.	PL/SQL에서 SQL 문장	251
1.1	PL/SQL에서 SQL 문장 사용	251
1.2	PL/SQL에서 SELECT 문장	251
1.3	PL/SQL을 이용한 데이터 조작	253
1.4	이름 지정 규약	255
1.5	COMMIT과 ROLLBACK 문장	255
1.6	SQL CURSOR	255
제 18 장	PL/SQL의 제어 구조	259
1.	개요	261
1.1	IF 문	261
1.2	논리적 조건 설정	265
1.3	LOOP 문	265
1.4	BASIC LOOP 문	266
1.5	EXIT 문	266
1.6	FOR LOOP 문	268
1.7	WHILE LOOP 문	270

1.8 중첩 LOOP 와 레이블-	271
제 19 장 CURSOR-	275
1. 커서의 개념-	277
1.1 CURSOR 의 종류-	277
1.2 명시적 CURSOR 의 제어-	277
1.3 DECLARE CURSOR-	278
1.4 OPEN CURSOR-	278
1.5 FETCH CURSOR-	279
1.6 CLOSE CURSOR-	279
1.7 명시적 CURSOR 의 속성-	280
1.8 복수 인출(FETCH) 제어-	280
1.9 CURSOR 와 RECORD-	281
1.10 CURSOR 와 FOR LOOP-	282
1.11 SUBQUERY 를 사용한 CURSOR FOR LOOP-	283
제 20 장 고급 명시적 CURSOR	287
1. 매개변수와 CURSOR -	289
1.1 Syntax -	289
1.2 FOR UPDATE 절-	292
1.3 WHERE CURRENT OF 절-	292
1.4 SUBQUERY -	294
제 21 장 예외처리	297
1. PL/SQL 로 예외 처리-	299
1.1 예외 처리란 -	299
1.2 예외를 발생시키는 두 가지 방법-	299
1.3 예외 처리-	299
1.4 예외의 유형-	300
1.5 예외 정의-	300
1.6 미리 정의된 ORACLE SERVER 에러-	301
1.7 미리 정의되지 않은 ORACLE SERVER 에러-	303
1.8 사용자 정의 예외 -	304
1.9 예외 트래핑 함수 -	305
1.10 RAISE_APPLICATION_ERROR -	306

제 22 장 BUILD PL/SQL SUBPROGRAMS	309
1. SUBPROGRAM	311
1.1 SUBPROGRAM 의 개요	311
1.2 SUBPROGRAM 작성 단계	311
1.3 PROCEDURE 생성	312
1.4 FUNCTION 생성	316
1.5 함수와 프로시저 비교	319
1.6 TRIGGER	320
부록 A	327
A. ORACLE DATA DICTIONARY	329
1. ORACLE DATA DICTIONARY 의 내용	329
2. ORACLE DATA DICTIONARY 의 내용 조회	329
B. 실습용 테이블을 생성하는 SCRIPT	333
1. 실습용 SCRIPT 란 ?	333
2. 실습용 TABLE 생성 절차	333
3. 실습용 SCRIPT 의 내용	333

1. 관계형 데이터베이스의 개념

관계형 데이터 모델은 1970년 6월 E.F.Codd 박사의 논문 “대량 공용 데이터베이스의 관계형 모델”(A Relational Model of Data for Large Shared Data Bank)에서 처음으로 데이터베이스 시스템으로 관계형 모델 도입을 제안했다. 그 당시 더 많이 알려진 데이터 모델은 Hierarchical Data Model, Network Data Model, 심지어 단순 형태의 파일 데이터 구조였다.

RDBMS(Relational Database Management System)가 특히 사용의 용이, 구조상의 융통성 때문에 쉽게 대중화 되었고 게다가 총체적 해결을 제공하는 강력한 응용 프로그램 개발과 사용자 제품 군들을 RDBMS로 가능하게 해주는 업체들이 나타났고 그 중 Oracle이 대표적인 업체로 발전했다.

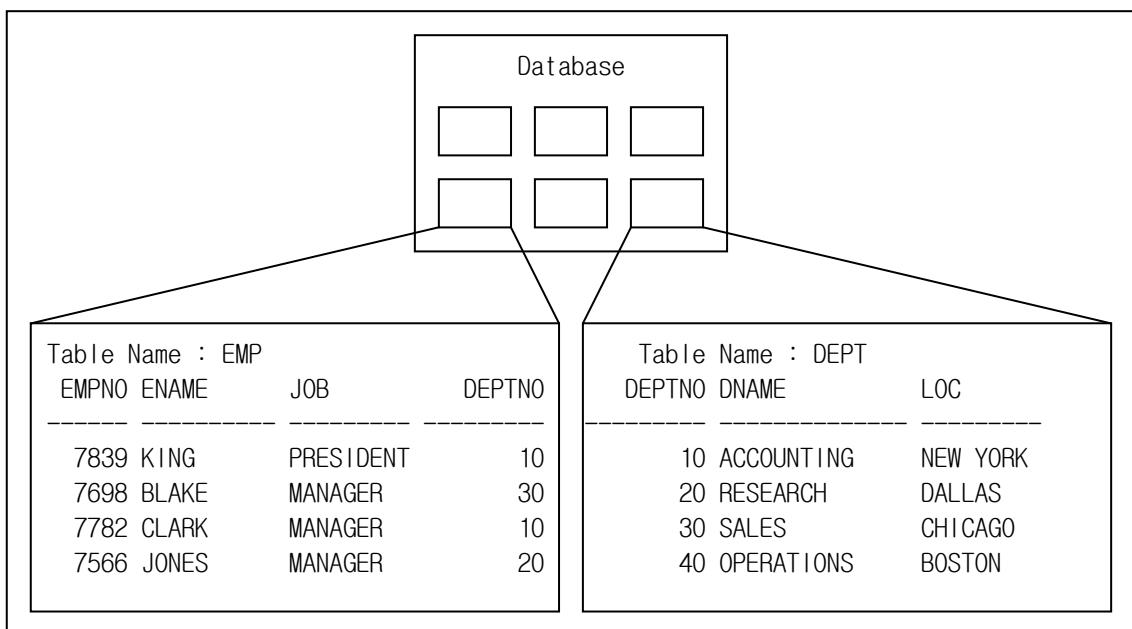
1.1 관계형 모델의 구성요소

- 1) 데이터를 저장하는 객체(object) 또는 관계(relation)들의 집합
- 2) 다른 관계를 생성하기 위해 관계에 가해지는 일련의 연산자 집합
- 3) 정확성과 일관성을 위한 데이터의 무결성(Integrity)

1.2 관계형 데이터베이스 기능

관계(relation)형 데이터베이스는 정보 저장을 위해 관계나 2차원 테이블을 이용한다.

- 1) 데이터의 저장을 관리한다.
- 2) 데이터에 대한 ACCESS을 통제한다.
- 3) 데이터를 검색 및 수정하기 위한 수단을 제공한다.



1.3 관계형 데이터베이스의 정리

- 1) E.F.Codd 박사는 1970년 데이터베이스 시스템용 관계형 모델을 제안
- 2) 제시한 관계형 모델은 관계형 데이터베이스 관리 시스템의 기본이 된다.
- 3) 관계형 모델링은 다음 구성요소를 포함하고 있다.
 - ① 객체(object) 또는 관계(relation)의 집합
 - ② 관계(relation)에 가해지는 연산의 집합
 - ③ 정확성 및 일관성을 위한 데이터의 무결성
- 4) 관계형 데이터베이스는 2차원 테이블 형태로 구성된다.
- 5) 각 테이블은 Row와 Column으로 구성되어 있다.
- 6) 각 형의 데이터는 유일하다.
- 7) 각 column은 데이터 무결성을 유지한다.
- 8) SQL 명령어를 실행함으로 행들의 데이터를 조작 가능하다.

1.4 관계 데이터베이스에서 사용하는 용어

관계형 데이터베이스는 한 개 이상의 테이블을 가지고 있고 이 테이블은 RDBMS에서 기본 저장 구조입니다.

```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000	10	
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	30	
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20

1. 특정 종업원에 대한 모든 데이터를 나타내는 단일 row 또는 tuple입니다. 기본 키(primary key)에 의해 식별되어져야 합니다.
2. 기본 키(primary key)인 종업원 번호를 포함하는 열(column) 또는 속성(attribute)입니다. 종업원 번호는 EMP 테이블에서 유일한(unique) 종업원을 식별합니다.

3. 키 값이 아닌 열입니다. 열은 테이블에서 한 종류의 데이터를 나타냅니다.
4. 부서 번호를 포함하는 열은 외래 키(foreign key)입니다. 외래 키는 테이블 간에 서로 어떻게 관련되었는가를 정의합니다. 외래 키는 다른 테이블의 기본 키 또는 고유 키를 참조합니다.
5. 필드(field)는 행과 열의 교차되는 곳에 있습니다.
6. 필드는 그 안에 값을 가지지 않을 수도 있습니다. 이것은 null value 라 불립니다.
EMP 테이블에서 영업 사원인 종업원만이 COMM(commission)필드에서 값을 가집니다.

1.5 다중 테이블 관계

Table Name : EMP			Table Name : DEPT		
EMPNO	ENAME	JOB	DEPTNO	DEPTNO	DNAME
7839	KING	PRESIDENT	10	10	ACCOUNTING
7698	BLAKE	MANAGER	30	20	RESEARCH
7782	CLARK	MANAGER	10	30	SALES
7566	JONES	MANAGER	20	40	OPERATIONS
7654	MARTIN	SALESMAN	30		
7499	ALLEN	SALESMAN	30		

각 테이블은 하나의 ENTITY를 기술하는 데이터를 포함합니다. 데이터의 카테고리들은 각 테이블의 상단을 가로질러 LIST 되며 개별 정보는 아래로 LIST 됩니다. 테이블 포맷을 사용하여 정보를 사용하고, 이해하며, 쉽게 시각화 할 수 있습니다. 다른 ENTITY에 대한 데이터는 다른 테이블에 저장되기 때문에, 특정 문제에 대한 해답을 얻기 위해 둘 또는 보다 많은 테이블을 결합시킬 필요가 있을 수 있습니다. RDBMS는 외래 키를 사용하여 하나의 테이블에 다른 데이터를 관련시킬 수 있도록 해 줍니다. 외래 키는 동일 테이블 또는 다른 테이블에서의 기본 키를 참조하는 열 또는 열의 집합입니다. 한 테이블의 데이터를 다른 테이블의 데이터로 관련시키는 능력은 개별적으로 관리되는 정보의 단위들을 조직할 수 있도록 해 줍니다. 종업원 데이터는 부서 테이블과 분리된 테이블에 저장됨으로써 부서 데이터로부터 논리적으로 구분될 수 있습니다.

☞ Guidelines

◆ 기본 키와 외래 키

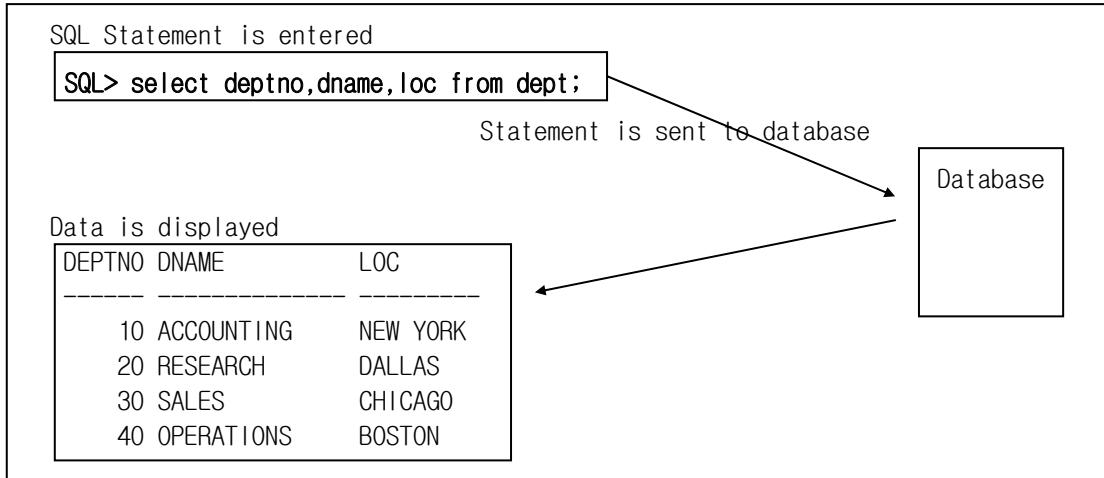
- 1) 기본 키에서 중복 값을 허용되지 않습니다.
- 2) 일반적으로 기본 키는 변경될 수 없습니다.
- 3) 외래 키는 데이터 값을 기초로 하며, 논리적이지 물리적이거나 포인터가 아닙니다.
- 4) 외래 키 값은 존재하는 기본 키 값 또는 고유 값과 일치해야하거나, 또는 NULL 이 될 수 있습니다.

1.6 관계형 데이터베이스의 속성

- 1) 관계형 데이터베이스에서는 테이블로의 액세스 루트를 지정하지 않으며 또 물리적으로 데이터가 어떻게 나열되는지 알 필요가 없다.
- 2) 데이터베이스를 이용하기 위해서는, 관계형 데이터베이스를 가동시킬 수 있는 ANSI(American National Standards Institute) 표준 언어인 SQL(Structured Query Language) 문을 실행해야 합니다. 이 언어는 관계(relation)를 조합, 분리 시킬 수 있는 일련의 큰 연산자 집합을 지니고 있습니다.
- 3) 데이터베이스는 SQL 문 사용으로 변경이 용이하다.
- 4) 데이터베이스는 데이터의 독립성을 보장한다.
- 5) 관계형 데이터베이스의 OBJECT의 종류는 다음과 같다

OBJECT	설명
TABLE	ROW와 COLUMN으로 구성된 기본적인 저장 단위
VIEW	하나 이상의 TABLE로부터 논리적으로 데이터를 분류한 부분집합
INDEX	포인터를 사용하여 행의 검색 속도를 향상
SEQUENCE	자동적으로 ORACLE SERVER가 유일 번호를 생성
SYNONYM	객체에 대체 이름을 부여
PROGRAM UNIT	SQL 또는 PL/SQL 문으로 작성한 PROCEDURE, FUNCTION, PACKAGE

1.7 SQL 문장



SQL로 서버와 통신이 가능하며 다음의 장점 등을 지니고 있습니다.

- 1) 효율적입니다.
- 2) 배우기 쉽고 사용하기 쉽습니다.
- 3) 기능적으로 완전합니다. SQL은 테이블에 있는 데이터를 정의, 검색, 조작할 수 있게 해 줍니다.

2. ORACLE 관계형 데이터베이스 관리 시스템

ORACLE SERVER의 특징은 관계형 구조에 근거한 모든 장점은 물론이고 프로그램 단위를 저장하고 실행하도록 하는 PL/SQL 엔진을 통해 정보를 저장하고 관리할 수 있게 해줍니다. 서버는 최적화를 바탕으로 데이터를 읽어 들이는 사용자 옵션을 제공, 데이터베이스가 어떻게 ACCESS 되고 사용되는지를 제어하는 보안 기능을 포함, Locking 기법을 통해 데이터의 일관성과 보호 기능을 가능하도록 하는 것입니다.

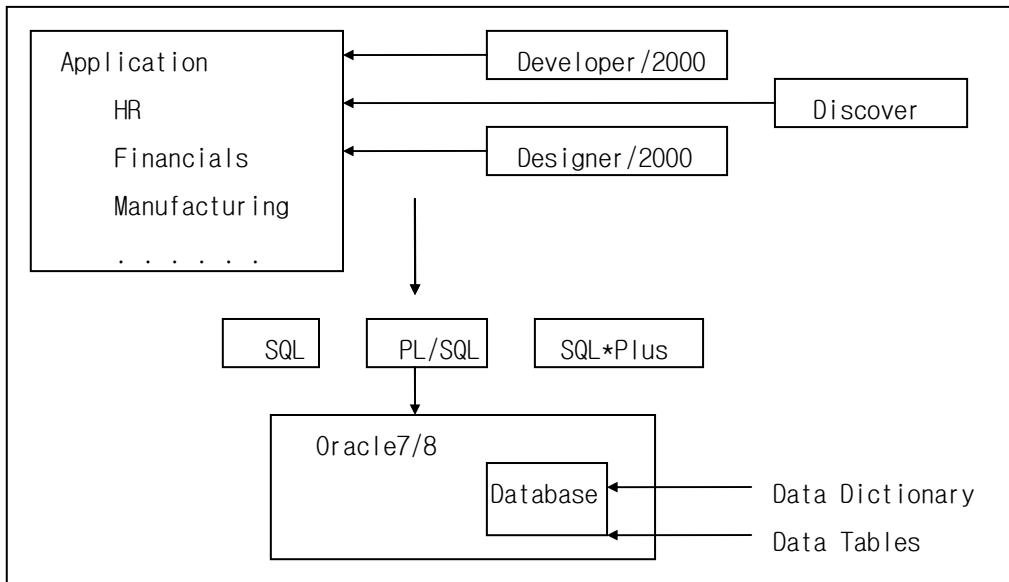
ORACLE 어플리케이션은 ORACLE SERVER과 동일한 컴퓨터에서 실행될 수 있으며 다른 방식으로는 Oracle7 Server는 다른 하나의 시스템에서 수행되고 어플리케이션은 사용자의 로컬 시스템에서 실행되는 클라이언트-서버 형태로 운영할 수 있습니다. 이런 클라이언트-서버 환경에서는 광범위한 컴퓨팅 리소스가 사용될 수 있습니다.

최근에는 Oracle8이 발표되었는데 이는 ORACLE에 의해 개발된 첫 번째 객체 가능 데이터베이스입니다. 이것은 새로운 ORDBMS를 지원하기 위하여 Oracle7의 데이터 모델링 능력을 확장합니다. Oracle8은 객체 지향 프로그래밍, 복잡한 데이터 타입, 복잡한 비즈니스 객체, 실 세계와 완전히 양립 가능한 새 엔진을 제공합니다. Oracle8은 많은 면에서 Oracle7을 확장합니다. 런타임 데이터 구조를 더 많이 공유하고, 더 큰 버퍼 캐쉬를 잡고, 제약 조건을 자연 가능케 하는 등 OLTP(Online Transaction Processing)의 기능성과 성능을 향상시킵니다. 데이터 웨어하우스 어플리케이션은 삽입, 갱신, 삭제의 병렬 수행, 분할, 병렬 인식 질의 최적화 같은 성능 향상 측면에서 장점이 있습니다. NCA(Network Computing Architecture)프레임 워크에서 작동 중인 Oracle8은 분산되고(distributed) 다중으로 고정된(multitiered) 클라이언트-서버와 웹 기반 어플리케이션을 지원합니다. Oracle8은 수십만의 동시 사용자를 조정할 수 있고 521petabytes 까지 지원하며 전통적으로 구조화된 데이터 뿐만 아니라 텍스트, 공간 이미지, 사운드, 비디오, 일련의 시간을 포함하는 어떤 타입의 데이터도 처리할 수 있습니다.

2.1 완전한 ORACLE 솔루션

ORACLE RDBMS는 ORACLE의 핵심 제품입니다. ORACLE SERVER과 데이터의 실제적인 사용, 모니터링, 유지 관리 등에 있어서 사용자를 돋기 위한 몇 가지 툴들을 포함하고 있습니다. ORACLE 데이터 사전은 서버의 가장 중요한 구성요소로서 데이터베이스를 읽기 전용으로 참조할 수 있게 하는 VIEW과 테이블로 구성되어집니다. ORACLE의 RDBMS는 다음과 같은 일을 처리합니다.

- 1) 데이터 정의와 저장을 관리
- 2) 데이터 이용과 동시성 제어 및 제한
- 3) 백업과 복구 제공
- 4) SQL과 PL/SQL 문 번역
- 5) 시스템 보안 유지



3. SQL, SQL*Plus, PL/SQL

SQL은 RDBMS를 사용하기 위해 ANSI에서 책정한 표준 언어로 Oracle 서버와 통신하기 위한 명령 언어이고 SQL*Plus는 SQL 및 PL/SQL 문장을 인식하고 실행시켜주는 Oracle TOOL이다. PL/SQL Application Logic 추가하여 SQL을 확장한 Oracle의 절차적인 언어이다.

3.1 SQL

ORACLE SQL은 ANSI(American Standards Institute)와 ISO(Internation Standards Organization) 표준을 따릅니다. ORACLE 회사는 SQL 표준 위원회에 주요 인사를 적극적으로 포함시켜, 변화, 발전하는 표준에 대한 미래 수용을 보증합니다. ANSI와 ISO는 관계형 데이터베이스 표준 언어로 SQL을 채택했습니다.

3.1.1 SQL 문장의 종류

구 분	문 장	설 명
Retrive(Query)	SELECT	데이터베이스로부터 데이터 검색
DML(Data Manipulation Language)	INSERT UPDATE DELETE	개별적으로 데이터베이스 테이블에서 새로운 행을 입력하고 기존의 행을 변경하고, 원치 않는 행을 제거 합니다.
DCL(Data Control Language)	CREATE ALTER DROP RENAME TRUNCATE	데이터 구조를 생성, 변경, 제거합니다.

Transaction Control Language	COMMIT ROLLBACK SAVEPOINT	DML 명령문으로 만든 변경을 관리, 데이터 변경은 논리적 Transaction 으로 함께 그룹화 될 수 있습니다.
DCL(Data Control Language)	GRANT REVOKE	ORACLE 데이터베이스와 그 구조에서 액세스 권한을 제공하거나 제거합니다.

3.1.2 SQL 문장의 특징

- 1) 사용하기가 쉬워 프로그래밍에 별로 경험이 없는 사용자는 물론 다양한 이용자들이 사용할 수 있다
- 2) 비절차적인 언어이다
- 3) 시스템 개발 및 유지 보수에 의해 소요되는 시간을 줄일 수 있다
- 4) 영어와 유사한 언어이다

3.2 SQL*Plus

SQL*Plus 는 SQL, PL/SQL 문장을 인식하고 실행을 위해 서버에 넘겨주는 Oracle TOOL로서 자체적인 명령 언어들을 갖고 있다

3.2.1 SQL*Plus 문장의 종류

구 분	종 류
Execution Commands	/, RUN, EXECUTE
Edit Commands	LIST, APPEND, CHANGE, DEL, INPUT, EDIT
Environment Commands	SET, SHOW, PAUSE
Report Format Commands	COLUMN, CLEAR, BREAK, COMPUTE, TTITLE, BTITLE
File Manipulation Commands	SAVE, GET, START, @, @@, SPPOOL
Interactive Commands	DEFINE, UNDEFINE, PROMPT, ACCEPT, VARIABLE, PRINT
Database Access Commands	CONNECT, COPY, DISCONNECT
Miscellaneous Commands	SQLPLUS, EXIT, HELP, DESCRIBE, HOST, REMARK, RUNFORM, TIMING, WHENEVER

3.2.2 SQL*Plus 의 특징

- 1) 수행하도록 입력한 문장들을 받아들인다
- 2) 파일에서 SQL 및 PL/SQL 입력 문을 받아들인다
- 3) 라인 편집기로 SQL 문장을 편집한다
- 4) 환경설정을 제어한다

- 5) 검색 결과를 보고서 형태로 형식 출력할 수 있도록 설정해 준다
- 6) 최종 사용자와 상호 작용한다
- 7) 원격지 데이터베이스(Remote database)를 액세스한다

3.3 PL/SQL(Procedural Language/SQL)

PL/SQL은 ORACLE 회사의 SQL에 대한 절차적 확장입니다. 객체 관계형 데이터베이스에 대한 표준 데이터 Access 언어입니다. PL/SQL은 데이터 캡슐화, 정보 숨김, 객체 지향 같은 현대적 소프트웨어 엔지니어링 기능을 제공하여 ORACLE SERVER과 툴셋에 대한 최신식의 프로그래밍을 가능하게 합니다.

PL/SQL은 1970년대와 1980년대 사이에 설계된 프로그래밍 언어의 여러 향상된 기능을 포함합니다. PL/SQL을 강력한 트랜잭션 프로세싱 언어로 만들기 위해, SQL 데이터 조작 및 질의 문장이 블록 구조와 절차적 코드 단위에 포함되도록 합니다. PL/SQL로 데이터를 처리하기 위해 PL/SQL 제어 명령문을 사용할 수도 있고, ORACLE 데이터를 처리하기 위해 SQL 명령문을 사용할 수도 있습니다.

3.3.1 PL/SQL 엔진과 Oracle Server

PL/SQL은 별개의 Oracle 제품이 아니라 Oracle 서버와 다른 Oracle 툴에 이용되고 있는 프로그래밍 언어이다. PL/SQL의 블록은 Oracle 서버나 툴에 내장되는 PL/SQL 엔진에 전달되어 처리된다. 사용하는 엔진은 PL/SQL이 수행되는 곳에 따라 다르다.

가) Oracle 서버에서의 PL/SQL 엔진

Pro*프로그램, USER-EXIT, SQL*Plus, 또는 Server Manager에서 PL/SQL 블록을 사용하면 Oracle 서버의 PL/SQL 엔진이 처리한다. 그리고 블록에 있는 SQL을 별도의 문장으로 분리하여 SQL 문 실행기로 보낸다. 이는 응용 프로그램의 블록을 한 번에 Oracle 서버에게 보낸다는 뜻이며 따라서 client/server 환경하에서 많은 성능 향상을 기대할 수 있다. 또한 데이터 베이스에 접속한 여러 응용 프로그램에서 stored subprogram들을 참조할 수 있다.

나) Oracle 툴에서의 PL/SQL

Developer/2000을 포함한 많은 Oracle 툴은 Oracle 서버에 있는 엔진과는 별도로 자체 PL/SQL 엔진을 갖고 있다. 이 엔진이 SQL 분장을 찾아서 Oracle 서버의 SQL 문 실행기로 보내고, PL/SQL engine은 응용 프로그램에 대해 지역적인 데이터를 처리한다. 이로써 Oracle 서버의 대한 작업량과 요구되는 메모리 커서의 수를 줄인다.

♣ 참고

Developer/2000 응용 프로그램의 부분으로 선언된 프로시저와 함수의 일반적인 구조는 동일하더라도 데이터베이스에 저장된 것과는 다르다. Stored Subprogram은 데이터베이스

액체이고 데이터 사전에 저장되며 여러 응용 프로그램이 사용할 수 있다. Application subprogram은 응용 프로그램의 지역적인 PL/SQL 엔진에 블록을 전달한다. 작업은 서버 쪽이 아닌 응용 프로그램 쪽에서 수행된다.

3.3.2 PL/SQL 문장의 종류

기본 PL/SQL은 선언부, 실행부, 예외 처리부의 블록과 섹션으로 구성되어 있다. 이들 중 실행부에서 사용할 수 있는 명령어들은 다음과 같다.

구 분	설 명
IF	조건에 따라 선택적으로 작업을 수행
Basic LOOP	LOOP와 END LOOP 사이의 문장을 반복 수행
FOR loop	반복되는 수를 정하여 문장을 반복 수행
WHILE loop	제어 조건이 TRUE가 아닐 때까지 문장을 반복 수행

3.3.3 PL/SQL의 특징

- 1) SQL로는 얻을 수 없는 PL/SQL의 절차적 프로그래밍 기능
- 2) 모듈화된 프로그램 개발
 - ① 블록 내에서 논리적으로 관련된 문장들의 그룹화
 - ② 강력한 프로그램을 작성하기 위해 서브 블록들을 큰 블록에 포함
 - ③ 복잡한 문제에 대한 프로그래밍이 적절히 나뉘어진 모듈들의 집합으로 구성
- 3) CURSOR, EXCEPTION
 - ① 변수, 상수 등을 선언하고 SQL과 절차적인 프로그램에서 사용
 - ② 데이터베이스의 테이블과 Record를 기반으로 하는 dynamic한 변수 선언이 가능
- 4) 절차적 언어 구조로 된 프로그램 작성
 - ① 조건에 따라 일련의 문장을 실행 (IF)
 - ② 루프에서 반복적으로 일련의 문장을 실행 (LOOP)
 - ③ Explicit Cursor를 이용한 Multi-row 질의 처리
- 5) ERROR 처리
 - ① Exception 처리 루틴을 이용하여 ORACLE8 서버 에러를 처리함
 - ② 사용자 정의 에러를 선언하고 Exception 처리 루틴으로 처리 가능 함

1. 기본적인 SELECT 문 사용법

SELECT 문장을 이용하여 데이터베이스로부터 저장되어 있는 데이터를 검색하는 방법에 대하여 알아보기로 한다.

1.1 SQL SELECT 문장의 성능

- 1) Selection : 질의에 대해 RETURN 하고자 하는 테이블의 행을 선택하기 위해 SQL 의 Selection 기능을 사용할 수 있습니다.
- 2) Projection : 질의에 대해 RETURN 하고자 하는 테이블의 열을 선택하기 위해 SQL 의 Projection 기능을 사용할 수 있습니다.
- 3) Join : 공유 테이블 양쪽의 열에 대해 링크를 생성하여 다른 테이블에 저장되어 있는 데이터를 함께 가져오기 위해 SQL의 join 기능을 사용할 수 있습니다.

1.2 Syntax

```
SELECT [DISTINCT]      {*, column [alias], . . .}
  FROM            table_name
  [WHERE          condition]
  [ORDER BY       {column, expression} [ASC | DESC]];
```

DISTINCT 중복 행 제거 옵션

* 테이블의 모든 column 출력

alias 해당 column에 대한 다른 이름 부여

table_name 테이블명 질의 대상 테이블 이름

WHERE 조건을 만족하는 행들만 검색

condition column, 표현식, 상수 및 비교 연산자

ORDER BY 질의 결과 정렬을 위한 옵션(ASC:오름차순(Default), DESC 내림차순)

1.3 SQL 문장 작성법

- 1) SQL 문장은 대소문자를 구별하지 않습니다.
- 2) SQL 문장은 한 줄 또는 여러 줄에 입력될 수 있습니다.
- 3) 하나의 명령어는 여러 줄에 나누거나 단축될 수 없습니다.
- 4) 절은 보통 읽고 편집하기 쉽게 줄을 나누도록 합니다.(권장)
- 5) 탭과 줄 넣기(들여쓰기)는 코드를 보다 읽기 쉽게 하기 위해 사용됩니다.(권장)
- 6) 일반적으로 키워드는 대문자로 입력합니다. 다른 모든 단어, 즉 테이블 이름, 열 이름은 소문자로 입력합니다.(권장)
- 7) SQL*Plus에서 SQL 문장은 SQL 프롬프트에 입력되며 1 라인 이후의 라인은 라인 번호가 붙습니다. 가장 최근의 명령어가 1개가 SQL buffer에 저장됩니다.

1.4 SQL 문장 실행

- 1) 마지막 줄의 끝에 “;”를 기술하여 명령의 끝을 표시한다.
- 2) 버퍼에서 마지막 라인에 슬래시를 넣습니다.(OS 의 Editor 사용시)
- 3) SQL 프롬프트에 슬래시를 입력합니다.(SQL Buffer 의 내용 실행)
- 4) SQL 프롬프트에서 SQL*Plus RUN 명령어를 실행합니다. (SQL Buffer 의 내용 실행)

1.5 모든 열 선택

SELECT 키워드에 “*” 을 사용하여 테이블의 열 데이터 모두를 조회할 수 있습니다.

문제 1) SCOTT OI 소유하고 있는 EMP Table 의 모든 자료를 출력하여라.

```
SQL> SELECT *
  2  FROM emp;

EMPNO ENAME      JOB          MGR HIREDATE      SAL      COMM  DEPTNO
----- -----
 7839 KING        PRESIDENT    7839 17-NOV-81   5000      10
 7698 BLAKE       MANAGER     7839 01-MAY-81   2850      30
 7782 CLARK       MANAGER     7839 09-JUN-81   2450      10
 7566 JONES       MANAGER     7839 02-APR-81   2975      20
.
.
.
14 rows selected.
```

1.6 특정 Column 선택

테이블의 특정 Column 을 검색하고자 할 경우 Column 이름을 “,”로 구분하여 명시함으로써 특정 Column 을 출력할 수 있습니다. 출력 순서는 SELECT 문 뒤에 기술한 Column 의 순서대로 출력됩니다.

문제 2) SCOTT OI 소유하고 있는 EMP Table 에서 사원 번호, 이름, 급여, 담당업무를 출력하여라.

```
SQL> DESC emp
Name           Null?    Type
----- -----
EMPNO          NOT NULL NUMBER(4)
ENAME          VARCHAR2(10)
JOB            VARCHAR2(9)
MGR            NUMBER(4)
HIREDATE       DATE
SAL             NUMBER(7,2)
COMM            NUMBER(7,2)
DEPTNO         NOT NULL NUMBER(2)
```

```

SQL> SELECT empno,ename,sal,job
  2 FROM emp;

EMPNO ENAME          SAL JOB
----- -----
 7839 KING            5000 PRESIDENT
 7698 BLAKE           2850 MANAGER
 7782 CLARK           2450 MANAGER
.
.
.
14 rows selected.

```

1.7 Column의 출력 형태

날짜 열 헤딩과 데이터 뿐만 아니라 문자열 헤딩과 데이터는 열 폭 내에서 좌측 정렬됩니다. 숫자 헤딩과 데이터는 우측 정렬입니다.

```

SQL> SELECT empno,ename,hiredate
  2 FROM emp;

EMPNO ENAME      HIREDATE
----- -----
 7839 KING        17-NOV-81
 7698 BLAKE       01-MAY-81
.
.
.
14 rows selected.

```

1.8 산술 표현식

데이터가 출력되는 방식을 수정하거나 계산을 수행하고자 할 때 산술 표현식을 사용한다. 산술 표현식은 열 이름, 숫자 상수, 문자 상수, 산술 연산자를 포함할 수 있으며 연산자는 +(Add), -(Subtract), *(Multiply), /(Divide)를 사용합니다. SELECT 문장에서는 FROM 절을 제외한 SQL 문장의 절에서 사용할 수 있습니다. 또한 산술 표현식이 하나 이상의 연산자를 포함한다면 일반적인 산술 연산자 우선 순위를 따른다.

문제 3) 모든 종업원의 급여를 \$300 증가시키기 위해 덧셈 연산자를 사용하고 결과에 SAL+300을 디스플레이 합니다.

```

SQL> SELECT ename, sal, sal+300
  2 FROM emp;

ENAME      SAL    SAL+300
----- -----
KING        5000   5300
BLAKE       2850   3150
.
.
.
14 rows selected.

```

♣ 참고

계산된 결과 열 SAL+300 은 EMP 테이블의 새로운 열이 아님을 유의하십시오. 이것은 단지 디스플레이를 위한 것일 뿐입니다. 디폴트로 새로운 열의 이름 sal+300 은 생성된 계산식으로부터 유래합니다. 또한 SQL*Plus 는 산술 연산자 앞뒤의 공백을 무시합니다.

1.9 Null 값의 처리

행이 특정 열에 대한 데이터 값이 없다면, 값은 null 이 됩니다. null 값은 이용할 수 없거나 지정되지 않았거나, 알 수 없거나 또는 적용할 수 없는 값입니다. null 값은 0이나 공백과는 다릅니다. 0 은 숫자이며 공백은 문자입니다. 열이 NOT NULL 로 정의되지 않았거나, 열이 생성될 때 PRIMARY KEY 로 정의되지 않았다면, 어떤 데이터형의 열은 null 값을 포함할 수 있습니다. EMP 테이블의 COMM 열에서 오직 SALESMAN 만이 보너스를 받을 수 있음을 주목하십시오.

♣ 참고

- 1) NULL은 이용할 수 없고 할당되지 않고 알려져 있지 않고 적용 불가한 값을 의미한다.
- 2) NULL이란 0 나 공백(space)과 다르다.
- 3) 널 값을 포함한 산술 표현식 결과는 NULL이 된다.
- 4) column에 데이터 값이 없으면 그 값 자체가 널 또는 널 값을 포함하고 있다.
- 5) 널 값은 1 바이트의 내부 저장 장치를 오버헤드로 사용하고 있으며 어떠한 datatype column 들이라도 널 값을 포함할 수 있다.

문제 4) EMP 테이블에서 사원번호, 이름, 급여, 보너스, 보너스 금액을 출력하여라

```
SQL> SELECT empno,ename,sal,comm,sal+comm/100
  2  FROM emp;
```

EMPNO	ENAME	SAL	COMM	SAL+COMM/100
7839	KING	5000		
7698	BLAKE	2850		
7782	CLARK	2450		
7566	JONES	2975		
7654	MARTIN	1250	1400	1264
7499	ALLEN	1600	300	1603
7844	TURNER	1500	0	1500
.				
14 rows selected.				

1.10 NVL 함수

- 1) Null 값을 어떤 특정한 값(실제 값)으로 변환하는데 사용한다.
- 2) 사용될 수 있는 데이터 타입은 날짜, 문자, 숫자입니다.

- 3) NVL 함수를 사용할 때 전환되는 값의 데이터 타입을 일치 시켜야 한다.

1.10.1 Syntax

NVL(expr1,expr2)

expr1	Null 값을 포함하고 있는 Column이나 표현식
expr2	Null 변환을 위한 목표 값

1.10.2 다양한 데이터형에 대한 NVL 변형

데이터형	변환 예
NUMBER	NVL(comm, 0)
DATE	NVL(hiredate, '01-JAN-99')
CHAR or VARCHAR2	NVL(job, '업무없음')

문제 5) EMP 테이블에서 이름, 급여, 보너스, 연봉을 출력하여라

<pre>SQL> SELECT ename,sal,comm,sal*12+NVL(comm,0) 2 FROM emp;</pre>
<pre>ENAME SAL COMM SAL*12+NVL(COMM,0) ----- ----- ----- . . . JONES 2975 35700 MARTIN 1250 16400 ALLEN 1600 19500 TURNER 1500 0 . . . 14 rows selected.</pre>

1.11 열에 별칭(Alias) 부여

질의의 결과를 출력할 때 보통 SQL*Plus 는 열 Heading 으로 선택된 열 이름을 사용합니다. 이 Heading 은 때로 사용자가 이해하기가 어려운 경우가 있기 때문에 열 Heading 을 변경하여 질의 결과를 출력하면 보다 쉽게 사용자가 이해할 수 있습니다.

1.11.1 열 별칭(Alias) 정의

- 1) 열 Heading 이름을 변경 합니다.
- 2) 계산에 유용합니다.
- 3) 열 이름 바로 뒤에 사용합니다. 열 이름과 별칭 사이에 키워드 AS 를 넣기도 합니다.
- 4) 공백이나 특수 문자 또는 대문자가 있으면 이중 인용부호(“ ”)가 필요 합니다.

문제 6) EMP 테이블에서 ENAME 를 NAME 로 SAL 을 SALARY 로 출력하여라.

```
SQL> SELECT ename AS name, sal salary  
2 FROM emp;
```

NAME	SALARY
KING	5000
BLAKE	2850
CLARK	2450
.....	
14 rows selected.	

문제 7) EMP 테이블에서 ENAME 를 Name 로 SAL*12 를 Annual Salary 로 출력하여라

```
SQL> SELECT ename "Name", sal*12 "Annual Salary"  
2 FROM emp;
```

Name	Annual Salary
KING	60000
BLAKE	34200
.....	
14 rows selected.	

문제 8) EMP 테이블에서 ENAME 를 Name 로 SAL*12 를 Annual Salary 로 출력하여라

```
SQL> SELECT ename "성 명", sal "급 여"  
2 FROM emp;
```

성 명	급 여
KING	5000
BLAKE	2850
CLARK	2450
JONES	2975
.....	
14 rows selected.	

1.12 연결 연산자

연결 연산자(||)를 사용하여 문자 표현식을 생성하기 위해 다른 열, 산술 표현식, 상수 값에 열을 연결 할 수 있습니다. 연결자의 왼쪽에 있는 열은 단일 결과 열을 만들기 위해 조합 됩니다.

- 1) 열이나 문자 STRING 을 다른 열에 연결 합니다.
- 2) 두개의 “||”로 연결 합니다.
- 3) 문자 표현식의 결과 열을 생성 합니다.

문제 9) EMP 테이블에서 이름과 업무를 연결하여 출력하여라.

```
SQL> SELECT ename || ' ' || job AS "Employees"
  2  FROM emp;

Employees
-----
KING PRESIDENT
BLAKE MANAGER
...
14 rows selected.
```

1.13 LITERAL 문자 STRING

LITERAL은 열 이름이나 열 별칭이 아닌 SELECT 목록에 포함되어 있는 문자, 표현식, 숫자입니다. 그것은 RETURN되는 각각의 행에 대해 출력됩니다. LITERAL과 STRING은 질의 결과에 포함될 수 있으며 SELECT 목록에서 열과 똑같이 취급됩니다. 날짜와 문자 LITERAL은 단일 인용 부호(‘ ’)를 사용하여야 하고 숫자 LITERAL은 사용하지 않습니다.

- 1) SELECT 절에 포함된 LITERAL은 문자, 표현식, 숫자입니다.
- 2) 날짜와 문자 LITERAL값은 단일 인용부호(‘ ’)안에 있어야 합니다.
- 3) 각각의 문자 STRING은 RETURN된 각 행에 대한 결과입니다.

문제 10) EMP 테이블에서 이름과 업무를 “KING is a PRESIDENT” 형식으로 출력하여라.

```
SQL> SELECT ename || ' ' || 'is a' || ' ' || job AS "Employees Details"
  2  FROM emp;

Employees Details
-----
KING is a PRESIDENT
BLAKE is a MANAGER
...
14 rows selected.
```

문제 11) EMP 테이블에서 이름과 연봉을 “KING: 1 Year salary = 60000” 형식으로 출력하여라.

```
SQL> SELECT ename || ': 1 Year salary = ' || sal * 12 Monthly
  2  FROM emp;

MONTHLY
-----
KING: 1 Year salary = 60000
...
14 rows selected.
```

1.14 중복 행의 제거

특별히 명시되지 않았다면, SQL*Plus 는 중복되지는 행을 제거하지 않고 Query 결과를 출력합니다. 결과에서 중복되는 행을 제거하기 위해서는 SELECT 키워드 바로 뒤에 DISTINCT 를 기술한다.

문제 12) EMP 테이블에서 JOB 을 모두 출력하여라

```
SQL> SELECT job  
2  FROM emp;
```

```
JOB  
-----  
PRESIDENT  
MANAGER  
MANAGER  
MANAGER  
SALESMAN  
SALESMAN  
SALESMAN  
CLERK  
SALESMAN  
ANALYST  
CLERK  
ANALYST  
CLERK  
CLERK
```

```
14 rows selected.
```

문제 13) EMP 테이블에서 담당하고 있는 업무의 종류를 출력하여라.

```
SQL> SELECT DISTINCT job  
2  FROM emp;
```

```
JOB  
-----  
ANALYST  
CLERK  
MANAGER  
PRESIDENT  
SALESMAN
```

문제 14) 부서별로 담당하는 업무를 한번씩 출력하여라.

```
SQL> SELECT DISTINCT deptno, job  
2  FROM emp;
```

```
DEPTNO JOB
```

```
-----  
10 CLERK  
10 MANAGER  
10 PRESIDENT  
20 ANALYST  
20 CLERK  
20 MANAGER  
30 CLERK  
30 MANAGER  
30 SALESMAN
```

```
9 rows selected.
```

☞ Guidelines

- 1) DISTINCT라는 키워드는 항상 SELECT 바로 다음에 기술한다.
- 2) DISTINCT 뒤에 나타나는 칼럼들은 모두 DISTINCT의 영향을 받는다.
- 3) DISTINCT 뒤에 여러 개의 칼럼을 기술하였을 때 나타나는 행은 칼럼의 조합들이 중복되지 않게 나타난다.
- 4) DISTINCT를 사용하여 나타나는 결과는 기본적으로 오름차순 정렬된다.

◆ 연습문제 ◆

1. 아래의 SELECT 문장이 성공적으로 수행 될까요? (참 / 거짓)

```
SQL> SELECT ename 이름, job 업무, sal 급여  
2  FROM emp;
```

2. 아래의 SELECT 문장이 성공적으로 수행 될까요? (참 / 거짓)

```
SQL> SELECT *  
2  FROM salgrade;
```

3. 이 문장에 에러가 있습니다. 올바르게 작성하시오.

```
SQL> SELECT empno,ename,sal X 12 년 봉  
2  FROM emp;
```

4. EMP 테이블의 구조와 내용을 조회하여라.

5. EMP 테이블에서 중복되지 않는 부서번호를 출력하시오.

6. EMP 테이블의 이름과 업무를 연결하여 출력하여라.

7. DEPT 테이블의 부서명과 위치를 연결하여 출력하여라.

8. EMP 테이블의 업무와 급여를 연결하여 출력하여라.

9. 6,7,8 번의 결과를 부석하여 설명하여라.

1. 특정 행의 검색

일반적인 경우 테이블에 있는 모든 자료를 조회할 필요 없이 사용자가 원하는 자료를 조회하는 경우가 대부분입니다. 이러한 질의를 만족하게 하는 것이 WHERE 절입니다. WHERE 절은 수행될 조건 절을 포함하며 FROM 절 바로 다음에 기술됩니다.

1.1 Syntax

```
SELECT [DISTINCT]      {*, column [alias], . . .}
      FROM          table_name
      [WHERE         condition]
      [ORDER BY     {column, expression} [ASC | DESC]];
```

DISTINCT	중복 행 제거 옵션
*	테이블의 모든 column 출력
alias	해당 column에 대한 다른 이름 부여
table_name	테이블명 질의 대상 테이블 이름
WHERE	조건을 만족하는 행들만 검색
condition	column 명, 표현식, 문자 상수, 숫자 상수, 비교 연산자로 구성된다.
ORDER BY	질의 결과 정렬을 위한 옵션(ASC:오름차순(Default), DESC 내림차순)

1.2 WHERE 절에 사용되는 연산자

- 1) WHERE 절을 사용하여 행들을 제한할 수 있다
- 2) WHERE 절은 FROM 절 다음에 온다.
- 3) 조건은 아래의 것으로 구성된다.
 - ① column 명, 표현식, 상수
 - ② 비교 연산자, SQL 연산자, 논리연산자
 - ③ 문자(Literal)

1.2.1 비교 연산자

연산자	의 미
=	같다
>	보다 크다
>=	보다 크거나 같다
<	보다 작다
<=	보다 작거나 같다
<>, !=, ^=	같지 않다
NOT Column_name =	같지 않다
NOT Column_name >	보다 크지 않다

문제 1) EMP 테이블에서 급여가 3000 이상인 사원의 정보를 사원번호, 이름, 담당업무, 급여를 출력하여라

```
SQL> SELECT empno,ename,job,sal
  2  FROM emp
  3 WHERE sal >= 3000;

EMPNO ENAME      JOB          SAL
----- -----
 7839 KING        PRESIDENT    5000
 7902 FORD        ANALYST     3000
 7788 SCOTT       ANALYST     3000
```

문제 2) EMP 테이블에서 담당 업무가 Manager 인 사원의 정보를 사원번호, 성명, 담당업무, 급여, 부서번호를 출력하여라.

```
SQL> SELECT empno,ename,job,sal,deptno
  2  FROM emp
  3 WHERE job = 'MANAGER';

EMPNO ENAME      JOB          SAL      DEPTNO
----- -----
 7698 BLAKE       MANAGER     2850      30
 7782 CLARK       MANAGER     2450      10
 7566 JONES       MANAGER     2975      20
```

```
SQL> SELECT empno,ename,job,sal,deptno
  2  FROM emp
  3 WHERE job = 'Manager';

no rows selected
```

문제 3) EMP 테이블에서 입사일자가 1982년 1월 1일 이후에 입사한 사원의 정보를 사원 번호, 성명, 담당업무, 급여, 입사일자, 부서번호를 출력하여라

```
SQL> SELECT empno,ename,job,sal,hiredate,deptno
  2  FROM emp
  3 WHERE hiredate >= '01-JAN-82';

EMPNO ENAME      JOB          SAL      HIREDATE           DEPTNO
----- -----
 7788 SCOTT       ANALYST     3000   09-DEC-82            20
 7876 ADAMS       CLERK       1100   12-JAN-83            20
 7934 MILLER      CLERK       1300   23-JAN-82            10
```

♣ 참고

- 1) 문자 STRING과 날짜 같은 단일 인용 부호(‘ ’)를 사용한다.
- 2) 문자 같은 대소문자를 구분하고 날짜 같은 날짜 형식을 구분합니다.
- 3) Default 날짜 형식은 ‘DD-MON-YY’ 입니다.

1.2.2 SQL 연산자

연산자	설명
BETWEEN a AND b	a 와 b 사이에 있다.(a, b 값 포함)
IN (list)	list 의 값 중 어느 하나와 일치한다.
LIKE	문자 형태와 일치한다.(%,_사용)
IS NULL	NULL 값을 가졌다.
NOT BETWEEN a AND b	a 와 b 사이에 있지않다.(a, b 값 포함하지 않음)
NOT IN (list)	list 의 값과 일치하지 않는다..
NOT LIKE	문자 형태와 일치하지 않는다.
NOT IS NULL	NULL 값을 갖지 않는다.

가) BETWEEN 연산자

두 값의 범위에 해당하는 행을 출력하기 위해 사용한다.

문제 4) EMP 테이블에서 급여가 1250 에서 1500 사이의 정보를 성명, 담당업무, 급여, 부서번호를 출력하여라

```
SQL> SELECT ename, job, sal, deptno  
  2 FROM emp  
  3 WHERE sal BETWEEN 1300 AND 1500;
```

ENAME	JOB	SAL	DEPTNO
TURNER	SALESMAN	1500	30
MILLER	CLERK	1300	10

```
SQL> SELECT ename, job, sal, deptno  
  2 FROM emp  
  3 WHERE sal >= 1300 AND sal <= 1500;
```

ENAME	JOB	SAL	DEPTNO
TURNER	SALESMAN	1500	30
MILLER	CLERK	1300	10

♣ 참고

BETWEEN 연산자를 기술할 경우 명시된 값도 포함 된다. 또한 작은 값을 앞에 기술하고 큰 값을 뒤에 기술하여야 한다.

```
SQL> SELECT ename, job, sal, deptno  
2  FROM emp  
3  WHERE sal BETWEEN 1500 AND 1300;  
  
no rows selected
```

나) IN 연산자

목록에 있는 값에 대해서 출력하기 위해 IN 연산자를 사용한다.

문제 5) EMP 테이블에서 사원번호가 7902, 7788, 7566 인 사원의 정보를 사원번호, 성명, 담당업무, 급여, 입사일자를 출력하여라

```
SQL> SELECT empno,ename,job,sal,hiredate  
2  FROM emp  
3  WHERE empno IN (7902,7788,7566);  
  
EMPNO ENAME      JOB          SAL HIREDATE  
----- -----  
7566 JONES       MANAGER     2975 02-APR-81  
7788 SCOTT       ANALYST    3000 09-DEC-82  
7902 FORD        ANALYST    3000 03-DEC-81  
  
SQL> SELECT empno,ename,job,sal,hiredate  
2  FROM emp  
3  WHERE empno = 7902 OR empno = 7788 OR empno = 7566;  
  
EMPNO ENAME      JOB          SAL HIREDATE  
----- -----  
7566 JONES       MANAGER     2975 02-APR-81  
7788 SCOTT       ANALYST    3000 09-DEC-82  
7902 FORD        ANALYST    3000 03-DEC-81
```

다) LIKE 연산자

- 1) 검색 STRING 값에 대한 와일드 카드 검색을 위해서 LIKE 연산자를 사용한다.
- 2) 검색 조건은 LITERAL 문자나 숫자를 포함할 수 있다.
- 3) '%'는 문자가 없거나 하나 이상의 문자를 '_'는 하나의 문자와 대치됩니다.
- 4) 패턴 일치 문자를 조합할 수 있습니다.
- 5) '%'나 '_'에 대해서 검색하기 위해서는 Escape 식별자를 이용할 수 있습니다.

예) name에 값이 X_Y가 포함되어 있는 문자열을 조회하고자 할 경우 Escape를 사용한다

```
WHERE name LIKE '%X\%Y%' ESCAPE '\';
```

문제 6) EMP 테이블에서 입사일자가 82년도에 입사한 사원의 정보를 사원번호, 성명, 담당업무, 급여, 입사일자, 부서번호를 출력하여라

```
SQL> SELECT empno,ename,job,sal,hiredate,deptno  
2 FROM emp  
3 WHERE hiredate LIKE '%82';
```

EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
7788	SCOTT	ANALYST	3000	09-DEC-82	20
7934	MILLER	CLERK	1300	23-JAN-82	10

♣ 참고

- 1) 기본 날짜 형식이 'YY-MM-DD'일 경우는 WHERE hiredate LIKE '82%';로 기술한다.
- 2) SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'YY-MM-DD'; 명령어로 연결되어 있는 SQL*Plus 창(SESSION)에서 날짜 타입을 바꿀 수 있다.

라) IS NULL 연산자

NULL값은 값이 없거나, 알 수 없거나, 적용할 수 없다는 의미이므로 NULL값을 조회하고자 할 경우에 사용한다.

문제 7) EMP 테이블에서 보너스가 NULL인 사원의 정보를 사원번호, 성명, 담당업무, 급여, 입사일자, 부서번호를 출력하여라

```
SQL> SELECT empno,ename,job,sal,comm,deptno  
2 FROM emp  
3 WHERE comm IS NULL;
```

EMPNO	ENAME	JOB	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	5000		10
.....					

10 rows selected.

1.2.3 논리 연산자

연산자	의미
AND	양쪽 컴포넌트의 조건이 TRUE이면 TRUE를 RETURN합니다.
OR	한쪽 컴포넌트의 조건만이 TRUE이면 TRUE를 RETURN합니다
NOT	이후의 조건이 FALSE이면 TRUE를 RETURN합니다

가. AND 연산자

- 1) 양쪽의 조건이 참이어야 TRUE 를 RETURN 한다.
- 2) 다음의 테이블은 AND 로 두개의 표현식을 조합한 결과를 보여 준다.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

문제 8) EMP 테이블에서 급여가 1100 이상이고 JOB 이 Manager 인 사원의 정보를 사원번호, 성명, 담당업무, 급여, 입사일자, 부서번호를 출력하여라

```
SQL> SELECT empno,ename,job,sal,hiredate,deptno
  2 FROM emp
  3 WHERE sal >= 1100 AND job = 'MANAGER';
```

EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
7698	BLAKE	MANAGER	2850	01-MAY-81	30
7782	CLARK	MANAGER	2450	09-JUN-81	10
7566	JONES	MANAGER	2975	02-APR-81	20

나) OR 연산자

- 1) 한쪽의 조건만 참이면 TRUE 를 RETURN 한다.
- 2) 다음의 테이블은 OR 로 두개의 표현식을 조합한 결과를 보여 준다.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

문제 9) EMP 테이블에서 급여가 1100 이상이거나 JOB 이 Manager 인 사원의 정보를 사원번호, 성명, 담당업무, 급여, 입사일자, 부서번호를 출력하여라

```
SQL> SELECT empno,ename,job,sal,hiredate,deptno
  2 FROM emp
  3 WHERE sal >= 1100 OR job = 'MANAGER';
```

EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
7839	KING	PRESIDENT	5000	17-NOV-81	10
7698	BLAKE	MANAGER	2850	01-MAY-81	30
7782	CLARK	MANAGER	2450	09-JUN-81	10
.....					
12 rows selected.					

다) NOT 연산자

- 1) NOT 연산자 우측의 값이 거짓이면 TRUE 를 RETURN 한다
- 2) 다음의 테이블은 NOT 으로 두개의 표현식을 조합한 결과를 보여 준다.

OR	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

문제 10) EMP 테이블에서 급여가 JOB 이 Manager,Clerk,Analyst 가 아닌 사원의 정보를 사원번호, 성명, 담당업무, 급여, 부서번호를 출력하여라

```
SQL> SELECT empno,ename,job,sal,deptno  
2 FROM emp  
3 WHERE job NOT IN ('MANAGER','CLERK','ANALYST');
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7839	KING	PRESIDENT	5000	10
7654	MARTIN	SALESMAN	1250	30
7499	ALLEN	SALESMAN	1600	30
7844	TURNER	SALESMAN	1500	30
7521	WARD	SALESMAN	1250	30

♣ 참고

NOT 연산자는 BETWEEN, LIKE, IS NULL 과 같은 다른 SQL 연산자와 함께 사용될 수 있습니다.

```
. . . . WHERE NOT job IN ('CLERK','ANALYST');  
. . . . WHERE sal NOT BETWEEN 1000 AND 1500;  
. . . . WHERE ename NOT LIKE '%A%';  
. . . . WHERE comm IS NOT NULL;
```

1.3.4 우선 순위 규칙

- 1) 괄호
- 2) 모든 비교 연산자
- 3) NOT
- 4) AND
- 5) OR

문제 11) 업무가 PRESIDENT 이고 급여가 1500 이상이거나 업무가 SALESMAN 인 사원의 정보를
사원번호, 이름, 업무, 급여를 출력하여라.

```
SQL> SELECT empno,ename,job,sal  
2 FROM emp  
3 WHERE job = 'SALESMAN' OR job = 'PRESIDENT' AND sal > 1500;
```

EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000
7654	MARTIN	SALESMAN	1250
7499	ALLEN	SALESMAN	1600
7844	TURNER	SALESMAN	1500
7521	WARD	SALESMAN	1250

문제 12) 업무가 PRESIDENT 또는 SALESMAN 이고 급여가 1500 이상이거나 사원의 정보를 사원
번호, 이름, 업무, 급여를 출력하여라.

```
SQL> SELECT empno,ename,job,sal  
2 FROM emp  
3 WHERE (job = 'SALESMAN' OR job = 'PRESIDENT') AND sal > 1500;
```

EMPNO	ENAME	JOB	SAL
7839	KING	PRESIDENT	5000
7499	ALLEN	SALESMAN	1600

2. ORDER BY 절

질의 결과에 RETURN 되는 행의 순서는 정의되지 않습니다. ORDER BY 절은 행을 정렬하는
데 사용할 수 있습니다. ORDER BY 절 사용하는 경우 SELECT 문의 맨 뒤에 기술되어야 합니다.
정렬을 위한 표현식이나 Alias를 명시할 수 있습니다.

2.1 Syntax

```
SELECT [DISTINCT]      {*, column [alias], . . . }  
      FROM            table_name  
      [WHERE           condition]  
      [ORDER BY        {column, expression} [ASC | DESC]];
```

ORDER BY 검색된 행이 출력되는 순서를 명시합니다.

ASC 행의 오름차순 정렬(Default)

DESC 행의 내림차순 정렬

♣ 참고

ORDER BY 절이 사용되지 않았다면, 정렬 순서가 정의되지 않은 것이며 오라클 서버는 똑같은 질의를 두 번 실행했을 때 행을 똑같은 순서로 나타내지 않을 수도 있습니다.

2.2 데이터의 정렬

1) 디폴트 정렬은 오름차순입니다:

- ① 숫자 값은 가장 적은 값이 먼저 출력됩니다.(예 : 1 ~ 999)
- ② 날짜 값은 가장 빠른 값이 먼저 출력됩니다.(예 : 01-JAN-92 ~ 01-JAN-95)
- ③ 문자 값은 알파벳 순서로 출력됩니다.(예 : A ~ Z ~ a ~ z)
- ④ Null 값은 오름차순에서는 제일 나중에 그리고 내림차순에서는 제일 먼저 옵니다.

2) 디플트 순서 변경

- ① 행이 디스플레이 되는 순서를 바꾸기 위해서, ORDER BY 절에서 열 이름 뒤에 DESC 키워드를 명시해야 합니다.

문제 13) EMP 테이블에서 입사일자 순으로 정렬하여 사원번호, 이름, 업무, 급여, 입사일자, 부서번호를 출력하여라.

```
SQL> SELECT hiredate,empno,ename,job,sal,deptno
  2  FROM emp
  3 ORDER BY hiredate;
```

HIREDATE	EMPNO	ENAME	JOB	SAL	DEPTNO
17-DEC-80	7369	SMITH	CLERK	800	20
20-FEB-81	7499	ALLEN	SALESMAN	1600	30
22-FEB-81	7521	WARD	SALESMAN	1250	30
02-APR-81	7566	JONES	MANAGER	2975	20
.....					
14 rows selected.					

문제 14) EMP 테이블에서 가장 최근에 입사한 순으로 사원번호, 이름, 업무, 급여, 입사일자, 부서번호를 출력하여라.

```
SQL> SELECT hiredate,empno,ename,job,sal, deptno
  2  FROM emp
  3 ORDER BY hiredate DESC;
```

HIREDATE	EMPNO	ENAME	JOB	SAL	DEPTNO
12-JAN-83	7876	ADAMS	CLERK	1100	20
09-DEC-82	7788	SCOTT	ANALYST	3000	20
23-JAN-82	7934	MILLER	CLERK	1300	10
.....					
14 rows selected.					

2.2.1 다양한 정렬 방법

```
SQL> SELECT empno,ename,job,sal,sal*12 annsal  
2  FROM emp  
3  ORDER BY annsal;
```

```
SQL> SELECT empno,ename,job,sal,sal*12 annsal  
2  FROM emp  
3  ORDER BY sal*12;
```

```
SQL> SELECT empno,ename,job,sal,sal*12 annsal  
2  FROM emp  
3  ORDER BY 5;
```

2.3 다중 열에 의한 정렬

- 1) 하나 이상의 열로 질의 결과를 정렬할 수 있습니다.
- 2) 주어진 테이블에 있는 개수까지만 가능합니다.
- 3) ORDER BY 절에서 열을 명시하고, 열 이름은 촘촘으로 구분합니다.
- 4) 열의 순서를 바꾸고자 한다면 열 이름 뒤에 DESC를 명시합니다.
- 5) SELECT 절에 포함되지 않는 열로 정렬할 수도 있습니다.

문제 15) EMP 테이블에서 부서번호로 정렬한 후 부서번호가 같을 경우 급여가 많은 순으로 정렬하여 사원번호, 성명, 업무, 부서번호, 급여를 출력하여라.

```
SQL> SELECT deptno,sal,empno,ename,job  
2  FROM emp  
3  ORDER BY deptno, sal DESC;
```

DEPTNO	SAL	EMPNO	ENAME	JOB
10	5000	7839	KING	PRESIDENT
10	2450	7782	CLARK	MANAGER
10	1300	7934	MILLER	CLERK
20		7788	SCOTT	ANALYST
20	3000	7902	FORD	ANALYST
20	2975	7566	JONES	MANAGER
20	1100	7876	ADAMS	CLERK
20	800	7369	SMITH	CLERK
30	2850	7698	BLAKE	MANAGER
30	1600	7499	ALLEN	SALESMAN
30	1500	7844	TURNER	SALESMAN
30	1250	7654	MARTIN	SALESMAN
30	1250	7521	WARD	SALESMAN
30	950	7900	JAMES	CLERK

14 rows selected.

문제 16) EMP 테이블에서 첫번째 정렬은 부서번호로 두번째 정렬은 업무로 세번째 정렬은 급여가 많은 순으로 정렬하여 사원번호, 성명, 입사일자, 부서번호, 업무, 급여를 출력하여라.

```
SQL> SELECT deptno, job, sal, empno, ename, hiredate  
2  FROM emp  
3  ORDER BY deptno, job, sal DESC;
```

DEPTNO	JOB	SAL	EMPNO	ENAME	HIREDATE
10	CLERK	1300	7934	MILLER	23-JAN-82
10	MANAGER	2450	7782	CLARK	09-JUN-81
10	PRESIDENT	5000	7839	KING	17-NOV-81
20	ANALYST		7788	SCOTT	09-DEC-82
20	ANALYST	3000	7902	FORD	03-DEC-81
20	CLERK	1100	7876	ADAMS	12-JAN-83
20	CLERK	800	7369	SMITH	17-DEC-80
20	MANAGER	2975	7566	JONES	02-APR-81
30	CLERK	950	7900	JAMES	03-DEC-81
30	MANAGER	2850	7698	BLAKE	01-MAY-81
30	SALESMAN	1600	7499	ALLEN	20-FEB-81
30	SALESMAN	1500	7844	TURNER	08-SEP-81
30	SALESMAN	1250	7654	MARTIN	28-SEP-81
30	SALESMAN	1250	7521	WARD	22-FEB-81

14 rows selected.

◆ 연습문제 ◆

1. WHERE 절에 HIREDATE 의 비교를 '01-JAN-82' 가 아닌 '01-jan-82' 로 기술하면 결과는 ?

```
SQL> SELECT *
  2  FROM emp
  3 WHERE hiredate = '23-jan-82';
```

2. EMP 테이블에서 급여가 3000 이상인 사원의 정보를 사원번호, 이름, 담당업무, 급여를 출력하는 SELECT 문장을 작성하시오.

3. EMP 테이블에서 사원번호가 7788 인 사원의 이름과 부서번호를 출력하는 SELECT 문장을 작성하시오.

4. EMP 테이블에서 입사일이 February 20, 1981 과 May 1, 19981 사이에 입사한 사원의 이름, 업무, 입사일을 출력하는 SELECT 문장을 작성하시오. 단 입사일 순으로 출력하시오.

5. EMP 테이블에서 부서번호가 10,20 인 사원의 모든 정보를 출력하는 SELECT 문장을 작성하시오. 단 이름순으로 정렬하여라.

6. EMP 테이블에서 급여가 1500 이상이고 부서번호가 10,30 인 사원의 이름과 급여를 출력하는 SELECT 문장을 작성하여라. 단 HEADING 을 Employee 과 Monthly Salary 로 출력하여라.

7. EMP 테이블에서 1982 년에 입사한 사원의 모든 정보를 출력하는 SELECT 문을 작성하여라.

8. EMP 테이블에서 COMM 에 NULL 이 아닌 사원의 모든 정보를 출력하는 SELECT 문을 작성하여라.

9. EMP 테이블에서 보너스가 급여보다 10%가 많은 모든 종업원에 대하여 이름, 급여, 보너스를 출력하는 SELECT 문을 작성하여라.

10. EMP 테이블에서 업무가 Clerk 이거나 Analyst 이고 급여가 1000,3000,5000 이 아닌 모든 사원의 정보를 출력하는 SELECT 문을 작성하여라.

11. EMP 테이블에서 이름에 L 이 두 자가 있고 부서가 30 이거나 또는 관리자가 7782 인 사원의 모든 정보를 출력하는 SELECT 문을 작성하여라.

1. SQL 함수

제공되는 함수들은 기본적인 Query 문을 더욱 강력하게 해주고 데이터 값을 조작하는데 사용된다. 여러분은 단일 행 함수를 이용하여 문자, 숫자, 날짜 함수에 대해 살펴볼 뿐만 아니라 형을 전환하는 함수들에 대해서도 살펴본다. 또한 복수 행 함수를 이용하여 복수의 행 조합하여 그룹 당 하나의 결과를 출력하는 그룹 함수에 대해서 살펴본다.

1.1 SQL 함수의 특징 및 이점

- 1) 데이터에 계산을 수행할 수 있다.
- 2) 개별적인 데이터 항목을 수정할 수 있다.
- 3) 행의 그룹에 대해 결과를 조작할 수 있다.
- 4) 출력을 위한 날짜와 숫자 형식을 조절할 수 있다.
- 5) 열의 자료형을 변환할 수 있다.

1.2 단일 행 함수(Single Row Function)

이 함수는 단일 행에 대해서만 적용 가능하고 행별로 하나의 결과를 RETURN 한다..

<code>Function_name (column expression [,arg1,arg2,])</code>

function_name 함수 명

column 데이터 베이스의 Column Name

expression 어떤 문자 스트링이거나 계산된 표현식

arg1,arg2 함수에 의해 사용될 수 있는 인수

1.2.1 단일 행 함수가 이용되는 곳

- 1) 데이터에 대해 계산을 수행할 경우
- 2) 각각의 데이터 항목을 변경할 경우
- 3) 출력할 날짜 형식을 변경할 경우
- 4) Column Data Type 을 변경할 경우

1.2.2 단일 행 함수의 종류

- 1) 문자형 함수 : 문자를 입력 받고 문자와 숫자 값 모두를 RETURN 할 수 있다.
- 2) 숫자형 함수 : 숫자를 입력 받고 숫자를 RETURN 한다.
- 3) 날짜형 함수 : 날짜형에 대해 수행하고 숫자를 RETURN 하는 MONTHS_BETWEEN 함수를 제외하고 모두 날짜 데이터형의 값을 RETURN 한다.
- 4) 변환형 함수 : 어떤 데이터형의 값을 다른 데이터형으로 변환한다.
- 5) 일반적인 함수 : NVL, DECODE

1.2.3 단일 행 함수의 특징

- 1) 질의에서 RETURN 되는 각각의 행에 대해 수행
- 2) 행별로 하나의 결과를 RETURN
- 3) 참조 시 사용한 데이터 형과 다른 데이터 형으로 결과를 RETURN 할 수 있다,
- 4) 하나 이상의 인수를 필요로 한다.
- 5) SELECT, WHERE, ORDER BY 절에서 사용할 수 있습니다.
- 6) 함수를 중첩할 수 있습니다.
 - ① 단일 행 함수들은 여러 LEVEL 에 걸쳐 중첩 사용이 가능하다.
 - ② 중첩된 함수들은 가장 하위 LEVEL 에서 가장 상위 LEVEL 순으로 진행된다.

1.3 문자형 함수(Character Function)

종 류	함 수	사 용 목 적
변환 함수	LOWER	알파벳 값을 소문자로 변환
	UPPER	알파벳 값을 대문자로 변환
	INITCAP	첫번째 글자만 대문자로 변환
문자 조작 함수	CONCAT	두 문자열을 연결(합성)
	SUBSTR	문자열 중 특정 문자 또는 문자열의 일부분을 선택
	LENGTH	문자열의 길이를 구함
	INSTR	명명된 문자의 위치를 구함
	LPAD	왼쪽 문자 자리 채움
	RPAD	오른쪽 문자 자리 채움
	LTRIM	왼쪽 문자를 지움
	RTRIM	오른쪽 문자를 지움
	TRANSLATE	특정 문자열을 대체
	REPLACE	특정 문자열을 대신

1.3.1 LOWER 함수

대소문자가 혼합되어 있거나 대문자인 문자열을 소문자로 변환 합니다.

Syntax	LOWER(column expression)
사 용 예	LOWER('MANAGER') → manager

문제 1) EMP 테이블에서 scott 의 정보를 사원번호, 성명, 담당업무(소문자로), 부서번호를 출력하여라.

```
SQL> SELECT empno,ename,LOWER(job),deptno
  2  FROM emp
  3 WHERE LOWER(ename) = 'scott';

    EMPNO ENAME      LOWER(JOB)   DEPTNO
----- ----- -----
    7788 SCOTT      analyst        20
```

1.3.2 UPPER 함수

대문자가 혼합되어 있거나 소문자인 문자열을 대문자로 변환 합니다.

Syntax	UPPER(column expression)
사용 예	UPPER('manager') → MANAGER

문제 2) EMP 테이블에서 scott 의 정보를 사원번호, 성명, 담당업무, 부서번호를 출력하여라.

```
SQL> SELECT empno,ename,job,deptno
  2  FROM emp
  3 WHERE ename = UPPER('scott');

    EMPNO ENAME      JOB          DEPTNO
----- ----- -----
    7788 SCOTT      ANALYST        20
```

1.3.3 INITCAP 함수

각 단어의 첫번째 문자를 대문자로 나머지 문자는 소문자로 변경합니다.

Syntax	INITCAP(column expression)
사용 예	INITCAP('ORACLE SERVER') → Oracle Server

문제 3) DEPT 테이블에서 첫 글자만 대문자로 변환하여 모든 정보를 출력하여라.

```
SQL> SELECT deptno,INITCAP(dname),INITCAP(loc)
  2  FROM dept;

    DEPTNO INITCAP(DNAME) INITCAP(LOC)
----- -----
    10 Accounting      New York
    20 Research        Dallas
    30 Sales           Chicago
    40 Operations      Boston
```

1.3.4 CONCAT 함수

두 개의 문자열을 합성합니다. CONCAT는 두개의 매개변수만 사용 가능합니다.

Syntax	CONCAT(column1 expression1, column2 expression2)
사용 예	INITCAP('ORACLE' , 'SERVER') → ORACLESERVER

문제 4) 두개의 SELECT 문이 있다. 결과의 차이점을 설명하여라

SQL> col e_name format a15 SQL> col e_empno format a15 SQL> col e_job format a15 SQL> SELECT empno,ename,job,CONCAT(empno,ename) e_name, 2 CONCAT(ename,empno) e_empno, 3 CONCAT(ename,job) e_job 4 FROM emp 5 WHERE deptno = 10;
<pre> EMPNO ENAME JOB E_NAME E_EMPNO E_JOB ----- ----- 7839 KING PRESIDENT 7839KING KING7839 KINGPRESIDENT 7782 CLARK MANAGER 7782CLARK CLARK7782 CLARKMANAGER 7934 MILLER CLERK 7934MILLER MILLER7934 MILLERCLERK </pre>
SQL> col no format 99 SQL> col d_name format a18 SQL> col d_deptno format a18 SQL> col d_loc format a25 SQL> SELECT deptno no, dname, loc, CONCAT(deptno,dname) d_name, 2 CONCAT(dname,deptno) d_deptno, CONCAT(dname,loc) d_loc 3 FROM dept;
<pre> NO DNAME LOC D_NAME D_DEPTNO D_LOC ---- ----- 10 ACCOUNTING NEW YORK 10ACCOUNTING ACCOUNTING 10 ACCOUNTING NEW YORK 20 RESEARCH DALLAS 20RESEARCH RESEARCH 20 RESEARCH DALLAS 30 SALES CHICAGO 30SALES SALES 30 SALES CHICAGO 40 OPERATIONS BOSTON 40OPERATIONS OPERATIONS 40 OPERATIONS BOSTON </pre>

♣ 참고

Column의 데이터 타입이 varchar2, number, char의 차이로 varchar2와 number는 가변 길이, char는 고정 길이입니다.

1.3.5 SUBSTR 함수

지정된 길이만큼의 문자열을 추출합니다.

Syntax	SUBSTR(column expression, m [,n])
사용 예	SUBSTR('000101-3234232', 8, 1) → 3

문제 5) EMP 테이블에서 이름의 첫글자가 'K' 보다 크고 'Y'보다 적은 사원의 정보를 사원번호, 이름, 업무, 급여, 부서번호를 출력하여라. 단 이름순으로 정렬하여라.

```
SQL> SELECT empno,ename,job,sal,deptno
  2  FROM emp
  3 WHERE SUBSTR(ename,1,1) > 'K' AND SUBSTR(ename,1,1) < 'Y'
  4 ORDER BY ename;
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7654	MARTIN	SALESMAN	1250	30
7934	MILLER	CLERK	1300	10
7788	SCOTT	ANALYST	3000	20
7369	SMITH	CLERK	800	20
7844	TURNER	SALESMAN	1500	30
7521	WARD	SALESMAN	1250	30

6 rows selected.

1.3.6 LENGTH 함수

문자열의 길이를 숫자 값으로 RETURN 한다.

Syntax	LENGTH(column expression)
사용 예	INITCAP('000101-3234232') → 14

문제 6) EMP 테이블에서 20 번 부서 중 이름의 길이 및 급여의 자릿수를 사원번호, 이름, 이름의 자릿수, 급여, 급여의 자릿수를 출력하여라.

```
SQL> SELECT empno,ename,LENGTH(ename),sal,LENGTH(sal)
  2  FROM emp
  3 WHERE deptno = 20;
```

EMPNO	ENAME	LENGTH(ENAME)	SAL	LENGTH(SAL)
7566	JONES	5	2975	4
7902	FORD	4	3000	4
7369	SMITH	5	800	3
7788	SCOTT	5	3000	4
7876	ADAMS	5	1100	4

1.3.7 INSTR 함수

명명된 문자의 위치를 숫자 값으로 RETURN 한다.

Syntax	INSTR(column expression, m[,n])
사용 예	INSTR('MILLER', 'L', 1, 2) → 4

문제 7) EMP 테이블에서 이름 중 'L'자의 위치를 출력하여라.

```
SQL> SELECT ename, INSTR(ename,'L') e_null, INSTR(ename,'L',1,1) e_11,
2      INSTR(ename,'L',1,2) e_12, INSTR(ename,'L',4,1) e_41,
3      INSTR(ename,'L',4,2) e_42
4  FROM emp
5 ORDER BY ename;

ENAME      E_NULL      E_11      E_12      E_41      E_42
-----  -----  -----  -----  -----  -----
ADAMS          0          0          0          0          0
ALLEN          2          2          3          0          0
BLAKE          2          2          0          0          0
CLARK          2          2          0          0          0
FORD            0          0          0          0          0
JAMES            0          0          0          0          0
JONES            0          0          0          0          0
KING            0          0          0          0          0
MARTIN          0          0          0          0          0
MILLER          3          3          4          4          0
.
.
.
14 rows selected.
```

문제 8) 파일명을 입력을 입력받아 확장자가 없으면 .SQL 을 붙여 출력하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_filename PROMPT '파일명을 입력하시오 : '
DECLARE
    v_filename      VARCHAR2(300) := '&p_filename';
BEGIN
    IF INSTR(v_filename, '.', 1, 1) = 0 THEN
        DBMS_OUTPUT.PUT_LINE('FILE NAME : ' || v_filename || '.SQL');
    ELSIF INSTR(v_filename, '.', 1, 1) >= 1 THEN
        DBMS_OUTPUT.PUT_LINE('FILE NAME : ' || v_filename);
    END IF;
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.3.8 LPAD 함수

문자값을 우측부터 채웁니다.

Syntax	LPAD(column expression, n, 'string')
사용 예	LPAD('MILLER', 10, '*') → ****MILLER

문제 9) 아래 두 문장의 결과를 보고 차이점을 설명하여라.

SQL> SELECT ename,LPAD(ename,15,'*'),sal,LPAD(sal,10,'*')
2 FROM emp
3 WHERE deptno = 10;
ENAME LPAD(ENAME,15,'*') SAL LPAD(SAL,10,'*')

KING *****KING 5000 *****5000
CLARK *****CLARK 2450 *****2450
MILLER *****MILLER 1300 *****1300
SQL> SELECT deptno,dname,LPAD(dname,20,'*')
2 FROM dept;
DEPTNO DNAME LPAD(DNAME,20,'*')

10 ACCOUNTING *****ACCOUNTING
20 RESEARCH *****RESEARCH
30 SALES *****SALES
40 OPERATIONS *****OPERATIONS

1.3.9 RPAD 함수

문자값을 좌측부터 채웁니다.

Syntax	RPAD(column expression, n, 'string')
사용 예	RPAD('MILLER', 10, '*') → MILLER****

문제 10) 아래 두 문장의 결과를 보고 차이점을 설명하여라.

SQL> SELECT ename,RPAD(ename,15,'*'),sal,RPAD(sal,10,'*')
2 FROM emp
3 WHERE deptno = 10;
ENAME RPAD(ENAME,15,'*') SAL RPAD(SAL,10,'*')

KING KING*****
CLARK CLARK*****
MILLER MILLER*****

```
SQL> SELECT deptno,dname,RPAD(dname,20,'*')
2  FROM dept;
```

DEPTNO	DNAME	RPAD(DNAME,20,'*')
10	ACCOUNTING	ACCOUNTING *****
20	RESEARCH	RESEARCH *****
30	SALES	SALES *****
40	OPERATIONS	OPERATIONS *****

1.3.10 LTRIM 함수

왼쪽 문자를 지우는 함수입니다.

Syntax	LTRIM(column1 expression1, column1 expression1)
사용 예	LTRIM('MILLER', 'M') → ILLER

문제 11) EMP 테이블에서 10 번 부서에 대하여 담당 업무 중 좌측에 'A'를 삭제하고 급여 중 좌측의 1을 삭제하여 출력하여라.

```
SQL> SELECT ename,job,LTRIM(job,'A'),sal,LTRIM(sal,1)
2  FROM emp;
```

ENAME	JOB	LTRIM(JOB)	SAL	LTRIM(SAL,1)
KING	PRESIDENT	PRESIDENT	5000	5000
BLAKE	MANAGER	MANAGER	2850	2850
CLARK	MANAGER	MANAGER	2450	2450
JONES	MANAGER	MANAGER	2975	2975
MARTIN	SALESMAN	SALESMAN	1250	250
ALLEN	SALESMAN	SALESMAN	1600	600
TURNER	SALESMAN	SALESMAN	1500	500
JAMES	CLERK	CLERK	950	950
WARD	SALESMAN	SALESMAN	1250	250
FORD	ANALYST	NALYST	3000	3000
SMITH	CLERK	CLERK	800	800
SCOTT	ANALYST	NALYST	3000	3000
ADAMS	CLERK	CLERK	1100	00
MILLER	CLERK	CLERK	1300	300

14 rows selected.

1.3.11 RTRIM 함수

오른쪽 문자를 지우는 함수입니다.

Syntax	RTRIM(column1 expression1, column2 expression2)
사용 예	RTRIM('MILLER', 'R') → MILLER

문제 12) EMP 테이블에서 10 번 부서에 대하여 담당 업무 중 우측에 ‘T’를 삭제하고 급여 중 우측의 0을 삭제하여 출력하여라.

```
SQL> SELECT ename, job, RTRIM(job, 'T'), sal, RTRIM(sal,0)
  2  FROM emp
  3 WHERE deptno = 10;
```

ENAME	JOB	RTRIM(JOB)	SAL	RTRIM(SAL,0)
KING	PRESIDENT	PRESIDEN	5000 5	
CLARK	MANAGER	MANAGER	2450 245	
MILLER	CLERK	CLERK	1300 13	

1.3.11 TRANSLATE 함수

특정 문자열을 대체하는 함수입니다. 즉 str1을 str2 문자로 대체하는 함수이다.

Syntax	TRANSLATE(column1 expression1, 'string1', 'string2')
사용 예	TRANSLATE('MILLER', 'L', '*') → M***ER

문제 13) EMP 테이블에서 성명을 소문자로 바꾸어 출력하여라.

```
SQL> var u_lower varchar2(10)
SQL> var n_h varchar2(10)
SQL> col u_lower format a10
SQL> col n_h format a10
SQL>
SQL> SELECT empno, ename, TRANSLATE(ename, 'ABCDEFGHIJKLMNPQRSTUVWXYZ',
  2  'abcdefghijklmnopqrstuvwxyz') u_lower,
  3  sal, TRANSLATE(sal, '0123456789',
  4  '영일이삼사오육칠팔구') n_h
  5  FROM emp
  6 WHERE deptno = 10;
```

EMPNO	ENAME	U_LOWER	SAL	N_H
7839	KING	king	5000	오영영영
7782	CLARK	clark	2450	이사오영
7934	MILLER	miller	1300	일삼영영

1.3.11 REPLACE 함수

특정 문자열을 대신하는 함수입니다.

Syntax	REPLACE(column1 expression1, 'string1', 'string2')
사용 예	REPLACE('JACK and JUE', 'J', 'BL') → BLACK and BLUE

문제 14) EMP 테이블에서 JOB 에 ‘A’를 ‘\$’로 바꾸어 출력하여라.

```
SQL> SELECT ename, job, REPLACE(job, 'A', '$'), sal  
2 FROM emp;
```

ENAME	JOB	REPLACE(J)	SAL
KING	PRESIDENT	PRESIDENT	5000
BLAKE	MANAGER	M\$N\$GER	2850
CLARK	MANAGER	M\$N\$GER	2450
JONES	MANAGER	M\$N\$GER	2975
MARTIN	SALESMAN	S\$LESMS\$N	1250
.....			
14 rows selected.			

1.4 숫자형 함수

함수	사용 목적
ROUND	숫자를 반올림
TRUNC	숫자를 절삭
MOD	나머지를 구함
POWER	거듭제곱
SQRT	제곱근
SIGN	양수, 음수, 0 인지를 구분
CHR	ASCII 값에 해당하는 문자를 구함

1.4.1 ROUND 함수

명시된 소수점으로 반올림하는 함수입니다. 숫자를 n 자리까지 반올림한다. n이 양수이면 소수 자리를, 음수이면 정수 자리를 사사오입합니다. 생략할 수 있으며 Default는 0입니다.

Syntax	ROUND(column1 expression1, n)
사용 예	ROUND(456.789, 2) → 456.79

문제 15) 다음의 결과를 분석하여라.

```
SQL> SELECT ROUND(4567.678),ROUND(4567.678,0),  
2      ROUND(4567.678,2),ROUND(4567.678,-2)  
3 FROM dual;
```

```
ROUND(4567.678) ROUND(4567.678,0) ROUND(4567.678,2) ROUND(4567.678,-2)
```

4568	4568	4567.68	4600
------	------	---------	------

♣ 참고

DUAL 테이블은 SYS User 가 Owner이며 모든 사용자가 사용할 수 있도록 권한을 부여하였다. Dummy라는 하나의 Column과 X 값을 가지는 하나의 행을 포함합니다. DUAL 테이블은 오직 하나의 값을 출력하고자 할 때 유용합니다. 예를 들어 데이터를 가진 테이블에서 발생되지 않은 상수, 의사열, 표현식의 값인 경우입니다. 즉 임의의 값을 알고자 할 경우 유용하게 사용할 수 있다. 위 SELECT 문장에서 dual이 아닌 dept를 사용하면 결과는 어떻게 될까?

1.4.2 TRUNC 함수

명시된 숫자를 절삭하는 함수입니다. 숫자를 n 자리까지 절삭한다. n이 양수이면 소수자리를, 음수이면 정수자리를 절삭합니다. 생략할 수 있으며 Default는 0입니다.

Syntax	TRUNC(column1 expression1 , n)
사용 예	TRUNC(456.789, 2) → 456.78

문제 16) 다음의 결과를 분석하여라.

```
SQL> SELECT TRUNC(4567.678),TRUNC(4567.678,0),
2      TRUNC(4567.678,2),TRUNC(4567.678,-2)
3 FROM dual;

TRUNC(4567.678) TRUNC(4567.678,0) TRUNC(4567.678,2) TRUNC(4567.678,-2)
-----
4567           4567        4567.67        4500
```

1.4.3 MOD 함수

숫자의 나머지를 구하는 함수입니다.

Syntax	MOD(column1 expression1 , n)
사용 예	MOD(10, 3) → 1

문제 17) EMP 테이블에서 급여를 30으로 나눈 나머지를 구하여 출력하여라.

```
SQL> SELECT sal, MOD(sal,30)
2  FROM emp
3 WHERE deptno = 10;

SAL  MOD(SAL,30)
-----
5000      20
2450      20
1300      10
```

1.4.4 POWER 함수

거듭제곱을 구하는 함수입니다.

Syntax	POWER(column1 expression1 , n)
사용 예	POWER(2, 3) → 8

1.4.5 SQRT 함수

제곱근을 구하는 함수입니다.

Syntax	SQRT(column1 expression1)
사용 예	SQRT(4) → 2

1.4.6 SIGN 함수

주어진 숫자가 양수인지 음수인지 또는 0 인지를 구하는 함수입니다.

Syntax	SIGN(column1 expression1)
사용 예	SIGN(100) → 1

1.4.7 CHR 함수

ASCII Code 값에 해당하는 문자를 구하는 함수입니다.

Syntax	CHR(column1 expression1)
사용 예	CHR(65) → A

문제 18) EMP 테이블에서 20 번 부서 중 이름과 담당 업무를 연결하여 출력하여라. 단 담당 업무를 한 줄 아래로 출력하여라

```
SQL> SELECT empno,ename,job,ename || CHR(10) || job
  2 FROM emp
  3 WHERE deptno = 20;
```

EMPNO	ENAME	JOB	ENAME CHR(10) JOB
7566	JONES	MANAGER	JONES MANAGER
7902	FORD	ANALYST	FORD ANALYST
7369	SMITH	CLERK	SMITH CLERK
7788	SCOTT	ANALYST	SCOTT ANALYST
7876	ADAMS	CLERK	ADAMS CLERK

1.5 날짜형 함수

1.5.1 오라클 날짜 형식

- 1) 오라클은 세기,년,월,일,시,분,초를 내부 숫자(7 Byte) 형식으로 날짜를 저장 합니다.
- 2) Default Date Type 은 DD-MON-YY(변경 가능)입니다.
- 3) 오라클 날짜의 범위는 B.C 4712년 1월 1일부터 A.D 9999년 12월 31일 사이입니다.
- 4) SYSDATE 는 오라클이 설치되어 있는 서버의 현재 날짜와 시간을 RETURN 하는 함수입니다.

1.5.2 날짜 연산

- 1) 날짜에서 숫자를 더하거나 빼어 날짜 결과를 출력
- 2) 날짜 사이의 일수를 알기 위해서 두개의 날짜를 뺍니다.
- 3) 시간을 24로 나누어서 시간을 날짜에 더합니다.

날짜 연산	결과	설명
Date + Number	Date	일수를 날짜에 더합니다.
Date - Number	Date	날짜에서 일수를 뺍니다.
Date - Date	일수	어떤 날짜에서 다른 날짜를 뺍니다
Date + Number / 24	Date	시간을 날짜에 더합니다.

문제 19) EMP 테이블에서 현재까지 근무일 수가 몇주 몇일 인가를 출력하여라. 단 근무 일 수가 많은 사람 순으로 출력하여라.

```
SQL> SELECT ename,hiredate,sysdate,sysdate - hiredate "Total Days",
  2 TRUNC((sysdate - hiredate) / 7, 0) Weeks,
  3 ROUND(MOD((sysdate - hiredate), 7), 0) DAYS
  4 FROM emp
  5 ORDER BY sysdate - hiredate DESC;
```

ENAME	HIREDATE	SYSDATE	Total Days	WEEKS	DAYS
SMITH	17-DEC-80	01-MAR-99	6648.5677	949	6
ALLEN	20-FEB-81	01-MAR-99	6583.5677	940	4
WARD	22-FEB-81	01-MAR-99	6581.5677	940	2
JONES	02-APR-81	01-MAR-99	6542.5677	934	5
BLAKE	01-MAY-81	01-MAR-99	6513.5677	930	4
CLARK	09-JUN-81	01-MAR-99	6474.5677	924	7
TURNER	08-SEP-81	01-MAR-99	6383.5677	911	7
MARTIN	28-SEP-81	01-MAR-99	6363.5677	909	1
KING	17-NOV-81	01-MAR-99	6313.5677	901	7
JAMES	03-DEC-81	01-MAR-99	6297.5677	899	5
FORD	03-DEC-81	01-MAR-99	6297.5677	899	5
.....					
			14 rows selected.		

1.5.3 날짜 함수

날짜 함수는 오라클 날짜에 대해 연산을 합니다. 모든 날짜 함수는 숫자값을 RETURN 하는 데 MONTHS_BETWEEN 을 제외하고는 DATE 형을 RETURN 합니다

날짜 함수	설명
MONTHS_BETWEEN	두 날짜 사이의 월수를 계산
ADD_MONTHS	월을 날짜에 더합니다.
NEXT_DAY	명시된 날짜로부터 다음 요일에 대한 날짜를 나타냅니다.
LAST_DAY	월의 마지막 날을 계산 합니다.
ROUND	날짜를 반올림 합니다.
TRUNC	날짜를 절삭 합니다.

1.5.4 MONTHS_BETWEEN 함수

- 1) 날짜와 날짜 사이의 월수를 계산합니다
- 2) 결과는 음수 또는 양수가 될 수 있습니다.
- 3) 결과의 비정수 부분을 월의 부분을 나타냅니다.

Syntax	MONTHS_BETWEEN(date1, date2)
사용 예	MONTHS_BETWEEN(sysdate,hiredate) → 212.04794

위 예에서 212는 월을 나타내고 .04794는 월의 일부분을 나타냅니다.

문제 20) EMP 테이블에서 10 번 부서 중 현재까지의 근무 월수를 계산하여 출력하여라.

```
SQL> SELECT ename,hiredate,SYSDATE,MONTHS_BETWEEN(SYSDATE,hiredate) m_between,
  2 TRUNC(MONTHS_BETWEEN(SYSDATE,hiredate),0) t_between
  3 FROM emp
  4 WHERE deptno = 10
  5 ORDER BY MONTHS_BETWEEN(SYSDATE,hiredate) DESC;
```

ENAME	HIREDATE	SYSDATE	M_BETWEEN	T_BETWEEN
CLARK	09-JUN-81	10-FEB-99	212.04812	212
KING	17-NOV-81	10-FEB-99	206.79005	206
MILLER	23-JAN-82	10-FEB-99	204.5965	204

1.5.5 ADD_MONTHS 함수

- 1) 날짜에 월을 더합니다(ADD_MONTHS(hiredate,10))
- 2) 날짜에 월을 뺍니다(ADD_MONTHS(hiredate,-10))
- 3) 결과의 날짜형입니다.

Syntax	ADD_MONTHS(date1, n)
--------	----------------------

사용 예	ADD_MONTHS(hiredate,5) → 23-JUN-82
------	------------------------------------

문제 21) EMP 테이블에서 10 번 부서 중 입사 일자로부터 5 개월이 지난 후 날짜를 계산하여 출력하여라.

```
SQL> SELECT ename,hiredate,ADD_MONTHS(hiredate,5) a_month
  2  FROM emp
  3 WHERE deptno = 10
  4 ORDER BY hiredate DESC;
```

ENAME	HIREDATE	A_MONTH
MILLER	23-JAN-82	23-JUN-82
KING	17-NOV-81	17-APR-82
CLARK	09-JUN-81	09-NOV-81

1.5.6 NEXT_DAY 함수

- 1) 명시된 요일의 돌아오는 날짜를 계산 합니다.
- 2) 요일이 아니라 숫자도 가능(SUNDAY:1, MONDAY:2,)
- 3) NLS_LANG OI KOREAN_KOREA.K016KSC5601로 되어 있으면 한글도 사용 가능(일요일, 월요일, 화요일,)

Syntax	NEXT_DAY(date1, 'string' n)
사용 예	NEXT_DAY(hiredate,'FRIDAY') → 29-JAN-82 NEXT_DAY(hiredate,'금요일') → 29-JAN-82

문제 22) EMP 테이블에서 10 번 부서 중 입사 일자로부터 돌아오는 금요일을 계산하여 출력하여라.

```
SQL> SELECT ename,hiredate,NEXT_DAY(hiredate,'FRIDAY') n_day,
  2  NEXT_DAY(hiredate,6) n_6,NEXT_DAY(hiredate,7) n_7
  3  FROM emp
  4 WHERE deptno = 10
  5 ORDER BY hiredate DESC;
```

ENAME	HIREDATE	N_DAY	N_6	N_7
MILLER	23-JAN-82	29-JAN-82	29-JAN-82	30-JAN-82
KING	17-NOV-81	20-NOV-81	20-NOV-81	21-NOV-81
CLARK	09-JUN-81	12-JUN-81	12-JUN-81	13-JUN-81

1.5.7 LAST_DAY 함수

- 1) 월의 마지막 날짜를 계산
- 2) 윤년, 평년은 자동 계산

Syntax	LAST_DAY(date1)
--------	-----------------

사용 예	LAST_DAY(hiredate) → 30-NOV-81
------	--------------------------------

문제 23) EMP 테이블에서 입사한 달의 근무 일수를 계산하여 출력하여라. 단 토요일과 일요일도 근무 일수에 포함한다.

```
SQL> SELECT empno,ename,hiredate, LAST_DAY(hiredate) l_last,
2      LAST_DAY(hiredate) - hiredate l_day
3   FROM emp
4 ORDER BY LAST_DAY(hiredate) - hiredate DESC;
```

EMPNO	ENAME	HIREDATE	L_LAST	L_DAY
7698	BLAKE	01-MAY-81	31-MAY-81	30
7566	JONES	02-APR-81	30-APR-81	28
7900	JAMES	03-DEC-81	31-DEC-81	28
7902	FORD	03-DEC-81	31-DEC-81	28
7844	TURNER	08-SEP-81	30-SEP-81	22
7788	SCOTT	09-DEC-82	31-DEC-82	22
7782	CLARK	09-JUN-81	30-JUN-81	21
7876	ADAMS	12-JAN-83	31-JAN-83	19
7369	SMITH	17-DEC-80	31-DEC-80	14
7839	KING	17-NOV-81	30-NOV-81	13
.....				
14 rows selected.				

1.5.8 ROUND 함수

- 1) 명시된 형식으로 반올림 합니다.
- 2) 날짜를 가장 가까운 년도 또는 월로 반올림할 수 있습니다.
 - ① fmt에 명시된 단위에 대해 반올림한 날짜를 계산
 - ② fmt가 생략되면 날짜를 가장 가까운 날짜로 반올림한다.

Syntax	ROUND(date1 [,fmt])
사용 예	ROUND('25-JUN-99','MONTH') → 01-AUG-99 ROUND('25-JUN-98','YEAR') → 01-JAN-99

1.5.9 TRUNC 함수

- 1) 명시된 형식으로 절삭 합니다.
- 2) 날짜를 가장 가까운 년도 또는 월로 절삭할 수 있습니다.
 - ① fmt에 명시된 단위에 대해 절삭한 날짜를 계산
 - ② fmt가 생략되면 날짜를 가장 가까운 날짜로 절삭한다.

Syntax	TRUNC(date1 [,fmt])
사용 예	TRUNC('25-JUN-99','MONTH') → 01-JUN-99 TRUNC('25-JUN-98','YEAR') → 01-JAN-98

문제 24) EMP 테이블에서 10 번 부서 중 입사한 달의 ROUND 과 TRUNC 함수를 비교합니다.

```
SQL> SELECT ename,hiredate,ROUND(hiredate,'MONTH') m_round,
2      TRUNC(hiredate,'MONTH') m_trunc, ROUND(hiredate,'YEAR') y_round,
3      TRUNC(hiredate,'YEAR') y_trunc
4  FROM emp
5 WHERE deptno = 10
6 ORDER BY hiredate DESC;
```

ENAME	HIREDATE	M_ROUND	M_TRUNC	Y_ROUND	Y_TRUNC
MILLER	23-JAN-82	01-FEB-82	01-JAN-82	01-JAN-82	01-JAN-82
KING	17-NOV-81	01-DEC-81	01-NOV-81	01-JAN-82	01-JAN-81
CLARK	09-JUN-81	01-JUN-81	01-JUN-81	01-JAN-81	01-JAN-81

1.6 변환 함수

1.6.1 데이터의 형 변환

오라클 서버는 어떤 일정한 데이터형의 데이터를 사용해야 하는 곳에, 그것과 다른 데이터형의 데이터를 사용할 수 있게 합니다. 이것은 오라클 서버가 자동적으로 데이터형을 변환할 수 있을 때 허용됩니다. 이 데이터형 변환은 오라클 서버에 의해서 암시적으로 행해지거나 또는 사용자에 의해서 명시적으로 행해질 수 있습니다.

가) 암시적인 데이터형 변환

값 할당(assignment)시, 오라클 서버는 다음을 자동으로 변환할 수 있습니다.

FROM	TO
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

♣ 참고

오라클 서버가 값 할당(assignment) 문장에서 사용된 값의 데이터형을 목표(target) 값의 데이터형으로 변환할 수 있을 경우에 할당(assignment) 문장은 올바로 수행됩니다. 또한 CHAR 가 NUMBER 로의 변환은 문자열이 적절한 숫자로 나타낼 수 있을 경우에만 가능하고 CHAR 가 DATE 로의 변환은 문자열이 Default Date Type 와 같을 경우에만 성공합니다.

☞ Guidelines

비록 암시적 데이터형 변환을 이용할 수 있더라도, SQL 문장의 안정성을 위해서 명시적 데이터형 변환을 할 것을 권장합니다.

나) 명시적인 데이터형 변환

SQL은 변환 함수를 통하여 어떤 데이터형의 값을 다른 데이터형의 값으로 변환하기 위하여 아래의 함수를 제공합니다.

함 수	사 용 목 적
TO_CHAR	숫자나 문자값을 지정한 형식의 VARCHAR2 문자열로 변환 합니다.
TO_NUMBER	숫자를 포함하는 문자열을 숫자로 변환 합니다.
TO_DATE	날짜를 나타내는 문자열을 명시된 날짜로 변환 합니다.

1.6.2 TO_CHAR 함수

숫자, 날짜, 문자열을 지정한 형식의 VARCHAR2 문자열로 변환하는 함수입니다.

1) 날짜 형식을 변환

Syntax	TO_CHAR(date, 'fmt')
사용 예	TO_CHAR(hiredate, 'YY/MM/DD') → 81/11/17

가) 특정 형식으로 날짜를 출력

이전의 모든 날짜 형식은 DD-MON-YY 형식이었다. TO_CHAR 함수는 이러한 형식의 날짜를 명시한 날짜 형식으로 변환하여 출력할 수 있다.

☞ Guidelines

- 1) 포맷(fmt) 모델은 단일 인용 부호로 둘러싸여 있어야 하고 대소문자를 구분한다.
- 2) 포맷(fmt) 모델은 어떤 타당한 날짜 형식도 포함 가능하다.
- 3) 추가된 공백을 제거하거나 앞부분의 0을 없애기 위해서 ‘f##’요소를 사용한다.
- 4) SQL*Plus COLUMN 명령어로 문자 필드 결과의 출력 폭의 크기를 조절할 수 있다.(DEFAULT는 80)

나) 날짜 형식 모델

구 성 요 소	설 명
SCC or CC	세기;BC 날짜에는 _S를 붙입니다.
Years in dates YYYY or SYYYY	년;BC 날짜에는 _S를 붙입니다.
YY or YY or Y	년의 마지막 3,2 또는 1 자리 수
Y,YYY	콤마가 있는 년
IYYY, IYY, IY, I	ISO 표준에 바탕을 둔 4,3,2 또는 1 자리 수
SYESR or YEAR	문자고 표현된 년;BC 날짜에는 _S를 붙입니다.

BC or AD	BC/AD 지시자
B.C or A.D	.이 있는 BC/AD 지시자
Q	년의 4분의 1
MM	두자리 값의 월
MONTH	9자리를 위해 공백을 추가한 월 이름
MON	세 자리의 약어로 된 월 이름
RM	로마 숫자 월
WW or W	년이나 월의 주
DDD or DD or D	년, 월 또는 주의 일
DAY	9자리를 위해 공백을 추가한 요일 이름
DY	세 자리 약어로된 요일 이름
J	Julian day: BC4713년 12월 31일 이후의 요일 수

다) 시간 형식

- 1) 시간 요소는 날짜의 시간 부분을 형식화(HH24:MI:SS AM → 15:34:32 PM)
- 2) 문자열에 이중 인용 부호를 사용하여 문자열을 추가(DD “of” MONTH→10 of OCTOBER)
- 3) 숫자 접미사는 숫자를 문자로 변환(ddspth → fourteenth)
- 4) 시간 형식의 종류

요 소	설 명
AM or PM	정오 지시자
A.M or P.M	.이 있는 정오 지시자
HH or HH12 or HH24	하루 중 시간(1-12, 0-23)
MI	분(0-59)
SS	초(0-59)
SSSSS	자정 이후의 초(0-86399)

라) 기타 형식

요 소	설 명
/ . ,	사용 문자가 결과에 다시 나타난다.
“of the”	인용 부호내의 문자가 결과에 출력

마) 숫자에 영향을 주는 접미사

요 소	설 명
TH	서수(DDTH → 4TH)
SP	명시한 수(DDSP → FOUR)

SPTH or THSP	명시한 서수(DDSPTH → FOURTH)
--------------	-------------------------

문제 25) EMP 테이블에서 10 번 부서 종 입사 일자를 ‘1 May 1981’와 ‘1998 년 1 월 1 일’의 형태로 출력하여라

```
SQL> var t_hiredate varchar2(30)
SQL> var t_kor varchar2(20)
SQL> col t_hiredate format a30
SQL> col t_kor format a20
SQL> SELECT ename,hiredate,TO_CHAR(hiredate, 'fmDD Month YYYY') t_hiredate,
2 TO_CHAR(hiredate, 'YYYY"년" MM"월" DD"일"') t_kor
3 FROM emp
4 WHERE deptno = 10
5 ORDER BY hiredate DESC;
```

ENAME	HIREDATE	T_HIREDATE	T_KOR
-			
MILLER	23-JAN-82	23 January 1982	1982 년 01 월 23 일
KING	17-NOV-81	17 November 1981	1981 년 11 월 17 일
CLARK	09-JUN-81	9 June 1981	1981 년 06 월 09 일

2) 숫자 형식을 변환

TO_CHAR 함수를 사용하여 숫자 값을 문자로 출력하기 위해 사용한다.

Syntax	TO_CHAR(number , 'fmt')
사용 예	TO_CHAR(sal, '\$999,999') → \$3,000

가) 숫자를 가진 TO_CHAR 함수

- 1) 숫자 값을 문자로 변환할 때 즉 NUMBER 형을 VARCHAR2 로 전환할 때
- 2) 이 기법은 연결(Concatenation) 시에 유용

☞ Guidelines

- 1) 형식에 의해 제공되는 자릿수를 초과하는 숫자에 대해서는 “#”을 출력
- 2) 지정된 소수 값을 형식에서 제공하는 소수점 자리로 반올림 한다.

나) 숫자 형식 모델

요 소	설 명	예	결 과
9	9 의 수는 출력 폭을 결정	999999	1234
0	무효의 0 을 출력	099999	001234
\$	달러 기호	\$999999	\$1234
L	지역 화폐 기호	L999999	₩1234
.	명시한 위치에 소수점	999999.99	1234.00

,	명시한 위치에 콤마	999,999	1,234
MI	우측에 마이너스 기호(음수 값)	999999MI	1234-
PR	음수를 “()”로 묶는다	999999PR	<1234>
EEEE	과학적인 부호 표기	99.999EEEE	1.234E+03
V	10 을 n 번 곱합니다.	9999V99	123400
B	0 을 0 이 아닌 공백으로 출력	B9999.99	1234.00

문제 26) EMP 테이블에서 부서 20 중 급여 앞에 \$를 삽입하고 3 자리마다 ,를 출력하여라

```
SQL> SELECT empno,ename,job,sal,TO_CHAR(sal,'$999,999')
  2  FROM emp
  3 WHERE deptno = 20
  4 ORDER BY sal DESC;
```

EMPNO	ENAME	JOB	SAL	TO_CHAR(S)
7902	FORD	ANALYST	3000	\$3,000
7788	SCOTT	ANALYST	3000	\$3,000
7566	JONES	MANAGER	2975	\$2,975
7876	ADAMS	CLERK	1100	\$1,100
7369	SMITH	CLERK	800	\$800

1.6.3 TO_NUMBER 함수

숫자를 포함하는 문자열을 숫자로 변환 합니다.

Syntax	TO_NUMBER(char)
사용 예	TO_NUMBER('1234') → 1234

1.6.4 TO_DATE 함수

날짜를 나타내는 문자열을 명시된 날짜로 변환 합니다.

Syntax	TO_DATE(char [, 'fmt'])
사용 예	TO_DATE('19990220181030','YYYYMMDDHH24MISS') → 1999/02/20 18:10:30

문제 27) February 22, 1981 에 입사한 사원의 정보를 이름, 업무, 입사일자를 출력하여라.

```
SQL> SELECT ename,job,TO_CHAR(hiredate, 'Month DD, YYYY') t_hire
  2  FROM emp
  3 WHERE hiredate = TO_DATE('February 22, 1981','Month DD, YYYY');
```

ENAME	JOB	T_HIRE
WARD	SALESMAN	February 22, 1981

1.7 기타 함수

1.7.1 DECODE 함수

CASE 나 IF-THEN-ELSE-END IF 문장의 조건적 조회를 가능하게 함

Syntax	DECODE(col expr, search1, result1[, search2, result2,...][,default])
사용 예	DECODE(deptno, 10, sal*1.1, 20, sal*1.5, 30, sal*1.2, sal)

문제 28) EMP 테이블에서 JOB 이 ANALYST 이면 급여 증가는 10%이고 JOB 이 CLERK 이면 급여 증가는 15%이고 JOB 이 MANAGER 이면 급여 증가는 20%입니다. 다른 업무에 대해서는 급여 증가가 없습니다. 사원번호, 이름, 업무, 급여, 증가된 급여를 출력하여라.

```
SQL> SELECT empno,ename,job,sal,DECODE(job,'ANALYST', sal*1.1,
 2 'CLERK', sal*1.15,'MANAGER', sal*1.2, sal) d_sal
 3 FROM emp
 4 ORDER BY sal DESC;

EMPNO ENAME      JOB          SAL      D_SAL
----- -----
 7839 KING        PRESIDENT    5000     5000
 7902 FORD        ANALYST     3000     3300
.
.
.
14 rows selected.
```

1.8 중첩 함수

- 1) 단일행 함수는 여러 LEVEL 에 걸쳐 중첩 가능
- 2) 중첩 함수는 가장 하위 LEVEL 에서 상위 LEVEL 순으로 진행

Syntax	F3(F2(F1(col,arg1), arg2), arg3)
사용 예	NVL(TO_CHAR(mgr), 'No Manager')

문제 28) 다음의 결과를 분석하여 보아라.

```
SQL> col t_rpad format a20
SQL> col r_r format a20
SQL> SELECT deptno,dname,RPAD(dname,20,'*') t_rpad,
 2 RPAD(RTRIM(dname),20,'*') r_r,loc
 3 FROM dept;

DEPTNO DNAME      T_RPAD          R_R          LOC
----- -----
 10 ACCOUNTING  ACCOUNTING***** ACCOUNTING***** NEW YORK
 20 RESEARCH    RESEARCH***** RESEARCH***** DALLAS
 30 SALES       SALES***** SALES***** CHICAGO
 40 OPERATIONS  OPERATIONS***** OPERATIONS***** BOSTON
```


◆ 연습문제 ◆

1. 현재 날짜를 출력하고 열 레이블은 Current Date로 출력하는 SELECT 문장을 기술하시오.
2. EMP 테이블에서 현재 급여에 15%가 증가된 급여를 사원번호, 이름, 업무, 급여, 증가된 급여(New Salary), 증가액(Increase)를 출력하는 SELECT 문장을 기술하시오.
3. EMP 테이블에서 이름, 입사일, 입사일로부터 6 개월 후 돌아오는 월요일 구하여 출력하는 SELECT 문장을 기술하시오.
4. EMP 테이블에서 이름, 입사일, 입사일로부터 현재까지의 월수, 급여, 입사일부터 현재까지의 급여의 총계를 출력하는 SELECT 문장을 기술하시오.
5. EMP 테이블에서 다음의 결과가 출력되도록 작성하시오.

Dream Salary

```
KING earns $5,000.00 monthly but wants $15,000.00
BLAKE earns $2,850.00 monthly but wants $8,550.00
CLARK earns $2,450.00 monthly but wants $7,350.00
. . . . .
14 rows selected
```

6. EMP 테이블에서 모든 사원의 이름과 급여(15 자리로 출력 좌측의 빈곳은 “*”로 대치)를 출력하는 SELECT 문장을 기술하시오.
7. EMP 테이블에서 모든 사원의 정보를 이름, 업무, 입사일, 입사한 요일을 출력하는 SELECT 문장을 기술하시오.
8. EMP 테이블에서 이름의 길이가 6 자 이상인 사원의 정보를 이름, 이름의 글자수, 업무를 출력하는 SELECT 문장을 기술하시오.
9. EMP 테이블에서 모든 사원의 정보를 이름, 업무, 급여, 보너스, 급여+보너스를 출력하는 SELECT 문장을 기술하시오.

1. 그룹 함수

단일 행 함수와는 달리 그룹 함수는 여러 행 또는 테이블 전체에 대해 함수가 적용되어 하나의 결과를 가져오는 함수를 말한다. 그룹 당 하나의 결과가 주어지도록 행의 집합에 대해 연산할 경우 GROUP BY 절을 이용하여 그룹화 할 수 있고 HAVING 를 이용하여 그룹에 대한 조건을 제한하는 방법을 배우기로 한다.

1.1 그룹 함수의 종류

함수	설명
AVG(DISTINCT ALL n)	NULL 값을 제외한 n 개 행의 평균값
COUNT(DISTINCT ALL expr *)	NULL 이 아닌 행의 개수
MAX(DISTINCT ALL expr)	최대값
MIN(DISTINCT ALL expr)	최소값
STDDEV(DISTINCT ALL n)	NULL 값을 제외한 n 의 표준편차
SUM(DISTINCT ALL n)	NULL 값을 제외한 n 의 합계
VARIANCE(DISTINCT ALL n)	NULL 값을 제외한 n 의 분산

☞ Guidelines

- 1) DISTINCT 는 해당 함수로 하여금 오직 중복되지 않는 값만 RETURN 하게 해준다. 그러나 ALL(Default)은 해당 함수로 하여금 모든 값을 고려하게 한다.
- 2) Expr 이 있는 인수들의 자료 형태는 CHAR, VARCHAR2, NUMBER, DATE 형이 될 수 있다.
- 3) COUNT(*)를 제외한 모든 그룹 함수들은 NULL 값을 무시한다. NULL 값을 하나의 값으로 치환하기 위해서는 NVL 함수를 사용하라.
- 4) 모든 자료형에 대하여 MAX 와 MIN 를 사용할 수 있다. 그러나 AVG, SUM, VARIANCE, STDDEV 는 NUMBER 만 사용 가능하다.

1.2 그룹 함수 사용

1.2.1 Syntax

```
SELECT      group_function(column) [,group_function(column), . . .]
  FROM        table_name
  [WHERE       condition]
  [ORDER BY column];
```

문제 1) EMP 테이블에서 모든 SALESMAN 에 대하여 급여의 평균, 최고액, 최저액, 합계를 구하여 출력하여라.

```
SQL> SELECT AVG(sal), MAX(sal), MIN(sal), SUM(sal)
  2  FROM emp
  3 WHERE job LIKE 'SALE%';

AVG(SAL)  MAX(SAL)  MIN(SAL)  SUM(SAL)
-----  -----  -----  -----
1400      1600      1250      5600
```

문제 2) 아래의 SELECT 문장을 분석하여라.

```
SQL> SELECT MIN(ename),MAX(ename),MIN(hiredate),
  2          MAX(hiredate),MIN(sal),MAX(sal)
  3  FROM emp;

MIN(ENAME)  MAX(ENAME)  MIN(HIREDATE)  MAX(HIREDATE)  MIN(SAL)  MAX(SAL)
-----  -----  -----  -----  -----
ADAMS      WARD        17-DEC-80       12-JAN-83      800       5000
```

문제 3) EMP 테이블에 등록되어 있는 인원수, 보너스에 NULL 이 아닌 인원수, 보너스의 평균, 등록되어 있는 부서의 수를 구하여 출력하여라.

```
SQL> SELECT COUNT(*) c_inwon,COUNT(comm) c_comm,AVG(comm) a_comm,
  2          AVG(NVL(comm,0)) n_comm,COUNT(deptno) c_dept,
  3          COUNT(DISTINCT deptno) c_dis
  4  FROM emp;

C_INWON    C_COMM     A_COMM     N_COMM     C_DEPT     C_DIS
-----  -----  -----  -----  -----
14          4          550  157.14286      14          3
```

1.3 데이터 그룹 생성

지금까지 모든 그룹 함수는 테이블을 하나의 큰 그룹으로 다루었다. 여기서는 테이블의 자료를 보다 작은 그룹으로 나누어 사용할 수 있는 GROUP BY 절을 배워보자.

1.3.1 Syntax

```
SELECT      [column,] group_function(column) [,group_function(column),...]
  FROM      table_name
  [WHERE      condition]
  [GROUP BY group_by_expression]
  [HAVING      condition]
  [ORDER BY column];
```

table_name 테이블명 질의 대상 테이블 이름

WHERE condition 을 만족하는 행들만 검색

ORDER BY	질의 결과 정렬을 위한 옵션(ASC:오름차순(Default), DESC 내림차순)
group_function	Group Function이 SELECT 절 뒤에서 Column과 같이 기술되면 반드시 GROUP BY 절이 기술되어야 한다.
GROUP BY group_by_expression	행을 그룹하기 위한 기준이 될 Column을 기술
HAVING condition	그룹에 대한 조건을 기술

1.3.2 SELECT 절

SELECT 절에 GROUP 함수와 Column이 같이 기술되면 반드시 GROUP BY 절이 기술되어야 한다. 그러나 SELECT 절에 GROUP 함수만 기술되고 Column은 기술되지 않으면 반드시 GROUP BY 절을 기술할 필요는 없다.

```
SQL> SELECT deptno,COUNT(*),AVG(sal),MIN(sal),MAX(sal),SUM(sal)
   2 FROM emp;
SELECT deptno,COUNT(*),AVG(sal),MIN(sal),MAX(sal),SUM(sal)
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

문제 4) EMP 테이블에서 부서별로 인원수, 평균 급여, 최저급여, 최고 급여, 급여의 합을 구하여 출력하여라.

```
SQL> SELECT deptno,COUNT(*),AVG(sal),MIN(sal),MAX(sal),SUM(sal)
   2 FROM emp
   3 GROUP BY deptno;
```

DEPTNO	COUNT(*)	AVG(SAL)	MIN(SAL)	MAX(SAL)	SUM(SAL)
10	3	2916.6667	1300	5000	8750
20	5	2175	800	3000	10875
30	6	1566.6667	950	2850	9400

문제 5) 각 부서별로 인원수, 급여의 평균, 최저 급여, 최고 급여, 급여의 합을 구하여 급여의 합이 많은 순으로 출력하여라.

```
SQL> SELECT deptno,COUNT(*),AVG(sal),MIN(sal),MAX(sal),SUM(sal)
   2 FROM emp
   3 GROUP BY deptno
   4 ORDER BY SUM(sal) DESC;
```

DEPTNO	COUNT(*)	AVG(SAL)	MIN(SAL)	MAX(SAL)	SUM(SAL)
20	5	2175	800	3000	10875
30	6	1566.6667	950	2850	9400
10	3	2916.6667	1300	5000	8750

문제 6) 각 부서별로 인원수, 급여의 평균, 최저 급여, 최고 급여, 급여의 합을 구하여 급여의 합이 많은 순으로 출력하여라.

```
SQL> SELECT COUNT(*), AVG(sal), MIN(sal), MAX(sal), SUM(sal)
  2  FROM emp
  3 GROUP BY deptno
  4 ORDER BY SUM(sal) DESC;
```

COUNT(*)	AVG(SAL)	MIN(SAL)	MAX(SAL)	SUM(SAL)
5	2175	800	3000	10875
6	1566.6667	950	2850	9400
3	2916.6667	1300	5000	8750

♣ 참고

위 결과는 어느 행이 어떤 부서의 결과인지 알 수 없다. 즉 GROUP BY 절에 기술된 Column은 SELECT 절에 반드시 기술될 필요는 없다. 그러나 결과를 구분하기가 어렵다.

☞ Guidelines

- 1) SELECT 절에 GROUP Function이 포함된다면 GROUP BY 절에 각각의 열이 명시되어야 함
- 2) WHERE 절을 사용하여 행을 그룹으로 나누기 전에 행을 제외
- 3) 그룹에 대한 조건은 HAVING 절을 사용(그룹에 대한 조건을 WHERE 절에서 기술 불가)
- 4) GROUP BY 절에 열을 포함(열의 별칭은 사용할 수 없다)
- 5) Default 는 GROUP BY 절 다음에 기술된 순서로 오름차순으로 정렬되지만 ORDER BY 절을 이용하여 변경 가능

1.3.3 하나 이상의 Column으로 그룹화

때로는 그룹 내의 그룹에 대한 결과를 검색할 필요가 있다. 이러한 경우 대그룹, 중그룹, 소그룹으로 나누고자 하는 순서대로 GROUP BY 절 뒤에 기술하면 순서대로 오름차순으로 정렬된다.

가) Syntax

```
SELECT      [column, ] group_function(column) [,group_function(column),...]
  FROM       table_name
  [WHERE      condition]
  [GROUP BY group_by_expr1[,group_by_expr2, . . . .]]
  [HAVING    condition]
  [ORDER BY column];
```

GROUP BY group_by_expr1[,group_by_expr2,] 행을 그룹하기 위한 기준이 될 Column을 기술한다.

문제 7) 부서별, 업무별 그룹하여 결과를 부서번호, 업무, 인원수, 급여의 평균, 급여의 합을 구하여 출력하여라.

```
SQL> SELECT deptno,job,COUNT(*),AVG(sal),SUM(sal)
  2 FROM emp
  3 GROUP BY deptno,job;
```

DEPTNO	JOB	COUNT(*)	AVG(SAL)	SUM(SAL)
10	CLERK	1	1300	1300
10	MANAGER	1	2450	2450
10	PRESIDENT	1	5000	5000
.....				
9 rows selected.				

문제 8) 업무별, 부서별 그룹하여 결과를 부서번호, 업무, 인원수, 급여의 평균, 급여의 합을 구하여 출력하여라.

```
SQL> SELECT job,deptno,COUNT(*),AVG(sal),SUM(sal)
  2 FROM emp
  3 GROUP BY job,deptno;
```

JOB	DEPTNO	COUNT(*)	AVG(SAL)	SUM(SAL)
ANALYST	20	2	3000	6000
CLERK	10	1	1300	1300
CLERK	20	2	950	1900
CLERK	30	1	950	950
MANAGER	10	1	2450	2450
MANAGER	20	1	2975	2975
MANAGER	30	1	2850	2850
.....				
9 rows selected.				

1.3.4 WHERE 절

WHERE 절을 이용하여 조회하고자 하는 ROW 를 선별할 수 있다. 그러나 그룹에 대한 제한 조건은 WHERE 절에서 기술할 수 없다. 즉 GROUP FUNCTION 는 문법상 WHERE 절에 기술할 수 없다. 그러므로 그룹에 대한 제한 조건은 HAVING 절에서 기술한다.

```
SQL> SELECT deptno,COUNT(*),SUM(sal)
  2 FROM emp
  3 WHERE COUNT(*) > 4
  4 GROUP BY deptno;
WHERE COUNT(*) > 4
*
ERROR at line 3:
ORA-00934: group function is not allowed here
```

문제 9) EMP 테이블에서 부서 인원이 4 명보다 많은 부서의 부서번호, 인원수, 급여의 합을 구하여 출력하여라.

```
SQL> SELECT deptno,COUNT(*),SUM(sal)
  2  FROM emp
  3  GROUP BY deptno
  4  HAVING COUNT(*) > 4;
```

DEPTNO	COUNT(*)	SUM(SAL)
20	5	10875
30	6	9400

1.3.5 HAVING 절

- 1) WHERE 절에 GROUP Function을 사용할 수 없다.
- 2) HAVING 절을 사용하여 출력될 그룹을 명시
- 3) 오라클 서버는 HAVING 절을 사용할 때 다음의 단계를 수행
 - ① 행을 그룹화
 - ② 그룹 함수를 적용
 - ③ HAVING 절과 일치하는 그룹을 출력

문제 10) EMP 테이블에서 급여가 최대 2900 이상인 부서에 대해서 부서번호, 평균 급여, 급여의 합을 구하여 출력하여라.

```
SQL> SELECT deptno, AVG(sal),SUM(sal)
  2  FROM emp
  3  GROUP BY deptno
  4  HAVING MAX(sal) > 2900;
```

DEPTNO	AVG(SAL)	SUM(SAL)
10	2916.6667	8750
20	2175	10875

문제 11) EMP 테이블에서 업무별 급여의 평균이 3000 이상인 업무에 대해서 업무명, 평균 급여, 급여의 합을 구하여 출력하여라.

```
SQL> SELECT job, AVG(sal),SUM(sal)
  2  FROM emp
  3  GROUP BY job
  4  HAVING AVG(sal) >= 3000;
```

JOB	AVG(SAL)	SUM(SAL)
ANALYST	3000	6000
PRESIDENT	5000	5000

문제 12) EMP 테이블에서 전체 월급이 5000 을 초과하는 각 업무에 대해서 업무와 월급여 합계를 출력하여라. 단 판매원은 제외하고 월 급여 합계로 정렬(내림차순)하여라.

```
SQL> SELECT job,SUM(sal) PAYROLL  
2 FROM emp  
3 WHERE job NOT LIKE 'SALE%'  
4 GROUP BY job  
5 HAVING SUM(sal) > 5000  
6 ORDER BY SUM(sal) DESC;
```

JOB	PAYROLL
MANAGER	8275
ANALYST	6000

☞ Guidelines

- 1) HAVING 절은 GROUP BY 절 앞에 기술 가능하지만 GROUP BY 절 다음에 기술하는 것이 논리적이므로 원장 됩니다. HAVING 절이 SELECT 절에 있는 그룹에 적용되기 전에 그룹은 구성되고 그룹 함수는 구성됩니다.
- 2) SELECT 절에 그룹 함수를 사용하지 않고 GROUP BY 절만 사용 가능하다. 그룹 함수의 결과로 행이 제한 된다면 HAVING 절 뿐만 아니라 GROUP BY 절을 사용해야 된다.

1.3.6 중첩 그룹 함수

그룹 함수는 어떤 LEVEL 까지도 중첩할 수 있습니다.

문제 13) 부서별 평균 중 최대 평균 급여, 부서별 급여의 합 중 최대 급여, 부서별 급여의 최소 급여, 부서별 급여의 최대 급여를 출력하여라.

```
SQL> SELECT MAX(AVG(sal)),MAX(SUM(sal)),MIN(MIN(sal)),MAX(MAX(sal))  
2 FROM emp  
3 GROUP BY deptno;
```

MAX(AVG(SAL))	MAX(SUM(SAL))	MIN(MIN(SAL))	MAX(MAX(SAL))
2916.6667	10875	800	5000

◆ 연습문제 ◆

1. EMP 테이블에서 인원수, 최대 급여, 최소 급여, 급여의 합을 계산하여 출력하는 SELECT 문장을 작성하여라.
2. EMP 테이블에서 각 업무별로 최대 급여, 최소 급여, 급여의 합을 출력하는 SELECT 문장을 작성하여라.
3. EMP 테이블에서 업무별 인원수를 구하여 출력하는 SELECT 문장을 작성하여라.
4. EMP 테이블에서 최고 급여와 최소 급여의 차이는 얼마인가 출력하는 SELECT 문장을 작성하여라.
5. EMP 테이블에서 아래의 결과를 출력하는 SELECT 문장을 작성하여라.

H_YEAR	COUNT(*)	MIN(SAL)	MAX(SAL)	AVG(SAL)	SUM(SAL)
80	1	800	800	800	800
81	10	950	5000	2282.5	22825
82	2	1300	3000	2150	4300
83	1	1100	1100	1100	1100

6. EMP 테이블에서 아래의 결과를 출력하는 SELECT 문장을 작성하여라.

TOTAL	1980	1981	1982	1983
14	1	10	2	1

7. EMP 테이블에서 아래의 결과를 출력하는 SELECT 문장을 작성하여라.

JOB	Deptno 10	Deptno 20	Deptno 30	Total
ANALYST		6000		6000
CLERK	1300	1900	950	4150
MANAGER	2450	2975	2850	8275
PRESIDENT	5000			5000
SALESMAN		5600	5600	

1. Join

하나 이상의 테이블로부터 자료를 검색하기 위하여 조인을 사용합니다. 일반적으로 Primary Key(이후 PK로 사용)와 Foreign Key(이후 FK로 사용)을 사용하여 Join하는 경우가 대부분이지만 때로는 논리적인 값들의 연관으로 Join하는 경우도 있습니다.

1.1 Syntax

```
SELECT      table1.column1 [,table2.column2, . . . .]
  FROM      table1, table2
 WHERE      table1.column1 = table2.column2;
```

☞ Guidelines

- 1) WHERE 절에 조인 조건을 기술한다.
- 2) 테이블을 조인하는 SELECT 문장을 작성할 경우 명확성을 위하여 또는 데이터베이스의 Performance 향상을 위하여 열 이름 앞에 테이블 명을 붙인다.
- 3) 똑 같은 열 이름이 존재하는 테이블이 있을 경우는 반드시 열 이름 앞에 테이블 명을 붙인다.
- 4) n개의 테이블을 조인 하려면 최소한 n-1 번의 조인 조건 문이 필요하다.

1.2 Join의 종류

Join 방법	설명
Cartesian Product	모든 가능한 행들의 Join
Equijoin	Join 조건이 정확히 일치하는 경우 사용(일반적으로 PK와 FK 사용)
Non-Equijoin	Join 조건이 정확히 일치하지 않는 경우에 사용(등급, 학점)
Outer Join	Join 조건이 정확히 일치하지 않는 경우에도 모든 행들을 출력
Self Join	하나의 테이블에서 행들을 Join하고자 할 경우에 사용
Set Operators	여러 개의 SELECT 문장을 연결하여 작성한다.

1.3 Cartesian Product

모든 가능한 행들의 Join으로 다음과 같은 경우에 발생한다.

- 1) 조인 조건이 생략된 경우
- 2) 조인 조건이 잘못된 경우
- 3) 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행과 두번째 테이블의 모든 행이 첫번째 테이블의 모든 행과 조인되는 경우.
- 4) 양쪽 ROW의 개수를 곱한 결과

♣ 참고

Cartesian Product 는 같은 수의 행을 생성하는 경향이 있고 결과도 거의 유용하지 못하다. 그러므로 모든 테이블로부터 모든 행을 조합할 필요가 없을 경우 WHERE 절에 조인 조건을 명확히 기술하여야 한다.

문제 1) EMP 테이블과 DEPT 테이블을 Cartesian Product 하여 사원번호, 이름, 업무, 부서번호, 부서명, 근무지를 출력하여라.

```
SQL> SELECT empno,ename,job,dept.deptno,dname,loc  
2  FROM dept,emp  
3  ORDER BY empno;
```

EMPNO	ENAME	JOB	DEPTNO	DNAME	LOC
7369	SMITH	CLERK	10	ACCOUNTING	NEW YORK
7369	SMITH	CLERK	20	RESEARCH	DALLAS
7369	SMITH	CLERK	30	SALES	CHICAGO
7369	SMITH	CLERK	40	OPERATIONS	BOSTON
7499	ALLEN	SALESMAN	10	ACCOUNTING	NEW YORK
7499	ALLEN	SALESMAN	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	30	SALES	CHICAGO
7499	ALLEN	SALESMAN	40	OPERATIONS	BOSTON
7521	WARD	SALESMAN	10	ACCOUNTING	NEW YORK
7521	WARD	SALESMAN	20	RESEARCH	DALLAS
7521	WARD	SALESMAN	30	SALES	CHICAGO
7521	WARD	SALESMAN	40	OPERATIONS	BOSTON
7566	JONES	MANAGER	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	20	RESEARCH	DALLAS
7566	JONES	MANAGER	30	SALES	CHICAGO
.					
56 rows selected.					

1.4 Equijoin

Equijoin 이란 조인 조건에서 “=”을 사용하여 값들이 정확하게 일치하는 경우에 사용하는 조인을 말합니다. 대부분 PK 와 FK 의 관계를 이용하여 조인 합니다. Equijoin 은 다른 말로 단순 조인 또는 내부 조인 이라고도 합니다.

1.4.1) Syntax

```
SELECT      table1.column1 [,table2.column2, . . . . .]  
  FROM       table1, table2  
  WHERE      table1.column1 = table2.column2;
```

table1.column1

조회할 자료가 있는 테이블과 열 이름을 기술

table1.column1=table2.column2

두 테이블들간에 논리적으로 연결하는 조인 조건 기술

1.4.2) Equijoin으로 자료 검색

- 1) SELECT 절은 검색할 열 이름을 명시
- 2) FROM 절은 데이터베이스가 Access 해야 하는 두개의 테이블을 명시
- 3) WHERE 절은 테이블의 조인 조건을 명시
- 4) 양쪽 테이블에 공통으로 존재하는 열 이름은 모호함을 피하기 위하여 열 이름 앞에 테이블명을 기술함

1.4.3) Equijoin 의 방법

종업원의 부서 이름을 결정하기 위해 EMP Table 의 DEPTNO 와 DEPT Table 의 DEPTNO 와 값을 비교하여야 합니다. EMP Table 과 DEPT Table 사이의 관계는 양쪽 테이블의 DEPTNO 열이 같아야 합니다. 이들이 PK 와 FK 로 연결되어 있습니다.

SQL> SELECT empno,ename,job,deptno 2 FROM emp; 2 FROM dept;			SQL> SELECT deptno,dname,loc DEPTNO DNAME LOC		
EMPNO	ENAME	JOB	DEPTNO	DNAME	LOC
7839	KING	PRESIDENT	10	10 ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	30	20 RESEARCH	DALLAS
7782	CLARK	MANAGER	10	30 SALES	CHICAGO
7566	JONES	MANAGER	20	40 OPERATIONS	BOSTON
7654	MARTIN	SALESMAN	30		
7499	ALLEN	SALESMAN	30		
.....					
14 rows selected.					

문제 2) EMP 테이블에서 사원번호, 이름, 업무, EMP 테이블의 부서번호, DEPT 테이블의 부서번호, 부서명, 근무지를 출력하여라.

SQL> SELECT empno,ename,job,emp.deptno,dept.deptno,dname,loc 2 FROM dept,emp 3 WHERE dept.deptno = emp.deptno 4 ORDER BY dept.deptno;					
EMPNO	ENAME	JOB	DEPTNO	DEPTNO	DNAME
7839	KING	PRESIDENT	10	10	ACCOUNTING
7782	CLARK	MANAGER	10	10	ACCOUNTING
7934	MILLER	CLERK	10	10	ACCOUNTING
7566	JONES	MANAGER	20	20	RESEARCH
7788	SCOTT	ANALYST	20	20	RESEARCH
.....					
14 rows selected.					

1.4.4) Table에 Alias 사용

- 1) 테이블 별칭을 사용하여 긴 테이블 명을 간단하게 사용한다.
- 2) 테이블 이름 대신에 Alias를 사용한다.
- 3) SQL 코드를 적게 사용하여 코딩 시간이 절약되고 메모리를 보다 적게 사용한다.

```
SQL> SELECT e.empno,e.ename,e.job,e.deptno,
2 d.deptno,d.dname,d.loc
3 FROM dept d,emp e
4 WHERE d.deptno = e.deptno
5 ORDER BY d.deptno;
```

EMPNO	ENAME	JOB	DEPTNO	DEPTNO	DNAME	LOC
7839	KING	PRESIDENT	10	10	ACCOUNTING	NEW YORK
7782	CLARK	MANAGER	10	10	ACCOUNTING	NEW YORK
7934	MILLER	CLERK	10	10	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	20	20	RESEARCH	DALLAS
7788	SCOTT	ANALYST	20	20	RESEARCH	DALLAS
7876	ADAMS	CLERK	20	20	RESEARCH	DALLAS
.						
14 rows selected.						

☞ Guidelines

- 1) 테이블 Alias는 30자까지 사용 가능하지만 짧을수록 더 좋다.
- 2) FROM 절에서 Alias가 사용되면 SELECT문 전체에서 사용 가능하다.
- 3) 테이블의 Alias에 가급적 의미를 부여
- 4) 테이블은 현재 SELECT문장에서만 유용

1.4.5) AND 연산자를 사용하여 추가적인 검색 조건

조인 이외의 WHERE 절에 추가적인 조건을 가질 수 있다.

문제 3) SALESMAN 사원만 사원번호, 이름, 급여, 부서명, 근무지를 출력하여라

```
SQL> SELECT e.empno,e.ename,e.sal ,d.dname,d.loc
2 FROM dept d,emp e
3 WHERE d.deptno = e.deptno AND e.job = 'SALESMAN';
```

EMPNO	ENAME	SAL	DNAME	LOC
7654	MARTIN	1250	SALES	CHICAGO
7499	ALLEN	1600	SALES	CHICAGO
7844	TURNER	1500	SALES	CHICAGO
7521	WARD	1250	SALES	CHICAGO

1.4.6) 두개 이상의 테이블 조인

때로는 두개 이상의 테이블을 조인 할 경우가 있다.

문제 4) 고객의 TKB SPORT SHOP 의 이름, 주문처, 항목수, 각 항목의 합계, 각 주문의 합계를 출력하여라.

SQL> SELECT name,custid 2 FROM customer;	SQL> SELECT ordid, itemid 2 FROM item;	SQL> SELECT custid,ordid 2 FROM ord;																																		
<table border="1"> <thead> <tr> <th>NAME</th><th>CUSTID</th></tr> </thead> <tbody> <tr><td>JOCKSPORTS</td><td>100</td></tr> <tr><td>TKB SPORT SHOP</td><td>101</td></tr> <tr><td>VOLLYRITE</td><td>102</td></tr> <tr><td>.....</td><td>.....</td></tr> <tr><td>9 rows selected.</td><td>64 rows selected.</td></tr> </tbody> </table>	NAME	CUSTID	JOCKSPORTS	100	TKB SPORT SHOP	101	VOLLYRITE	102	9 rows selected.	64 rows selected.	<table border="1"> <thead> <tr> <th>ORDID</th><th>ITEMID</th></tr> </thead> <tbody> <tr><td>610</td><td>3</td></tr> <tr><td>611</td><td>1</td></tr> <tr><td>612</td><td>1</td></tr> <tr><td>.....</td><td>.....</td></tr> <tr><td>21 rows selected.</td><td>21 rows selected.</td></tr> </tbody> </table>	ORDID	ITEMID	610	3	611	1	612	1	21 rows selected.	21 rows selected.	<table border="1"> <thead> <tr> <th>CUSTID</th><th>ORDID</th></tr> </thead> <tbody> <tr><td>101</td><td>610</td></tr> <tr><td>102</td><td>611</td></tr> <tr><td>104</td><td>612</td></tr> <tr><td>.....</td><td>.....</td></tr> </tbody> </table>	CUSTID	ORDID	101	610	102	611	104	612
NAME	CUSTID																																			
JOCKSPORTS	100																																			
TKB SPORT SHOP	101																																			
VOLLYRITE	102																																			
.....																																			
9 rows selected.	64 rows selected.																																			
ORDID	ITEMID																																			
610	3																																			
611	1																																			
612	1																																			
.....																																			
21 rows selected.	21 rows selected.																																			
CUSTID	ORDID																																			
101	610																																			
102	611																																			
104	612																																			
.....																																			
SQL> SELECT c.name, o.ordid, i.itemid, i.itemtot, o.total 2 FROM customer c, ord o, item i 3 WHERE c.custid = o.custid AND o.ordid = i.ordid 4 AND c.name = 'TKB SPORT SHOP';																																				
	<table border="1"> <thead> <tr> <th>NAME</th><th>ORDID</th><th>ITEMID</th><th>ITEMTOT</th><th>TOTAL</th></tr> </thead> <tbody> <tr><td>TKB SPORT SHOP</td><td>610</td><td>3</td><td>58</td><td>101.4</td></tr> <tr><td>TKB SPORT SHOP</td><td>610</td><td>1</td><td>35</td><td>101.4</td></tr> <tr><td>TKB SPORT SHOP</td><td>610</td><td>2</td><td>8.4</td><td>101.4</td></tr> </tbody> </table>	NAME	ORDID	ITEMID	ITEMTOT	TOTAL	TKB SPORT SHOP	610	3	58	101.4	TKB SPORT SHOP	610	1	35	101.4	TKB SPORT SHOP	610	2	8.4	101.4															
NAME	ORDID	ITEMID	ITEMTOT	TOTAL																																
TKB SPORT SHOP	610	3	58	101.4																																
TKB SPORT SHOP	610	1	35	101.4																																
TKB SPORT SHOP	610	2	8.4	101.4																																

1.5 Non-Equijoin

EMP 와 SALGRADE 사이의 관련성은 EMP 테이블의 어떠한 column 도 직접적으로 SALGRADE 테이블의 한 column에 상응하지 않기 때문에 Non-Equijoin이다. 두 테이블 사이의 관련성은 EMP 테이블의 SAL 열이 SALGRADE 테이블의 LOSAL 과 HISAL 열 사이에 있다는 것이다. 조인 조건은 등등(=) 이외의 연산자(BETWEEN ~ AND ~)를 갖는다.

SQL> SELECT empno,ename,sal 2 FROM emp;	SQL> SELECT grade,losal,hisal 2 FROM salgrade;																																										
<table border="1"> <thead> <tr> <th>EMPNO</th><th>ENAME</th><th>SAL</th></tr> </thead> <tbody> <tr><td>7839</td><td>KING</td><td>5000</td></tr> <tr><td>7698</td><td>BLAKE</td><td>2850</td></tr> <tr><td>7782</td><td>CLARK</td><td>2450</td></tr> <tr><td>7566</td><td>JONES</td><td>2975</td></tr> <tr><td>7654</td><td>MARTIN</td><td>1250</td></tr> <tr><td>.....</td><td>.....</td><td>.....</td></tr> <tr><td>14 rows selected.</td><td></td><td></td></tr> </tbody> </table>	EMPNO	ENAME	SAL	7839	KING	5000	7698	BLAKE	2850	7782	CLARK	2450	7566	JONES	2975	7654	MARTIN	1250	14 rows selected.			<table border="1"> <thead> <tr> <th>GRADE</th><th>LOSAL</th><th>HISAL</th></tr> </thead> <tbody> <tr><td>1</td><td>700</td><td>1200</td></tr> <tr><td>2</td><td>1201</td><td>1400</td></tr> <tr><td>3</td><td>1401</td><td>2000</td></tr> <tr><td>4</td><td>2001</td><td>3000</td></tr> <tr><td>5</td><td>3001</td><td>9999</td></tr> </tbody> </table>	GRADE	LOSAL	HISAL	1	700	1200	2	1201	1400	3	1401	2000	4	2001	3000	5	3001	9999
EMPNO	ENAME	SAL																																									
7839	KING	5000																																									
7698	BLAKE	2850																																									
7782	CLARK	2450																																									
7566	JONES	2975																																									
7654	MARTIN	1250																																									
.....																																									
14 rows selected.																																											
GRADE	LOSAL	HISAL																																									
1	700	1200																																									
2	1201	1400																																									
3	1401	2000																																									
4	2001	3000																																									
5	3001	9999																																									

문제 5) EMP 테이블에서 사원번호, 이름, 업무, 급여, 급여의 등급, 하한값, 상한값을 출력하여라.

```
SQL> SELECT e.empno,e.ename,e.job,e.sal,s.grade,s.losal,s.hisal
2 FROM salgrade s,emp e
3 WHERE e.sal BETWEEN s.losal AND s.hisal AND e.deptno = 10;
```

EMPNO	ENAME	JOB	SAL	GRADE	LOSAL	HISAL
7839	KING	PRESIDENT	5000	5	3001	9999
7782	CLARK	MANAGER	2450	4	2001	3000
7934	MILLER	CLERK	1300	2	1201	1400

♣ 참고

위 질의가 실행될 때 한번만 조인되는 것을 알 수 있다. 이에 대한 두 가지 이유가 있다.

- 1) SALGRADE 테이블에서 중복되는 등급을 포함하는 행이 없다.
- 2) EMP 테이블에 있는 SAL의 값은 SALGRADE 테이블에서 제공하는 값 범위에 있다.

☞ Guidelines

<= 및 >= 같은 다른 연산자를 사용 가능하나 BETWEEN 이 가장 단순하다. 또한 테이블에 Alias 를 사용하였는데 이는 모호성 때문이 아니라 성능 때문에 사용하였다. BETWEEN 사용 시 하한값을 먼저 명시하고 상한값을 나중에 명시한다는 것을 명심하라.

1.6 Outer Join

행이 조인 조건을 만족하지 않으면, 행은 질의 결과에 나타나지 않을 것입니다. 예를 들어 EMP 와 DEPT 테이블의 equijoin 조건에서 부서 OPERATIONS(40 번 부서)는 해당 부서에 아무도 없기 때문에 나타나지 않습니다. 이런 경우 모든 행을 전부 출력하고자 할 경우 Outer Join 을 사용한다. 즉 정상적으로 조인 조건을 만족하지 못하는 행들을 보기 위해 Outer join 을 사용한다.

```
SQL> SELECT empno,ename,job,deptno
2 FROM emp;    2 FROM dept;
```

EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7566	JONES	MANAGER	20
7654	MARTIN	SALESMAN	30
.	.	.	.
14 rows selected.			

```
SQL> SELECT deptno,dname,loc
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

40 번 부서는 조인시
조인되지 않는다.

☞ Guidelines

- 1) 행인 조건을 만족하지 않을 시 해당 행은 질의 결과에 나타나지 않는다.
- 2) Outer join 연산자를 조인 조건에 사용시 조인 조건을 만족하지 않는 행들도 결과에 나타날 수 있다.
- 3) 연산자는 괄호로 묶인 플러스 기호(+)이며 조인 시킬 값이 없는 조인 쪽에 "(+)"를 위치 시킨다.
- 4) (+)연산자는 한 개 이상의 NULL 행을 생성하고 정보가 충분한 테이블의 한 개 이상의 행들이 이런 NULL 행에 조인된다.
- 5) Outer join 연산자는 표현식의 한 편에만 올 수 있다.
- 6) Outer join 을 포함하는 조건은 IN 연산자, OR 연산자를 사용하여 다른 하나의 조건에 연결될 수 없다.

1.6.1) Syntax

```
SELECT      table1.column1 [,table2.column2, . . . . .]
  FROM      table1, table2
 WHERE      table1.column1 = table2.column2(+);
```



```
SELECT      table1.column1 [,table2.column2, . . . . .]
  FROM      table1, table2
 WHERE      table1.column1(+) = table2.column2;
```

table1.column 테이블을 함께 조인(또는 관련)시키는 조건입니다.
table2.column(+) (+)는 outer join 기호입니다. WHERE 절 조건의 양쪽이 아니라
 어느 한쪽에 둘 수 있습니다. 즉 양측 모두에는 올 수 없습니다.
 일치하는 행이 없는 테이블의 열 이름 뒤에 outer join 연산자
 를 사용합니다.

1.6.2) Outer Join 제약 사항

- 1) Outer Join 연산자는 정보가 부재하는 쪽의 표현식 한 쪽에만 둡니다. 다른 테이블의 어떠한 열과도 직접적으로 일치하는 것이 없는 한 테이블의 행을 리턴합니다.
- 2) Outer Join 을 포함하는 조건은 IN 연산자를 사용할 수 없고, OR 연산자에 의해 다른 조건과 연결될 수 없습니다.

```
SQL> select * from emp,dept
  2 where dept.deptno(+) = emp.deptno(+);
where dept.deptno(+) = emp.deptno(+)
      *
ERROR at line 2:
ORA-01468: a predicate may reference only one outer-joined table
```



```
SQL>
```

문제 6) EMP 테이블과 DEPT 테이블에서 DEPT 테이블에 있는 모든 자료를 사원번호, 이름, 업무,EMP 테이블의 부서번호,DEPT 테이블의 부서번호,부서명,근무지를 출력하여라

```
SQL> SELECT e.empno,e.ename,e.job,e.deptno,
2 d.deptno,d.dname,d.loc
3 FROM dept d,emp e
4 WHERE d.deptno = e.deptno(+);
```

EMPNO	ENAME	JOB	DEPTNO	DEPTNO	DNAME	LOC
7900	JAMES	CLERK	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	30	30	SALES	CHICAGO
				40	OPERATIONS	BOSTON

15 rows selected.

1.7 Self Join

때때로 자체적으로 테이블을 조인할 필요가 있습니다. 각 종업원의 관리자 명을 알기 위해서 자체적으로 EMP 테이블을 조인하는 것이 필요합니다.

- 1) ENAME 열을 검사하여 EMP 테이블에서 Blake 를 검색한다.
- 2) MGR 열을 검사하여 Blake 에 대한 관리자 번호를 검색한다.(Blake 관리자 번호:7839)
- 3) ENAME 열을 검사하여 EMPNO 가 7839 인 관리자를 검색한다.7839 는 King 이므로 Blake 의 관리자는 King 이다.

```
SQL> SELECT empno,ename,mgr
2 FROM emp;
```

EMPNO	ENAME	MGR
7839	KING	
7698	BLAKE	7839
7782	CLARK	7839
7566	JONES	7839
7654	MARTIN	7698
7499	ALLEN	7698
7844	TURNER	7698
7900	JAMES	7698
7521	WARD	7698
7902	FORD	7566
7369	SMITH	7902
7788	SCOTT	7566
7876	ADAMS	7788
7934	MILLER	7782

14 rows selected.

```
SQL> SELECT empno,ename,mgr
2 FROM emp;
```

EMPNO	ENAME	MGR
7839	KING	
7698	BLAKE	7839
7782	CLARK	7839
7566	JONES	7839
7654	MARTIN	7698
7499	ALLEN	7698
7844	TURNER	7698
7900	JAMES	7698
7521	WARD	7698
7902	FORD	7566
7369	SMITH	7902
7788	SCOTT	7566
7876	ADAMS	7788
7934	MILLER	7782

14 rows selected.

☞ Guidelines

- 1) Self join 을 사용하여 한 테이블의 행들을 같은 테이블의 행들과 조인한다.
- 2) 같은 테이블에 대해 두 개의 alias 를 작성(테이블 구분)함으로 FROM 절에 두개의 테이블을 사용하는 것과 같이 한다.
- 3) Column 에 대해서도 어떤 테이블에서 왔는지 반드시 Alias 명을 기술하여야 한다.
- 4) 테이블 하나를 두개 또는 그 이상으로 Self join 할 수 있다.

문제 7) EMP 테이블에서 Self join 하여 관리자를 출력하여라.

```
SQL> SELECT worker.ename || '의 관리자는 ' || manager.ename || '이다'  
2  FROM emp worker, emp manager  
3 WHERE worker.mgr = manager.empno;  
  
WORKER.ENAME||'의관리자는'||MANAGER.  
-----  
BLAKE 의 관리자는 KING 이다  
CLARK 의 관리자는 KING 이다  
.....  
13 rows selected.
```

1.8 Set Operators

하나 이상의 테이블로부터 자료를 검색하는 또 다른 방법은 SET 연산자를 이용하는 방법이 있다. 즉 SET 연산자를 이용하여 여러 개의 SELECT 문장을 연결하여 작성할 수 있다.

1.8.1) Syntax

```
SELECT      * | column1[, column2, column3, . . . ]  
FROM       table1  
.  
.  
.  
SET        operator  
SELECT      * | column1[, column2, column3, . . . ]  
FROM       table2  
.  
.  
.  
[ORDER BY    column | expression];
```

☞ Guidelines

- 1) 첫번째 SELECT 구문에서 기술된 열과 두번째 SELECT 구문에서 기술된 열들은 좌측부터 1 대 1 대응하여 그 개수와 타입이 일치해야 한다.
- 2) FROM 절 뒤에 기술되는 테이블은 같을 수도 있고 다를 수도 있다.
- 3) 출력되는 HARDING 을 첫번째 SELECT 구문에서 기술된 열이 출력된다.
- 4) ORDER BY 는 단 한번만 기술 가능하고 SELECT 구문의 마지막에 기술한다.
- 5) SELECT 문장은 위에서 아래로 수행되고 이를 변경하고자 할 경우는 괄호를 사용한다.

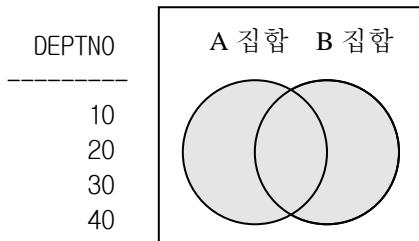
1.8.2) SET 연산자의 종류

종 류	설 명
UNION	각 결과의 합(합집합: 중복되는 값은 한번 출력)
UNION ALL	각 결과의 합(합집합)
INTERSECT	각 결과의 중복되는 부분만 출력(교집합)
MINUS	첫번째 결과에서 두번째 결과를 뺀(차집합)

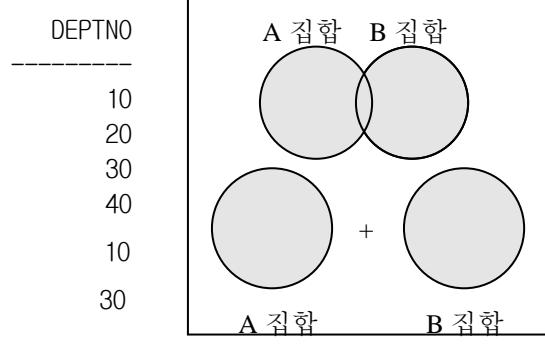
1.8.3) UNION과 UNION ALL의 차이

양쪽에서 검색된 결과를 모두 출력한다. 아래의 두개의 SELECT 문장을 참조하여라.

```
SQL> SELECT deptno
  2  FROM dept
  3 UNION
  4 SELECT deptno
  5  FROM emp;
```



```
SQL> SELECT deptno
  2  FROM dept
  3 UNION ALL
  4 SELECT deptno
  5  FROM emp;
```



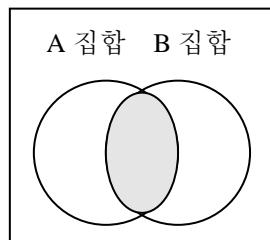
18 rows selected.

1.8.4) INTERSECT 연산자

양쪽에서 검색된 자료만 출력한다. 아래의 SELECT 문장을 참조하여라.

```
SQL> SELECT deptno
  2  FROM dept
  3 INTERSECT
  4 SELECT deptno
  5  FROM emp;
```

```
DEPTNO
-----
10
20
30
```

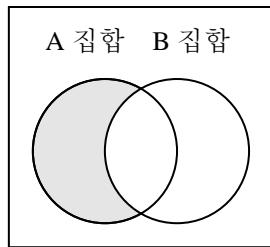


1.8.5) MINUS 연산자

두번째 SELECT 문장에서 검색되지 않았던 값을 첫번째 SELECT 문장에서 출력한다. 즉 첫 번째 SELECT 문장에서 두번째 SELECT 문장에의 값을 뺀것을 출력한다. 아래의 SELECT 문장을 참조하여라.

```
SQL> SELECT deptno  
2   FROM dept  
3   MINUS  
4   SELECT deptno  
5   FROM emp;
```

```
DEPTNO  
-----  
40
```



◆ 연습문제 ◆

1. EMP 테이블에서 모든 사원에 대한 이름, 부서번호, 부서명을 출력하는 SELECT 문장을 작성하여라.
2. EMP 테이블에서 NEW YORK 에서 근무하고 있는 사원에 대하여 이름, 업무, 급여, 부서명을 출력하는 SELECT 문장을 작성하여라.
3. EMP 테이블에서 보너스를 받는 사원에 대하여 이름, 부서명, 위치를 출력하는 SELECT 문장을 작성하여라.
4. EMP 테이블에서 이름 중 L 자가 있는 사원에 대하여 이름, 업무, 부서명, 위치를 출력하는 SELECT 문장을 작성하여라.
5. 아래의 결과를 출력하는 SELECT 문장을 작성하여라.(관리자가 없는 King 을 포함하여 모든 사원을 출력)

Employee	Emp#	Manager	Mgr#
KING	7839		
BLAKE	7698	KING	7839
CLARK	7782	KING	7839
.			
14 rows selected.			

6. EMP 테이블에서 그들의 관리자 보다 먼저 입사한 사원에 대하여 이름, 입사일, 관리자 이름, 관리자 입사일을 출력하는 SELECT 문장을 작성하여라.
7. EMP 테이블에서 사원의 급여와 사원의 급여 양만큼 “*”를 출력하는 SELECT 문장을 작성하여라. 단 “*”는 100 을 의미한다.

Employee and their salary	
KING	*****
BLAKE	*****
CLARK	*****
JONES	*****
MARTIN	*****
ALLEN	*****
TURNER	*****
.	
14 rows selected.	

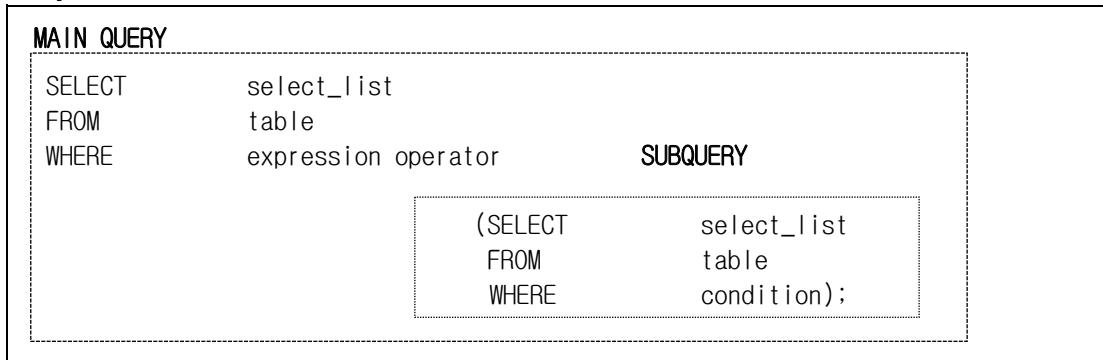
1. SUBQUERY

알려지지 않은 기준에 의한 데이터 검색을 위한 NESTED SUBQUERY 와 데이터 조작 문장에 SUBQUERY 를 사용하는 방법을 배우고 SUBQUERY 에 의해 검색된 데이터 정렬에 대해 다루기로 한다.

1.1 SUBQUERY 의 개념

다른 SELECT 문장의 절에 내장된 SELECT 문장입니다. SUBQUERY 는 여러 절에서 사용 가능하며 SELECT 문장 안에 기술된 SELECT 문장이다. NESTED SUBQUERY 는 MAIN QUERY 이전에 한번만 수행되며 SUBQUERY 의 결과를 MAIN QUERY 에 의해 조건으로 사용된다. SUBQUERY 를 사용하여 간단한 문장을 강력한 문장으로 만들 수 있고 테이블 자체의 데이터에 의존하는 조건으로 테이블의 행을 검색할 필요가 있을 때 아주 유용하다.

1.2 Syntax



- 1) SUBQUERY 는 다른 하나의 SQL 문장의 절에 NESTED 된 SELECT 문장이다.
- 2) 알려지지 않은 조건에 근거한 값을 검색하는 SELECT 문장을 작성하는데 유용하다.
- 3) SUBQUERY 는 MAIN QUERY 이전에 한 번 실행한다.
- 4) SUBQUERY 의 결과는 MAIN OUTER QUERY 에 의해 사용된다.

☞ Guidelines

- 1) SUBQUERY 는 괄호로 묶어야 한다.
- 2) 두 종류의 비교 연산자들이 SUBQUERY 에 사용된다.
 - ① 단일 행 연산자
=,>, >=, <, <=, <>, !=
 - ② 복수 행 연산자
IN, NOT IN, ANY, ALL, EXISTS
- 3) SUBQUERY 는 연산자의 오른쪽에 나타나야 한다.
- 4) SUBQUERY 는 많은 SQL 명령에서 사용 가능하다.
- 5) SUBQUERY 는 ORDER BY 절을 포함할 수 없다.

1.3 SUBQUERY 를 사용할 수 있는 절

- 1) WHERE, HAVING, UPDATE
- 2) INSERT 구문의 INTO
- 3) UPDATE 구문의 SET
- 4) SELECT 나 DELETE 의 FROM 절

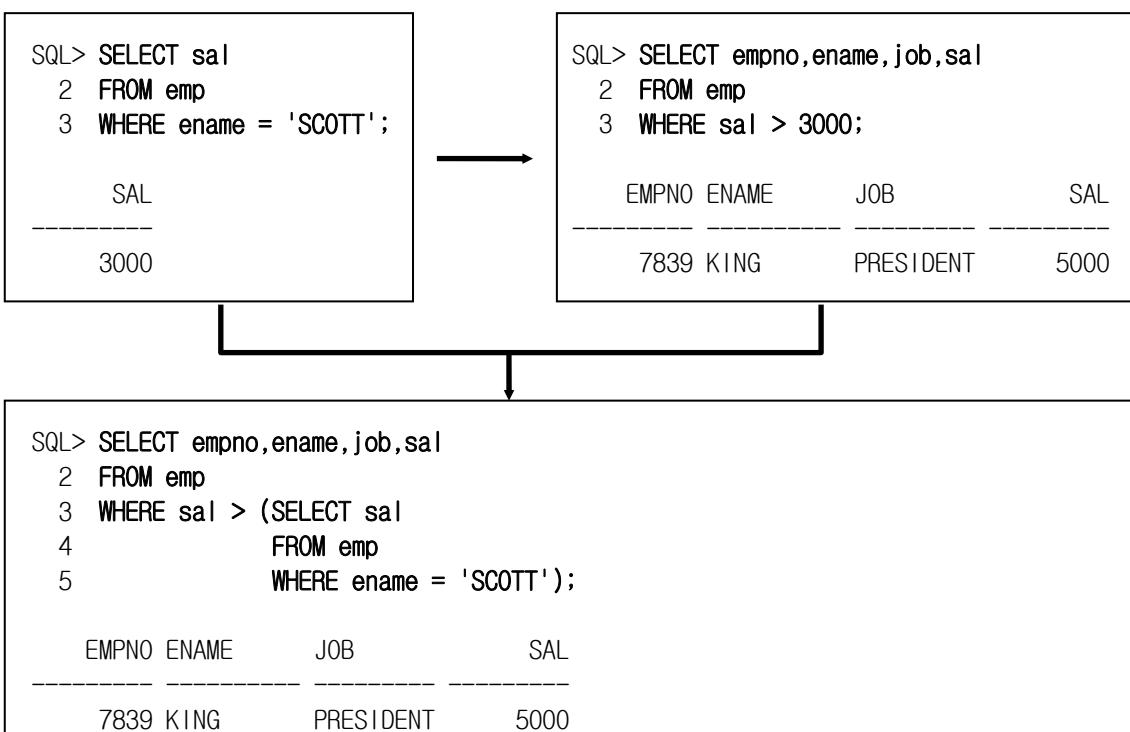
1.4 SUBQUERY 의 유형

- 1) 단일 행 SUBQUERY : SELECT 문장으로부터 오직 하나의 행만을 검색하는 질의입니다
- 2) 다중 행 SUBQUERY : SELECT 문장으로부터 하나 이상의 행을 검색하는 질의입니다
- 3) 다중 열 SUBQUERY : SELECT 문장으로부터 하나 이상의 열을 검색하는 질의입니다

1.5 단일 행 SUBQUERY

단일 행 SUBQUERY 는 내부 SELECT 문장으로부터 하나의 행을 검색하는 질의입니다. 이런 유형의 SUBQUERY 는 단일 행 연산자를 사용합니다. 이때 WHERE 절에 기술된 열의 개수와 데이터 타입은 SELECT 절에 기술된 열과 좌측부터 1 대 1 대응되며 데이터 타입이 일치해야 합니다.

문제) EMP 테이블에서 SCOTT 의 급여보다 많은 사원의 정보를 사원번호, 이름, 담당업무, 급여를 출력하여라.



문제 1) EMP 테이블에서 사원번호가 7521의 업무와 같고 급여가 7934보다 많은 사원의 정보를 사원번호, 이름, 담당업무, 입사일자, 급여를 출력하여라.

```
SQL> SELECT empno,ename,job,hiredate,sal
  2  FROM emp
  3 WHERE job = (SELECT job
  4                   FROM emp
  5                   WHERE empno = 7521)
  6 AND
  7       sal > (SELECT sal
  8                   FROM emp
  9                   WHERE empno = 7934);
```

EMPNO	ENAME	JOB	HIREDATE	SAL
7499	ALLEN	SALESMAN	20-FEB-81	1600
7844	TURNER	SALESMAN	08-SEP-81	1500

1.5.1) SUBQUERY에서 그룹 함수 사용

단일 행을 RETURN하는 SUBQUERY에 그룹 함수를 사용하여 MAIN QUERY로부터 데이터를 출력할 수 있다.

문제 2) EMP 테이블에서 급여의 평균보다 적은 사원의 정보를 사원번호, 이름, 담당업무, 급여, 부서번호를 출력하여라.

```
SQL> SELECT empno,ename,job,sal,deptno
  2  FROM emp
  3 WHERE sal < (SELECT AVG(sal)
  4                   FROM emp);
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7654	MARTIN	SALESMAN	1250	30
7499	ALLEN	SALESMAN	1600	30
7844	TURNER	SALESMAN	1500	30
7900	JAMES	CLERK	950	30
7521	WARD	SALESMAN	1250	30
7369	SMITH	CLERK	800	20
7876	ADAMS	CLERK	1100	20
.				
8 rows selected.				

1.5.2) SUBQUERY 가진 HAVING 절

SUBQUERY를 WHERE 절 뿐만 아니라 HAVING 절에서도 사용 가능하다. 오라클 서버는 SUBQUERY를 실행하고 MAIN QUERY의 HAVING 절에 RETURN 한다.

문제 3) EMP 테이블에서 20 번 부서의 최소 급여보다 많은 모든 부서를 출력하여라.

```
SQL> SELECT deptno,MIN(sal)
  2  FROM emp
  3  GROUP BY deptno
  4 HAVING MIN(sal) > (SELECT MIN(sal)
  5           FROM emp
  6           WHERE deptno = 20);
```

DEPTNO	MIN(SAL)
10	1300
30	950

문제 4) EMP 테이블에서 업무별로 가장 적은 급여를 출력하여라.

```
SQL> SELECT job,avg(sal)
  2  FROM emp
  3  GROUP BY job
  4 HAVING AVG(sal) = (SELECT MIN(AVG(sal))
  5           FROM emp
  6           GROUP BY job);
```

JOB	AVG(SAL)
CLERK	1037.5

1.6 다중 행 SUBQUERY

하나 이상의 행을 RETURN 하는 SUBQUERY 를 다중 행 SUBQUERY 라고 부릅니다. 다중 행 SUBQUERY 는 단일 행 연산자 대신에 다중 행 연산자를 사용합니다. 다중 행 연산자는 하나 이상의 값을 요구합니다.

```
SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp
  3 WHERE sal = (SELECT MIN(sal)
  4           FROM emp
  5           GROUP BY deptno);
ERROR:
ORA-01427: single-row subquery returns more than one row

no rows selected
```

위 SELECT 문의 문제점은 SUBQUERY 에서 RETURN 되는 ROW 가 1 개 이상이다. 이런 경우는 다중 행 SUBQUERY 연산자를 이용하여야 한다

```

SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp
  3 WHERE sal IN (SELECT MIN(sal)
  4                   FROM emp
  5                   GROUP BY deptno);

```

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	17-DEC-80	800	20
7900	JAMES	CLERK	03-DEC-81	950	30
7934	MILLER	CLERK	23-JAN-82	1300	10

1.6.1) 다중 행 SUBQUERY 연산자

- 1) IN 연산자
- 2) ANY 연산자
- 3) ALL 연산자
- 4) EXISTS 연산자

1.6.2) IN 연산자

2 개 이상의 값을 RETURN 하는 SUBQUERY 에 대하여 비교 연산자(=, !=, <, <=, >, >=)를 기술하면 ERROR 가 발생한다. 이런 경우 SUBQUERY 에서 RETURN 된 목록의 각각과 비교하여 QUERY 를 수행하는 연산자가 IN 이다.

문제 5) EMP 테이블에서 업무별로 최소 급여를 받는 사원의 정보를 사원번호, 이름, 업무, 입사일자, 급여, 부서번호를 출력하여라

```

SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp
  3 WHERE sal IN (SELECT MIN(sal)
  4                   FROM emp
  5                   GROUP BY job);

```

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	17-DEC-80	800	20
7654	MARTIN	SALESMAN	28-SEP-81	1250	30
7521	WARD	SALESMAN	22-FEB-81	1250	30
7782	CLARK	MANAGER	09-JUN-81	2450	10
7902	FORD	ANALYST	03-DEC-81	3000	20
7788	SCOTT	ANALYST	09-DEC-82	3000	20
7839	KING	PRESIDENT	17-NOV-81	5000	10

7 rows selected.

1.6.3) ANY 연산자

2 개 이상의 값을 RETURN 하는 SUBQUERY 에 대하여는 그런 값들을 어떻게 사용하는가를 지정해 두어야 한다. 비교 연산자(=, !=, <, <=, >, >=)와 SUBQUERY 사이에 ANY 연산자를 기술하여 RETURN 된 목록의 각각의 값과 비교한다.

문제 6) EMP 테이블에서 30 번 부서의 최소 급여를 받는 사원 보다 많은 급여를 받는 사원의 정보를 사원번호, 이름, 업무, 입사일자, 급여, 부서번호를 출력하여라. 단 30 번은 제외

```
SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp
  3 WHERE deptno != 30 AND sal > ANY (SELECT sal
  4                                     FROM emp
  5                                     WHERE deptno = 30);
```

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
7839	KING	PRESIDENT	17-NOV-81	5000	10
7782	CLARK	MANAGER	09-JUN-81	2450	10
7566	JONES	MANAGER	02-APR-81	2975	20
7902	FORD	ANALYST	03-DEC-81	3000	20
7788	SCOTT	ANALYST	09-DEC-82	3000	20
7876	ADAMS	CLERK	12-JAN-83	1100	20
.					
7 rows selected.					

1.6.4) ALL 연산자

2 개 이상의 값을 RETURN 하는 SUBQUERY 에 대하여는 그런 값들을 어떻게 사용하는가를 지정해 두어야 한다. 비교 연산자(=, !=, <, <=, >, >=)와 SUBQUERY 사이에 ALL 연산자를 기술하여 RETURN 된 목록의 모든 값과 비교한다.

문제 7) EMP 테이블에서 30 번 부서의 최고 급여를 받는 사원 보다 많은 급여를 받는 사원의 정보를 사원번호, 이름, 업무, 입사일자, 급여, 부서번호를 출력하여라. 단 30 번은 제외

```
SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp
  3 WHERE deptno != 30 AND sal > ALL (SELECT sal
  4                                     FROM emp
  5                                     WHERE deptno = 30);
```

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
7839	KING	PRESIDENT	17-NOV-81	5000	10
7566	JONES	MANAGER	02-APR-81	2975	20
7902	FORD	ANALYST	03-DEC-81	3000	20
7788	SCOTT	ANALYST	09-DEC-82	3000	20

1.6.5) EXISTS 연산자

SUBQUERY에서 적어도 1개의 행을 RETURN하면 논리식은 참이고 그렇지 않으면 거짓이다.

문제 8) EMP 테이블에서 적어도 한명의 사원으로부터 보고를 받을 수 있는 사원의 정보를
사원번호, 이름, 업무, 입사일자, 급여를 출력하여라. 단 사원번호 순으로 정렬하여라.

```
SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp e
  3 WHERE EXISTS (SELECT *
  4                   FROM emp
  5                   WHERE e.empno = mgr)
  6 ORDER BY empno;
```

EMPNO	ENAME	JOB	HIREDATE	SAL	DEPTNO
7566	JONES	MANAGER	02-APR-81	2975	20
7698	BLAKE	MANAGER	01-MAY-81	2850	30
7782	CLARK	MANAGER	09-JUN-81	2450	10
7788	SCOTT	ANALYST	09-DEC-82	3000	20
7839	KING	PRESIDENT	17-NOV-81	5000	10
7902	FORD	ANALYST	03-DEC-81	3000	20

6 rows selected.

1.7 다중 열 SUBQUERY

SUBQUERY의 구문을 작성할 때 WHERE 절에 비교되는 열이 하나가 아니라 여러 개의 열을
동시에 비교하는 경우가 있다. 이런 경우 다중 열 SUBQUERY 라하여 Pairwise 되었다고 한다.

1.7.1) Syntax

```
SELECT      * | column1[,column2,...]
  FROM      table
  WHERE      (column1,column2,...) IN ( SELECT column1,column2,...
                                         FROM      table
                                         WHERE      condition);
```

SUBQUERY로 작성된 구문을 실행하면 의외의 결과가 검색되는 경우가 있을 것이다. 이런 경우는 반드시 다중 열 SUBQUERY를 사용하여 조회하여야 올바른 자료를 검색할 수 있다.
아래의 예를 보고 정확한 개념을 이해하기 바란다.

문제 9) EMP 테이블에서 급여와 보너스가 부서 30에 있는 어떤 사원의 보너스와 급여에 일치하는 사원의 이름,부서번호,급여,보너스를 출력하여라.

여러분의 이해를 돋기 위해 자료를 수정하였다. 아래의 UPDATE 문장을 수행한 후 SELECT 문장을 수행하여라. UPDATE 문장은 다음 장에서 기술된다.

```
SQL> UPDATE emp  
2 SET sal = 1500, comm = 300  
3 WHERE empno = 7934;
```

1 row updated.

```
SQL> SELECT ename,deptno,sal,comm  
2 FROM emp  
3 WHERE sal IN (SELECT sal  
4 FROM emp  
5 WHERE deptno = 30)  
6 AND NVL(comm,-1) IN (SELECT NVL(comm,-1)  
7 FROM emp  
8 WHERE deptno = 30);
```

ENAME	DEPTNO	SAL	COMM
JAMES	30	950	
BLAKE	30	2850	
TURNER	30	1500	0
MILLER	10	1500	300
ALLEN	30	1600	300
WARD	30	1250	500
MARTIN	30	1250	1400

7 rows selected.

```
SQL> SELECT deptno,ename,job,sal,comm  
2 FROM emp  
3 ORDER BY deptno,ename;
```

DEPTNO	ENAME	JOB	SAL	COMM
30	BLAKE	MANAGER	2850	
30	JAMES	CLERK	950	
30	MARTIN	SALESMAN	1250	1400
30	TURNER	SALESMAN	1500	0
30	WARD	SALESMAN	1250	500

14 rows selected.

♣ 중요

위 결과는 30 번 부서에 급여가 1500이고 보너스가 300인 사원이 없는데도 출력되었다. 이는 조건을 각각 별도로 조회할 경우에 발생되는 문제점이다. 이를 해결하기 위해서는 Pairwise SUBQUERY를 이용하여야 한다.

문제 10) 업무별로 최소 급여를 받는 사원의 정보를 사원번호, 이름, 업무, 부서번호를 출력하여라. 단 업무별로 정렬하여라.

여러분의 이해를 돋기 위해 자료를 수정하였다. 아래의 UPDATE 문장을 수행한 후 SELECT 문장을 수행하여라. UPDATE 문장은 다음 장에서 설명된다.

```
SQL> UPDATE emp  
  2 SET sal = 2450  
  3 WHERE empno = 7900;
```

1 row updated.

```
SQL> SELECT empno,ename,job,sal,deptno  
  2 FROM emp  
  3 WHERE sal IN (SELECT MIN(sal)  
  4                      FROM emp  
  5                      GROUP BY job)  
  6 ORDER BY job;
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7902	FORD	ANALYST	3000	20
7788	SCOTT	ANALYST	3000	20
7369	SMITH	CLERK	800	20
7900	JAMES	CLERK	2450	30
7782	CLARK	MANAGER	2450	10
7839	KING	PRESIDENT	5000	10
7654	MARTIN	SALESMAN	1250	30
7521	WARD	SALESMAN	1250	30

8 rows selected.

```
SQL> SELECT job,MIN(sal)  
  2 FROM emp  
  3 GROUP BY job;
```

JOB	MIN(SAL)
ANALYST	3000
CLERK	800
MANAGER	2450
PRESIDENT	5000
SALESMAN	1250

♣ 중요

위 결과를 보면 사번이 7900 인 사원은 업무가 CLERK 이다. CLERK 의 업무의 최소 급여는 800에도 불구하고 출력되었다. 이는 업무별 최소 급여만 RETURN 되지 어느 업무가 어떤 최소값을 가지는지를 알 수 없다.

1.7.2) Pairwise SUBQUERY

앞의 결과를 보면 알 수 있듯이 어떤 업무의 급여가 최소인지 업무와 최소 급여를 동시에 비교하여야 한다.

문제 11) EMP 테이블에서 급여와 보너스가 부서 30에 있는 어떤 사원의 보너스와 급여에 일치하는 사원의 이름,부서번호,급여,보너스를 출력하여라.

```
SQL> SELECT ename,deptno,sal,comm
  2  FROM emp
  3 WHERE (sal,NVL(comm,-1)) IN (SELECT sal,NVL(comm,-1)
  4                                     FROM emp
  5                                     WHERE deptno = 30);
```

ENAME	DEPTNO	SAL	COMM
WARD	30	1250	500
MARTIN	30	1250	1400
TURNER	30	1500	0
ALLEN	30	1600	300
CLARK	10	2450	
JAMES	30	2450	
BLAKE	30	2850	

7 rows selected.

문제 12) 업무별로 최소 급여를 받는 사원의 정보를 사원번호,이름,업무,부서번호를 출력하여라. 단 업무별로 정렬하여라.

```
SQL> SELECT empno,ename,job,sal,deptno
  2  FROM emp
  3 WHERE (job,sal) IN (SELECT job,MIN(sal)
  4                           FROM emp
  5                           GROUP BY job)
  6 ORDER BY job;
```

EMPNO	ENAME	JOB	SAL	DEPTNO
7902	FORD	ANALYST	3000	20
7788	SCOTT	ANALYST	3000	20
7369	SMITH	CLERK	800	20
7782	CLARK	MANAGER	2450	10
7839	KING	PRESIDENT	5000	10
7654	MARTIN	SALESMAN	1250	30
7521	WARD	SALESMAN	1250	30

7 rows selected.

1.8 SUBQUERY에서의 NULL 값

NULL 값을 비교하는 모든 조건은 NULL이다.

```
SQL> SELECT e.empno,e.ename,e.job,e.sal  
2 FROM emp e  
3 WHERE e.empno IN (SELECT m.mgr  
4                      FROM emp m);
```

EMPNO	ENAME	JOB	SAL
7566	JONES	MANAGER	2975
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7788	SCOTT	ANALYST	3000
7839	KING	PRESIDENT	5000
7902	FORD	ANALYST	3000

6 rows selected.

위 문장의 SELECT 문장은 1명 이상으로부터 보고를 받을 수 있는 사원의 정보를 출력한 것이다. 그러면 말단 직원을 출력할 경우에는 SELECT 문장을 어떻게 기술해야 할까. 일반 사용자가 실수하기 쉬운 부분이다. 우선 IN 대신 NOT IN 을 사용하면 쉽게 해결될 것이라 생각한다. 그러나 여기에는 다음과 같은 문제점이 있다. 우선 다음의 SELECT 문장을 보기 바란다.

```
SQL> SELECT e.empno,e.ename,e.job,e.sal  
2 FROM emp e  
3 WHERE e.empno NOT IN (SELECT m.mgr  
4                           FROM emp m);
```

no rows selected

위의 SELECT 문장의 결과 SUBQUERY에서 RETURN 되는 값 중에는 NULL(KING은 MGR이 NULL이다)이 있다. NULL에 어떠한 연산을 하여도 모든 조건은 NULL이므로 전체 값이 존재하지 않는다고 RETURN 한다. SUBQUERY의 결과 집합의 일부분으로서 NULL값은 IN 연산자를 사용할 경우는 문제가 되지 않지만 NOT IN 연산자를 사용하면 안된다.

☞ Guidelines

SUBQUERY의 결과 집합의 일부분으로서 NULL값은 IN(= ANY) 연산자를 사용할 수 있다. 그러나 NOT IN(!= ALL)연산자를 사용할 수 없다.

1.9 FROM 절에서의 SUBQUERY

SUBQUERY 는 FROM 절에서도 사용 가능하다. 하나의 테이블에서 자료의 양이 많을 경우 FROM 절에 테이블 전체를 기술하여 사용하면 효율이 떨어질 수 있다. 이런 경우 필요한 행과 열만 선택하여 FROM 절에 기술하면 오라클 서버가 최적화 단계에서 효율적인 검색을 할 수 있다. 이처럼 FROM 절에 기술한 SUBQUERY 는 마치 VIEW 와 같은 역할을 한다. 이런 VIEW 를 INLINE VIEW 라 한다.

문제 13) EMP 과 DEPT 테이블에서 업무가 MANAGER 인 사원의 정보를 이름,업무,부서명,근무지를 출력하여라

```
SQL> SELECT e.ename,e.job,d.dname,d.loc
  2  FROM dept d,emp e
  3 WHERE job = 'MANAGER' AND e.deptno = d.deptno;
```

ENAME	JOB	DNAME	LOC
BLAKE	MANAGER	SALES	CHICAGO
CLARK	MANAGER	ACCOUNTING	NEW YORK
JONES	MANAGER	RESEARCH	DALLAS

```
SQL> SELECT e.ename,e.job,d.dname,d.loc
  2  FROM (SELECT ename,job,deptno
  3        FROM emp
  4       WHERE job = 'MANAGER') e, dept d
  5 WHERE e.deptno = d.deptno;
```

ENAME	JOB	DNAME	LOC
BLAKE	MANAGER	SALES	CHICAGO
CLARK	MANAGER	ACCOUNTING	NEW YORK
JONES	MANAGER	RESEARCH	DALLAS

◆ 연습문제 ◆

1. EMP 테이블에서 Blake 와 같은 부서에 있는 모든 사원의 이름과 입사일자를 출력하는 SELECT 문을 작성하시오.
2. EMP 테이블에서 평균 급여 이상을 받는 모든 종업원에 대해서 종업원 번호와 이름을 출력하는 SELECT 문을 작성하시오. 단 급여가 많은 순으로 출력하여라.
3. EMP 테이블에서 이름에 “T”가 있는 사원이 근무하는 부서에서 근무하는 모든 종업원에 대해 사원 번호, 이름, 급여를 출력하는 SELECT 문을 작성하시오. 단 사원번호 순으로 출력하여라.
4. EMP 테이블에서 부서 위치가 Dallas 인 모든 종업원에 대해 이름, 업무, 급여를 출력하는 SELECT 문을 작성하시오.
5. EMP 테이블에서 King 에게 보고하는 모든 사원의 이름과 급여를 출력하는 SELECT 문을 작성하시오.
6. EMP 테이블에서 SALES 부서 사원의 이름, 업무를 출력하는 SELECT 문을 작성하시오.
7. EMP 테이블에서 월급이 부서 30 의 최저 월급보다 높은 사원을 출력하는 SELECT 문을 작성하시오.
8. EMP 테이블에서 부서 10 에서 부서 30 의 사원과 같은 업무를 맡고 있는 사원의 이름과 업무를 출력하는 SELECT 문을 작성하시오.
9. EMP 테이블에서 FORD 와 업무도 월급도 같은 사원의 모든 정보를 출력하는 SELECT 문을 작성하시오.

10. EMP 테이블에서 업무가 JONS 와 같거나 월급이 FORD 이상인 사원의 정보를 이름,업무,부서번호,급여를 출력하는 SELECT 문을 작성하시오. 단 업무별, 월급이 많은 순으로 출력하여라.
11. EMP 테이블에서 SCOTT 또는 WARD 와 월급이 같은 사원의 정보를 이름,업무,급여를 출력하는 SELECT 문을 작성하시오.
12. EMP 테이블에서 CHICAGO 에서 근무하는 사원과 같은 업무를 하는 사원의 이름,업무를 출력하는 SELECT 문을 작성하시오.
13. EMP 테이블에서 부서별로 월급이 평균 월급보다 높은 사원을 부서번호,이름,급여를 출력하는 SELECT 문을 작성하시오.
14. EMP 테이블에서 업무별로 월급이 평균 월급보다 낮은 사원을 부서번호,이름,급여를 출력하는 SELECT 문을 작성하시오.
15. EMP 테이블에서 적어도 한명 이상으로부터 보고를 받을 수 있는 사원을 업무,이름,사원번호,부서번호를 출력하는 SELECT 문을 작성하시오.
16. EMP 테이블에서 말단 사원의 사원번호,이름,업무,부서번호를 출력하는 SELECT 문을 작성하시오.

1 SQL*Plus 명령어

SQL*Plus 명령어는 오라클 서버(데이터베이스)와 연관되어 작용하는 명령어는 아니며 사용자가 보다 효율적으로 SQL을 활용할 수 있도록 환경을 제공하는 TOOL이다.

1.1 SQL과 SQL*Plus의 차이점

1.1.1 SQL의 특징

- 1) RDBMS의 표준 언어
- 2) SQL Buffer에 바로 전에 실행한 SQL 문장이 저장
- 3) 명령어의 끝은 :을 사용한다

가) 명령어의 종류

구 분	종 류
Data Retrieval Language	SELECT
Data Manipulation Language	INSERT, UPDATE, DELETE, COMMIT, ROLLBACK
Data Definition Language	CREATE, ALTER, DROP, RENAME
Data Control Language	GRANT, REVOKE, AUDIT, NO AUDIT, LOCK
Miscellaneous Language	COMMENT

1.1.2 SQL*Plus의 특징

- 1) SQL 문장을 실행 한다.
- 2) SQL Buffer에 저장되지 않는다.
- 3) SQL 문장을 편집한다.
- 4) 환경 및 기본 조회 결과를 FORMATTING

가) 명령어의 종류

구 분	설 명
Execution Commands	/, RUN, EXECUTE
Edit Commands	LIST, APPEND, CHANGE, DEL, INPUT, EDIT
Environment Commands	SET, SHOW, PAUSE
Report Format Commands	COLUMN, CLEAR, BREAK, COMPUTE, TTITLE, BTITLE
File Manipulation Commands	SAVE, GET, START, @, @@, SPOOL
Interactive Commands	DEFINE, UNDELETE, PROMPT, ACCEPT, VARIABLE, PRINT
Database Access Commands	CONNECT, COPY, DISCONNECT
Miscellaneous Commands	SQLPLUS, EXIT, HELP, DESCRIBE, HOST, REMARK, WHENEVER

1.2 SQL 명령 편집 및 실행

1.2.1 SQL Buffer에는 SQL 명령어의 편집

SQL Buffer에 있는 명령어를 Line 단위로 편집할 수 있다.

가) 종류

명령어	설명
A[PPEND] text	현재 편집 라인의 끝에 text를 추가
C[HANGE]/old/new	현재 편집 라인의 old 문자를 new 문자로 바꿈
DEL [n]	n 라인을 삭제
I[NPUT] [text]	현재 편집 라인 다음에 라인을 추가하여 text를 추가
L[IST] [n]	SQL 문장을 보여주고, 편집 라인을 이동
n text	n 번째 라인을 text로 바꿈
R[UN]	Buffer에 있는 명령어를 실행한다.(/와 동일)

나) 사용 예

```
SQL> SELECT empno,ename,jb,dept
  2  FROM emp;
SELECT empno,ename,jb,dept
*
ERROR at line 1:
ORA-00904: invalid column name
SQL> c/jb/job
  1* SELECT empno,ename,job,dept
SQL> a no
  1* SELECT empno,ename,job,deptno
SQL> /
EMPNO ENAME      JOB          DEPTNO
----- -----
  7839 KING        PRESIDENT    10
  7698 BLAKE       MANAGER     30
  7782 CLARK       MANAGER     10
  7566 JONES       MANAGER     20
  7654 MARTIN     SALESMAN    30
  7499 ALLEN       SALESMAN    30
  7844 TURNER     SALESMAN    30
  7900 JAMES        CLERK      30
  7521 WARD        SALESMAN    30
  7902 FORD         ANALYST    20
  7369 SMITH       CLERK      20
  7788 SCOTT       ANALYST    20
...
14 rows selected.
```

1.2.2 파일에 있는 명령어 편집과 실행

SCRIPT 를 생성하거나 LINE 단위의 편집이 아니라 SCREEN 편집을 할 경우에 유용하다. SQL*Plus 파일 명령어는 파일을 저장, 획득, 적재 그리고 실행하는데 사용할 수 있다.

가) 종류

명령어	설명
EDIT [filename[.ext]]	지정된 파일의 내용이나 버퍼의 내용을 운영체제의 문자 편집기로 불러온다.
SAV[E] [filename[.ext]] [REP[LACE] APP[END]]	SQL 버퍼의 내용을 파일에 저장하고 기존 파일에 추가하기 위해서 APPEND 를, 기존 파일에 중복해서 쓰려면 REPLACE 를 사용한다. 기본적인 파일 확장자는 sql 이다.
STA[RT] [filename[.ext]]	지정된 파일을 수행한다. START 라는 명령 대신에 @을 사용할 수 있다. 파일 확장자가 .sql 이 아니면, 파일 확장자를 써야 한다.
GET [filename[.ext]]	SQL 버퍼에 파일의 내용을 기록한다. 파일명의 기본적인 확장자는 .lis 또는 .lst 이다.
SPO[OL] [filename[.ext]] OFF OUT]	SQL*Plus 의 내용(Query 결과)을 파일에 저장한다. OFF 는 스크립트 파일을 닫는다.
HOST	SQL*Plus 안에서 호스트 운영체제의 명령어를 실행한다.
!	UNIX Shell 로 나들이
!vi file_name.sql	file_name.sql 을 vi 편집기로 부름

♣ 참고

파일명을 쓰지 않고 EDIT 명령어를 사용할 때, 기본적인 파일명은 afiedt.buf 이다. 이 파일은 버퍼를 편집할 때마다 이 파일에 겹쳐 쓰게 된다. 현재의 디렉토리에 없는 파일명을 명시하면 SQL*Plus 는 파일 이름을 물는다.

나) 사용 예

```
SQL> SELECT e.empno,e.ename,e.job,e.sal,d.dname,d.loc  
2 FROM dept d,emp e  
3 WHERE d.deptno = e.deptno;
```

EMPNO	ENAME	JOB	SAL	DNAME	LOC
7839	KING	PRESIDENT	5000	ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	2850	SALES	CHICAGO
7782	CLARK	MANAGER	2450	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	2975	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	1250	SALES	CHICAGO
7499	ALLEN	SALESMAN	1600	SALES	CHICAGO
7844	TURNER	SALESMAN	1500	SALES	CHICAGO
7900	JAMES	CLERK	2450	SALES	CHICAGO

.....
14 rows selected.

```
SQL> save emp_dept  
Created file emp_dept  
SQL> ed emp_dept
```

File의 마지막 부분에 “and e.job = 'MANAGER'”를 추가 후 저장하고 종료한다.

```
SELECT e.empno,e.ename,e.job,e.sal,d.dname,d.loc  
FROM dept d,emp e  
WHERE d.deptno = e.deptno  
AND e.job = 'MANAGER'  
/
```

```
SQL> @emp_dept
```

EMPNO	ENAME	JOB	SAL	DNAME	LOC
7698	BLAKE	MANAGER	2850	SALES	CHICAGO
7782	CLARK	MANAGER	2450	ACCOUNTING	NEW YORK
7566	JONES	MANAGER	2975	RESEARCH	DALLAS

1.2.3 SQL*Plus 의 환경 설정

SQL*Plus 를 사용한 환경 설정은 SET 명령을 이용한다.

가) Syntax

```
SET 시스템변수 값
```

시스템변수 세션 환경을 제어하는 변수

값 시스템 변수의 값

나) 종류

SET 변수와 값	설명
ARRAY[SIZE] {20 n}	데이터베이스 데이터 패치의 크기를 설정
COLSEP { text}	열 사이에 출력되는 문자를 설정(Default:space)
FEED[BACK] {6 n OFF ON}	질의가 최소한 n 개이어야 ROW의 수를 출력
HEADING {OFF ON}	열의 HEADING을 출력할 지의 여부를 결정
LIN[ESIZE] {80 n}	LINE 당 문자의 수를 결정
LONG {80 n}	LONG 값을 출력하기 위해 최대 폭을 설정
PAGES[IZE] {24 n}	PAGE 당 LINE 수를 지정
PAU[SE] {OFF ON text}	화면 제어를 함
TERM[OUT] {OFF ON}	결과를 화면에 출력할 지의 여부를 결정

☞ Guidelines

- 1) SET 명령어는 현재의 세션(운영 중인 SQL*Plus) 환경을 제어
- 2) login.sql에는 로그온시 실행되는 표준 SET 명령과 그 외의 SQL*Plus 명령을 기술
- 3) login.sql을 수정하여 부가적인 SET 명령을 사용

♣ 참고

login.sql 파일에는 접속할 때마다 필요한 표준 SET 명령과 그 외의 SQL*Plus 명령어들이 들어 있다. 접속할 때에 이 파일을 읽어서 명령어가 수행된다. 로그 아웃을 하면 사용자 정의 설정이 상실된다. 설정 값을 영구적으로 변경하려면 login.sql 파일에 추가한다.

다) 사용 예

```
SQL> SET HEADING OFF
SQL> SET COLSEP '**'
SQL> SELECT *
  2 FROM ITEM;

      610**      3**    100890**      58**      1**       58
      611**      1**    100861**      45**      1**       45
      612**      1**    100860**      30**    100**     3000
      601**      1**    200376**      2.4**      1**       2.4
      602**      1**    100870**      2.8**    20**       56
      604**      1**    100890**      58**      3**       174
      604**      2**    100861**      42**      2**       84
      604**      3**    100860**      44**    10**     440
      603**      2**    100860**      56**      4**       224
. . . . .
64 rows selected.
```

1.2.4 SQL*Plus FORMAT 명령어

다음의 명령어를 사용하여 리포트의 속성을 제어할 수 있다.

명령어	설명
COL[UMN] [column option]	열 포맷을 제어한다.
TTI[TLE] [text OFF ON]	리포트의 머리말을 명시한다.
BTI[TLE] [text OFF ON]	리포트의 꼬리말을 명시한다.
BRE[AK] [ON report_element]	중복값을 제거하고 라인 피드로 행들을 단락 짓는다.

☞ Guidelines

- 1) 모든 포맷 명령은 SQL*Plus SESSION의 마지막이나 변경 전까지 효력을 유지한다.
- 2) 모든 리포트 후에는 SQL*Plus Default 값으로 RESET 하는 것을 원칙으로 한다.
- 3) SQL*Plus 변수 설정을 Default 값으로 해주는 명령은 없다. 특정 값을 알거나 로그 아웃한 후 로그인을 해야 한다.
- 4) 열에 별칭을 사용하였다면 열 이름이 아닌 별칭을 사용하여야 한다.

1.2.5 COLUMN 명령어

열의 출력을 제어합니다.

가) Syntax

COL[UMN] [{column alias} [option]]

나) COLUMN 명령의 OPTION

OPTION	설명
CLE[AR]	어떤 열의 형식을 해제합니다.
FOR[MAT] format	열 데이터의 디스플레이를 변경합니다.
HEA[DING] text	열 헤딩을 설정합니다. 수직 바()는 헤딩 라인을 나눕니다.
JUS[TIFY] [align]	열 HEADING 을 정렬(왼쪽, 가운데, 오른쪽)합니다.
NOPRI[NT]	열을 숨깁니다.
NUL[L] text	Null 값에 대해 디스플레이 되는 텍스트를 명시합니다.
PRI[NT]	열을 보여줍니다.
TRU[NCATED]	디스플레이 되는 첫번째 라인의 마지막 문자열을 절삭합니다.
WRA[PPED]	문자열이 끝나면 다음 라인으로 이동합니다.

다) 사용 예

```
SQL> COL ename HEADING '사원의|이름' FORMAT a15
SQL> COL ename HEA '사원의|이름' FOR a15
SQL> COL sal JUS LEFT FOR $999,990.00
SQL> COL mgr FOR 9999999999 NUL 'No manager'
SQL> SELECT empno, ename, mgr, sal, deptno
2 FROM emp
3 WHERE deptno = 10;
```

EMPNO	사원의 이름	MGR	SAL	DEPTNO
7839	KING	No manager	\$5,000.00	10
7782	CLARK	7839	\$2,450.00	10
7934	MILLER	7782	\$1,500.00	10

```
SQL> COL ename
column ename ON
heading '사원의|이름' headsep '||'
format a15
SQL> col ename CLE
SQL> COL ename
column 'ename' not defined
```

♣ 참고

SQL*Plus 명령이 너무 길다면 현재 라인의 마지막에 하이픈(–)을 두어서 다음 라인에 계속적으로 작성할 수 있습니다. 오라클 서버는 형식 모델에서 제공된 숫자의 자리 수를 초과하게 되면 자리 전체에 파운드 기호(#)의 문자열을 출력합니다. 또한 포맷 모델이 알파벳이지만 실제 값은 숫자인 값의 자리에도 파운드 기호(#)를 출력합니다.

1.3 SQL*Plus 를 이용하여 보고서 작성

SQL*Plus 명령어를 이용하여 간단한 보고서를 작성할 수 있다.

```
SQL> SET PAGESIZE 42
SQL> SET LINESIZE 54
SQL> SET FEEDBACK OFF
SQL> TTITLE '사원의 업무별|보고서 양식'
SQL> BTITLE 'GOOD BY'
SQL> COLUMN job HEADING 'Job|Category' FORMAT A20
SQL> COLUMN ename HEADING 'Employee' FORMAT A20
SQL> COLUMN sal HEADING 'Salary' FORMAT $999,990.00
SQL> BREAK ON job SKIP 1 ON REPORT
SQL> COMPUTE SUM OF sal ON job REPORT
SQL> SPOOL salary
SQL> SELECT job, ename, sal
2 FROM emp
3 ORDER BY job,ename,sal DESC;
```

Fri Feb 12	사원의 업무별 보고서 양식	page 1	Display a Header, Date and Page Number
Job Category	Employee	Salary	Format Column Headings
ANALYST	FORD	\$3,000.00	
	SCOTT	\$3,000.00	
*****		-----	
sum		\$6,000.00	Display Subtotal
CLERK	ADAMS	\$1,100.00	
	JAMES	\$950.00	
	MILLER	\$1,500.00	
	SMITH	\$800.00	
*****		-----	
sum		\$4,350.00	
MANAGER	BLAKE	\$2,850.00	
	CLARK	\$2,450.00	
	JONES	\$2,975.00	
*****		-----	
sum		\$8,275.00	
PRESIDENT	KING	\$5,000.00	
*****		-----	
sum		\$5,000.00	
SALESMAN	ALLEN	\$1,600.00	
	MARTIN	\$1,250.00	
	TURNER	\$1,500.00	
	WARD	\$1,250.00	
*****		-----	
sum		\$5,600.00	Suppress Duplicate Values

sum		\$29,225.00	Display a Grand Total

	GOOD BY		Display a Footer

1.3.1 BREAK 명령

행 단락을 구분 짓고 중복 값을 제거하기 위해서 BREAK 명령을 사용합니다. BREAK 명령이 효과적으로 수행되기 위해서 BREAK 되는 열에 대해서 ORDER BY 하십시오.

가) Syntax

BREAK ON column[alias row] [skip n dup page] on [ON REPORT]

page break 값이 변경될 때 새로운 PAGE로 SKIP

skip n break 값이 변경될 때 n 만큼 줄을 SKIP(COLUMN,ROW,PAGE,REPORT)

duplicate 중복되는 값을 출력

나) 사용 예

```
SQL> BREAK ON deptno ON job : 중복 제거의 경우  
SQL> BREAK ON REPORT : 전체 합계의 경우  
SQL> BREAK ON deptno SKIP 2 ON job SKIP 1 : BREAK 값에서 행을 단락짓는 경우
```

다) CLEAR 명령

CLEAR 명령은 BREAK POINT 설정을 해제한다.

```
CLEAR BREAK
```

1.3.2 BREAK 를 이용한 SELECT 문장 기술

SELECT 문의 ORDER BY 절로 BREAK 를 제어한다.

가) Syntax

```
SELECT      . . . . .  
  FROM       . . . . .  
  ORDER BY   break_column;
```

나) 사용 예

```
SQL> BREAK ON job SKIP 1 ON REPORT  
SQL> SELECT job, ename, sal  
2  FROM emp  
3  ORDER BY job,ename,sal DESC;
```

☞ Guidelines

- 1) BREAK 명령을 사용할 때 DATA 의 의미 있는 SUBSET 를 만들기 위해 BREAK 명령에 참조 된 COLUMN 을 ORDER BY 절에 기술한다.
- 2) BREAK 명령은 마지막에 기술된 오직 하나의 명령어만 유효하다.

1.3.4 COMPUTE 명령

SQL*Plus 명령어를 이용하여 요약된 계산을 더한다.

가) Syntax

```
COMPUTE function OF compute_column ON break_column
```

function COUNT,NUM,MAX,MIN,SUM,AVG,STD,VAR 중 하나의 함수를 기술
compute_column 계산에 사용되는 COLUMN 이나 식
break_column BREAK 명령으로 기술된 COLUMN

나) COMPUTE 명령 취소

현재 설정된 COMPUTE 명령을 Clear 한다.

```
CLEAR COMPUTE
```

다) 사용 예

```
SQL> BREAK ON job SKIP 1 ON REPORT  
SQL> COMPUTE SUM OF sal ON job REPORT
```

1.3.5 TTITLE 과 BTITLE 명령

머리말과 꼬리말을 출력합니다.

가) Syntax

```
TTITLE [text|OFF|ON]
```

나) 사용 예

```
SQL> TTITLE '사원의 업무별|보고서 양식'  
SQL> BTITLE 'GOOD BY'
```

1.3.6 REPORT 를 실행 하기 위한 SCRIPT FILE 작성

SQL 프롬프트에서 각각의 SQL*Plus 명령을 입력하거나 명령(또는 스크립트)파일 SELECT 문장을 포함하는 모든 명령어를 입력할 수 있습니다. 전형적인 스크립트는 최소한 하나의 SQL 문장과 여러 개의 SQL*Plus 명령어들로 구성되어 있습니다.

가) 작성 절차

```
SQL> ed rep
```

“ORACLE_HOME\BIN” DIRECTORY 에 rep.sql FILE 이 존재하지 않으면 생성 여부를 요구하고 rep.sql FILE 이 있으면 FILE 을 OPEN 한다. OPEN 된 FILE 에 내용을 입력한 후 저장하고 종료한다.

```
SQL> @rep
```

위 명령은 SQL> START rep 와 동일한 명령으로 SCRIPT 의 내용을 실행한다.

♣ 참고

SCRIPT 작성 중 SQL*Plus Window 를 클릭하여 사용할 수 없다. 반드시 편집중인 Window 를 종료 후 SQL*Plus 사용할 수 있다.

문제 1) 아래의 SCRIPT 를 분석하여라

```
SET PAGESIZE 42
SET LINESIZE 54
SET FEEDBACK OFF
TTITLE '사원의 업무별|보고서 양식'
BTITLE 'GOOD BY'
COLUMN job HEADING 'Job|Category' FORMAT A20
COLUMN ename HEADING 'Employee' FORMAT A20
COLUMN sal HEADING 'Salary' FORMAT $999,990.00
BREAK ON job SKIP 1 ON REPORT
COMPUTE SUM OF sal ON job REPORT
SPOOL salary
SELECT job, ename, sal
FROM emp
ORDER BY job,ename,sal DESC
/
```

문제 2) 아래의 SCRIPT 를 분석하여라

```
SET PAGESIZE 36
SET LINESIZE 64
SET FEEDBACK OFF
TTITLE '사원의 업무별|보고서 양식'
BTITLE '수고 하였습니다'
COLUMN job HEADING '담당업무' FORMAT A20
COLUMN ename HEADING '성 명' FORMAT A20
COLUMN sal HEADING '급 여' FORMAT $999,990.00
BREAK ON deptno SKIP 2 ON job SKIP 1 ON REPORT
COMPUTE SUM OF sal ON deptno REPORT
SPOOL emp_sal
SELECT deptno,job, ename, sal
FROM emp
ORDER BY deptno,job,ename,sal DESC;
SPOOL OFF
CREATE COMPUTE
CREATE BREAK
COLUMN job CLEAR
COLUMN ename CLEAR
COLUMN sal CLEAR
TTITLE OFF
BTITLE OFF
SET PAGESIZE 14
SET LINESIZE 80
SET FEEDBACK ON
```

1.4 상호작용 리포트

SQL*Plus 를 사용하여 리턴된 자료의 범위를 제한하는데 사용자가 입력하는 값을 이용하기 위해서 사용자에게 PROMPT 라는 리포트를 생성할 수 있다. 상호 작용 리포트를 생성하기 위해 명령어 파일이나 단일 SQL 문장에 치환 변수를 내장할 수 있다.

1.4.1 치환 변수

1) 값을 임시로 저장하기 위해서 SQL*Plus 치환 변수를 사용합니다.

- ① Single ampersand(&)
- ② Double ampersand(&&)
- ③ DEFINE 과 ACCEPT 명령어

2) SQL 문장간에 변수 값을 전달 합니다.

3) 머리말과 꼬리말을 동적으로 변경합니다.

♣ 참고

SQL*Plus 는 사용자 입력에 대한 타당성 검사를 하지 않는다. 사용자에 대해서 만드는 PROMPT 는 단순하고 모호하지 않게 하십시오.

1.4.2 치환 변수 사용할 수 있는 절

- 1) WHERE 절
- 2) ORDER BY
- 3) COLUMN 표현식
- 4) 테이블 이름
- 5) 전체 SELECT 문장

1.4.3 Single ampersand(&)의 치환 변수

리포트에서 실행할 때 사용자는 종종 리턴되는 데이터를 동적으로 제한 하기를 원한다. SQL*Plus 는 사용자 변수로써 이러한 융통성을 제공합니다. SQL 에서 각각의 변수를 인식하기 위해서 “&”를 사용합니다. 숫자는 &dept_no 와 같이 인용 부호를 사용하지 않고 사용하고 문자와 날짜에 대해서는 단일 인용 부호(‘&name’)를 사용하면 보다 편리하게 사용할 수 있다.

문제 3) EMP 테이블에서 부서번호를 입력받아 동적 조회할 수 있는 SELECT 문장을 기술하여라.

```
SQL> SELECT empno,ename,job,hiredate,deptno
  2  FROM emp
  3 WHERE deptno = &dept_no;
Enter value for dept_no: 10
old  3: WHERE deptno = &dept_no
new  3: WHERE deptno = 10

EMPNO ENAME      JOB        HIREDATE          DEPTNO
----- -----      -----      -----          -----
 7839 KING        PRESIDENT 17-NOV-81          10
 7782 CLARK       MANAGER   09-JUN-81          10
 7934 MILLER     CLERK    23-JAN-82          10
```

참고) SET VERIFY ON/OFF 명령어를 이용하여 old 와 new 의 출력을 조절할 수 있다.

```
SQL> SET VERIFY OFF
SQL> SELECT empno,ename,job,hiredate,deptno
  2  FROM emp
  3 WHERE deptno = &dept_no;
Enter value for dept_no: 10

EMPNO ENAME      JOB        HIREDATE          DEPTNO
----- -----      -----      -----          -----
 7839 KING        PRESIDENT 17-NOV-81          10
 7782 CLARK       MANAGER   09-JUN-81          10
 7934 MILLER     CLERK    23-JAN-82          10

SQL> SET VERIFY ON
```

문제 4) EMP 테이블에서 이름을 입력받아 동적 조회할 수 있는 SELECT 문장을 기술하여라.

```
SQL> SELECT empno,ename,job,hiredate,sal,deptno
  2  FROM emp
  3 WHERE UPPER(ename) = UPPER('&name');
Enter value for name: scott
old  3: WHERE ename = UPPER('&name')
new  3: WHERE ename = UPPER('scott')

EMPNO ENAME      JOB        HIREDATE          SAL       DEPTNO
----- -----      -----      -----          -----      -----
 7788 SCOTT      ANALYST   09-DEC-82        3000          20
```

문제 5) EMP 테이블에서 사원번호는 반드시 출력하고 나머지 열은 입력받아 출력하고 조건도 입력받아 출력하여라

```
SQL> SELECT empno,&column_name
  2  FROM emp
  3 WHERE &condition;
Enter value for column_name: ename,job,deptno
old  1: SELECT empno,&column_name
new  1: SELECT empno,ename,job,deptno
Enter value for condition: deptno = 10
old  3: WHERE &condition
new  3: WHERE deptno = 10

EMPNO ENAME      JOB          DEPTNO
----- ----- -----
    7839 KING        PRESIDENT      10
    7782 CLARK       MANAGER       10
    7934 MILLER     CLERK        10
```

1.4.4 Double ampersand(&&)의 치환 변수

매번 사용자에게 입력받지 않고 입력된 변수의 값을 사용하고자 할 경우에 사용한다. 사용자는 오직 한번만 입력하면 된다.

문제 6) EMP 테이블에서 사원번호, 이름, 업무는 반드시 출력하고 나머지 열은 입력받아 출력하고 입력받은 열을 정렬 조건으로 사용한다.

```
SQL> SELECT empno,ename,job,&&column_name
  2  FROM emp
  3 ORDER BY &column_name;
Enter value for column_name: deptno
old  1: SELECT empno,ename,job,&&column_name
new  1: SELECT empno,ename,job,deptno
old  3: ORDER BY &column_name
new  3: ORDER BY deptno

EMPNO ENAME      JOB          DEPTNO
----- ----- -----
    7839 KING        PRESIDENT      10
    7782 CLARK       MANAGER       10
    . . . . .
14 rows selected.
```

1.4.5 사용자 변수의 정의

SELECT 문장을 실행하기 전에 사용자 변수를 미리 정의해 사용할 수 있습니다. SQL*Plus는 사용자 변수를 정의하고 설정하기 위해 두개의 명령어를 제공합니다.

명령어	설명
DEFINE variable = value	CHAR 데이터형 사용자 변수를 생성하고 값을 할당합니다.
DEFINE variable	변수, 변수 값, 변수 데이터형을 출력합니다.
DEFINE	값과 데이터형을 가진 모든 데이터형을 출력합니다.
ACCEPT	사용자 입력 라인을 읽고 그것을 변수에 저장합니다

가) ACCEPT 명령어

- 1) 사용자 입력을 받을 때 사용자가 원하는 프롬프트를 생성합니다.
- 2) NUMBER 또는 DATE 데이터형 변수를 명시적으로 정의합니다.
- 3) 보안의 이유 때문에 사용자 입력을 숨깁니다.

```
ACCEPT variable [datatype] [FOR[MAT] format] [PROMPT text] [HIDE]
```

variable 값을 저장하는 변수의 이름입니다.
 존재하지 않으면 SQL*Plus 가 그것을 생성하여 사용합니다.

Datatype NUMBER,CHAR 또는 DATE.CHAR 는 최대 길이 제한이 240bytes 입니다.
 DATE 는 형식 모델을 다시 검사하고 데이터형은 CHAR 입니다.

format 형식 모델을 명시합니다.(예:A10, 9.999)

text 사용자가 값을 입력하기 전에 값을 출력합니다.

HIDE 사용자 입력을 숨긴다.(예:패스워드)

♣ 주의

ACCEPT 명령에서 치환 매개변수를 참조할 때 SQL*Plus 치환 매개변수 앞에 앤퍼샌드(&)를 두어서는 안됩니다.

문제 7) ACCEPT 명령으로 업무를 입력받아 사원번호, 이름, 업무, 급여를 출력하여라

```
SQL> ACCEPT job PROMPT '담당업무를 입력하시오: '
담당업무를 입력하시오: manager
SQL> SELECT empno,ename,job,sal
  2 FROM emp
  3 WHERE UPPER(job) = UPPER('&job');
old  3: WHERE UPPER(job) = UPPER('&job')
new  3: WHERE UPPER(job) = UPPER('manager')
```

EMPNO	ENAME	JOB	SAL
7698	BLAKE	MANAGER	2850
7782	CLARK	MANAGER	2450
7566	JONES	MANAGER	2975

나) DEFINE과 UNDEFINE 명령어

- 1) 변수는 다음의 경우까지 계속 정의 됩니다.
 - ① 선언된 변수에 대해 UNDEFINE 명령을 사용
 - ② SQL*Plus 종료
- 2) DEFINE 명령으로 변수 내용을 검사할 수 있습니다.
- 3) 모든 SESSION에 대해 변수를 정의하기 위해서는 login.sql file 을 수정하면 변수는 Startup 시 생성되어 사용할 수 있습니다.

문제 8) 부서이름(sales) 유지하기 위해 변수(dept_name)를 선언한 다음 이 변수를 이용하여 DEPT 테이블을 검색하여라

```
SQL> DEFINE dept_name = sales
SQL> DEFINE dept_name
DEFINE DEPT_NAME      = "sales" (CHAR)
SQL> SELECT *
  2  FROM dept
  3 WHERE UPPER(dname) = UPPER('&dept_name');
old   3: WHERE UPPER(dname) = UPPER('&dept_name')
new   3: WHERE UPPER(dname) = UPPER('sales')

DEPTNO DNAME          LOC
----- -----
      30 SALES        CHICAGO
```

◆ 연습문제 ◆

1. 단일 앤퍼샌드(&) 치환 변수를 설명하시오.
2. EMP 테이블에서 이름과 업무는 ,로 구분하여 출력하고 입사일자는 YYYY 년 MM 월 DD 일 X
요일형태로 출력하는 SELECT 문장을 기술하시오.
3. EMP 테이블에서 실행시 이름을 입력받아 이름,업무,급여를 출력하는 SCRIPT 를 작성하
시오.

```
Please enter the location name : Dallas
old  4: AND loc = UPPER('&p_name')
new  4: AND loc = UPPER('Dallas')
```

EMPLOYEE	JOB	DEPT_NAME
JONES	MANAGER	RESEARCH
FORD	ANALYST	RESEARCH
SMITH	CLERK	RESEARCH
SCOTT	ANALYST	RESEARCH
ADAMS	CLERK	RESEARCH

4. EMP 테이블에서 입력되는 지역에 대해서 부서명,이름,입사일,급여를 출력하는 SCRIPT
를 작성하시오.

```
Please enter the location name : chicago
old  6: AND loc = UPPER('&p_name')
new  6: AND loc = UPPER('chicago')
```

DEPARTMENT NAME	EMPLOYEE NAME	START DATE	ANNUAL SALARY
SALES	BLAKE	01-MAY-81	\$2,850.00
	MARTIN	28-SEP-81	\$1,250.00
	ALLEN	20-FEB-81	\$1,600.00
	TURNER	08-SEP-81	\$1,500.00
	JAMES	03-DEC-81	\$950.00
	WARD	22-FEB-81	\$1,250.00

6 rows selected.

1. 테이블 생성

CREATE TABLE 문장을 실행하여 테이블을 생성 합니다. 이 문장은 DDL 문장으로 Oracle8 데이터베이스 구조를 생성, 수정, 삭제하는데 사용되는 SQL 문장입니다. 이러한 문장은 데이터베이스에 즉각 영향을 미치며 데이터베이스 사전(DATA DICTIONARY)에 정보를 기록 합니다. CREATE TABLE 문장을 실행 후 SQL*Plus에서 “DESC table_name”으로 확인 할 수 있습니다. 테이블을 생성하기 위해서는 SYSTEM PRIVILEGE(다음 과정)인 CREATE TABLE 권한(SQL> SELECT * FROM role_sys_privs;)을 가지고 있어야 합니다. 또한 사용자가 테이블을 만들 수 있는 공간(SQL> SELECT * FROM user_free_space;)을 확보하여야 합니다.

♣ 참고

SYSTEM PRIVILEGE, OBJECT PRIVILEGE, 사용자의 권한, 사용 가능한 공간을 사용자는 DATA DICTIONARY를 검색하여 알 수 있다. 물론 뒤 장에서 좀더 자세히 다루도록 하겠다.

1.1 ORACLE에서 사용하는 객체

ORACLE 데이터베이스는 여러 개의 데이터 구조를 가지고 있습니다. 데이터베이스 설계에서 각각의 구조는 데이터베이스 개발 단계에서부터 생성할 수 있습니다.

액체	설명
TABLE	행과 열로 구성된 기본적인 저장 구조
VIEW	하나 이상의 테이블에서 데이터의 부분집합을 논리적으로 표현
SEQUENCE	고유한 번호를 자동으로 발생시키는 객체로 주로 PK 값 생성에 사용
INDEX	질의(SELECT) 성능을 향상시키기 위하여 사용하는 물리적인 저장 구조
SYNONYM	객체에 대한 이름을 부여

☞ Guidelines

- 1) 데이터베이스를 사용하고 있는 동안에도 언제든지 테이블을 생성할 수 있다.
- 2) 테이블의 크기는 명시할 필요가 없다.
- 3) 테이블 구조는 언제든지 수정 가능하다.

문제 1) ORACLE DATA DICTIONARY에서 SYSTEM PRIVILEGE를 조회하여라.

```
SQL> SELECT *
  2  FROM system_privilege_map;
PRIVILEGE NAME
-----
-3 ALTER SYSTEM
-4 AUDIT SYSTEM
. . .
86 rows selected.
```

문제 2) ORACLE DATA DICTIONARY에서 OBJECT PRIVILEGE를 조회하여라.

```
SQL> SELECT *  
  2 FROM table_privilege_map;
```

PRIVILEGE NAME

```
-----  
0 ALTER  
1 AUDIT  
2 COMMENT  
.....  
13 rows selected.
```

문제 3) 현재 SESSION을 이루고 있는 사용자의 SYSTEM PRIVILEGE 중 ROLE에 관련된 사항을 ORACLE DATA DICTIONARY에서 조회하여라.

```
SQL> CONN scott/tiger  
Connected.  
SQL> SELECT *  
  2 FROM role_sys_privs;
```

ROLE	PRIVILEGE	ADM
CONNECT	ALTER SESSION	NO
CONNECT	CREATE CLUSTER	NO
CONNECT	CREATE DATABASE LINK	NO
.....		
13 rows selected.		

문제 4) 현재 SESSION을 이루고 있는 사용자가 사용할 수 있는 FREE SPACE가 얼마인지 ORACLE DATA DICTIONARY에서 조회하여라.

```
SQL> CONN scott/tiger  
Connected.  
SQL> SELECT *  
  2 FROM user_free_space;
```

TABLESPACE_NAME	FILE_ID	BLOCK_ID	BYTES	BLOCKS
SYSTEM	1	8370	71680	35
.....				
USER_DATA	2	57	61440	30
ROLLBACK_DATA	3	802	3602432	1759
TEMPORARY_DATA	4	2	2095104	1023

10 rows selected.

1.1.1 Syntax

```
CREATE TABLE [schema.]table_name  
  ( column datatype [DEFAULT expr] [column_constraint],  
    . . . . .  
  [table_constraint]);
```

schema	테이블의 소유자
table_name	생성하고자 하는 테이블 이름. 사용자 단위로 유일한 이름
column	테이블에서 사용하는 열 이름. 테이블 단위로 유일한 이름
datatype	열의 자료형
DEFAULT expr	INSERT 문장에서 값을 생략시 기본적으로 입력되는 값을 명시
column_constraint	열정의 부분에서 무결성 제약 조건을 기술
table_constraint	테이블 정의 부분에서 무결성 제약 조건을 기술

1.1.2 이름 지정 규칙

객체 이름을 지정하는 표준 규칙에 따라 데이터베이스 테이블과 열의 이름을 정합니다.

- 1) 문자로 시작하여야 한다
- 2) 문자의 길이는 1 ~ 30 이내를 사용한다.
- 3) 오직 A ~ Z, a ~ z, 0 ~ 9, _, \$, # 만을 사용 가능하다. 단 한글 데이터베이스에서는 한글 사용 가능하다.
- 4) 동일한 사용자가 소유한 객체 이름은 중복될 수 없다.
- 5) 예약어는 사용할 수 없다.

1.1.3 DATA TYPE

DATA TYPE	설명
VARCHAR2(n)	가변 길이 문자 데이터(1~4000byte)
CHAR(n)	고정 길이 문자 데이터(1~2000byte)
NUMBER(p,s)	전체 p 자리 중 소수점 이하 s 자리(p:1~38, s:-84~127)
DATE	7Byte(BC 4712년 1월 1일부터 AD 9999년 12월 31일)
LONG	가변 길이 문자 데이터(1~2Gbyte)
CLOB	단일 바이트 가변 길이 문자 데이터(1~4Gbyte)
RAW(n)	n Byte 의 원시 이진 데이터(1~2000)
LONG RAW	가변 길이 원시 이진 데이터(1~2Gbyte)
BLOB	가변 길이 이진 데이터(1~4Gbyte)
BFILE	가변 길이 외부 파일에 저장된 이진 데이터(1~4Gbyte)

☞ Guidelines

- 1) 테이블이나 다른 데이터베이스 객체에 대한 서술적인 이름을 사용하여라.
- 2) 다른 테이블에도 일관되게 똑같은 이름을 지정하여라(예:EMP 와 DEPT 의 DEPTNO)
- 3) 객체 이름은 대소문자를 구분하지 않는다.

1.1.4 DEFAULT OPTION

열은 DEFAULT OPTION 을 사용하여 DEFAULT 값을 부여할 수 있다. 이 OPTION 은 열에 대한 값없이 어떤 행을 입력할 경우 NULL 값이 입력되지 않게 해 줍니다.

- 1) 삽입시 열에 대한 Default 값을 명시한다.
- 2) 기술 가능한 값은 literal 값, 표현식, SQL 함수(SYSDATE,USER 등)이다.
- 3) 불가능한 값은 다른 열의 이름이나 의사(NEXTVAL,CURRVAL 등)열입니다.
- 4) DEFAULT DATA TYPE 은 열의 DATA TYPE 과 일치해야 한다.

1.2 제약 조건

ORACLE SERVER 은 부적절한 자료가 입력되는 것을 방지하기 위하여 constraint 을 사용한다.

- 1) 제약 조건은 테이블 LEVEL 에서 규칙을 적용합니다.
- 2) 제약 조건은 종속성이 존재할 경우 테이블 삭제를 방지 합니다.
- 3) 테이블에서 행이 삽입,갱신,삭제될 때마다 테이블에서 규칙을 적용합니다.
- 4) Developer/2000 같은 ORACLE TOOL 에 대한 규칙을 제공 합니다.
- 5) 제약 조건의 유형은 ORACLE 에서 유효합니다.

☞ Guidelines

모든 제약 조건은 DATA DICTIONARY 에 저장 됩니다. 제약 조건의 이름을 의미 있게 부여 했다면 참조하기가 보다 쉽습니다. 제약 조건의 이름은 표준 객체 이름 규칙을 따릅니다. 제약 조건을 명명하지 않는다면 ORACLE SERVER 이 SYS_Cnxxxx 의 형태로 이름을 생성합니다.

제약 조건은 테이블 생성시나 테이블이 생성된 후에 정의될 수 있습니다. 또는 일시적으로 DISABLE 할 수 있고 ENABLE 할 수도 있습니다. User_constraints 의 DATA DICTIONARY VIEW 을 조회하므로 지정 테이블에 대해 정의된 제약 조건을 볼 수 있다.

1.2.1 제약 조건 정의 방법

제약 조건 정의하는 방법에는 COLUMN LEVEL 과 TABLE LEVEL 두 가지 방법이 있다.

가) 컬럼 LEVEL 제약 조건(COLUMN LEVEL CONSTRAINT)

- 1) 열별로 제약 조건을 정의한다.
- 2) 무결성 제약 조건 5 가지를 모두 적용할 수 있다.

- 3) NOT NULL 제약 조건은 컬럼 LEVEL에서만 가능하다.

◆ Syntax

Column datatype [CONSTRAINT constraint_name] constraint_type

나) 테이블 LEVEL 제약 조건(TABLE LEVEL CONSTRAINT)

- 1) 테이블의 컬럼 정의와는 개별적으로 정의한다.
- 2) 하나 이상의 열을 참조할 경우에 사용
- 3) NOT NULL을 제외한 나머지 제약 조건만 정의 가능하다.

◆ Syntax

column datatype, [CONSTRAINT constraint_name] constraint_type (column1[, column2,])

☞ Guidelines

constraint name 만 보고도 어떤 용도의 CONSTRAINT 인가를 식별할 수 있으면 사용자는 쉽게 데이터베이스를 운용할 것이다. 다음의 규칙에 따라 constraint name 을 부여하기를 권장한다. tablename_columnname_constrainttype (예 : emp_empno_pk, emp_deptno_fk)

1.2.2 데이터 무결성 제약 조건의 종류

제약 조건	설명
PRIMARY KEY(PK)	유일하게 테이블의 각 행을 식별(NOT NULL과 UNIQUE 조건을 만족)
FOREIGN KEY(FK)	열과 참조된 열 사이의 외래키 관계를 적용하고 설정합니다.
UNIQUE key(UK)	테이블의 모든 행을 유일하게 하는 값을 가진 열(NULL을 허용)
NOT NULL(NN)	열은 NULL 값을 포함할 수 없습니다.
CHECK(CK)	참이어야 하는 조건을 지정함(대부분 업무 규칙을 설정)

가) PRIMARY KEY(PK)

- 1) 테이블에 대한 기본 키를 생성합니다.
- 2) 하나의 기본 키만이 각 테이블에 대해 존재할 수 있다.
- 3) PRIMARY KEY 제약 조건은 테이블에서 각 행을 유일하게 식별하는 열 또는 열의 집합입니다.(UNIQUE와 NOT NULL 조건을 만족)
- 4) 이 제약 조건은 열 또는 열의 집합의 유일성을 요구하고 NULL 값을 포함할 수 없음을 보증 합니다.
- 5) UNIQUE INDEX 가 자동 생성된다.

◆ Syntax

column datatype [CONSTRAINT constraint_name] PRIMARY KEY (col1[,col2,...])
column datatype, , [CONSTRAINT constraint_name] PRIMARY KEY (column1[,column2,...])

문제 5) 아래의 두 문장의 차이점을 설명하여라.

SQL> CREATE TABLE test_tab1(2 id NUMBER(2) CONSTRAINT test_id_pk PRIMARY KEY, 3 name VARCHAR2(10));

Table created.

SQL> CREATE TABLE test_tab2(2 id NUMBER(2), 3 name VARCHAR2(10), 4 CONSTRAINT test_id_pk PRIMARY KEY (id));

Table created.

나) FOREIGN KEY(FK)

- 1) FOREIGN KEY 는 DETAIL 쪽에서 정의한다.
- 2) MASTER TABLE 의 PRIMARY KEY,UNIQUE KEY 를 정의된 열을 지정할 수 있으며 열의 값과 일치하거나 NULL 값이어야 한다.
- 3) FOREIGN KEY 는 열 또는 열의 집합을 지정할 수 있으며 동일 테이블 또는 다른 테이블간의 관계를 지정할 수 있다.
- 4) ON DELETE CASCADE 을 사용하여 DETAIL TABLE 에서 관련된 행을 삭제하고 MASTER TABLE 에서 삭제를 허용할 수 있다.

◆ Syntax

column datatype [CONSTRAINT constraint_name] REFERENCES table_name (column1[,column2,...] [ON DELETE CASCADE])
column datatype, , [CONSTRAINT constraint_name] FOREIGN KEY (column1[,column2,...]) REFERENCES table_name (column1[,column2,...] [ON DELETE CASCADE])

문제 6) 아래의 두 문장의 차이점을 설명하여라.

```
SQL> CREATE TABLE DEPT_TAB (
  2  DEPTNO      NUMBER(2),
  3  DNAME       CHAR(14),
  4  LOC         CHAR(13),
  5  CONSTRAINT DEPT_DEPTNO_PK PRIMARY KEY (DEPTNO);
```

Table created.

```
SQL> CREATE TABLE EMP_TAB (
  2  EMPNO       NUMBER(4),
  3  ENAME       VARCHAR2(10),
  4  JOB         VARCHAR2(9),
  5  MGR         NUMBER(4) CONSTRAINT EMP_SELF_KEY REFERENCES EMP (EMPNO),
  6  HIREDATE    DATE,
  7  SAL          NUMBER(7,2),
  8  COMM         NUMBER(7,2),
  9  DEPTNO      NUMBER(2) NOT NULL,
 10  CONSTRAINT EMP_DEPTNO_FK FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO),
 11  CONSTRAINT EMP_EMPNO_PK PRIMARY KEY (EMPNO));
```

Table created.

♣ 주의

FOREIGN KEY 값은 MASTER TABLE에서 존재하는 값과 일치해야 하거나 NULL이 되어야 한다. FOREIGN KEY 값은 데이터 값을 기초로 하여 순전히 논리적이지 물리적이거나 포인터가 아니다. MASTER TABLE(parent)은 참조 당하는 쪽(DEPT TABLE)을 테이블을 의미하고 DETAIL TABLE(child)은 참조하는 쪽(EMP TABLE)의 테이블을 의미한다.

☞ Guidelines

- 1) MASTER TABLE(참조 당하는 쪽)을 먼저 생성하여야 한다.
- 2) MASTER TABLE에 PRIMARY KEY 또는 UNIQUE KEY로 설정된 열을 DETAIL TABLE에서 참조하여야 한다.
- 3) MASTER TABLE과 DETAIL TABLE의 참조하는 열과 참조 당하는 쪽의 자료형과 크기가 일치해야 한다.

다) UNIQUE key(UK)

- 1) UNIQUE Key 무결성 제약 조건은 열 또는 열 집합의 모든 값들이 유일해야 한다.
- 2) 중복된 값을 가지는 행이 존재할 수 없음을 의미한다.
- 3) PRIMARY KEY와 유사하나 NULL을 허용한다.
- 4) 열이 하나 이상 포함되어 있다면 composite unique key 라 부릅니다.
- 5) UNIQUE Key에 대하여 UNIQUE INDEX가 자동 생성된다.

◆ Syntax

```
column datatype [CONSTRAINT constraint_name] UNIQUE  
column datatype,  
. . . . .,  
[CONSTRAINT constraint_name] UNIQUE (column1[,column2,...])
```

문제 7) 아래의 두 문장의 차이점을 설명하여라.

```
SQL> CREATE TABLE UNI_TAB1 (  
 2 DEPTNO          NUMBER(2) CONSTRAINT UNI_TAB_DEPTNO_UK UNIQUE,  
 3 DNAME           CHAR(14),  
 4 LOC             CHAR(13));
```

Table created.

```
SQL> CREATE TABLE UNI_TAB2 (  
 2 DEPTNO          NUMBER(2),  
 3 DNAME           CHAR(14),  
 4 LOC             CHAR(13),  
 5 CONSTRAINT UNI_TAB_DEPTNO_UK UNIQUE (DEPTNO));
```

Table created.

라) NOT NULL(NN)

- 1) NOT NULL 제약 조건은 열에서 NULL 을 허용하지 않도록 보증한다.
- 2) NOT NULL 제약 조건이 없는 열은 DEFAULT 로 NULL 을 허용한다.
- 3) NOT NULL 제약 조건은 COLUMN CONSTRAINT 에서만 기술 가능하다.

◆ Syntax

```
column datatype [CONSTRAINT constraint_name] NOT NULL  
column datatype,  
. . . . .,  
[CONSTRAINT constraint_name] NOT NULL (column1[,column2,...])
```

문제 8) 아래의 두 문장의 차이점을 설명하여라.

```
SQL> CREATE TABLE NN_TAB1 (  
 2 DEPTNO          NUMBER(2) CONSTRAINT UNI_TAB_DEPTNO_NN NOT NULL,  
 3 DNAME           CHAR(14),  
 4 LOC             CHAR(13));
```

Table created.

```

SQL> CREATE TABLE NN_TAB2 (
  2 DEPTNO          NUMBER(2),
  3 DNAME           CHAR(14),
  4 LOC              CHAR(13),
  5 CONSTRAINT UNI_TAB_DEPTNO_NN NOT NULL (DEPTNO));
CONSTRAINT UNI_TAB_DEPTNO_NN NOT NULL (DEPTNO))
*
ERROR at line 5:
ORA-00904: invalid column name

```

▷) CHECK(CK)

- 1) CHECK 제약 조건은 행이 만족해야 하는 조건을 정의한다.
- 2) 다음과 같은 표현식은 허용되지 않습니다.
 - ① CURRVAL, NEXTVAL, LEVEL, ROWNUM에 대한 참조
 - ② SYSDATE, UID, USER, USERENV 함수에 대한 호출
 - ③ 다른 행에 있는 값을 참조하는 질의
 - ④ ORACLE SERVER 가 사용하는 예약어

◆ Syntax

column datatype [CONSTRAINT constraint_name] CHECK (condition)
column datatype,, [CONSTRAINT constraint_name] CHECK (condition)

문제 9) 아래의 두 문장의 차이점을 설명하여라.

<pre> SQL> CREATE TABLE CK_TAB1 (2 DEPTNO NUMBER(2) 3 CONSTRAINT UNI_TAB_DEPTNO_CK CHECK (DEPTNO IN (10,20,30,40,50)), 4 DNAME CHAR(14), 5 LOC CHAR(13)); Table created. </pre>
<pre> SQL> CREATE TABLE CK_TAB2 (2 DEPTNO NUMBER(2), 3 DNAME CHAR(14), 4 LOC CHAR(13), 5 CONSTRAINT UNI_TAB_DEPTNO_CK CHECK (DEPTNO IN (10,20,30,40,50))); Table created. </pre>

1.3 테이블 차트에 의한 테이블 생성

1.3.1 테이블 차트

가) TABLE NAME : POST

Column name	POST1	POST2	ADDR
Key Type	PK		
Nulls/Unique			NN
Data Type	CHAR	CHAR	VARCHAR2
Maximun Length	3	3	60
Sample			경기도 성남시 분당구 정자동

나) TABLE NAME : MEMBER

Column name	ID	NAME	SEX	JUMIN1	JUMIN2	TEL	POST1	POST2	ADDR
Key Type	PK						FK		
Nulls/Unique		NN		UK					
FK Ref Table							POST		
FK Ref Column							POST1,POST2		
Check			1,2						
Data Type	NUM	VAR	CHAR	CHAR	CHAR	VAR	CHAR	CHAR	VAR
Maximun Length	4	10	1	6	7	15	3	3	60
Sample	1234	홍길동	1	990101	123234 4	712- 1234	100	010	

1.3.2 테이블 생성 문

가) TABLE NAME : POST

```
SQL> CREATE TABLE post(
  2      post1    CHAR(3),
  3      post2    CHAR(3),
  4      addr     VARCHAR2(60) CONSTRAINT post_addr_nn NOT NULL,
  5  CONSTRAINT post_post12_pk PRIMARY KEY (post1,post2));
```

```
Table created.
```

나) TABLE NAME : MEMBER

```
SQL> CREATE TABLE member(
  2      id      NUMBER(4) CONSTRAINT member_id_pk PRIMARY KEY,
  3      name    VARCHAR(10) CONSTRAINT member_name_nn NOT NULL,
  4      sex     CHAR(1) CONSTRAINT member_sex_ck CHECK ( sex IN ('1','2') ),
  5      jumin1  CHAR(6),
  6      jumin2  CHAR(7),
  7      tel     VARCHAR2(15),
  8      post1   CHAR(3),
  9      post2   CHAR(3),
 10      addr    VARCHAR2(60),
 11  CONSTRAINT member_jumin12_uk UNIQUE (jumin1,jumin2),
 12  CONSTRAINT member_post12_fk FOREIGN KEY (post1,post2)
 13      REFERENCES post (post1,post2));
```

```
Table created.
```

1.4 SUBQUERY 을 사용한 테이블 생성

테이블 생성시 이미 만들어져 있는 기존의 테이블을 이용하여 특정 열 또는 특정 행들만을 추출하여 사용자가 필요로 하는 새로운 테이블을 만들 수 있다.

1.4.1 SUBQUERY 을 이용하여 테이블 생성 방법

- 1) CREATE TABLE 문장과 AS SUBQUERY 를 사용하여 테이블을 생성하고 행을 삽입합니다.
- 2) SUBQUERY 의 열의 개수와 명시된 열의 개수를 좌측부터 일치시킨다.
- 3) 열 이름과 DEFAULT VALUE 를 가진 열을 정의한다.

1.4.2 Syntax

```
CREATE TABLE table_name [column1[,column2, . . . . .]]
AS subquery
```

table_name 테이블의 이름

column1 열 이름, DEFAULT VALUE, 무결성 제약 조건

subquery 새로운 테이블에 삽입할 행의 집합을 정의한 SELECT 문장

☞ Guidelines

- 1) 테이블은 명시된 열 이름으로 생성, SQL 문장에 의해 RETURN 된 행들이 테이블에 삽입.
- 2) 열 정의는 오직 열 이름과 DEFAULT VALUE 만 정의 가능
- 3) 열이 기술되었다면 열의 수는 SUBQUERY의 열과 좌측부터 1 대 1 대응
- 4) 열이 기술되지 않았다면 테이블의 열 이름은 SUBQUERY의 열 이름과 동일
- 5) SUBQUERY 에서 계산식이나 함수를 사용하면 계산식과 함수를 열 이름으로 사용할 수 없기 때문에 반드시 Alias 을 지정하거나 table_name 옆에 열 이름을 기술하여야 한다.

문제 10) EMP 테이블에서 30 부서에 근무하는 사원의 정보만 추출하여 EMP_30 테이블을 생성하여라. 단 열은 사원번호, 이름, 업무, 입사일자, 급여, 보너스를 포함한다,

```
SQL> CREATE TABLE emp_30
  2 AS SELECT empno,ename,job,hiredate,sal,comm
  3 FROM emp
  4 WHERE deptno = 30;

Table created.

SQL> SELECT *
  2 FROM emp_30;

  EMPNO ENAME      JOB          HIREDATE        SAL       COMM
----- -----
    7698 BLAKE     MANAGER   01-MAY-81      2850
    7654 MARTIN   SALESMAN  28-SEP-81      1250      1400
    . . .
6 rows selected.
```

문제 11) EMP 테이블에서 부서별로 인원수, 평균 급여, 급여의 합, 최소 급여, 최대 급여를 포함하는 EMP_DEPTNO 테이블을 생성하여라.

```
SQL> CREATE TABLE emp_deptno
  2 AS SELECT deptno,COUNT(*),AVG(sal),SUM(sal),MIN(sal),MAX(sal)
  3 FROM emp
  4 GROUP BY deptno;
AS SELECT deptno,COUNT(*),AVG(sal),SUM(sal),MIN(sal),MAX(sal)
*
ERROR at line 2:
ORA-00998: must name this expression with a column alias

SQL> CREATE TABLE emp_deptno (deptno,e_count,e_avg,e_sum,e_min,e_max)
  2 AS SELECT deptno,COUNT(*),AVG(sal),SUM(sal),MIN(sal),MAX(sal)
  3 FROM emp
  4 GROUP BY deptno;

Table created.

SQL> SELECT *
  2 FROM emp_deptno;

  DEPTNO E_COUNT      E_AVG      E_SUM      E_MIN      E_MAX
----- -----
    10         3 2983.3333      8950      1500      5000
    20         5    2175      10875      800      3000
    30         6 1566.6667      9400      950      2850
```

문제 12) EMP 테이블에서 사원번호, 이름, 업무, 입사일자, 부서번호만 포함하는 EMP_TEMP 테이블을 생성하는데 자료는 포함하지 않고 구조만 생성하여라.

```
SQL> CREATE TABLE emp_temp  
2 AS SELECT empno,ename,job,hiredate,deptno  
3 FROM emp  
4 WHERE 1 = 2;
```

Table created.

```
SQL> SELECT *  
2 FROM emp_temp;  
  
no rows selected
```

1.5 데이터 사전(DATA DICTIONARY) 질의

사용자가 소유한 다양한 데이터베이스 객체를 보기 위해서 데이터 사전을 질의하여 알 수 있다.

- 1) DDL 문장을 실행하면 그 정보는 데이터 사전(DATA DICTIONARY)에 등록
- 2) 사용자가 소유한 테이블을 조회

문제 13) 현재 SESSION을 이루고 있는 사용자가 소유하고 있는 TABLE을 조회하여라.

```
SQL> SELECT table_name,tablespace_name  
2 FROM user_tables;  
  
TABLE_NAME          TABLESPACE_NAME  
-----  
BONUS                USER_DATA  
CUSTOMER              USER_DATA  
.....  
15 rows selected.
```

문제 14) 현재 SESSION을 이루고 있는 사용자가 소유한 모든 객체를 조회하여라

```
SQL> COL object_name FORMAT a20  
SQL> COL timestamp FORMAT a25  
SQL> SELECT object_name,object_type,timestamp  
2 FROM user_objects;  
  
OBJECT_NAME      OBJECT_TYPE      TIMESTAMP  
-----  
BONUS            TABLE          1999-02-11:17:21:32  
CUSTID           SEQUENCE       1999-02-11:17:21:41  
.....  
29 rows selected.
```

문제 15) 현재 SESSION 을 이루고 있는 사용자가 소유한 테이블, 뷰, 동의어, 시퀀스를 조회하여라

```
SQL> SELECT *
  2  FROM user_catalog;

TABLE_NAME          TABLE_TYPE
-----
BONUS                TABLE
CUSTID               SEQUENCE
CUSTOMER              TABLE
.
.
.
19 rows selected.
```

2. 테이블을 수정

테이블을 생성한 이후에 열이 생략 되었거나, 열 정의를 변경할 필요가 있을 수 있다. 테이블의 구조를 변경할 경우 ALTER TABLE 명을 사용하여 변경 한다.

2.1 새로운 열 추가

- 1) 새로운 열을 추가 할 수는 있지만 테이블에 있는 기존의 열은 DROP 할 수 없다.
- 2) 열이 위치를 기술할 수 없으며 항상 테이블에서 마지막에 위치 합니다.
- 3) 열을 추가할 때 테이블이 행을 포함하고 있다면 새로운 열은 이미 존재하는 열을 NULL로 초기화 한다.

2.1.1 Syntax

```
ALTER TABLE table_name
ADD (column datatype [DEFAULT expr]
     [,column datatype [DEFAULT expr]] . . . . .)
```

문제 16) BONUS 테이블에 ETC COLUMN 을 추가하여라. 단 자료형은 VARCHAR2(50) 사용하여라.

```
SQL> ALTER TABLE bonus
  2 ADD (etc VARCHAR2(50));

Table altered.
SQL> desc bonus
Name           Null?    Type
-----
ENAME          CHAR(10)
JOB            CHAR(9)
SAL             NUMBER
COMM            NUMBER
ETC             VARCHAR2(50)
```

문제 17) EMP_30 테이블에 DEPTNO NUMBER(2)을 추가하여라.

```
SQL> ALTER TABLE emp_30  
2 ADD (deptno number(2));
```

Table altered.

```
SQL> SELECT *  
2 FROM emp_30;
```

EMPNO	ENAME	JOB	HIREDATE	SAL	COMM
7698	BLAKE	MANAGER	01-MAY-81	2850	
7654	MARTIN	SALESMAN	28-SEP-81	1250	1400
7499	ALLEN	SALESMAN	20-FEB-81	1600	300
7844	TURNER	SALESMAN	08-SEP-81	1500	0
7900	JAMES	CLERK	03-DEC-81	2450	
7521	WARD	SALESMAN	22-FEB-81	1250	500

6 rows selected.

```
SQL> ALTER TABLE emp_30  
2 ADD (deptno number(2));  
ALTER TABLE emp_30  
*  
ERROR at line 1:  
ORA-00942: table or view does not exist
```

위와 같이 ERROR 가 발생되면 EMP_30 테이블이 존재하지 않는 경우입니다. 이전 문제 (EMP_30 테이블을 생성)를 수행 후 다시 한번 위 예를 실행하여 보시오

2.2 열 수정

ALTER TABLE 의 MODIFY 절을 사용하여 열의 정의를 수정할 수 있습니다. 열의 수정은 열의 자료형, 크기, DEFAULT VALUE 입니다.

2.2.1 Syntax

```
ALTER TABLE table_name  
MODIFY (column datatype [DEFAULT expr]  
[,column datatype [DEFAULT expr]] . . . . .)
```

문제 18) 아래 구문들을 설명하여라.

```
SQL> DESC emp_30
```

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)

```
SQL> ALTER TABLE emp_30
```

```
2 MODIFY (ename VARCHAR2(15));
```

Table altered.

```
SQL> DESC emp_30
```

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(15)
JOB		VARCHAR2(9)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)

```
SQL> ALTER TABLE emp_30
```

```
2 MODIFY (empno CHAR(4));
```

```
MODIFY (empno CHAR(4))
```

```
*
```

```
ERROR at line 2:
```

```
ORA-01439: column to be modified must be empty to change datatype
```

```
SQL> ALTER TABLE emp_30
```

```
2 MODIFY (JOB char(9));
```

Table altered.

```
SQL> DESC emp_30;
```

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(15)
JOB		CHAR(9)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)

```
SQL> DESC emp_30
```

Object does not exist.

위와 같이 MESSAGE 가 발생되면 EMP_30 테이블이 존재하지 않는 경우입니다. 이전 문제(EMP_30 테이블을 생성)를 수행 후 다시 한번 위 예를 실행하여 보시오

☞ Guidelines

- 1) 숫자 열의 정밀도나 폭을 증가할 수 있다.
- 2) 열이 모두 NULL 이거나 테이블에 자료가 없으면 열의 폭을 감소시킬 수 있다.
- 3) 열이 NULL 을 포함하면 열의 자료형을 변경할 수 있다.
- 4) 열이 NULL 을 포함하거나 크기를 변경하지 않으면 CHAR 을 VARCHAR2 로 변경하거나 그 반대의 경우도 가능하다.
- 5) 열의 DEFAULT VALUE 를 변경하는 것은 이후의 INSERT 문장에만 영향을 미칩니다.

2.3 제약 조건 추가

ADD 절을 가지는 ALTER TABLE 문장을 사용하여 기존의 테이블에 대한 제약 조건을 추가할 수 있다.

2.3.1 Syntax

```
ALTER TABLE table_name  
ADD [CONSTRAINT constraint_name] constraint_type (column);
```

제약 조건의 선택 사항이지만 기술하기를 권장한다. 기술하지 않을 경우는 ORACLE SERVER ID 생성(SYS_Cnnnnn)하여 부여한다.

문제 19) EMP 테이블에서 이름 필드에 UNIQUE CONSTRAINT 를 설정하고 DATA DICTIONARY 에서 확인 하여라.

```
SQL> ALTER TABLE emp  
  2 ADD CONSTRAINT emp_ename_pk UNIQUE (ename);  
  
Table altered.  
SQL> SELECT constraint_name,table_name,status  
  2 FROM user_constraints;  
  
CONSTRAINT_NAME          TABLE_NAME        STATUS  
-----  
...  
EMP_PRIMARY_KEY           EMP            ENABLED  
EMP_SELF_KEY              EMP            ENABLED  
EMP_FOREIGN_KEY           EMP            ENABLED  
EMP_ENAME_UK              EMP            ENABLED  
...  
33 rows selected.
```

☞ Guidelines

- 1) 제약 조건의 추가, 삭제는 가능하지만 변경은 불가능하다.
- 2) 제약 조건의 활성화 또는 비활성화가 가능하다.
- 3) MODIFY 절을 사용하여 NOT NULL 제약 조건을 추가한다.

♣ 주의

데이터는 열이 추가되는 시점에서 기존의 열에 대해 명시될 수 없기 때문에 테이블에 행이 하나도 없을 경우에만 NOT NULL 열을 정의할 수 있다.

2.4 제약 조건 삭제

DROP 절을 가지는 ALTER TABLE 문장을 사용하여 기존의 테이블에 대한 제약 조건을 삭제할 수 있다.

2.4.1 Syntax

```
ALTER TABLE table_name  
DROP PRIMARY KEY | UNIQUE (column) | CONSTRAINT constraint_name [CASCADE];
```

♣ 참고

제약 조건을 삭제하면 USER_CONSTRAINTS, USER_CONS_COLUMNS 데이터 사전 뷰에서 제약 조건 이름을 삭제한다. DROP의 CASCADE 문장은 모든 종속적인 제약 조건이 삭제됩니다.

문제 20) 전 문제에서 생성한 EMP 테이블에 있는 emp_ename_uk 을 삭제하고 DATA DICTIONARY에서 확인하여라.

```
SQL> ALTER TABLE emp  
2  DROP CONSTRAINT emp_ename_uk;  
  
Table altered.  
  
SQL> SELECT constraint_name,table_name,status  
2  FROM user_constraints;  
  
CONSTRAINT_NAME          TABLE_NAME        STATUS  
-----  
...  
EMP_PRIMARY_KEY           EMP             ENABLED  
EMP_SELF_KEY              EMP             ENABLED  
EMP_FOREIGN_KEY           EMP             ENABLED  
...  
32 rows selected.
```

♣ 참고

무결성 제약 조건을 삭제할 때, 그 제약 조건은 더 이상 ORACLE SERVER 에 의해 적용되지 않으며, 데이터 사전에서 확인할 수 없다.

2.5 제약 조건 비활성화

DISABLE 절을 가지는 ALTER TABLE 문장을 사용하여 삭제 또는 재생성 없이 제약 조건을 비활성화할 수 있다.

☞ Guidelines

- 1) 무결성 제약 조건을 비활성화 하기 위하여 ALTER TABLE 문장을 사용하여 DISABLE 할 수 있다.
- 2) 종속적인 무결성 제약 조건을 비활성화 하기 위하여 CASCADE를 사용한다.
- 3) CREATE TABLE 문장과 ALTER TABLE 문장으로 DISABLE 할 수 있다.

2.5.1 Syntax

```
ALTER TABLE table_name  
DISABLE CONSTRAINT constraint_name [CASCADE];
```

문제 21) EMP 테이블에 있는 PRIMARY KEY(EMP_PRIMARY_KEY)를 DISABLE 하여라.

```
SQL> SELECT constraint_name,table_name,status  
2 FROM user_constraints;  
  
CONSTRAINT_NAME          TABLE_NAME        STATUS  
-----  
...  
EMP_PRIMARY_KEY           EMP             ENABLED  
EMP_SELF_KEY              EMP             ENABLED  
EMP_FOREIGN_KEY           EMP             ENABLED  
...  
32 rows selected.  
  
SQL> ALTER TABLE emp  
2 DISABLE CONSTRAINT emp_primary_key;  
ALTER TABLE emp  
*  
ERROR at line 1:  
ORA-02297:  cannot disable constraint (SCOTT.EMP_PRIMARY_KEY) - dependencies exist  
  
SQL> ALTER TABLE emp  
2 DISABLE CONSTRAINT emp_primary_key CASCADE;  
  
Table altered.  
  
SQL> SELECT constraint_name,table_name,status  
2 FROM user_constraints;  
  
CONSTRAINT_NAME          TABLE_NAME        STATUS
```

EMP_PRIMARY_KEY	EMP	DISABLED
EMP_SELF_KEY	EMP	DISABLED
EMP_FOREIGN_KEY	EMP	ENABLED
.....		
32 rows selected.		

2.6 제약 조건 활성화

ENABLE 절을 가지는 ALTER TABLE 문장을 사용하여 삭제 또는 재생성 없이 제약 조건을 활성화할 수 있다.

☞ Guidelines

- 1) 제약 조건이 활성화 된다면 그 제약 조건은 테이블의 모든 데이터에 적용된다. 테이블의 모든 자료는 데이터의 제약 조건과 일치해야 한다.
- 2) UNIQUE key 와 PRIMARY key 는 제약 조건이 활성화 된다면 UNIQUE INDEX 가 자동 생성된다.
- 3) CREATE TABLE 문장과 ALTER TABLE 문장으로 ENABLE 할 수 있다.

2.6.1 Syntax

```
ALTER TABLE table_name
ENABLE CONSTRAINT constraint_name;
```

문제 22) EMP 테이블에 있는 PRIMARY KEY(EMP_PRIMARY_KEY), FOREIGN KEY(EMP_SELF_KEY)를 ENABLE 하여라.

CONSTRAINT_NAME	TABLE_NAME	STATUS
.....		
EMP_PRIMARY_KEY	EMP	DISABLED
EMP_SELF_KEY	EMP	DISABLED
EMP_FOREIGN_KEY	EMP	ENABLED
.....		
32 rows selected.		

```
SQL> ALTER TABLE emp
  2  ENABLE CONSTRAINT EMP_PRIMARY_KEY;
```

Table altered.

```

SQL> SELECT constraint_name,table_name,status
2  FROM user_constraints;

CONSTRAINT_NAME          TABLE_NAME      STATUS
-----  -----
.
.
.
EMP_PRIMARY_KEY          EMP            ENABLED
EMP_SELF_KEY              EMP            DISABLED
EMP_FOREIGN_KEY           EMP            ENABLED
.
.
.
32 rows selected.

SQL> ALTER TABLE emp
2  ENABLE CONSTRAINT emp_self_key;

Table altered.

SQL> SELECT constraint_name,table_name,status
2  FROM user_constraints;

CONSTRAINT_NAME          TABLE_NAME      STATUS
-----  -----
.
.
.
EMP_PRIMARY_KEY          EMP            ENABLED
EMP_SELF_KEY              EMP            ENABLED
EMP_FOREIGN_KEY           EMP            ENABLED
.
.
.
32 rows selected.

```

2.7 제약 조건 조회

테이블 소유자가 소유자 이름을 붙이지 않는 제약 조건은 시스템이 이름을 부여한다. 제약 조건 유형에서 C 는 CHECK, P 는 PRIMARY KEY, R 은 REFERENCE, U 는 UNIQUE 를 담당하고 NOT NULL 은 CHECK 가 담당한다.

문제 23) EMP 테이블에 있는 각종 제약 조건을 조회하여라.

```

SQL> COL search_condition FORMAT a30
SQL> SELECT constraint_name,constraint_type,search_condition
2  FROM user_constraints
3  WHERE table_name = 'EMP';

CONSTRAINT_NAME          C SEARCH_CONDITION
-----  -----
SYS_C00613                C EMPNO IS NOT NULL
SYS_C00614                C DEPTNO IS NOT NULL
EMP_PRIMARY_KEY            P
EMP_SELF_KEY               R
EMP_FOREIGN_KEY            R

```

```

SQL> SELECT constraint_name, column_name
  2  FROM user_cons_columns
  3 WHERE table_name = 'EMP';

CONSTRAINT_NAME          COLUMN_NAME
-----
EMP_FOREIGN_KEY          DEPTNO
EMP_PRIMARY_KEY           EMPNO
EMP_SELF_KEY               MGR
SYS_C00613                  EMPNO
SYS_C00614                  DEPTNO

```

2.8 객체 이름 변경

테이블, 뷰, 시퀀스, 동의어를 변경하기 위해 RENAME 문장을 실행 합니다. 단 객체 소유자 이어야 합니다.

2.8.1) Syntax

```
RENAME old_name TO new_name;
```

문제 24) 이전에 생성한 EMP_30 테이블의 이름을 EMP_TEMP30 으로 변경하여라.

```
SQL> RENAME emp_30 TO emp_temp30;
```

```
Table renamed.
```

2.9 TRUNCATE TABLE 문장

테이블의 OWNER 이거나 DELETE TABLE 권한을 가진 사용자가 테이블의 모든 행을 삭제(구조는 삭제되지 않는다) 하고 사용하고 있던 기억 공간을 모두 해제할 경우에 사용합니다. 삭제된 행은 복구(ROLLBACK)할 수 없습니다.

2.9.1 Syntax

```
TRUNCATE TABLE table_name;
```

문제 25) 이전에 생성한 EMP_TEMP30 테이블의 모든 자료를 삭제하고 사용하고 있던 기억 공간을 모두 해제하여라.

```
SQL> TRUNCATE TABLE emp_temp30;
```

```
Table truncated.
```

♣ 참고

`DELETE` 문장은 테이블의 모든 행을 삭제할 수 있지만, 저장 공간을 해제할 수 없습니다.

2.10 테이블에 주석문 추가

`COMMENT` 문장을 사용하여 열, 테이블, 뷰, 스냅샷에 대하여 2000Byte 까지 주석을 추가할 수 있다. 주석은 데이터 사전 VIEW(`All_col_comments`, `User_col_comments`, `All_tab_comments`, `User_tab_comments`)를 통하여 볼 수 있다.

2.10.1) Syntax

```
COMMENT ON TABLE table_name | COLUMN table.column IS 'text';
```

문제 26) EMP 테이블에 “Employee Information”라는 주석을 추가하여라.

```
SQL> COMMENT ON TABLE emp
  2  IS 'Employee Information';

Comment created.

SQL> COL comments FORMAT a30
SQL> SELECT *
  2  FROM user_tab_comments;

TABLE_NAME          TABLE_TYPE   COMMENTS
-----
.
.
.
DUMMY               TABLE
EMP                 TABLE      Employee Information
EMP_DEPTNO          TABLE
.
.
.
15 rows selected.
```

3. 테이블 삭제

`DROP TABLE` 문장은 Oracle8 테이블의 정의를 삭제 합니다. 테이블을 삭제할 때 데이터베이스는 테이블에 있는 모든 자료와 그와 연관된 모든 INDEX를 DROP하고 사용하고 있던 공간을 돌려줍니다.

☞ Guidelines

- 1) 테이블의 모든 구조와 데이터가 삭제 됩니다.
- 2) DDL 문장이기 때문에 TRANSACTION이 COMMIT됩니다.
- 3) 모든 인덱스가 삭제 됩니다.
- 4) VIEW나 SYNONYM은 남지만 사용시 ERROR가 발생합니다.
- 5) 테이블의 OWNER나 DROP ANY TABLE 권한을 가진 사용자만이 테이블을 삭제 할 수 있습니다.

3.1 Syntax

```
DROP TABLE table_name [CASCADE CONSTRAINT]
```

♣ 주의

일단 실행된 DROP TABLE 문장은 복구(ROLLBACK)할 수 없습니다. ORACLE SERVER는 DROP TABLE 문장을 실행할 때 삭제 여부를 질문하지 않습니다.

문제 27) EMP_TEMP30의 테이블을 삭제하여라.

```
SQL> DROP TABLE emp_30;
```

```
Table dropped.
```

문제 28) DEPT 테이블을 삭제하여라.

```
SQL> DROP TABLE dept;
```

```
DROP TABLE dept
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02266: unique/primary keys in table referenced by enabled foreign keys
```

```
SQL> DROP TABLE dept CASCADE CONSTRAINT;
```

```
Table dropped.
```

```
SQL> @c:\Worawin95\ dbs\demobld
```

♣ 참고

SQL> @c:\Worawin95\ dbs\demobld 은 DEMO SCRIPT 가 있는 절대 패스를 사용하여 데모 테이블을 다시 생성합니다.

◆ 연습문제 ◆

1. EMP 테이블에 있는 모든 CONSTRAINT 를 조회하는 SELECT 문을 작성하여라.
2. EMP 테이블에 SAL, COMM 을 제외한 모든 COLUMN 과 행을 포함하는 EMP_DEMO 테이블을 생성하는 SQL 문을 작성하여라.
3. EMP 테이블과 DEPT 테이블을 이용하여 아래의 내용을 포함하는 테이블(EMP_DEPT)을 생성하여라.

EMPNO	ENAME	JOB	DNAME	LOC
7839	KING	PRESIDENT	ACCOUNTING	NEW YORK
7698	BLAKE	MANAGER	SALES	CHICAGO
7782	CLARK	MANAGER	ACCOUNTING	NEW YORK
.				
14 rows selected.				

4. EMP 테이블과 SALGRADE 테이블을 이용하여 아래의 내용을 포함하는 테이블(EMP_GRADE)을 생성하여라.

EMPNO	ENAME	JOB	SAL	COMM	GRADE
7839	KING	PRESIDENT	5000		5
7698	BLAKE	MANAGER	2850		4
7782	CLARK	MANAGER	2450		4
7566	JONES	MANAGER	2975		4
.					
14 rows selected.					

5. DEPT 테이블의 PRIMARY KEY 를 DISABLE 하는 SQL 문을 작성하여라.
6. 3 번에서 생성한 테이블에 EMPNO 를 PRIMARY KEY 로 설정하는 SQL 문을 작성하여라.
7. 4 번에서 생성한 테이블에 SAL 의 정밀도를 정수 부분을 12 자리 소수 이하 4 자리를 기억할 수 있도록 변경하는 SQL 문을 작성하여라.
8. 3 번에서 작성한 테이블의 내용과 기억 장소를 모두 해제하는 SQL 문을 작성하여라.
9. 3 번과 4 번에서 생성한 테이블을 모두 삭제하는 SQL 문을 작성하여라.

1. 데이터(DML) 조작어

DML(Data Manipulation Language) 명령은 데이터를 입력, 수정, 삭제하는 SQL 명령어이다. 데이터베이스에 영구적으로 반영되지 않은 데이터 조작 명령어들을 TRANSACTION 이라고 하며 오라클에서는 이를 하나의 논리적 작업 단위로 사용한다.

명령어	설명
INSERT	테이블에 새로운 행 추가
UPDATE	테이블의 행 내용을 변경
DELETE	테이블의 행 삭제
COMMIT	저장되지 않은 모든 변경 사항을 Database에 저장
SAVEPOINT	savepoint 설정
ROLLBACK	저장되지 않은 모든 변경 사항을 취소

1.1 INSERT 문장

테이블에 사용하여 테이블에 새로운 행을 삽입(INSERT)할 수 있다.

1.1.1 Syntax

```
INSERT INTO table_name [(column1[, column2, . . . . .])]  
VALUES (value1[, value2, . . . . .]);
```

1.1.2 사용 예

- ◆ 모든 column에 대해 값을 갖는 새로운 행을 삽입한다.

```
SQL> DESC emp  
Name           Null?    Type  
-----  
EMPNO          NOT NULL NUMBER(4)  
ENAME           VARCHAR2(10)  
JOB             VARCHAR2(9)  
MGR             NUMBER(4)  
HIREDATE       DATE  
SAL             NUMBER(7,2)  
COMM            NUMBER(7,2)  
DEPTNO          NOT NULL NUMBER(2)  
  
SQL> INSERT INTO emp  
2  VALUES (1111,'YOUNJB','','NULL',SYSDATE,3000,NULL,10);  
1 row created.
```

- ◆ INSERT 절의 column 은 선택적으로 기입할 수 있다. 이럴 경우 열중 NOT NULL 제약 조건이 있는 열은 반드시 포함하여야 한다.

```
SQL> INSERT INTO emp(empno,ename,hiredate,deptno)
  2 VALUES (2222,'홍길동',SYSDATE,10);
1 row created.
```

- ◆ 문자와 날짜 값은 단일 따옴표('')를 둘러싼다.

```
SQL> INSERT INTO emp(empno,ename,job,hiredate,deptno)
  2 VALUES (3333,'HONGKD','SALESMAN',
  3 to_date('19990215213812','YYYYMMDDHH24MISS'),10);
1 row created.

SQL> INSERT INTO emp(empno,ename,job,hiredate,deptno)
  2 VALUES (4444,'JBY','ANALYST','13-FEB-99',20);
1 row created.
```

☞ Guidelines

- 1) VALUES 절을 가지는 INSERT 문장은 한번에 오직 하나의 행만을 추가한다.
- 2) 모든 행에 값을 갖는 새로운 행을 삽입할 수 있기 때문에 column list 는 INSERT 절에 필요하지 않다. 하지만 테이블에 정의된 순서에 따라 값을 나열해야 한다.
- 3) 명확성을 위해 INSERT 절에 column list 를 사용하면 좋다.(천장)
- 4) 문자와 날짜는 단일 따옴표 안에 쓰나, 수치 같은 사용하지 않는다.

1.1.3 NULL 값을 새로운 행에 추가

자료형에 관계없이 사용 가능하다.

가) 암시적 방법

```
SQL> INSERT INTO dept(deptno,dname)
  2 VALUES (50,'DEVELOPMENT');
1 row created.
```

나) 명시적 방법

```
SQL> INSERT INTO dept
  2 VALUES(60,'',NULL);
1 row created.
```

1.1.4 특정 날짜 값 삽입

형식 DD-MON-YY 는 항상 날짜 값을 입력할 때 사용한다. 이 형식은 현재 세기에 대한 DEFAULT 세기를 다시 호출한다. 또한 날짜가 시간 정보를 포함하므로 DEFAULT 시간은 자정 (00:00:00)이다. 날짜를 다른 세기로 입력하거나 또는 특정 시간을 요구 한다면 TO_DATE 함수를 사용하여라.

예) 아래의 INSERT 문장은 날짜를 입력하는 방법이다.

INSERT 명령	입력된 입사일자의 결과
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (5555,TO_DATE('1999','YYYY'),30);	1999/02/01 00:00:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (6666,TO_DATE('99','YY'),20);	1999/02/01 00:00:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (7777,TO_DATE('02','MM'),20);	1999/02/01 00:00:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (8888,TO_DATE('13','DD'),30);	1999/02/13 00:00:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (9999,TO_DATE('10','HH24'),20);	1999/02/01 10:00:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (1122,TO_DATE('10','MI'),20);	1999/02/01 00:10:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (2211,TO_DATE('10','SS'),20);	1999/02/01 00:00:10
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (1133,TO_DATE('JUN 3,99','MON DD,YY'),10);	1999/06/03 00:00:00
SQL> INSERT INTO emp(empno,hiredate,deptno) 2 VALUES (3311,TO_DATE('2000/02/01 17:35:10', 3 'YYYY/MM/DD HH24:MI:SS'),20);	2000/02/01 17:35:10

1.1.5 치환 변수를 사용하여 값 입력

SQL*Plus 의 치환 변수를 사용하여 사용자가 상호 작용으로 값을 추가할 수 있다. 날짜와 문자 값은 단일 인용 부호를 사용하여 감싸는 것이 사용하기에 편하고 SCRIPT 로 작성하여 사용하면 입력시 매우 유용하다.

예) SCRIPT 안에서 치환 변수 사용하는 방법이다.

```
SQL> ed dept_insert
```

```
dept_insert SCRIPT 안에 다음의 명령어를 작성하여 저장하고 실행 한다.
```

```
INSERT INTO dept (deptno,dname,loc)
VALUES (&department_id,'&department_name','&location');
```

```
SQL> @dept_insert
```

```
Enter value for department_id: 70
```

```
Enter value for department_name: EDUCATION
```

```
Enter value for location: ATLANTA
```

```
old  2: VALUES (&department_id,'&department_name','&location')
```

```
new  2: VALUES (70,'EDUCATION','ATLANTA')
```

```
1 row created.
```

1.1.6 다른 테이블로부터 행 복사

기존의 테이블로부터 값을 가져와 테이블에 추가하기 위해서 INSERT 문장을 사용할 수 있다. 즉 VALUES 절에서 subquery를 사용할 수 있다.

가) Syntax

```
INSERT INTO table_name [column1[,column2, . . . .]] subquery;
```

문제 1) EMP 테이블에서 EMPNO,ENAME,SAL,HIREDATE 의 COLUMN 만 선택하여 EMP_10 테이블을 생성한 후 10 번 부서만 선택하여 이에 대응하는 값을 EMP_10 테이블에 입력하여라.

```
SQL> CREATE TABLE emp_10(id,name,salary,hiredate)
  2 AS SELECT empno,ename,sal,hiredate
  3 FROM emp
  4 WHERE 1 = 2;
```

```
Table created.
```

```
SQL> INSERT INTO emp_10
  2 SELECT empno,ename,sal,hiredate
  3 FROM emp
  4 WHERE deptno = 10;
```

```
7 rows created.
```

♣ 참고

INSERT 절의 열의 개수와 서브쿼리의 열의 개수가 좌측부터 1 대 1 대응하여 자료형과 길이가 같아야 한다.

1.1.7 INSERT 시 무결성 제약 조건의 오류

무결성 제약 조건(5 가지)이 위배되면 INSERT 시 ERROR 가 발생합니다.

```
SQL> INSERT INTO emp
  2 VALUES (7788,'YOUNG','MANAGER',NULL,SYSDATE,NULL,NULL,10);
INSERT INTO emp
*
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.EMP_PRIMARY_KEY) violated

SQL> INSERT INTO emp
  2 VALUES (1144,'YOUNG','MANAGER',NULL,SYSDATE,NULL,NULL,91);
INSERT INTO emp
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.EMP_FOREIGN_KEY) violated - parent key
not found

SQL> INSERT INTO emp
  2 VALUES (1144,'YOUNG','MANAGER',1234,SYSDATE,NULL,NULL,10);
INSERT INTO emp
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.EMP_SELF_KEY) violated - parent key not
found

SQL> INSERT INTO emp (empno,ename,job)
  2 VALUES (1144,'YOUNG','MANAGER');
INSERT INTO emp (empno,ename,job)
*
ERROR at line 1:
ORA-01400: mandatory (NOT NULL) column is missing or NULL during insert
```

1.2 UPDATE 문장

- 1) UPDATE 문장으로 기존의 행을 갱신합니다.
- 2) 일반적으로 단일 행을 식별하기 위해서 기본 키(primary key)를 사용합니다. 다른 열을 사용하면 원하지 않는 여러 행이 갱신될 수 있습니다.
- 3) 특정 열이나 행은 WHERE 절을 이용하여 갱신할 수 있다.

1.2.1 Syntax

UPDATE	table_name
SET	column1 = value1 [,column2 = value2,]
[WHERE	condition];

♣ 참고

DEMO TABLE 를 새로 생성하여 실습을 하자

```
SQL> @c:\Worawin95\ dbs\demobld
```

문제 2) EMP 테이블에서 사원 번호가 7788 인 사원의 부서를 10 번으로 변경하여라.

```
SQL> UPDATE emp  
  2 SET deptno = 10  
  3 WHERE empno = 7788;  
  
1 row updated.
```

문제 3) EMP 테이블에서 사원 번호가 7788 인 사원의 부서를 20, 급여를 3500 으로 변경하여라.

```
SQL> UPDATE emp  
  2 SET deptno = 20, sal = 3500  
  3 WHERE empno = 7788;  
  
1 row updated.
```

문제 4) EMP 테이블에서 부서를 모두 10 으로 변경하여라.

```
SQL> UPDATE emp  
  2 SET deptno = 10;  
  
14 rows updated.
```

1.2.2 다중 열 SUBQUERY 로 갱신

다중 열 SUBQUERY 는 UPDATE 문장의 SET 절로 구현할 수 있다.

가) Syntax

```
UPDATE      table_name  
SET         (column1, column2, . . . ) =  
           ( SELECT      column1, column2, . . .  
             FROM       table_name  
             WHERE      condition)  
[WHERE      condition];
```

문제 4) EMP 테이블에서 SCOTT 의 업무와 급여가 일치하도록 JONES 업무와 급여를 개인하여 라

```
SQL> UPDATE emp
  2 SET (job,sal) = (SELECT job,sal
  3   FROM emp
  4 WHERE ename = 'SCOTT')
  5 WHERE ename = 'JONES';

1 row updated.
```

1.2.3 다른 테이블을 근거로 한 행 갱신

다른 테이블의 값을 근거로 하는 테이블에서 행을 갱신하기 위해 UPDATE 문장에서 SUBQUERY를 사용한다.

문제 4) EMP 테이블을 근거로 EMPLOYEE 테이블을 생성한 후 7902 의 업무에 해당하는 사원의 부서번호를 7902 의 부서번호로 갱신하여라.

```
SQL> @c:\orawin95\ dbs\demobld

SQL> UPDATE employee
  2 SET deptno = (SELECT deptno
  3   FROM emp
  4 WHERE empno = 7902)
  5 WHERE job = (SELECT job
  6   FROM emp
  7 WHERE empno = 7902);

2 rows updated.
```

1.2.4 UPDATE 시 무결성 제약 조건 ERROR

무결성 제약 조건이 위배되는 값으로 UPDATE 할 경우 ERROR 가 발생한다.

문제 5) EMP 테이블에서 10 번 부서의 사원을 모두 91 번 부서로 갱신하여라.

```
SQL> UPDATE emp
  2 SET deptno = 91
  3 WHERE deptno = 10;
UPDATE emp
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.EMP_FOREIGN_KEY) violated - parent key
not found
```

문제 6) DEPT 테이블에서 부서 번호 10 을 15 로 갱신하여라.

```
SQL> UPDATE dept
  2  SET deptno = 15
  3 WHERE deptno =10;
UPDATE dept
      *
ERROR at line 1:
ORA-02292: integrity constraint (SCOTT.EMP_FOREIGN_KEY) violated - child record
found
```

1.3 DELETE 문장

- 1) DELETE 문장을 사용하여 테이블로부터 기존의 자료를 삭제할 수 있다.
- 2) WHERE 절을 명시하여 특정 행이나 행들을 삭제할 수 있다.
- 3) WHERE 절을 생략하면 테이블의 모든 행이 삭제 된다.

1.3.1 Syntax

```
DELETE [FROM] table_name
[WHERE] condition;
```

문제 7) EMP 테이블에서 사원번호가 7499 인 사원의 정보를 삭제하여라.

```
SQL> DELETE emp
  2 WHERE empno = 7499;
1 row deleted.
```

문제 8) EMP 테이블에서 입사일자가 83 년인 사원의 정보를 삭제하여라.

```
SQL> DELETE emp
  2 WHERE TO_CHAR(hiredate,'YY') = '83';
1 row deleted.
```

1.3.2 다른 테이블을 근거로 한 행 삭제

다른 테이블을 값을 근거로 행을 삭제하기 위해서는 SUBQUERY 를 사용하여야 한다.

문제 9) EMP 테이블의 자료 중 부서명이 'SALES' 인 사원의 정보를 삭제하여라.

```
SQL> DELETE emp
  2 WHERE deptno = (SELECT deptno
  3                   FROM dept
  4                   WHERE dname = 'SALES');

5 rows deleted.
```

1.3.3 무결성 제약 조건 ERROR

무결성 제약 조건을 위반하도록 행을 삭제하면 ERROR 가 발생한다.

문제 10) EMP 테이블에서 사원 번호가 7902 인 사원의 정보를 삭제하여라.

```
SQL> DELETE emp
  2 WHERE empno = 7902;
DELETE emp
*
ERROR at line 1:
ORA-02292: integrity constraint (SCOTT.EMP_SELF_KEY) violated - child record
found
```

문제 11) DEPT 테이블에서 부서명이 ‘ACCOUNTING’ 부서의 정보를 삭제하여라.

```
SQL> DELETE dept
  2 WHERE dname = 'ACCOUNTING';
DELETE dept
*
ERROR at line 1:
ORA-02292: integrity constraint (SCOTT.EMP_FOREIGN_KEY) violated - child record
found
```

1.4 데이터베이스 TRANSACTION

ORACLE SERVER 는 TRANSACTION 을 근거로 하는 데이터의 일관성을 보증한다. TRANSACTION 은 데이터 변경시에 보다 나은 융통성과 제어를 제공하며 그들은 사용자 프로세스 실패나 시스템 실패 같은 이벤트에서 데이터의 일관성을 책임집니다. TRANSACTION 은 데이터를 일관되게 변경하는 DML 문장으로 구성됩니다. 예를 들면, 하나의 예금에 대한 차변과 똑같은 금액이 있는 다른 예금에 대한 대변을 포함해야 하는 두 개의 예금 사이에 예금을 전달합니다. 액션은 둘 모두 실패하거나 둘 모두 성공해야 합니다. 대변은 차변 없이는 인증될 수 없습니다.

1.4.1 TRANSACTION 의 유형

유형	설명
DML	작업의 논리적인 단위로 취급하는 임의의 수의 DML 문장으로 구성됩니다.
DDL	오직 하나의 DDL 문장으로 구성합니다.
DCL	오직 하나의 DCL 문장만으로 구성합니다.

1.4.2 TRANSACTION 의 시작과 종료

가) TRANSACTION 의 시작

- 1) 실행 가능한 SQL 문장이 제일 처음 실행될 때

나) TRANSACTION의 종료

- 1) COMMIT이나 ROLLBACK
- 2) DDL이나 DCL 문장의 실행(자동 COMMIT)
- 3) 기계 장애 또는 시스템 충돌(crash)
- 4) deadlock 발생
- 5) 사용자가 정상 종료

1.5.2 COMMIT과 ROLLBACK의 장점

- 1) 데이터의 일관성 제공
- 2) 데이터를 영구적으로 변경하기 전에 데이터 변경을 확인하게 한다.
- 3) 관련된 작업을 논리적으로 그룹화 할 수 있다.

1.4.3 TRANSACTION 제어

COMMIT, SAVEPOINT, ROLLBACK 문장으로 TRANSACTION의 논리를 제어할 수 있다.

명령어	설명
COMMIT	모든 미결정 데이터를 영구적으로 변경 함으로서 현재 TRANSACTION을 종료합니다.
SAVEPOINT name	현재 TRANSACTION 내에 savepoint를 표시합니다.
ROLLBACK [TO SAVEPOINT name]	ROLLBACK은 모든 미결정 데이터 변경을 버림으로써 현재의 TRANSACTION을 종료합니다. ROLLBACK TO SAVEPOINT name은 savepoint와 모든 연이은 변경을 버립니다.

♣ 참고

하나의 TRANSACTION이 끝난 후에 다음의 실행 가능한 SQL 문장을 자동적으로 다음 TRANSACTION을 시작할 것이다. DDL과 DCL은 자동적으로 COMMIT되므로 TRANSACTION을 임시적으로 종료한다. SAVEPOINT는 ANSI 표준 SQL이 아니다.

1.4.3 암시적 TRANSACTION 처리

- 1) 자동 COMMIT은 다음의 경우 발생
 - ① DDL, DCL 문장이 완료시
 - ② 명시적인 COMMIT이나 ROLLBACK 없이 SQL*Plus를 정상 종료
- 2) 자동 ROLLBACK은 다음의 경우 발생
 - ① SQL*Plus를 비정상 종료 또는 시스템 실패

1.4.4 COMMIT이나 ROLLBACK 이전의 데이터 상태

- 1) 데이터 이전의 상태로 복구가 가능하다.
- 2) 현재 사용자는 SELECT 문장으로 DML 작업의 결과를 확인할 수 있다.
- 3) 다른 사용자는 SELECT 문장으로 현재 사용자 사용한 DML 문장의 결과를 확인할 수 없다.
- 4) 변경된 행은 LOCK이 설정되어서 다른 사용자가 변경할 수 없다.

1.4.5 COMMIT 이후의 데이터 상태

- 1) 데이터베이스에 데이터를 영구적으로 변경
- 2) 데이터의 이전 상태는 완전히 상실
- 3) 모든 사용자가 결과를 볼 수 있다.
- 4) 변경된 행의 LOCK이 해제되고 다른 사용자가 변경할 수 있다.
- 5) 모든 SAVEPOINT는 제거된다.

문제 12) EMP 테이블에서 7788 인 사원의 부서번호를 30 번 부서로 갱신한 후 자료를 확정 하여라.

```
SQL> UPDATE emp  
  2 SET deptno = 30  
  3 WHERE empno = 7788;  
  
1 row updated.  
  
SQL> commit;  
  
Commit complete.
```

1.4.6 ROLLBACK 이후의 데이터 상태

- 1) 데이터의 변경이 취소
- 2) 데이터의 이전 상태로 복구
- 3) 변경된 행들의 LOCK이 해제되어 다른 사용자가 변경할 수 있다.

```
SQL> DELETE emp;  
  
14 rows deleted.  
SQL> ROLLBACK;  
  
Rollback complete.
```

1.4.7 SAVEPOINT로 변경을 ROLLBACK

- 1) SAVEPOINT를 사용하여 현재 TRANSACTION 내에 표시자를 생성한다
- 2) ROLLBACK TO SAVEPOINT 명령을 사용하여 표시자까지 ROLLBACK

```
SQL> UPDATE emp
  2  SET deptno = 30
  3  WHERE empno = 7788;

1 row updated.

SQL> SAVEPOINT point_1;

Savepoint created.

SQL> UPDATE emp
  2  SET job = 'MANAGER';

14 rows updated.

SQL> ROLLBACK TO SAVEPOINT point_1;

Rollback complete.
```

1.4.8 문장 단위 ROLLBACK

- 1) 실행 동안에 단일 DML 문장이 실패하면 단지 그 문장만을 ROLLBACK 한다.
- 2) ORACLE SERVER은 암시적 SAVEPOINT를 구현 한다.
- 3) 모든 다른 변경들은 유지된다.
- 4) 사용자는 COMMIT이나 ROLLBACK을 실행하여 명시적으로 TRANSACTION을 종료한다.

1.5 읽기 일관성

- 1) 읽기 일관성은 항상 데이터의 검색이 일관되게 보증한다.
- 2) 사용자에 의해 행해진 변경은 다른 사용자에 의해 행해진 변경과 충돌하지 않는다.
- 3) 데이터를 똑같게 보증한다.

1.6 Locking

- 1) 동시 TRANSACTION 사이의 상호 작용이 파괴되지 않도록 막아 줍니다.
- 2) 사용자 액션을 요구하지 않습니다.
- 3) 자동적으로 낮은 LEVEL의 제약 조건을 사용합니다.
- 4) TRANSACTION이 지속되도록 합니다.
- 5) 두가지 기본적인 모드를 가집니다.
 - ① Exclusive

② Share

1.6.1 Locking Mode

LOCK MODE	설명
Exclusiv	자원이 공유되는 것을 막아 줍니다. 자원을 배타적으로 lock 하는 첫 번째 TRANSACTION 은 배타적 잠금이 해제되기 전까지는 자원을 변경 할 수 있는 유일한 TRANSACTION입니다.
Share	자원이 공유되도록 허용합니다. 데이터를 읽는 다중 사용자는 데이터를 공유하고, writer 에 의해 동시에 액세스 되는 것을 막기 위해 공유 잠금을 유지 합니다. 똑같은 지원상에서 여러 개의 TRANSACTION 은 공유 잠금을 구할 수 있습니다.

◆ 연습문제 ◆

1. 아래의 구조를 만족하는 MY_DATA 테이블을 생성하시오. 단 ID 가 PRIMARY KEY 이다.

SQL> DESC my_data		
Name	Null?	Type
ID	NOT NULL	NUMBER(4)
NAME		VARCHAR2(10)
USERID		VARCHAR2(30)
SALARY		NUMBER(10,2)

2. 1번에 의해 생성된 테이블에 아래의 값을 입력하여라.

ID	NAME	USERID	SALARY
1	Scott	sscott	10,000.00
2	Ford	fford	13,000.00
3	Patel	ppatel	33,000.00
4	Report	r report	23,500.00
5	Good	ggood	44,450.00

3. 2 번에서 입력한 자료를 확인 하여라.

4. 2 번에서 입력한 자료를 모두 삭제하고 INSERT 하기 위한 SCRIPT 를 작성하여 대화식으로 입력하여라. 단 내용은 2번과 동일하다.

5. 4 번에서 바꾼 자료를 영구적으로 데이터베이스에 등록하여라.

6. ID 가 3 번인 사람의 급여를 65,000.00 으로 갱신하고 영구적으로 데이터베이스에 반영 하여라.

7. 이름이 Ford 인 사원을 영구 제명하여라.

8. 급여가 15,000 이하인 사람의 급여를 15,000 로 변경하여라.

9. 1번에서 생성한 테이블을 삭제하여라.

1. SEQUENCE

SEQUENCE 는 테이블의 행에 대한 SEQUENCE 번호를 자동적으로 생성하기 위해 사용될 수 있다. SEQUENCE 는 사용자가 생성한 데이터베이스 객체이다. SEQUENCE 에 대한 전형적인 사용은 각 행에 대해 유일해야 하는 PRIMARY KEY 값을 생성하기 위해서입니다. SEQUENCE 는 Oracle8 에 의해 발생되고 증가(또는 감소) 됩니다.

1.1 SEQUENCE 특징

- 1) 자동적으로 유일 번호를 생성합니다.
- 2) 공유 가능한 객체
- 3) 주로 기본 키 값을 생성하기 위해 사용됩니다.
- 4) 어플리케이션 코드를 대체합니다.
- 5) 메모리에 CACHE 되면 SEQUENCE 값을 액세스 하는 효율성을 향상시킵니다.

1.2 Syntax

```
CREATE SEQUENCE sequence_name  
  [INCREMENT BY n]  
  [START WITH n]  
  [{MAXVALUE n | NOMAXVALUE}]  
  [{MINVALUE n | NOMINVALUE}]  
  [{CYCLE | NOCYCLE}]  
  [{CACHE | NOCACHE}];
```

sequence_name	SEQUENCE 의 이름입니다.
INCREMENT BY n	정수 값인 n 으로 SEQUENCE 번호 사이의 간격을 지정. 이 절이 생략되면 SEQUENCE 는 1 씩 증가.
START WITH n	생성하기 위해 첫번째 SEQUENCE 를 지정. 이 절이 생략되면 SEQUENCE 는 1 로 시작.
MAXVALUE n	SEQUENCE 를 생성할 수 있는 최대 값을 지정.
NOMAXVALUE	오름차순용 10^{27} 최대값과 내림차순용-1 의 최소값을 지정.
MINVALUE n	최소 SEQUENCE 값을 지정.
NOMINVALUE	오름차순용 1 과 내림차순용-(10^{26})의 최소값을 지정.
CYCLE NOCYCLE	최대 또는 최소값에 도달한 후에 계속 값을 생성할지의 여부를 지정. NOCYCLE 이 디폴트.
CACHE NOCACHE	얼마나 많은 값이 메모리에 오라클 서버가 미리 할당하고 유지하는가를 지정. 디폴트로 오라클 서버는 20 을 CACHE.

문제 1) DEPT 테이블의 PRIMARY KEY에 사용되는 DEPT_DEPTNO SEQUENCE를 생성 합니다.

```
SQL> CREATE SEQUENCE dept_deptno
  2  INCREMENT BY 1
  3  START WITH 91
  4  MAXVALUE 99
  5  NOCACHE
  6  NOCYCLE;
```

♣ 참고

CYCLE OPTION을 사용해서는 안됩니다.(PRIMARY KEY으로 사용될 경우)

1.3 SEQUENCE 확인

한번 SEQUENCE를 생성했으면 데이터 사전에 등록 됩니다. SEQUENCE가 데이터베이스 객체가 된 이후에 USER_OBJECTS DATA DICTIONARY에서 식별할 수 있습니다. 또한 데이터 사전의 USER_SEQUENCES 테이블을 검색함으로써 SEQUENCE의 설정 값을 확인할 수 있다.

문제 2) 현재 SESSION을 이루고 있는 사용자가 소유하고 있는 SEQUENCE를 조회하여라.

```
SQL> SELECT sequence_name,min_value,max_value,increment_by,last_number
  2  FROM user_sequences;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
CUSTID	1	1.000E+27	1	109
DEPT_DEPTNO	1	99	1	91
ORDID	1	1.000E+27	1	622
PRODID	1	1.000E+27	1	200381

1.4 SEQUENCE 사용법

테이블에 사용할 절차적인 번호를 생성하기 위해 SEQUENCE를 사용할 수 있다. NEXTVAL과 CURRVAL의 차별을 사용하여 SEQUENCE 값을 참조한다.

1.4.1 NEXTVAL과 CURRVAL의 차별

가) 특징

- 1) NEXTVAL은 다음 사용 가능한 SEQUENCE 값을 반환 한다.
- 2) SEQUENCE가 참조될 때마다, 다른 사용자에게 조차도 유일한 값을 반환한다.
- 3) CURRVAL은 현재 SEQUENCE 값을 얻는다.
- 4) CURRVAL이 참조되기 전에 NEXTVAL이 사용되어야 한다.

나) NEXTVAL 과 CURRVAL 의 사용 규칙

- 1) NEXTVAL 과 CURRVAL 을 사용할 수 있는 경우
 - ① SUBQUERY 가 아닌 SELECT 문
 - ② INSERT 문 dml SELECT 문
 - ③ INSERT 문의 VALUES 절
 - ④ UPDATE 문의 SET 절
- 2) NEXTVAL 과 CURRVAL 사용할 수 없는 경우
 - ① VIEW 문의 SELECT 문
 - ② DISTINCT 키워드를 사용한 SELECT 문
 - ③ GROUP BY, HAVING, ORDER BY 를 이용한 SELECT 문
 - ④ SELECT, DELETE, UPDATE 문장에서의 SUBQUERY
 - ⑤ CREATE TABLE, ALTER TABLE 명령문의 DEFAULT 절

문제 3) DEPT 테이블에 부서명을 영업부, 위치를 분당구 정자동을 입력하여라.

```
SQL> INSERT INTO dept
  2  VALUES (DEPT_DEPTNO.NEXTVAL, '영업부', '분당구 정자동');
1 row created.
```

문제 4) DEPT_DEPTNO SEQUENCE 의 현재 값을 확인하시오.

```
SQL> SELECT dept_deptno.CURRVAL
  2  FROM dual;
CURRVAL
-----
91
```

1.5 SEQUENCE 값 CACHE

SEQUENCE 값에 대해 보다 빠른 액세스를 허용하기 위해 메모리에 SEQUENCE 를 CACHE 합니다. CACHE 는 SEQUENCE 를 처음 참조할 때 형성됩니다. 다음 SEQUENCE 값에 대한 요구는 CACHE 된 SEQUENCE 에서 읽어 들입니다. 마지막 SEQUENCE 가 사용된 후에 SEQUENCE 에 요구하면 CACHE 된 SEQUENCE 를 메모리에 갖다 높습니다.

1.6 SEQUENCE 에서 간격의 경계

- 1) SEQUENCE 값에서 간격(gap)은 아래의 상황에서 발생합니다.
 - ① ROLLBACK
 - ② SYSTEM CRASH
 - ③ SEQUENCE 가 다른 테이블에서 사용될 때

1.7 SEQUENCE 수정

INCREMENT BY, MAXVALUE, MINVALUE, CYCLE, CACHE 를 변경할 수 있습니다.

1.7.1 Syntax

```
ALTER SEQUENCE sequence_name
  [INCREMENT BY n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE | NOCACHE}];
```

1.7.2 SEQUENCE 수정 지침

- 1) SEQUENCE 에 대한 ALTER 권한을 가지거나 소유자여야 합니다.
- 2) 이후의 SEQUENCE 번호만 영향을 받습니다.
- 3) SEQUENCE 는 다른 번호에서 SEQUENCE 를 다시 시작하기 위해서는 제거하고 다시 생성해야 합니다.
- 4) 유효한 검사를 수행합니다.

1.8 SEQUENCE 제거

데이터 사전에서 SEQUENCE 를 제거하기 위해 DROP SEQUENCE 문장을 사용합니다. SEQUENCE 를 제거하기 위해서는 소유자이거나 DROP ANY SEQUENCE 권한을 가져야 합니다.

1.8.1 Syntax

```
DROP SEQUENCE sequence_name;
```

문제) DEPT_DEPTNO SEQUENCE 를 삭제하여라

```
SQL> DROP SEQUENCE dept_deptno;
```

```
Sequence dropped.
```

◆ 연습문제 ◆

1. 초기값 1부터 최대값 999,999 까지 1씩 증가하는 TEST_SEQ SEQUENCE 를 생성하여라.
2. 현재 SESSION 을 이루고 있는 사용자가 사용할 수 있는 SRQUENCE 를 조회하여라.
3. 1번에서 작성한 SRQUENCE 의 현재 값을 조회하여라.
4. CURRVAL 과 NEXTVAL 을 설명하여라.
5. CACHE 와 NOCACHE 의 차이점을 설명하여라.
6. CYCLE 와 NOCYCLE 의 차이점을 설명하여라.
7. 1번에서 생성한 SRQUENCE 를 삭제하여라.

1. VIEW의 개념

테이블이나 다른 VIEW를 기초로 한 논리적인 테이블이고 VIEW는 자체의 데이터는 없지만 테이블의 데이터를 보거나 변경할 수 있는 장과 같다. VIEW은 실제적으로는 질의 문장을 가진다.

EMP TABLE

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

EMP_10 VIEW

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		17-NOV-81	5000		10
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

1.1 VIEW의 장점

- 1) VIEW은 데이터베이스의 선택적인 내용을 보여줄 수 있기 때문에 데이터베이스에 대한 액세스를 제한한다.
- 2) 복잡한 질의어를 통해 얻을 수 있는 결과를 간단한 질의어를 써서 구할 수 있게 한다.
- 3) 데이터 독립성을 허용한다.
- 4) 동일한 데이터의 다른 VIEW를 나타낸다.
- 5) 조인을 한 것처럼 여러 테이블에 대한 데이터를 VIEW를 통해 볼 수 있다.
- 6) 한 개의 VIEW로 여러 테이블에 대한 데이터를 검색할 수 있다.
- 7) 특정 평가기준에 따른 사용자 별로 다른 데이터를 액세스할 수 있다.

1.2 VIEW의 종류

1.2.1 Simple VIEW

- 1) 오직 하나의 테이블에서만 데이터가 유래된다.
- 2) 데이터 그룹 또는 함수를 포함하지 않는다.
- 3) VIEW를 통해 DML 수행 가능

1.2.3 Complex VIEW

- 1) 다중 테이블에서 데이터가 유래된다.
- 2) 데이터 그룹 또는 함수를 포함한다.
- 3) VIEW를 통한 DML을 항상 허용하지 않는다.

1.3 VIEW의 생성

- 1) CREATE VIEW 문장 내에서 SUBQUERY을 내장하여 VIEW를 생성한다.
- 2) SUBQUERY은 복합 SELECT 구문을 포함할 수 있고 ORDER BY 절을 포함할 수 없다.

1.3.1 Syntax

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name [(alias[,alias,...])]  
AS Subquery  
[WITH CHECK OPTION [CONSTRAINT constraint ]]  
[WITH READ ONLY]
```

OR REPLACE	이미 존재한다면 다시 생성한다.
FORCE	Base Table 유무에 관계없이 VIEW를 만든다.
NOFORCE	기본 테이블이 존재할 경우에만 VIEW를 생성한다.
view_name	VIEW의 이름
Alias	Subquery를 통해 선택된 값에 대한 Column명이 된다.
Subquery	SELECT 문장을 기술한다.
WITH CHECK OPTION	VIEW에 의해 액세스 될 수 있는 행만이 입력, 갱신될 수 있다.
Constraint	CHECK OPTION 제약 조건에 대해 지정된 이름이다.
WITH READ ONLY	이 VIEW에서 DML이 수행될 수 없게 한다.

☞ Guidelines

- 1) VIEW를 정의하는 질의어는 조인, 그룹, Subquery를 포함하는 복잡한 SELECT 문장으로 구성될 수 있다.
- 2) VIEW를 정의하는 질의어에는 ORDER BY 절을 쓸 수 없다.
- 3) 제약 조건의 이름을 명시하지 않으면 시스템이 SYS_Cn 형태의 이름을 지정한다.
- 4) VIEW를 삭제하거나 재생성하지 않고 VIEW의 정의를 변경하려면 OR REPLACE 옵션을 쓸 수 있다.

문제 1) EMP 테이블에서 20 번 부서의 세부 사항을 포함하는 EMP_20 VIEW를 생성 하여라

```
SQL> CREATE VIEW emp_20
  2 AS SELECT *
  3   FROM emp
  4 WHERE deptno = 20;
```

View created.

```
SQL> DESC emp_20
```

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO	NOT NULL	NUMBER(2)

```
SQL> SELECT *
  2   FROM emp_20;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20

문제 2) EMP 테이블에서 30 번 부서만 EMPNO 를 EMP_NO 로 ENAME 를 NAME 로 SAL 를 SALARY 로 바꾸어 EMP_30 VIEW를 생성 하여라.

```
SQL> CREATE VIEW EMP_30
  2 AS SELECT empno emp_no,ename name, sal salary
  3   FROM emp
  4 WHERE deptno = 30;
```

View created.

```
SQL> CREATE VIEW EMP_30
  2 AS SELECT empno emp_no,ename name, sal salary
  3   FROM emp
  4 WHERE deptno = 30;
FROM emp
*
ERROR at line 3:
ORA-00955: name is already used by an existing object
```

위와 같은 ERROR는 이미 EMP_30이라는 OBJECT(table)가 존재한다는 의미이므로 EMP_30을 DROP한 후 다시 생성하여라.

```
SQL> DROP TABLE emp_30;  
  
Table dropped.  
SQL> CREATE VIEW EMP_30  
2 AS SELECT empno emp_no,ename name, sal salary  
3 FROM emp  
4 WHERE deptno = 30;  
  
View created.
```

1.4 VIEW의 구조 및 이름 확인

일단 VIEW가 생성되면, VIEW의 이름과 VIEW 정의를 보기 위해 USER_VIEWS라는 데이터 사전 테이블을 질의할 수 있습니다. VIEW를 만드는 SELECT 문장의 텍스트는 LONG 열에 저장됩니다.

문제 3) 현재 SESSION를 이루고 있는 사용자가 소유한 VIEW를 조회하시오.

```
SQL> COL view_name FORMAT a15  
SQL> COL text_length FORMAT 99,990  
SQL> COL text FORMAT a40  
SQL> SELECT *  
2 FROM user_views;  
  
VIEW_NAME      TEXT_LENGTH TEXT  
-----  
EMP_20          103 SELECT "EMPNO", "ENAME", "JOB", "MGR", "HIRE  
                      DATE", "SAL", "COMM", "DEPTNO"  
                      FROM emp  
EMP_30          70  SELECT empno emp_no,ename name, sal sala  
                      ry  
                      FROM emp  
                      WHERE deptno = 30  
SALES           291 SELECT REPID, ORD.CUSTID, CUSTOMER.NAME  
                      CUSTNAME, PRODUCT.PRODID,  
                      DESCRIP PRODNA
```

1.5 데이터 액세스 VIEW

VIEW를 사용하여 데이터를 액세스할 때 ORACLE SERVER은 다음 작업을 수행합니다.

- 1) USER_VIEWS 데이터 사전 테이블에서 VIEW 정의를 검색합니다.
- 2) VIEW 기반 테이블에 대한 액세스 권한을 확인합니다.
- 3) VIEW 질의를 기본 테이블 또는 테이블들에서의 동등한 작업으로 전환합니다.

문제 4) emp_30 의 VIEW에서 자료를 조회하여라.

```
SQL> SELECT *
  2  FROM emp_30;

EMP_NO NAME          SALARY
----- -----
 7698 BLAKE        2850
 7654 MARTIN       1250
 7499 ALLEN        1600
 7844 TURNER       1500
 7900 JAMES         2450
 7521 WARD          1250

6 rows selected.
```

1.5.1 VIEW의 수정

OR REPLACE 옵션은 비록 이 이름이 이미 존재할지라도 VIEW 가 생성될 수 있도록 해주므로 그 소유자에 대한 오래된 VIEW 버전업할 수 있다.

문제 5) 부서번호 10 번만 포함하고 있는 이미 생성되어 있는 VIEW 의 내용을 사원번호(employee_no), 이름(employee_name), 업무(job_title)의 내용으로 변경하여라.

```
SQL> CREATE OR REPLACE VIEW emp_10
  2  (employee_no,employee_name,job_title)
  3  AS SELECT empno,ename,job
  4  FROM emp
  5  WHERE deptno = 10;

View created.
```

♣ 주의

CREATE VIEW 절에서 열 별칭을 지정할 때 별칭은 SUBQUERY 의 열과 동일한 명령으로 나열됨을 명심하십시오.

1.6 복합 VIEW 생성

두 테이블로부터 값을 출력하는 그룹 함수를 포함하는 복잡한 VIEW 를 생성합니다. VIEW 의 어떤 열이 함수나 표현식에서 유래되었다면 별칭은 필수적입니다.

문제 6) 부서별로 부서명, 최소 급여, 최대 급여, 부서의 평균 급여를 포함하는 DEPT_SUM VIEW를 생성하여라.

```
SQL> CREATE VIEW dept_sum (name,minsal,maxsal,avgsal)
  2 AS SELECT d.dname,MIN(e.sal),MAX(e.sal),AVG(e.sal)
  3 FROM dept d,emp e
  4 WHERE d.deptno = e.deptno
  5 GROUP BY d.dname;
```

```
View created.
```

1.7 VIEW에서 DML 연산 수행

- 1) 단순 VIEW에서 DML 연산을 수행할 수 있습니다.
- 2) VIEW가 다음을 포함한다면 행을 제거할 수 없습니다.
 - ① 그룹 함수
 - ② GROUP BY 절
 - ③ DISTINCT 키워드
- 3) 다음을 포함한다면 VIEW에서 데이터를 수정할 수 없습니다.
 - ① 그룹 함수
 - ② GROUP BY 절
 - ③ DISTINCT 키워드
 - ④ 표현식으로 정의된 열
 - ⑤ ROWNUM 의사열
- 1) 다음을 포함한다면 VIEW에서 데이터를 추가할 수 없습니다.
 - ① 그룹 함수
 - ② GROUP BY 절
 - ③ DISTINCT 키워드
 - ④ 표현식으로 정의된 열
 - ⑤ ROWNUM 의사열
 - ⑥ VIEW에 의해 선택되지 않은 NOT NULL 열이 기본 테이블에 있을 경우

1.8 WITH CHECK OPTION 절 사용

VIEW를 통해 참조 무결성 체크를 수행하는 것이 가능합니다. 또한 데이터베이스 LEVEL에서 제약 조건을 적용할 수 있습니다. VIEW는 데이터 무결성을 보호하기 위해 사용될 수 있지만, 사용은 매우 제한됩니다. VIEW를 통해 수행되는 INSERT와 UPDATE는 WITH CHECK OPTION 절이 있으면 VIEW를 가지고 검색할 수 없는 행 생성을 허용하지 않음을 명시합니다. 그러므로 삽입되거나 갱신되는 데이터에 대해서 무결성 제약 조건과 데이터 검증 체크를 허용합니다. VIEW가 선택하지 않은 행에 대해 DML 작업을 수행하려고 하면, 지정된 제약 조건 명과 함께 에러가 출력됩니다.

문제 7) EMP 테이블과 동일한 EMP_20(20 번 부서만)이라는 VIEW 를 WITH CHECK OPTION 을 사용하여 생성하여라.

```
SQL> CREATE OR REPLACE VIEW emp_20
  2 AS SELECT *
  3 FROM emp
  4 WHERE deptno = 20
  5 WITH CHECK OPTION CONSTRAINT emp_20_ck;

View created.

SQL> UPDATE emp_20
  2 SET deptno = 30
  3 WHERE empno = 7566;
UPDATE emp_20
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

♣ 참고

부서번호가 30 으로 변경된다면 VIEW 은 더 이상 그 종업원들을 볼 수 없기 때문에 아무 행동도 개신되지 않습니다. 그러므로 WITH CHECK OPTION 절로 VIEW 은 부서 20 종업원만 볼 수 있고, 이 종업원들에 대한 부서번호가 VIEW 을 통해 변경되는 것을 허용하지 않습니다.

1.9 DML 연산 부정

- 1) WITH READ ONLY 옵션으로 VIEW를 생성하면 VIEW에서 DML 연산을 수행할 수 없습니다.
- 2) VIEW에서 임의의 행에서 DML 연산을 수행하려고 하면 ORACLE SERVER 에러 ORA-01752 가 발생합니다.

문제 8) EMP 테이블에서 EMP_10(10 번 부서 중 EMPNO,ENAME,JOB)이라는 VIEW 를 WITH READ ONLY OPTION 을 사용하여 생성하여라.

```
SQL> CREATE OR REPLACE VIEW emp_10
  2 AS SELECT empno,ename,job
  3 FROM emp
  4 WHERE deptno = 10
  5 WITH READ ONLY;

View created.

SQL> DELETE FROM emp_10
  2 WHERE empno = 7782;
DELETE FROM emp_10
*
ERROR at line 1:
ORA-01752: cannot delete from view without exactly one key-preserved table
```

1.10 VIEW의 제거

VIEW는 데이터베이스에서 기본 테이블을 기반으로 하기 때문에 데이터 손실 없이 VIEW를 제거할 수 있다.

1.10.1 Syntax

```
DROP VIEW view_name;
```

문제 9) 앞에서 생성한 EMP_10,EMP_20 VIEW를 삭제하여라.

```
SQL> DROP VIEW emp_10;
```

```
View dropped.
```

```
SQL> DROP VIEW emp_20;
```

```
View dropped.
```

◆ 연습문제 ◆

1. EMP 테이블에서 사원 번호, 이름, 업무를 포함하는 EMP_VIEW VIEW를 생성하여라.
2. 1번에서 생성한 VIEW를 이용하여 10 번 부서의 자료만 조회하여라.
3. 1번에서 생성한 VIEW를 DATA DICTIONARY에서 조회하여라.
4. EMP 테이블과 DEPT 테이블을 이용하여 이름, 업무, 급여, 부서명, 위치를 포함하는 EMP_DEPT_NAME이라는 VIEW를 생성하여라.
5. VIEW 생성시 WITH READ ONLY OPTION에 대하여 설명하여라.
6. VIEW 생성시 WITH CHECK OPTION에 대하여 설명하여라.
7. VIEW를 이용하여 자료를 수정할 수 있는 경우와 없는 경우에 대하여 설명하여라.

1. 인덱스(INDEX)의 개요

Oracle8 Server 의 인덱스는 포인터를 사용하여 행의 검색을 촉진시킬 수 있는 스키마 객체입니다. 인덱스는 명시적 또는 자동적으로 생성 할 수 있으며 Column 에 대한 인덱스가 없으면 한 테이블 전체를 검색하게 될 것입니다. 인덱스는 QUERY 의 성능을 향상시키려면 인덱스 생성을 고려해야 한다. 또한 Column 이나 Column 의 집합에 값이 유일하도록 하기 위해 인덱스를 사용할 수 있다.

1.1 인덱스의 특징

- 1) 인덱스는 테이블의 값을 빠르게 액세스 하도록 하는 데이터베이스 객체이다.
- 2) 데이터를 빠르게 찾기 위한 B*TREE 를 써서 디스크 입출력 횟수를 줄인다.
- 3) Oracle8 Server 가 인덱스를 자동적으로 사용하고 유지 보수 한다.
- 4) 인덱스를 만들면 사용자가 직접 조작할 필요가 없게 된다.
- 5) 인덱스는 논리적으로도 물리적으로도 테이블과는 독립적이다.
- 6) 언제든지 생성하거나 삭제할 수 있으며 이는 테이블이나 다른 인덱스에 영향을 주지 않는다는 의미이다.

♣ 참고

B* INDEX에 대하여

- 1) 어떤 행에 대한 액세스 횟수도 동일하게 하는 이진의 균형 탐색 TREE 이다.
- 2) 행이 테이블 시작이나 중간, 또는 끝에 있어도 거의 같은 횟수 내에 지정된 값을 액세스 하는 효율적인 방법이다.
- 3) Oracle8 Server 가 만드는 인덱스는 TREE 에 정렬된 여러 개의 저장소 PAGE 로 구성된다.
- 4) 각 페이지는 키 값이 데이터 자체를 가리킬 때까지 주조의 아래 쪽으로 향하는 페이지에 대한 포인터와 일련의 키 값을 갖고 있다.

1.2 인덱스 생성 방법

1.2.1 자동 생성

테이블 정의에 PRIMARY KEY 나 UNIQUE 제약 조건을 정의할 때 unique 인덱스가 자동적으로 생성된다.

1.2.2 사용자가 생성

행에 대한 액세스 속도를 빠르게 하기 위해 Column 에 non_unique 인덱스 또는 unique 인덱스를 생성한다.

1.3 인덱스의 종류

종 류	설 명
Unique index	지정된 열의 값이 고유함을 보장
Non-unique index	데이터를 검색할 때 가장 빠른 결과를 보장
Single column index	하나의 열만 인덱스에 존재
Composite Index	여러 열을 결합하여 하나의 인덱스를 생성(16 개의 열까지)

1.4 사용자가 인덱스 생성

CREATE INDEX 문장을 이용함으로써 하나 이상의 열의 인덱스를 생성합니다.

1.4.1 Syntax

```
CREATE INDEX index_name  
ON table_name (column1[,column2,.....]);
```

1.4.2 인덱스 생성을 위한 지침

가) 많은 것이 항상 더 좋은 것은 아니다.

테이블의 많은 인덱스가 질의의 스피드 향상을 꼭 의미하는 것은 아닙니다. 인덱스를 가지고 있는 테이블에 대한 각 DML 작업은 인덱스도 갱신되어야 함을 의미합니다. 많은 인덱스가 테이블과 관련되어 있으면, ORACLE SERVER 은 DML 후에 모든 인덱스를 갱신하기 위해 더 많은 노력이 필요하게 됩니다.

나) 언제 인덱스를 생성하는가?

- 1) 열은 WHERE 절 또는 조인 조건에서 자주 사용됩니다.
- 2) 열은 광범위한 값을 포함합니다.
- 3) 열은 많은 수의 null 값을 포함합니다.
- 4) 둘 또는 이상의 열은 WHERE 절 또는 조인 조건에서 자주 함께 사용됩니다.
- 5) 테이블은 대형이고 대부분의 질의들은 행의 2~4%보다 적게 읽어 들일 것으로 예상됩니다.

다) 언제 인덱스를 생성해서는 안되는가

- 1) 테이블이 작다.
- 2) 열이 질의의 조건으로 자주 사용되지 않는다.
- 3) 대부분의 질의들은 행의 2~4% 이상을 읽어 들일 것으로 예상된다.
- 4) 테이블은 자주 갱신됩니다. 테이블에 하나 이상 인덱스를 가지고 있다면 테이블을 엑세스하는 DML 문장은 인덱스의 유지 때문에 상대적으로 더 많은 시간이 걸리게 됩니다.

♣ 참고

유일성을 강행하기를 원한다면, 테이블 정의에 유일한 제약 조건을 정의해야 함을 명심하십시오. 이때 유일한 인덱스는 자동으로 생성됩니다.

문제 1) EMP 테이블에서 ename 열에 인덱스를 생성하여라

```
SQL> CREATE INDEX emp_ename_idx  
2 ON emp (ename);  
  
Index created.
```

1.5 인덱스 생성 확인

- 1) USER_INDEXES 는 인덱스 이름과 고유성 정보를 가지고 있다.
- 2) USER_IND_COLUMNS 는 인덱스 명, 테이블 명, 열명을 가지고 있다.

문제 2) EMP 테이블에 이미 생성되어 있는 인덱스, 관련된 열명, 고유성 여부를 출력하여라.

```
SQL> SELECT c.index_name,c.column_name,  
2 c.column_position,i.uniqueness  
3 FROM user_indexes i,user_ind_columns c  
4 WHERE c.index_name = i.index_name  
5 AND c.table_name = 'EMP';  
  
INDEX_NAME          COLUMN_NAME          COLUMN_POSITION UNIQUENESS  
-----            -----          -----  
EMP_PRIMARY_KEY    EMPNO                  1 UNIQUE
```

♣ 참고

Oracle에서의 최적화 방법

가) 규칙 기준(Rule-base) 최적화

SQL 문장을 실행하기 위한 계획(Execution plan)을 선택할 때 내부적인 규칙에 근거하여 적절한 INDEX를 선정하여 사용하는 방식

나) 비용 기준(Cost-base) 최적화

SQL 문장을 실행하기 위한 계획(Execution plan)을 선택할 때 인덱스가 있는 테이블의 통계치를 분석하여 가장 비용이 적게 드는 방식으로 액세스 경로를 결정한다.

1.6 인덱스 제거

인덱스를 수정할 수 없습니다. 인덱스를 변경하기 위해서는, 그것을 제거하고 다시 작성해야 합니다. DROP INDEX 문장을 생성하여 데이터 사전에서 인덱스 정의를 제거합니다. 인덱스를 제거하기 위해서는 인덱스의 소유자이거나 DROP ANY INDEX 권한을 가지고 있어야 합니다.

1.6.1 Syntax

```
DROP INDEX index_name;
```

문제 3) emp_ename_idx 인덱스를 삭제하여라.

```
SQL> DROP INDEX emp_ename_idx;  
Index dropped.
```

☞ Guidelines

- 1) 인덱스를 수정할 수는 없다.
- 2) 인덱스를 변경하려면 삭제한 다음 다시 만들어야 한다.
- 3) DROP INDEX 명령을 사용하여 인덱스를 삭제하라.
- 4) 인덱스를 삭제하려면 그 인덱스의 소유자이거나 DROP ANY INDEX 권한을 가지고 있어야 한다.

2. 동의어

다른 사용자가 소유한 테이블을 참조하기 위해서는 동의어를 생성한 이름 뒤에 점을 찍고 테이블 이름을 써야 합니다. 동의어 생성은 스키마 이름까지 명시할 필요를 제거시키고 테이블, 뷰, 시퀀스, 프로시저, 또는 다른 객체에 대한 또 다른 이름을 제공 합니다. 이 방법은 뷰처럼 긴 이름을 가진 객체한테 유용하게 사용될 수 있습니다.

2.1 Syntax

```
CREATE [PUBLIC] SYNONYM synonym_name  
FOR object_name;
```

PUBLIC 모든 사용자에 대해 액세스 가능한 동의어를 생성

synonym_name 생성 되어야 할 동의어 이름

object_name 생성된 동의어에 대한 객체를 식별합니다.

☞ Guidelines

- 1) 객체는 패키지에 포함될 수 없습니다.
- 2) 개별 동의어 이름은 동일 사용자가 소유한 모든 다른 객체의 이름과 달라야 합니다.

문제 4) SALGRADE 동의어로 GUBUN 를 생성하고 동의어로 조회하여라.

```
SQL> CREATE SYNONYM gubun  
  2 FOR salgrade;  
  
Synonym created.  
  
SQL> SELECT *  
  2 FROM gubun;  
  
  GRADE      LOSAL      HISAL  
-----  -----  -----  
       1        700      1200  
       2       1201      1400  
       . . . . .
```

2.2 동의어 삭제

동의어를 제거하기 위해 DROP SYNONYM 문장을 사용합니다. DBA 만 공용(Public) 동의어를 제거할 수 있습니다.

2.2.1 Syntax

```
DROP [PUBLIC] SYNONYM synonym_name;
```

문제 5) 앞에서 생성한 동의어를 삭제하여라.

```
SQL> DROP SYNONYM gubun;  
  
Synonym dropped.
```

◆ 연습문제 ◆

1. EMP 테이블에서 이름을 가지고 INDEX(emp_ename_idx)를 생성하여라.
2. INDEX의 장단점을 기술하여라.
3. 테이블 생성시 자동적으로 생성되는 INDEX가 있다. 어떤 제약 조건을 기술하면 생성되는가?
4. INDEX 생성 지침을 설명하여라.
5. 여러 개의 COLUMN으로 인덱스를 사용하는 경우가 있다. 몇개의 COLUMN까지 가능한가?
6. 동의어는 어떤 경우에 사용하는가.
7. EMP 테이블을 EMPLOYEE라는 동의어를 생성하여라.

1. 사용자 접근 제어

다중 사용자 환경에서는 데이터베이스 액세스와 사용의 보안 유지가 요구 됩니다.

- 1) 데이터베이스 액세스 제어
- 2) 데이터베이스에서 특정 객체에 대한 액세스 제공
- 3) 오라클 데이터 사전으로 주어지고 받는 Privilege 확인
- 4) 데이터베이스 객체에 대한 동의어 생성

1.1 데이터베이스 보안의 두 범주

1.1.1 시스템 보안

사용자에 의해 허용된 시스템 작업 같은 시스템 수준에서의 데이터베이스의 액세스와 사용을 규정합니다.

- 1) 사용자 명
- 2) 사용자의 비밀 번호
- 3) 사용자에게 할당된 디스크 공간

1.1.2 데이터 보안

객체에 대해 사용자가 할 수 있는 작업을 규정합니다.

1.2 사용자 생성

DBA 는 CREATE USER 문장을 사용하여 사용자를 생성 합니다. 사용자는 생성성 후 어떠한 권한도 가지지 않습니다. DBA 는 이때 그 사용자에게 여러 권한을 부여 합니다. 이 권한은 데이터베이스 수준에서 사용자가 할 수 있는 것이 무엇인가를 결정 합니다.

1.2.1 Syntax

```
CREATE USER user_name  
IDENTIFIED BY password;
```

문제 1) 사용자 명은 YJB, 패스워드는 YOON 인 사용자를 생성하고 CONNECT,RESOURCE 권한을 부여하여라.

```
SQL> conn system/manager  
Connected.  
SQL> CREATE USER yjb  
2 IDENTIFIED BY yoon;  
  
User created.  
SQL> GRANT connect,resource TO yjb;  
  
Grant succeeded.
```

1.3 권한

권한은 특정 SQL 문장을 실행하기 위한 권한입니다. 데이터베이스 관리자는 데이터베이스와 그 객체에 대한 액세스를 사용자에게 부여하는 능력을 가진 상급 사용자입니다. 사용자는 데이터베이스에 액세스하기 위해 system privilege 가 필요하고 데이터베이스에서 객체의 내용을 조작하기 위해 object privilege 가 필요합니다. 사용자는 관련 권한들의 이름있는 그룹인 role 이나 다른 사용자에게 추가적으로 권한을 부여하기 위해 권한을 가질 수 있습니다.

♣ 참고

스키마는 테이블, 뷰, 시퀀스 같은 객체의 모음입니다. 스키마는 데이터베이스 사용자에 의해 소유되고 사용자와 동일 이름을 가집니다.

1.3.1 시스템 권한

- 1) 사용자와 ROLE에 대해 부여할 수 있는 시스템 권한의 종류는 80개 이상의 있다.
- 2) 시스템 권한은 주로 DBA가 부여한다.
- 3) DBA는 상급의 시스템 권한을 가집니다.
 - ① 새로운 사용자 생성(CREATE USER)
 - ② 사용자 제거(DROP USER)
 - ③ 테이블 제거(DROP ANY TABLE)
 - ④ 테이블 백업(BACKUP ANY TABLE)

가) Syntax

GRANT	system_privilege1[,system_privilege2,]
TO	user_name1[,user_name2,]
[WITH ADMIN OPTION];	

system_privilege 시스템 권한

user_name 사용자 명

WITH ADMIN OPTION 받은 시스템 권한을 다른 사용자에게 부여할 수 있는 권한

나) 시스템 권한의 종류

SQL> SELECT * 2 FROM system_privilege_map;	PRIVILEGE NAME ----- -3 ALTER SYSTEM -4 AUDIT SYSTEM 86 rows selected.
-----------------------------------------------	-------------------------------------------------------------------------------------------------

시스템 권한	허가된 내용(Grantee:권한을 받은 사용자)
ALTER ANY TABLE	Grantee 가 Schema 에 있는 Index 를 Alter 할 수 있다.
ALTER ANY PROCEDURE	Grantee 가 Schema 에 내장 프로시저,함수,또는 패키지 바꾸기를 할 수 있다.
ALTER ANY ROLE	Grantee 가 데이터베이스에서 역할 바꾸기를 할 수 있다.
ALTER ANY TABLE	Grantee 가 Schema 에서 TABLE 이나 VIEW 를 바꾸도록 할 수 있다.
ALTER ANY TRIGGER	Grantee 가 Schema 에서 데이터베이스 TRIGGER 를 활성화,비활성화 또는 Compile 하게할 수 있다.
ALTER DATABASE	Grantee 가 데이터베이스 바꾸기를 허용한다.
ALTER USER	Grantee 가 사용자 바꾸기를 할 수 있다. 이 권한은 Grantee 가 다른 사용자의 Password 나 확인 방법을 바꾸도록 권한을 주고 DEFAULT TABLESPACE, TEMPORARY TABLESPACE, PROFILE, QUOTA 의 양을 바꿀 수 있도록 한다.
CREATE ANY INDEX	Grantee 가 어떤 Schema 에서나 테이블에 인덱스 만들기를 허용한다.
CREATE ANY PROCEDURE	Grantee 가 어떤 Schema 에서 내장 프로시저,함수,패키지를 만들 수 있도록 허용한다.
CREATE ANY TABLE	Grantee 가 어떤 Schema 에서나 테이블을 만들 수 있도록 허용한다.
CREATE ANY TRIGGER	Grantee 가 어떤 Schema 에서나 테이블과 연관된 Schema 에서 데이터베이스 트리거를 만들 수 있도록 허용한다.
CREATE ANY VIEW	Grantee 가 어떤 Schema 에서나 VIEW 를 만들 수 있도록 허용한다.
CREATE PROCEDURE	Grantee 가 자체 Schema 에서 내장 프로시저,함수,패키지를 만들 수 있도록 허용한다.
CREATE PROFILE	Grantee 가 PROFILE 을 만들 수 있도록 허용한다.
CREATE ROLE	Grantee 가 ROLE 을 만들 수 있도록 허용한다.
CREATE SYNONYM	Grantee 가 자체 Schema 에서 시너임을 만들 수 있도록 허용한다.
CREATE TABLE	Grantee 가 자체 Schema 에서 테이블을 만들 수 있도록 허용한다.
CREATE TRIGGER	Grantee 가 자체 Schema 에서 트리거를 만들 수 있도록 허용한다.
CREATE USER	Grantee 가 사용자를 만들 수 있도록 허용한다.

CREATE VIEW	Grantee 가 자체 Schema 에서 VIEW 를 만들 수 있도록 허용한다.
DELETE ANY TABLE	Grantee 가 어떤 Schema 에서 테이블의 자료를 삭제할 수 있도록 허용한다.
DROP ANY INDEX	Grantee 가 어떤 Schema 에서나 인덱스를 삭제할 수 있다.
DROP ANY PROCEDURE	Grantee 가 어떤 Schema 에서나 내장 프로시저, 함수, 패키지를 삭제할 수 있도록 허용한다.
DROP ANY ROLE	Grantee 가 ROLE 을 삭제하도록 허용한다.
DROP ANY SYNONYM	Grantee 가 어떤 Schema 에서나 시너임을 삭제할 수 있도록 허용한다.
DROP ANY TABLE	Grantee 가 어떤 Schema 에서나 테이블을 삭제할 수 있도록 허용한다.
DROP ANY TRIGGER	Grantee 가 어떤 Schema 에서나 데이터베이스 트리거를 삭제할 수 있도록 허용한다.
DROP ANY VIEW	Grantee 가 어떤 Schema 에서나 VIEW 를 삭제할 수 있도록 허용한다.
DROP USER	Grantee 가 사용자를 삭제할 수 있도록 허용한다.
EXECUTE ANY PROCEDURE	Grantee 가 어떤 Schema 에서나 프로시저, 함수, 패키지를 실행할 수 있도록 허용한다.
TRANSACTION	Local Database 에서 자체의 불안정한 분산 Transaction 의 BACK 을 허용한다.
GRANT ANY PRIVILEGE	Grantee 가 시스템 권한을 주는 것을 허용한다.
GRANT ANY ROLE	Grantee 가 데이터베이스에서 어떠한 ROLE 이라도 GRANT 할 수 있는 권한을 허용한다.
INSERT ANY TABLE	Grantee 가 어떠한 Schema 에서나 테이블과 VIEW 에 자료를 삽입할 수 있도록 허용한다.
LOCK ANY TABLE	Grantee 가 어떤 Schema 에서나 테이블과 VIEW 에 LOCK 을 걸도록 허용한다.
SELECT ANY SEQUENCE	Grantee 가 어떤 Schema 에서나 시퀀스를 참조할 수 있도록 허용한다.
SELECT ANY TABLE	Grantee 가 어떤 Schema 에서나 테이블, VIEW, Snapshot 을 참조할 수 있도록 허용한다.
UPDATE ANY	Grantee 가 테이블에서 행을 수정하도록 허용한다.

문제 2) SCOTT에게 CREATE ROLE 권한을 부여하여라.

```
SQL> conn system/manager
Connected.
SQL> GRANT create role TO scott;

Grant succeeded.
```

1.3.2 시스템 권한 최소

REVOKE 명령으로 시스템 권한을 취소할 수 있다. WITH ADMIN OPTION을 통해 부여된 권한은 취소되지 않는다.

가) Syntax

```
REVOKE system_privilege1[,system_privilege2, . . . .] | role1[,role2, . . . .]
FROM {user1[,user2, . . . .] | role1[,role2, . . . .] | PUBLIC};
```

문제 3) SCOTT에게 부여된 CREATE ROLE 권한을 취소하여라.

```
SQL> conn system/manager
Connected.
SQL> REVOKE create role FROM scott;

Revoke succeeded.
```

☞ Guidelines

사용자가 WITH ADMIN OPTION으로 권한을 부여 받았다면 그 사용자는 WITH ADMIN OPTION으로 권한을 부여해줄 수 있어 수여자 간의 전 체인이 가능하지만 소유자가 다른 사용자에게 부여한 권한을 취소하면 모든 권한을 연이어 취소되지 않는다.

1.3.3 객체 권한

- 1) 객체 권한은 객체마다 다양하다.
- 2) 객체 소유자는 객체에 대한 모든 권한을 가지고 있다.
- 3) 소유자는 사용자 객체에 대한 특정 권한을 제공할 수 있습니다.

♣ 주의

DBA는 일반적으로 시스템 권한을 할당합니다. 객체를 소유한 모든 사용자는 객체 권한을 부여할 수 있습니다. WITH GRANT OPTION으로 부여 받은 권한은 부여자에 의해 다른 사용자와 ROLE에게 다시 부여될 수 있습니다. WITH GRANT OPTION으로 테이블을 질의할 수 있고 테이블에 행을 추가할 수 있도록 해 줍니다. 테이블의 소유자는 PUBLIC 키워드를 사용하여 모든 사용자에게 액세스 권한을 부여할 수 있습니다.

객체 권한	TABLE	VIEW	SEQUENCE	PROCEDURE	SNAPSHOT
ALTER	♣		♣		
DELETE	♣	♣			
EXECUTE				♣	
INDEX	♣				
INSERT	♣	♣			
REFERENCES	♣				
SELECT	♣	♣	♣		♣
UPDATE	♣	♣			

나) OBJECT 권한의 종류

```
SQL> SELECT *
  2 FROM table_privilege_map;

PRIVILEGE NAME
-----
0 ALTER
1 AUDIT
2 COMMENT
3 DELETE
.
.
.
13 rows selected.
```

OBJECT 권한	허가된 내용(Grantee:권한을 받은 사용자)
ALTER	Grantee 가 OBJECT 에 대해 ALTER 할 수 있도록 허용한다.
AUDIT	Grantee 가 OBJECT 에 대해 감사할 수 있도록 허용한다.
COMMENT	Grantee 가 OBJECT 에 대해 COMMENT 할 수 있도록 허용한다.
DELETE	Grantee 가 OBJECT 에 대해 자료를 삭제할 수 있도록 허용한다.
GRANT	Grantee 가 OBJECT 에 대해 GRANT 할 수 있도록 허용한다.
INDEX	Grantee 가 OBJECT 에 대해 인덱스를 생성할 수 있도록 허용한다.
INSERT	Grantee 가 OBJECT 에 대해 자료를 삽입할 수 있도록 허용한다.
LOCK	Grantee 가 OBJECT 에 대해 Locking 할 수 있도록 허용한다.
RENAME	Grantee 가 OBJECT 에 대해 이름을 변경할 수 있도록 허용한다.
SELECT	Grantee 가 OBJECT 에 대해 자료를 조회할 수 있도록 허용한다.
UPDATE	Grantee 가 OBJECT 에 대해 자료를 갱신할 수 있도록 허용한다.
REFERENCES	Grantee 가 OBJECT 에 대해 자료를 참조할 수 있도록 허용한다.
EXECUTE	Grantee 가 프로시저, 함수, 패키지에 대해 실행할 수 있도록 허용한다.

문제 4) 앞에서 생성한 YJB 사용자에게 SCOTT 이 소유하고 있는 EMP 테이블을 조회하고 삽입할 수 있는 권한을 부여하여라.

```
SQL> conn scott/tiger
Connected.
SQL> GRANT select,insert ON emp TO yjb;

Grant succeeded.
SQL> conn yjb/yoon
Connected.
SQL> SELECT empno,ename,job,hiredate,sal
  2  FROM scott.emp
  3 WHERE deptno = 10;

EMPNO ENAME      JOB        HIREDATE          SAL
----- -----
 7839 KING        PRESIDENT 17-NOV-81       5000
 7782 CLARK       MANAGER   09-JUN-81       2450
 7934 MILLER     CLERK    23-JAN-82       1300
```

```
SQL> conn scott/tiger
Connected.
SQL> GRANT select,insert ON emp TO yjb;

Grant succeeded.
SQL> conn yjb/yoon
Connected.
SQL> CREATE SYNONYM emp FOR scott.emp;

Synonym created.
SQL> SELECT empno,ename,job,hiredate,sal
  2  FROM emp
  3 WHERE deptno = 10;

EMPNO ENAME      JOB        HIREDATE          SAL
----- -----
 7839 KING        PRESIDENT 17-NOV-81       5000
 7782 CLARK       MANAGER   09-JUN-81       2450
 7934 MILLER     CLERK    23-JAN-82       1300
```

1.3.4 객체 권한 철회

다른 사용자에게 부여된 권한을 철회하기 위하여 REVOKE 문장을 사용합니다. WITH GRANT OPTION을 통해 다른 사용자에게 부여된 권한도 같이 취소된다.

가) Syntax

```
REVOKE {object_privilege1[,object_privilege2, . . . .] | ALL}
ON object_name
FROM {user1[,user2, . . . .] | role1[,role2, . . . .] | PUBLIC}
[CASCADE CONSTRAINTS];
```

CASCADE CONSTRAINTS REFERENCES 권한을 사용하여 만들어진 객체에 대한 참조 무결성 제약 조건을 제거하기 위해 사용한다.

문제 5) 앞에서 EMP 테이블에 부여한 SELECT 권한을 YJB에게서 취소하여라.

```
SQL> REVOKE select ON emp FROM yjb;

Revoke succeeded.
SQL> conn yjb/yoon;
Connected.
SQL> SELECT *
  2  FROM scott.emp;
  FROM scott.emp
*
ERROR at line 2:
ORA-00942: table or view does not exist
```

☞ Guidelines

사용자가 WITH GRANT OPTION으로 권한을 부여 받았다면 그 사용자는 WITH GRANT OPTION으로 권한을 부여해줄 수 있어 수여자 간의 전 체인이 가능하지만 원형 부여는 허용되지 않는다. 소유자가 다른 사용자에게 부여한 권한을 취소하면 모든 권한을 연이어 취소된다.

1.3.5 부여된 권한 확인

데이터 사전 테이블	설명
ROLE_SYS_PRIVS	ROLE에게 부여된 시스템 권한
ROLE_TAB_PRIVS	ROLE에게 부여된 테이블 권한
USER_ROLE_PRIVS	사용자에 의해 액세스 가능한 ROLE.
USER_TAB_PRIVS_MADE	사용자가 부여된 객체 권한
USER_TAB_PRIVS_REC0	사용자에게 부여된 객체 권한
USER_COL_PRIVS_MADE	사용자가 객체의 열에 대해 부여한 객체 권한
USER_COL_PRIVS_REC0	특정 열에 대해 사용자에게 부여된 객체 권한

문제 6) SCOTT 에게 할당되어 있는 SYSTEM ROLE 를 확인 하여라.

```
SQL> conn scott/tiger
Connected.
SQL> SELECT *
  2 FROM role_sys_privs;

ROLE                      PRIVILEGE          ADM
-----                    -----
CONNECT                   ALTER SESSION      NO
CONNECT                   CREATE CLUSTER    NO
13 rows selected.
```

1.4 ROLE 의 개념

ROLE 은 사용자에게 부여할 수 있는 관련된 권한들의 그룹이다. 이러한 ROLE 을 이용하면 권한 부여와 회수를 쉽게 할 수 있다. 한 사용자가 여러 ROLE 을 액세스할 수 있고 다른 여러 사용자에게 같은 ROLE 을 지정할 수 있다. ROLE 을 생성하기 위해서는 CREATE ROLE 권한 또는 DBA 권한이 필요하다.

1.4.1 ROLE 의 작성과 지정 순서

- 1) 먼저 DBA 가 ROLE 을 생성한다.
- 2) ROLE 에 권한을 지정한다.
- 3) 사용자에게 ROLE 을 부여한다.

1.4.2 Syntax

```
CREATE ROLE role_name;
```

role_name 생성되는 ROLE 의 이름

문제 7) LEVEL1 이라는 ROLE 을 생성하여라.

```
SQL> CONN SYSTEM/MANAGER
Connected.
SQL> CREATE ROLE level1;
Role created.
```

문제 8) LEVEL1 이라는 ROLE 에 CREATE SESSION,CREATE TABLE,CREATE VIEW 의 권한을 부여 하여라.

```
SQL> GRANT CREATE SESSION,CREATE TABLE,CREATE VIEW,  
2 TO level1;
```

```
Grant succeeded.
```

문제 9) TEST1/TIGER1 과 TEST2/TIGER2 라는 사용자를 생성하여라

```
SQL> CREATE USER test1  
2 IDENTIFIED BY tiger1;
```

```
User created.
```

```
SQL> CREATE USER test2  
2 IDENTIFIED BY tiger2;
```

```
User created.
```

문제 10) TEST1,TEST2 에 LEVEL1 이라는 ROLE 를 부여하여라.

```
SQL> GRANT level1 TO test1,test2;
```

```
Grant succeeded.
```

```
SQL> conn test1/tiger1
```

```
Connected.
```

◆ 연습문제 ◆

1. Oracle8에 로그 온하기 위해 필요한 권한은 무엇인가 ?
2. 테이블을 생성하기 위해 필요한 권한은 무엇인가 ?
3. SYSTEM PRIVILEGE 와 OBJECT PRIVILEGE 란 ?
4. 테이블을 생성한 OWNER 는 어떠한 권한을 가지는가 ?
5. 사용자가 비밀 번호를 갱신하려면 어떤 문장을 기술하여야 하는가 ?
6. 사용자가 액세스할 수 있는 테이블을 조회하여라 ?
7. CONNECT ROLE 와 RESOURCE ROLE 에 대하여 설명하여라 .
8. 현재 SESSION 을 이루고 있는 사용자가 가지고 있는 OBJECT 권한을 조회하여라 .
9. 사용자는 KSH 이고 패스워드는 KIM 인 사용자를 생성하여라 .
10. 9 번에서 생성된 사용자에게 CONNECT 와 RESOURCE 권한을 부여하여라 .
11. 10 번에서 부여한 권한을 취소하고 KSH 사용자를 삭제하여라 .

1. PL/SQL 개요

PL/SQL(Procedural Language/SQL)은 최근의 프로그래밍 언어의 특성을 수용한, SQL의 확장이라 할 수 있다. SQL의 데이터 조작(DML)과 질의문(QUERY)을 블록 구조에 절차적 단위(IF, LOOP, FOR 등)로 된 코드를 포함할 수 있으며 절차적 프로그래밍을 가능하게 한 강력한 TRANSACTION 처리 언어이다.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_name PROMPT '이름: '
DECLARE
    v_empno emp.empno%TYPE;
    v_name  emp.ename%TYPE := UPPER('&p_name');
    v_sal   emp.sal%TYPE;
    v_job   emp.job%TYPE;
BEGIN
    SELECT empno, job
        INTO v_empno, v_job
        FROM emp
        WHERE ename = v_name;
    IF v_job IN ('MANAGER', 'ANALYST') THEN
        v_sal := v_sal * 1.5;
    ELSE
        v_sal := v_sal * 1.2;
    END IF;
    UPDATE emp
        SET sal = v_sal
        WHERE empno = v_empno;
    IF SQL%FOUND THEN
        DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || '개의 행이 갱신되었습니다.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('갱신된 자료가 없습니다.');
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 자료가 없습니다.');

    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 동명 이인입니다.');

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러가 발생 했습니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.1 PL/SQL 의 장점

PL/SQL은 SQL로는 얻을 수 없는 절차적 언어의 기능을 가지고 있다.

1.1.1 프로그램 개발의 모듈화

- 1) 블록 내에서 논리적으로 관련된 문장들의 그룹화할 수 있다.
- 2) 강력한 프로그램을 작성하기 위해 서브 블록들을 큰 블록에 포함할 수 있다.
- 3) 복잡한 문제에 대한 프로그래밍이 적절히 나뉘어진 모듈들의 집합으로 구성된다.

1.1.2 식별자 선언

- 1) 변수, 상수 등을 선언하고 SQL과 절차적인 프로그램에서 사용한다.
- 2) 데이터베이스의 테이블과 Record를 기반으로 하는 dynamic한 변수 선언이 가능하다.
- 3) 단일형 데이터 타입과 복합형 데이터 타입을 선언할 수 있다.

1.1.3 절차적 언어 구조로 된 프로그램 작성

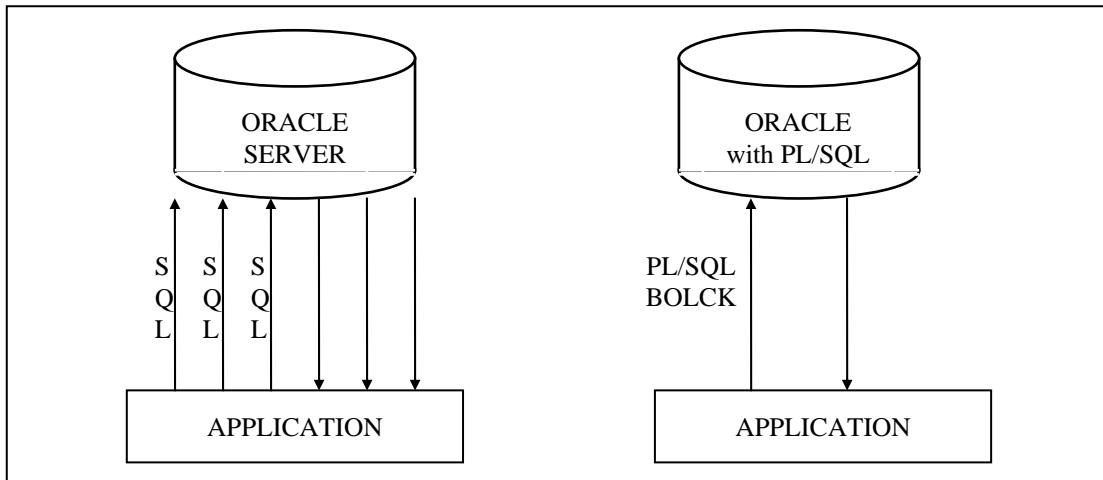
- 1) IF 문은 조건에 따라 일련의 문장을 실행한다.
- 2) LOOP 문을 사용하여 일련의 문장을 반복적으로 실행한다.
- 3) Explicit Cursor를 이용한 Multi-row 질의 처리한다.

1.1.4 ERROR 처리

- 1) Exception 처리 루틴을 이용하여 Oracle Server 에러를 처리한다.
- 2) 사용자 정의 에러를 선언하고 Exception 처리 루틴으로 처리 가능하다.

1.1.5 성능 향상

- 1) PL/SQL은 응용 프로그램의 성능을 향상 시킬 수 있다.
- 2) PL/SQL은 여러 SQL 문장을 BLOCK으로 묶고 한번에 BLOCK 전부를 서버로 전송하기 때문에 통신량을 줄일 수 있다.



1.1.6 PL/SQL 의 처리

PL/SQL 로 작성된 Block 을 Oracle Server 로 보내면 그 안에 있는 PL/SQL Engine 이 SQL 문과 Non SQL 문을 구분하여 Non SQL 문은 PL/SQL Engine 내의 Procedural statement executor 가 수행하고 SQL 문은 SQL statement executor 가 처리하게 된다. 즉 Non SQL 문은 Client 환경에서 처리되고 SQL 문은 서버에서 실행하게 된다. 따라서 PL/SQL 을 사용하게 되면 서버쪽으로 작업의 양을 줄이게 되므로 Network traffic 를 감소시켜 Performance 를 증가시키는 이점이 있다.

1.2 PL/SQL Block 구조

PL/SQL 은 프로그램을 논리적인 블록으로 나누게 하는 구조화된 블록 언어이다. PL/SQL 블록은 선언부(선택적), 실행부(필수적), 예외 처리부(선택적)로 구성되고 BEGIN 과 END 는 키워드로 반드시 기술하여야 한다. PL/SQL 블록에서 사용하는 변수는 블록에 대해 논리적으로 선언할 수 있고 사용할 수 있다. 변수들과 그 외의 식별자를 참조하고 선언함으로써 PL/SQL 블록 내에서 값을 저장하고 변경할 수 있다.

- ◆ DECLARE
 - ✧ variables, cursor, user_defined, exception
- ◆ BEGIN
 - ✧ SQL,PL/SQL statements;
- ◆ EXCEPTION
 - ✧ actions to perform when errors occur
- ◆ END;

기본적인 PL/SQL Block 은 세 부분으로 구성된다.

섹션	설명	포함
선언부	실행부에서 참조할 모든 변수, 상수, CURSOR, EXCEPTION 을 선언	선택
실행부	데이터베이스의 데이터를 처리할 SQL 문과 PL/SQL 블록을 기술	필수
예외 처리부	실행부에서 예외가 발생했을 때 수행될 문장을 기술	선택

☞ Guidelines

- 1) PL/SQL Block 내에서는 한 문장이 종료할 때마다 세미콜론(:)을 기술한다.
- 2) END 뒤에 세미콜론(:)을 사용하여 하나의 Block 이 끝났다는 것을 명시한다.
- 3) PL/SQL Block 의 작성은 편집기를 통해 파일로 작성할 수 있고 SQL*Plus 에서 바로 작성할 수 있다.
- 4) SQL Buffer 에서 PL/SQL 을 실행하기 위해 “/”을 사용하여 성공적으로 실행 된다면 PL/SQL procedure successfully completed 라는 Message 가 출력된다.

1.2.1 PL/SQL 블록의 유형

PL/SQL의 모든 단위는 하나 또는 그 이상의 블록을 포함합니다. 이 블록은 다른 것으로부터 하나로 완전히 분리되어 중첩될 수 있습니다. 기본 단위(프로시저, 함수, 서브 프로그램, 익명의 블록)는 임의의 수의 중첩된 서브 블록을 포함할 수 있는 논리적인 블록으로 구성된다. 그러므로 하나의 블록은 다른 블록의 작은 부분이 되기도 하고 또는 코드 단위의 전체중 일부가 될 수도 있습니다.

Anonymous	Procedure	Function
[DECLARE] BEGIN statements; statements; statements; [EXCEPTION] END;	CREATE PROCEDURE name IS BEGIN statements; statements; [EXCEPTION] END;	CREATE FUNCTION name RETURN datatype IS BEGIN statements; statements; RETURN value; [EXCEPTION] END;

가) Anonymous Block(익명 블록)

이름이 없는 블록을 의미한다. 그것은 실행하기 위해 프로그램 안에서 선언되고 실행 시에 실행을 위해 PL/SQL 엔진으로 전달됩니다. 실행 컴파일러 프로그램과 SQL*Plus 또는 서버 관리자에서 익명의 블록을 내장할 수 있습니다.

나) Subprogram(procedure, function)

Subprogram은 매개변수를 사용할 수 있고 호출할 수 있는 PL/SQL BLOCK이다. Procedure 또는 Function로 선언될 수 있습니다. 일반적으로 어떤 작업을 수행하기 위해 Procedure를 사용하고 값을 계산하기 위해 Function을 사용합니다. 서버 또는 Application 수준에서 Subprogram을 저장할 수 있습니다. Developer/2000을 사용하여 Application의 일부로써 Procedure와 Function을 선언할 수 있고 필요할 때마다 Trigger를 이용하여 사용할 수 있다.

1.2.2 프로그램의 구성

프로그램 구성	설 명	사용 환경
익명 블록	Application에 내장되거나 대화식으로 호출	모든 PL/SQL 환경
내장된 Procedure or Function	매개변수를 받아들일 수 있고 이름을 이용하여 반복적으로 호출할 수 있는 이름이 있는 PL/SQL 블록은 Oracle 서버에 저장된다.	Oracle Server
Application Procedure or Function	매개변수를 받아들일 수 있고 이름을 이용하여 반복적으로 호출할 수 있는 이름이 있는 PL/SQL 블록은 Developer/2000 어플리케이션에 저장되거나 Shared Library에 저장된다.	Developer/2000
Package	관련된 Procedure or Function을 묶어 이름을 붙인 PL/SQL 모듈입니다.	Oracle Server 와 Developer/2000
Database Trigger	Database Table과 관련된 DML 명령문에 의해 Trigger될 때 자동적으로 실행됩니다.	Oracle Server
Application Trigger	PL/SQL 블록은 Application Event와 관련되고 자동적으로 실행됩니다.	Developer/2000

1.2.3 PL/SQL 환경

PL/SQL은 별개의 Oracle 제품이 아니라 Oracle8 서버와 다른 Oracle TOOL에 이용되고 있는 프로그래밍 언어이다. PL/SQL의 블록은 Oracle8 서버나 툴에 내장되는 PL/SQL 엔진에 전달되어 처리된다. 사용하는 엔진은 PL/SQL이 수행되는 곳에 따라 다르다.

가) Oracle8 서버에서의 PL/SQL 엔진

Pro*프로그램, USER-EXIT, SQL*Plus, 또는 Server Manager에서 PL/SQL 블록을 사용하면 Oracle8 서버의 PL/SQL 엔진이 처리한다. 그리고 블록에 있는 SQL을 별도의 문장으로 분리하여 SQL 문 실행기로 보낸다. 이는 응용 프로그램의 블록을 한 번에 Oracle8 서버에게 보낸다는 뜻이며 따라서 client/server 환경하에서 많은 성능 향상을 기대할 수 있다.

나) Oracle 툴에서의 PL/SQL

Developer/2000을 포함한 많은 Oracle 툴은 Oracle7 서버에 있는 엔진과는 별도로 자체 PL/SQL 엔진을 갖고 있다. 이 엔진이 SQL 문장을 찾아서 Oracle7 서버의 SQL 문 실행기로 보내고, PL/SQL engine은(데이터베이스라기보다는 이미 클라이언트 환경에 있는) 응용 프로그램에 대해 지역적인 데이터를 처리한다. 이로써 Oracle7 서버의 대한 작업량과 요구되는 메모리 커서의 수를 줄인다.

♣ 참고

Developer/2000 응용 프로그램의 부분으로 선언된 프로시저와 함수의 일반적인 구조는 동일하더라도 데이터베이스에 저장된 것과는 다르다. Stored Subprogram은 데이터베이스 객체이고 데이터 사전에 저장되며 여러 응용 프로그램이 사용할 수 있다. Application subprogram은 응용 프로그램의 지역적인 PL/SQL 엔진에 블록을 전달한다. 작업은 서버 쪽이 아닌 응용 프로그램 쪽에서 수행된다.

1.3 SQL*Plus로 하는 일

- 1) SQL 명령과 PL/SQL 블록의 입력, 편집, 저장, 검색 및 실행을 해준다.
- 2) 데이터 베이스의 데이터 access를 가능하게 한다.
- 3) 계산수행, query 결과를 보고서 양식으로 출력한다.
- 4) SQL 데이터베이스 언어와 절차적 언어의 확장인 PL/SQL을 SQL*Plus 프로그램을 이용하여 사용할 수 있다.
- 5) SQL*Plus는 SQL 명령과 PL/SQL 블럭을 조작하는 등 많은 부가적인 작업을 수행할 수 있게 해 준다.

◆ 연습문제 ◆

1. PL/SQL의 특징을 설명하여라.
2. PL/SQL에서 사용하는 BLOCK란
3. PL/SQL의 3 가지 유형을 설명하여라.
4. SQL*Plus 와 PL/SQL의 관계를 설명하여라

1. 변수

SQL 과 절차적인 문장 안에서 PL/SQL 로써 변수를 선언할 수 있고 그것을 사용할 수 있다.

1.1 변수 사용

변수는 자료를 일시적으로 저장하고 변경하고 검증하기 위해 하나 또는 그 이상의 변수를 선언하여 사용한다. 또한 변수는 데이터베이스를 액세스하지 않고 계산이나 다른 데이터 조작에 사용할 수 있다. 이러한 변수는 일단 선언되면 다른 선언적 문장을 포함한 다른 문장에서 간단하게 그것을 반복적으로 참조하여 사용할 수 있다.

%TYPE 와 %ROWTYPE 을 사용하여 변수를 선언하면 테이블의 구조가 변경(데이터형과 길이)되어도 Application 에서는 실행 시간에 테이블을 참조하여 변수가 정의되므로 데이터의 독립성, 유지비용 절감을 제공하고, 새로운 업무 요구에 충족시키기 위해 데이터베이스 변경에 따라 프로그램의 적응, 수정되는 것을 허용한다.

1.2 PL/SQL에서 변수 처리

- 1) 선언 섹션 내에서 변수를 선언하고 초기화하여 사용합니다.
- 2) 실행 섹션에서 변수에 대한 새 값을 할당 합니다.
- 3) 매개변수를 통해 PL/SQL 블록으로 값을 전달합니다.
- 4) 출력 변수를 통해 결과를 봅니다.

문제 1) EMP 테이블에 EMPNO_SEQUENCE 의 SEQUENCE 를 이용하여 이름.급여,부서번호를 입력 받아 등록하는 SCRIPT 를 작성하여라. 단 10 번부서는 입력된 급여에 20%의 가산하여 등록하고 30 번부서는 10% 가산 점이 있다.

```
SET VERIFY OFF
ACCEPT p_name  PROMPT  '이    름: '
ACCEPT p_sal   PROMPT  '급    여: '
ACCEPT p_deptno PROMPT  '부서번호: '
DECLARE
    v_name          VARCHAR2(10) := UPPER('&p_name');
    v_sal           NUMBER(7,2)  := &p_sal;
    v_deptno        NUMBER(2)   := &p_deptno;
BEGIN
    IF v_deptno = 10 THEN
        v_sal := v_sal * 1.2;
    ELSIF v_deptno = 30 THEN
        v_sal := v_sal * 1.1;
    END IF;
    INSERT INTO emp(empno,ename,sal,deptno)
        VALUES (empno_sequence.NEXTVAL,v_name,v_sal,v_deptno);
    COMMIT;
END;
/
```

```
SET VERIFY ON
```

1.3 변수 유형

모든 PL/SQL 변수는 저장 포맷, 제약 조건, 값의 유효 범위를 지정하는 데이터형을 가지고 있다. PL/SQL은 변수, 상수, 포인터를 선언하기 위해 사용할 수 있는 4 가지 데이터형 (Scalar, Composite, Reference, LOB(large objects:Oracle8))을 지원합니다.

1.3.1 PL/SQL 변수

- 1) Scalar : 주로 단일 값을 보유합니다. 주요 데이터형은 ORACLE SERVER 테이블의 열 유형에 대응하는 것들입니다.
- 2) Composite : 레코드 같은 조합 데이터형은 PL/SQL 블록에서 조작되고 정의되는 필드 그룹을 허용합니다.
- 3) Reference : 참조 데이터형은 pointer 라 불리며 다른 프로그램 항목을 지시하는 값을 보유합니다.
- 4) LOB(large objects) : LOB 데이터형은 locator 라 불리며 라인 밖에서 지정된 큰 객체의 위치를 지정하는 값을 보유합니다.

1.3.2 Non-PL/SQL 변수

- 1) Bind 와 host variables

1.4 PL/SQL 변수 선언

PL/SQL 블록에서 그것을 참조하기 전에 선언 섹션에서 모두 PL/SQL 식별자를 선언할 필요가 있습니다. 초기값을 할당하기 위해 옵션을 가집니다. 변수를 선언하기 위해 변수에 대한 값을 할당할 필요는 없습니다. 선언에서 다른 변수를 참조한다면 이전 문장에서 개별적으로 그것들을 반드시 선언해 놓아야 합니다.

1.4.1 Syntax

```
identifier [CONSTANT] datatype [NOT NULL]  
[:= | DEFAULT expression];
```

identifier 변수의 이름

CONSTANT 변수의 값을 변경할 수 없도록 제약합니다.

datatype Scalar, Composite, Reference, LOB(large objects)

NOT NULL 값을 포함해야만 하도록 하기 위해 변수를 제약 합니다.

Expression Literal, 다른 변수, 연산자나 함수를 포함하는 표현식

1.4.2 사용 예

```
DECLARE  
    v_hiredate      DATE;  
    v_deptno        NUMBER(2) NOT NULL := 10;  
    v_loc            VARCHAR2(13) := 'ATLANTA';  
    v_com            CONSTANT NUMBER := 1400;
```

☞ Guidelines

- 1) SQL 객체에 대해 사용된 동일한 이름 지정 규칙에 따라 식별자의 이름을 지정한다.
- 2) 이름 지정 규약을 사용할 수 있습니다.(예:v_name 는 변수를 나타내고 c_name 는 상수를 나타낸다)
- 3) NOT NULL로 지정된 변수를 초기화 합니다.
- 4) 지정 연산자(:=)를 사용하거나 예약어 DEFAULT를 사용하여 식별자를 초기화 합니다.
- 5) 한 라인에 하나의 식별자만 선언 합니다.
- 6) 상수 선언에서 CONSTRAINT는 형 지정자보다 먼저 기술되어야 한다.

1.5 이름 지정 규칙

- 1) 하나의 블록에서 동일 이름의 변수를 선언할 수 없습니다.
- 2) 블록이 다르면 동일 이름을 선언할 수 있습니다. 객체들이 동시에 존재하는 곳에서는 현재 블록에서 정의된 객체들만이 사용될 수 있습니다.
- 3) 변수에 대한 이름을 블록에서 사용되는 테이블 열의 이름과 동일하게 선택해서는 안 됩니다. PL/SQL 변수가 SQL 명령에서 사용되고 열과 동일 이름을 가지면 ORACLE SERVER은 참조되는 열로 간주 합니다.

1.6 변수의 값 지정

변수의 값을 지정하거나 재지정하기 위해 PL/SQL 지정 문자를 사용합니다. 지정 연산자 (:=)의 좌측에 새 값을 받기 위한 변수를 적습니다.

1.6.1 Syntax

```
identifier := expression;
```

1.6.2 사용 예

```
DECLARE
    v_hiredate      DATE;
    v_ename         VARCHAR2(10);
BEGIN
    v_hiredate := '30-DEC-98';
    v_ename := 'MADURO';
```

♣ 주의

Oracle Server에서 DEFAULT 날짜 형식 설정이 데이터베이스마다 차이가 있을 수 있기 때문에 DEFAULT 날짜 형식을 알아야 한다. DEFAULT 날짜 형식이 YY-MM-DD 이면 v_hiredate := '99-01-01'로 값을 설정하여야 한다. 일반적으로 날짜의 형식에 의존하지 않고 사용하고자 할 경우에는 v_hiredate := TO_DATE('99-01-01', 'YY-MM-DD')을 사용한다.

1.7 스칼라 데이터 형

- 1) 단일 값을 유지 합니다.
- 2) 내부적인 구성 요소는 없습니다.

1.7.1 기본 스칼라 데이터 형

데이터 형	설 명
VARCHAR2(n)	변수 길이 문자 데이터에 대한 기본형은 32767Byte 까지입니다. VARCHAR2 변수와 상수에 대한 디플트 크기는 없습니다.
NUMBER(p,s)	고정(fixed)과 유동(floating)포인트 숫자에 대한 기본형
DATE	날짜와 시간에 대한 기본형. DATE 값은 지정 이후의 초 단위로 날에 대한 시간을 포함합니다. 날짜의 범위는 BC 4712년 1월 1일부터 AD 9999년 12월 31일사이입니다.
CHAR(n)	고정 길이 문자에 대한 기본형은 32767 바이트까지입니다. 지정하지 않는다면 디플트 길이는 1로 설정됩니다.
LONG	고정 길이 문자에 대한 기본형은 32760 바이트까지입니다. LONG 데이터베이스 열의 최대 폭은 2147483647 바이트입니다.
LONG RAW	이진 데이터와 바이트 문자열에 대한 기본형은 32760Byte 까지입니다.

	다. LONG RAW 데이터는 PL/SQL 에 의해 해석되지 않습니다.
BOOLEAN	계산에 사용되는 3 가지 가능한 값 가운데 기본형(TRUE,FALSE,NULL)
BINARY_INTEGER	-2147483647~2147483647 사이의 정수에 대한 기본 형
PLS_INTEGER	-2147483647~2147483647 사이의 signed 정수에 대한 기본형으로 PLS_INTEGER 같은 NUMBER 와 BINARY_INTEGER 값보다 적은 기억장치를 필요로 합니다.

♣ 참고

LONG 데이터 형은 LONG 값의 최대 길이가 32767 바이트인 것을 제외하고는 VARCHAR2 와 유사합니다. 그러므로 32760 바이트보다 더 긴 값은 LONG 데이터베이스 열에서 LONG PL/SQL 변수로 사용할 수 없습니다.

1.7.2 스칼라 변수 선언의 예

```
DECLARE
    v_job          VARCHAR2(9);
    v_count        BINARY_INTEGER := 0;
    v_total_sal   NUMBER(9,7) := 0;
    v_order_date  DATE := SYSDATE + 7;
    v_tax_rate    CONSTANT NUMBER(3,2) := 8.25;
    v_valid        BOOLEAN NOT NULL := TRUE;
    v_sex          CHAR(1);
```

1.7.3 %TYPE 속성

변수의 데이터 형과 정밀도를 직접 코딩하기 보다는 이전에 선언된 다른 변수 또는 데이터베이스 열에 맞추어 변수를 선언하기 위해 %TYPE 속성을 사용할 수 있습니다. 변수에 저장되는 값이 데이터베이스의 테이블에서 오거나 변수가 테이블에 쓰여지기로 되었다면 %TYPE 속성은 자주 사용됩니다. 변수 선언에서 필요한 데이터형 대신에 속성을 사용하려면 데이터베이스 테이블과 열 이름을 절두어로 사용합니다. 또한 이전에 선언된 변수를 참조한다면 속성 앞에 변수명을 기술합니다. 데이터베이스 수준에서 테이블의 데이터형을 변경하여도 PL/SQL 을 고칠 필요가 없습니다.

가) 사용 예

```
DECLARE
    v_empno        emp.empno%TYPE;
    v_ename        emp.ename%TYPE;
    v_deptno      emp.deptno%TYPE := 10;
```

1.7.4 BOOLEAN 변수 선언

- 1) TRUE, FALSE, NULL 값만을 BOOLEAN 변수에 대해 지정할 수 있습니다.
- 2) 변수는 논리연산자 AND, OR, NOT에 의해 접속 됩니다.
- 3) 변수는 항상 TRUE, FALSE, NULL을 생성 합니다.
- 4) 산술, 문자, 날짜 표현식은 BOOLEAN 값을 리턴하기 위해 사용될 수 있습니다.

가) 사용 예

```
DECLARE
    v_sal1      NUMBER(5) := 5000;
    v_sal2      NUMBER(5) := 6000;
    v_flag      BOOLEAN := TRUE;
BEGIN
    v_flag := (v_sal1 > v_sal2);
    IF v_flag THEN
        . . .
    END IF;
END;
```

1.8 조합 데이터 형(Composite Datatype)

Composite Datatype은 내부 구성 요소를 갖고 있고 PL/SQL에서 사용할 수 있는 Composite Datatype은 RECORD, TABLE, 중첩 TABLE, VARRAY입니다. RECORD 데이터형은 관련은 있으나 서로 다른 데이터형들을 논리적인 하나의 단위로 묶기 위해 사용하고 TABLE 데이터형은 전체 객체로써 데이터형이 같은 데이터의 모음을 참조하고 조작하기 위해 사용한다. 한번 정의되면 테이블과 레코드는 재이용할 수 있다.

1.8.1 PL/SQL TABLE TYPE

테이블형의 객체는 PL/SQL 테이블이라 불립니다. PL/SQL 테이블은 행에 대해 배열처럼 액세스하기 위해 기본키를 사용합니다. 배열과 유사하고 PL/SQL 테이블을 액세스하기 위해 BINARY_INTEGER 데이터형의 기본키와 PL/SQL 테이블 요소를 저장하는 스칼라 또는 레코드 데이터형의 열을 포함하여야 한다. 또한 이들은 동적으로 자유롭게 증가할 수 있습니다.

Primary key	Column
.
1	Jones
2	Smith
3	Maduro
.

BINARY_INTEGER

스칼라

가) Syntax

```

TYPE table_type_name IS TABLE OF
    {column_type | variable%TYPE | table.column%TYPE} [NOT NULL]
    [INDEX BY BINARY_INTEGER];
identifier      table_type_name;

```

table_type_name	테이블형의 이름
column_type	VARCHAR2, DATE, NUMBER 과 같은 스칼라 데이터 형
identifier	전체 PL/SQL 테이블을 나타내는 식별자의 이름

문제 2) TABLE 변수를 사용하여 EMP 테이블에서 이름과 업무를 출력하여라.

```

SET SERVEROUTPUT ON
DECLARE
    TYPE ename_table_type IS TABLE OF emp.ename%TYPE
        INDEX BY BINARY_INTEGER;
    TYPE job_table_type IS TABLE OF emp.job%TYPE
        INDEX BY BINARY_INTEGER;
    ename_table    ename_table_type;
    job_table     job_table_type;
    i             BINARY_INTEGER := 0;
BEGIN
    FOR k IN (SELECT ename, job FROM emp) LOOP
        i := i + 1;
        ename_table(i) := k.ename;
        job_table(i) := k.job;
    END LOOP;
    FOR j IN 1..i LOOP
        DBMS_OUTPUT.PUT_LINE(RPAD(ename_table(j),12) ||
            RPAD(job_table(j),9));
    END LOOP;
END;
/
SET SERVEROUTPUT OFF

```

1.8.2 PL/SQL RECORD TYPE

PL/SQL RECORD TYPE은 데이터베이스의 테이블 ROW와 다르고 3GL에서의 RECORD나 STRUCTURE와 유사하다. PL/SQL RECORD는 Scalar, PL/SQL RECORD, PL/SQL TABLE 데이터 타입 중 하나 이상의 요소를 갖고 있어야 하며, 다른 데이터 타입을 가질 수도 있다. 또한

FIELD(ITEM)들의 집합을 하나의 논리적 단위로 처리할 수 있게 해 주므로 테이블의 ROW를 읽어올 때 편리하다.

가) Syntax

```
TYPE type_name IS RECORD  
  (field_name1 {scalar_datatype|record_type} [NOT NULL] [{:= | DEFAULT} expr],  
   (field_name2 {scalar_datatype|record_type} [NOT NULL] [{:= | DEFAULT} expr],  
    . . . . .);  
  identifier_name      type_name;
```

type_name RECODE 형의 이름, 이 식별자는 RECODE 를 선언하기 위해 사용한다.
field_name RECODE 내의 필드명

나) RECORD 참조

RECORD 에서 필드는 이름으로 액세스 됩니다. 개별 필드를 참조하거나 초기화 하기 위해 “.”을 사용합니다.

```
Record_name.field_name
```

다) RECORD에 대한 값 할당

SELECT 또는 FETCH 문장을 사용함으로써 RECORD 에 값을 지정할 수 있다. 열이름의 RECORD 의 필드와 동일한 순서로 1 대 1 대응을 하여야 합니다. 두 RECORD 가 동일한 구조를 가지면 하나의 RECORD 를 다른 RECORD 에 지정할 수 있습니다.

```
Record_name1.field_name1 := expression;
```

```
Record_name1.field_name1 := Record_name2.field_name2;
```

문제 3) EMP 테이블에서 이름을 입력 받아 아래의 형태로 출력하는 SCRIPT를 작성하여라.

```
조회하고자 하는 사원의 이름을 입력하시오 : scott  
사원번호 : 7788  
이    름 : SCOTT  
담당업무 : ANALYST  
메    니    저 : 7566  
입사일자 : 09-DEC-82  
급    여 :  
보    너    스 : $0  
부서번호 : 20
```

```

SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_ename PROMPT '조회하고자 하는 사원의 이름을 입력하시오 : '
DECLARE
    TYPE emp_record_type IS RECORD(
        v_empno      emp.empno%TYPE,
        vENAME       emp.ename%TYPE,
        v_job        emp.job%TYPE,
        v_mgr         emp.mgr%TYPE,
        v_hiredate   emp.hiredate%TYPE,
        v_sal         emp.sal%TYPE,
        v_comm        emp.comm%TYPE,
        v_deptno     emp.deptno%TYPE);
        emp_record    emp_record_type;
        v_ename        emp.ename%TYPE := '&p_ename';
BEGIN
    SELECT *
        INTO emp_record
        FROM emp
        WHERE ename = UPPER(v_ename);
    DBMS_OUTPUT.PUT_LINE('사원번호 : ' || TO_CHAR(emp_record.v_empno));
    DBMS_OUTPUT.PUT_LINE('이    름 : ' || emp_record.vENAME);
    DBMS_OUTPUT.PUT_LINE('담당업무 : ' || emp_record.v_job);
    DBMS_OUTPUT.PUT_LINE('메 니 저 : ' || TO_CHAR(emp_record.v_mgr));
    DBMS_OUTPUT.PUT_LINE('입사일자 : ' || TO_CHAR(emp_record.v_hiredate));
    DBMS_OUTPUT.PUT_LINE('급    여 : ' ||
        LTRIM(TO_CHAR(emp_record.v_sal,'$999,990.00')));
    DBMS_OUTPUT.PUT_LINE('보너스 : ' ||
        LTRIM(TO_CHAR(NVL(emp_record.v_comm,0),'$999,990')));
    DBMS_OUTPUT.PUT_LINE('부서번호 : ' || TO_CHAR(emp_record.v_deptno));
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '의 자료는 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '자료가 2건 이상입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

☞ Guidelines

- 1) 각 RECORD는 필요한 만큼 많은 필드를 가질 수 있다.
- 2) RECORD는 초기값을 지정할 수 있고 NOT NULL로 정의될 수 있다.
- 3) 초기값이 없는 필드는 NULL로 초기화 됩니다.
- 4) DEFAULT 키워드는 필드를 정의할 때 사용될 수 있습니다.

- 5) 임의의 블록 서브 프로그램, 패키지의 선언 부분에 RECORD 형을 정의하고 사용자 정의 RECORD를 선언할 수 있다.
- 6) 중첩 RECORD를 선언하고 사용할 수 있다. RECORD는 다른 RECORD의 구성 요소가 될 수 있다.

1.8.3 %ROWTYPE의 속성

데이터베이스의 테이블 또는 VIEW의 일련의 열을 RECORD로 선언하기 위하여 %ROWTYPE를 사용합니다. 데이터베이스 테이블 이름을 %ROWTYPE 앞에 접두어를 붙여 RECORD를 선언하고 FIELD는 테이블이나 VIEW의 COLUMN명과 데이터 타입과 LENGTH을 그대로 가져올 수 있다.

가) Syntax

identifier	reference%ROWTYPE;
------------	--------------------

identifier RECORD에 대해 지정된 이름

reference RECORD의 기초가 되는 테이블, VIEW, CURSOR, 변수 명을 기술

나) 개별 필드를 참조하는 방법

record_name.field_name

다) %ROWTYPE을 사용 시 장점

- 1) 알지 못하는 데이터베이스 COLUMN의 개수와 데이터 형식을 모르게 지정할 수 있다.
- 2) 실행 시 변경되는 데이터베이스 COLUMN의 개수와 데이터 형식을 지정할 수 있다.
- 3) SELECT 문장으로 행을 검색할 때 유리하다.

문제 4) EMP 테이블에서 이름을 입력 받아 아래의 형태로 출력하는 SCRIPT를 작성하여라.

조회하고자 하는 사원의 이름을 입력하시오 : scott

사원번호 : 7788

이 름 : SCOTT

담당업무 : ANALYST

메 니 저 : 7566

입사일자 : 09-DEC-82

급 여 :

보 너 스 : \$0

부서번호 : 20

```

SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_ename PROMPT '조회하고자 하는 사원의 이름을 입력하시오 : '
DECLARE
    emp_record      emp%ROWTYPE;
    v_ename         emp.ename%TYPE := '&p_ename';
BEGIN
    SELECT *
        INTO emp_record
        FROM emp
        WHERE ename = UPPER(v_ename);
    DBMS_OUTPUT.PUT_LINE('사원번호 : ' || TO_CHAR(emp_record.empno));
    DBMS_OUTPUT.PUT_LINE('이    름 : ' || emp_record.ename);
    DBMS_OUTPUT.PUT_LINE('담당업무 : ' || emp_record.job);
    DBMS_OUTPUT.PUT_LINE('메니저 : ' || TO_CHAR(emp_record.mgr));
    DBMS_OUTPUT.PUT_LINE('입사일자 : ' || TO_CHAR(emp_record.hiredate));
    DBMS_OUTPUT.PUT_LINE('급    여 : ' ||
                           LTRIM(TO_CHAR(emp_record.sal,'$999,990.00')));
    DBMS_OUTPUT.PUT_LINE('보너스 : ' ||
                           LTRIM(TO_CHAR(NVL(emp_record.comm,0),'$999,990')));
    DBMS_OUTPUT.PUT_LINE('부서번호 : ' || TO_CHAR(emp_record.deptno));
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '의 자료는 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '자료가 2 건 이상입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

1.9 LOB Datatype 변수

LOB Datatype 변수는 Oracle8 데이터형으로 구조화 되지 않은 데이터(텍스트, 그래픽, 이미지, 비디오 클립, 소리 웨이브 폼 같은) 블록을 4기가 바이트 크기까지 저장할 수 있습니다. LOB Datatype은 데이터에 대한 랜덤 액세스를 지원합니다.

- 1) CLOB(character large object) 데이터형은 데이터베이스 내의 단일 바이트 문자 데이터의 대형 블록을 저장하기 위해 사용한다.
- 2) BLOB(binary large object) 데이터형은 행의 안팎에 데이터베이스 내의 대형 이진 객체를 저장하기 위해 사용됩니다.
- 3) BFILE(binary file) 데이터형은 데이터베이스 외부의 운영 시스템 파일의 대형 이진 객체를 저장하기 위해 사용됩니다.
- 4) NCLOB(national language character large object) 데이터형은 데이터베이스 내의 단

일 바이트, 또는 고정 길이의 멀티바이트 NCHAR 데이터를 행의 안팎에 저장하기 위해 사용됩니다.

1.10 바인드 변수

바인드 변수는 호스트 환경에서 선언된 변수이며, 실행 시간 값을, 그것이 숫자이든지 또는 문자이든지 임의의 다른 변수처럼 그것을 사용할 수 있는 하나 이상의 PL/SQL 프로그램의 내부나 외부에서 전달하기 위해 사용합니다. 문장이 프로시저, 함수, 패키지 안에 있지 않는다면, 호출 환경 또는 호스트에서 선언된 변수를 PL/SQL 문장에서 참조할 수 있습니다. 이것은 실행 컴파일러 프로그램에서 선언된 호스트 언어 변수, Developer/2000Forms 어플리케이션의 화면 필드, SQL*Plus 바인드 변수를 포함합니다.

1.10.1 바인드 변수 생성

SQL*Plus 환경에서, 바인드 변수를 선언하기 위해 VARIABLE 명령을 사용합니다. SQL 과 SQL*Plus는 바인드 변수를 참조할 수 있고 SQL*Plus는 그 값을 출력할 수 있습니다.

가) Syntax

```
VAR[!ABLE] [ variable [ NUMBER | CHAR (n) | VARCHAR2 (n) ] ]
```

1.10.2 바인드 변수 출력

SQL*Plus 환경에서 바인드 변수의 현재 값을 출력하기 위해 PRINT 명령을 사용합니다.

```
SQL> VARIABLE return_code NUMBER
SQL> DECLARE
 2 BEGIN
 3   :return_code := 100;
 4 END;
 5 /
PL/SQL procedure successfully completed.
SQL> PRINT return_code
RETURN_CODE
-----
100
```

1.11 Non-PL/SQL 변수 참조

호스트 변수를 참조하기 위해 선언된 PL/SQL 변수와 호스트 변수를 구별하기 위해 콜론(:)으로 참조 접두어를 기술하여야 한다.

```
:host_var1 := v_sal;
:global_val1 := 'YES';
```

2. PL/SQL 블록

PL/SQL은 프로그램을 논리적인 블록으로 나누게 하는 구조화된 블록 언어이다. PL/SQL 블록에서 사용하는 변수는 블록에 대해 논리적으로 선언할 수 있고 사용할 수 있다. 변수들과 그 외의 식별자를 참조하고 선언함으로써 PL/SQL 블록 내에서 값을 저장하고 변경할 수 있는 논리적인 단위이다.

2.1 PL/SQL 블록 구문과 지침

PL/SQL은 SQL의 확장이기 때문에 SQL에 적용하는 일반적인 구문은 PL/SQL 언어에 대해서도 적용 가능하다. 문장은 몇 라인 이상 계속될 수 있습니다. 문법적인 요소는 공백으로 분리될 수 있습니다.

2.1.1 식별자

식별자는 상수, 변수, 예외, 커서, 커서 변수, SUBPROGRAM, 패키지를 포함하는 PL/SQL 프로그램 항목과 요소를 명명하기 위해 사용됩니다.

- 1) 식별자는 30 문자까지 포함할 수 있지만 알파벳 문자로 시작해야 합니다.
- 2) 블록에서 사용된 테이블의 열 이름과 동일한 이름으로 식별자를 사용할 수 없습니다.
- 3) PL/SQL 식별자가 동일 SQL 명령에 있고 열로 동일 이름을 가지고 있다면, 이때 ORACLE은 참조 중인 열로 간주합니다.
- 4) 예약어는 더블 인용 부호에 둘러싸여 있지 않으면 식별자로서 사용될 수 없습니다.
- 5) 예약어는 읽기 쉽도록 대문자로 사용합니다.

2.1.2 구분 문자

구분 문자는 PL/SQL에 대한 특수한 의미를 가지는 단순 상징이거나 혼합 상징입니다.

단순 상징		혼합 상징	
기호	의미	기호	의미
+	덧셈 연산자	<>	관계형 연산자
-	뺄셈 연산자	!=	관계형 연산자
*	곱셈 연산자		접속 연산자
/	나눗셈 연산자	--	단일 라인 주석 지시자
=	관계형 연산자	/*	주석 구분 문자 시작
@	원격 액세스 지시자	*/	주석 구분 문자 종료
:	문장 종결자	:=	지정 연산자(치환 연산자)

2.1.3 Literal

Literal은 식별자(identifier)로 표현되지 않은 숫자, 문자, 문자열, BOOLEAN 값입니다.

- 1) 문자 Literal은 PL/SQL 문자 집합 내에서 인쇄 가능한 모든 문자를 포함합니다.

- 2) 숫자 Literal 은 단순 값(예:-32.5) 또는 과학적인 표기법

2.1.4 주석 코드

주석 코드 각 단계를 문서화하고 디버깅을 돋기 위해 코드에 주석을 기술합니다. 주석이 단일 라인에 있으면 두 개의 대쉬(--)을 기술하면 – 뒤에 기술된 것은 주석으로 인식 하고, 주석 범위가 여러 줄이라면 기호 /*와 */사이에 주석을 기술 한다. 주석은 철저하게 정보를 제공해야 하고 기능적인 논리 또는 데이터에 대한 어떤 조건 또는 기능을 강요해서는 안됩니다. 좋은 주석은 읽기 쉽게 하고 코드 유지를 위해 매우 중요합니다.

```
DECLARE
    v_sal    NUMBER(9,2);
BEGIN
    /* Compute the annual salary based on
       the monthly salary input from the user. */
    v_sal := v_sal * 12;
END;    -- This is the end of the transaction.
/
```

2.1.5 PL/SQL에서 SQL 함수

- 1) SQL에서 이용 가능한 대부분의 함수는 PL/SQL 표현식에서도 유효합니다.
- ① 단일 행 숫자 함수
 - ② 단일 행 문자 함수
 - ③ 데이터형 변환 함수
 - ④ 데이터 형식 그 밖의 함수
- 2) 아래 함수는 절차적인 문장에서는 사용 불가능합니다.
- ① GREATEST, LEAST, DECODE.
 - ② 그룹 함수(AVG, MIN, MAX, COUNT, SUM, STDEV, VARIANCE)는 테이블에서 행 그룹에 적용되므로 PL/SQL 블록에 있는 SQL 문장에서만 이용 가능합니다.

```
V_total := SUM(number_table); -- Error 가 발생한다.
```

2.2 데이터형 변환

PL/SQL은 데이터형이 문장에서 혼합되었다면 동적으로 데이터형 변환을 시도 합니다. 예를 들면 NUMBER 값을 CHAR 변수가 지정되었다면 그것이 CHAR 변수에 저장될 수 있도록 하기 위해 PL/SQL은 동적으로 숫자를 문자로 변환 합니다. 문자 표현을 숫자 값으로 나타내는 역의 상황도 적용됩니다. DATE 변수에 대해 문자 값을 지정할 수 있고 역의 상황도 적용됩니다.

니다. 혼합된 데이터형이 표현식에서 생성되면 데이터를 전환하기 위해 해당 변환 함수를 사용하여야 합니다.

2.2.1 데이터형 변환 함수의 종류

- 1) TO_CHAR(value, fmt) : value를 문자로 전환
- 2) TO_DATE(value, fmt) : value를 날짜 형식으로 전환
- 3) TO_NUMBER(value, fmt) : value를 숫자로 전환

☞ Guidelines

PL/SQL은 가능한 한 데이터형 변환을 시도하지만 성공은 수행 중인 작업에 달려있다. 명시적으로 데이터의 형 변환을 시도하는 것은 상당히 성능에 영향을 미치고 소프트웨어 버전이 변경되더라도 유효하게 유지될 수 있기 때문에 좋은 프로그램이 될 수 있다.

2.3 중첩 블록과 변수 범위

- 1) 문장은 실행 명령이 허용하는 곳 어디에서든지 중첩될 수 있습니다.
- 2) 중첩 블록은 하나의 문장이 됩니다.
- 3) 예외 �кции에서도 중첩 블록을 사용할 수 있습니다.
- 4) 객체 범위는 객체를 참조할 수 있는 프로그램 영역입니다.
- 5) 선언된 변수를 실행 �кции에서 참조할 수 있습니다.
- 6) 한정시키지 않은 변수는 참조할 수 있는 영역에서 변수를 참조 가능합니다.
 - ① 블록은 둘러싸는 블록을 참조할 수 있습니다.
 - ② 블록은 둘러싸인 블록을 참조할 수 없습니다.

```
DECLARE
    x      NUMBER;
    y      NUMBER;
BEGIN
    x := 1;
    y := 2;
    -- 이곳에서 x, y의 값은 x := 1, y := 2
    DECLARE
        y      NUMBER;
        z      NUMBER;
    BEGIN
        -- 이곳에서 x, y, z의 값은 x := 1, y := NULL, z := NULL
        x := 3;
        y := 4;
        z := 5;
        -- 이곳에서 x, y, z의 값은 x := 3, y := 4, z := 5
    END;
    -- 이곳에서 x, y, z의 값은 x := 3, y := 2, z는 참조 불가능하다.
```

```
END;  
/
```

2.4 PL/SQL에서 연산자

연산자는 논리, 산술, 연결, 연산 제어 명령인 괄호, 지수 연산자가 있다.

2.4.1 연산 명령

표현식에서의 연산은 그것들의 우선 순위에 따라 특별한 순서로 행해 집니다.

우선순위	연 산 자	설 명
1	**, NOT	지수 승, 논리 부정 연산자
2	+, -	식별, 부정 연산자
3	*, /	곱셈, 나눗셈 연산자
4	+,-,	덧셈, 뺄셈, 연결 연산자
5	=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	비교 연산자
6	AND	논리곱
7	OR	논리합

2.4.2 사용 예

```
DECALRE  
    v_cnt          NUMBER := 0;  
    v_eq           BOOLEAN;  
    v_valid        BOOLEAN;  
BEGIN  
    V_cnt := v_cnt + 1;  
    v_eq := (v_n1 = v_n2);  
    v_valid := (v_empno IS NOT NULL);  
END;  
/
```

☞ Guidelines

NULL로 작업할 때 아래 규칙을 명심함으로써 몇 가지 일반적인 실수를 피할 수 있습니다.

- 1) 관계 연산자로 NULL을 비교하면 항상 NULL이 됩니다.
- 2) NULL에 논리 연산자 NOT을 적용하면 항상 NULL이 됩니다.

3) 조건 제어 문장에서 조건이 NULL 이 되면 관련된 문장은 실행되지 않습니다.

2.5 프로그래밍 지침 사항

PL/SQL 블록을 개발할 때 명확한 코드 생성과 유지를 경감하기 위하여 프로그래밍 지침을 수행 합니다.

2.5.1 코드 규약

범 주	대소문자 규약	사용 예
SQL 명령어	대문자	SELECT, INSERT, UPDATE
PL/SQL 키워드	대문자	DECLARE, BEGIN, END
데이터형	대문자	NUMBER, VARCHAR2, CHAR
식별자와 매개변수	소문자	v_sal, v_ename, v_job
데이터베이스 테이블과 열명	소문자	emp, dept, salgrade

2.5.2 코드명 지정 규약

식별자	명명 규약	사용 예
변수	v_name	v_sal
상수	c_name	c_company_name
커서	name_cursor	emp_cursor
예외	e_name	e_too_many
테이블 형	name_table_type	amount_table_type
테이블	name_table	order_total_table
레코드 형	name_record_type	emp_record_type
레코드	name_record	customer_record
SQL*Plus 치환 매개변수	p_name	p_sal
SQL*Plus 전역변수	g_name	g_year_sal

2.5.3 코드 들여쓰기

명확성을 위해 또는 읽기 쉽도록 하기 위해 코드를 각 단계별로 들여 씁니다. 구조를 보여주기 위해 Carriage return을 사용합니다.

```
DECLARE
    v_deptno emp.deptno%TYPE;
BEGIN
    SELECT deptno
        INTO v_deptno
        FROM emp
        WHERE empno = 7788;
```

```
IF v_deptno IN (10,20) THEN
    UPDATE emp
        SET sal = 3500
        WHERE empno = 7788;
ELSE
    UPDATE emp
        SET sal = 2500
        WHERE empno = 7788;
END IF;
END;
/
```

◆ 연습문제 ◆

1. %TYPE 와 %ROWTYPE 를 사용시 장점을 설명하여라.
2. PL/SQL 에서 사용할 수 없는 데이터형이 있는가. 있다면 어떤 데이터형인가?
3. 아래의 선언문들을 평가하여라.

```
DECLARE
    v_id      NUMBER;
DECLARE
    v_emp, v_sal, v_comm      NUMBER;
DECLARE
    v_flag      BOOLEAN := 1;
DECLARE
    TYPE emp_table_type IS TABLE OF VARCHAR2(20)
        INDEX BY BINARY_INTEGER;
    emp_table    emp_table_type;
```

4. 아래와 같은 RECORD TYPE 의 변수를 선언하여라.

v_empno	v_ename	v_job	v_sal	v_comm
NUMBER(4)	VARCHAR2(10)	VARCHAR2(9)	NUMBER(7,2)	NUMBER(7,2)

5. PL/SQL 에서 사용하는 연산자의 종류를 설명하시오.

6. NULL 값이 연산에 참여하면 결과는 어떻게 되는가 ?

7. PL/SQL 에서는 들여쓰기를 권장한다 그 이유를 설명하여라.

1. PL/SQL에서 SQL 문장

데이터베이스에서 정보를 추출할 필요가 있을 때 또는 데이터베이스로 변경된 내용을 적용할 필요가 있을 때 SQL 을 사용합니다. PL/SQL 은 SQL 에 있는 DML 과 TRANSACTION 제어 명령을 모두 지원합니다. 테이블의 행에서 질의된 값을 변수에 할당 시키기 위해 SELECT 문장을 사용합니다. DML 문장은 다중 행 처리를 할 수 있지만 SELECT 문장은 하나의 행만을 처리할 수 있습니다.

1.1 PL/SQL에서 SQL 문장 사용

- 1) SELECT 명령어를 사용하여 데이터베이스에서 한 행의 데이터를 추출합니다.
- 2) DML 명령어를 사용하여 데이터베이스의 행에 대해 간신할 수 있습니다.
- 3) COMMIT,ROLLBACK,SAVEPOINT 명령어로 TRANSACTION 을 제어 합니다.
- 4) 암시적인 커서로 DML 결과를 결정합니다.

1.1.1 SQL과 PL/SQL 문장의 유형 비교

- 1) PL/SQL 블록은 TRANSACTION 단위가 아닙니다. COMMIT,ROLLBACK,SAVEPOINT 는 블록과는 독립적이지만 블록에서 이 명령어를 사용할 수 있습니다.
- 2) PL/SQL 은 CREATE TABLE, ALTER TABLE, DROP TABLE 같은 DDL 을 지원하지 않습니다.
- 3) PL/SQL 은 GRANT,REVOKE 과 같은 DCL 을 지원하지 않습니다.

1.2 PL/SQL에서 SELECT 문장

데이터베이스에서 데이터를 읽어 들이기 위해 SELECT 문장을 사용합니다. SELECT 문장은 INTO 절이 필요한데, INTO 절에는 데이터를 저장할 변수를 기술한다. SELECT 절에 있는 Column 수와 INTO 절에 있는 변수의 수는 좌측부터 1 대 1 대응을 하며 개수와 데이터의 형, 길이가 일치하여야 한다. SELECT 문은 INTO 절에 의해 하나의 행만을 저장할 수 있다. 그러므로 SELECT 문장에서 조건을 만족하는 ROW 가 한 개도 없거나 여러 행이 있으면 에러를 발생한다.

1.2.1 Syntax

SELECT	select_list
INTO	{variable_name1[,variable_name2,...] record_name}
FROM	table_name
WHERE	condition;

select_list 열의 목록이며 행 함수, 그룹 함수, 표현식을 기술할 수 있다.

variable_name 읽어 들인 값을 저장하기 위한 스칼라 변수

record_name 읽어 들인 값을 저장하기 위한 PL/SQL RECORD 변수

Condition PL/SQL 변수와 상수를 포함하여 열명, 표현식, 상수, 비교 연산자로 구성되며 오직 하나의 값을 RETURN 할 수 있는 조건이어야 한다.

♣ 참고

질의는 하나의 행만 RETURN 해야 합니다. PL/SQL 블록 내의 SELECT 문장은 다음 규칙을 적용하는 Embedded SQL 의 ANSI 범주에 속합니다. 질의의 결과는 하나의 행만을 RETURN 해야 하고 하나의 행 이상 또는 행이 없는 것은 에러를 생성합니다. PL/SQL 은 NO_DATA_FOUND 와 TOO_MANY_ROWS 를 예외로 블록의 예외 섹션에서 추적할 수 있는 표준 예외를 조성하여 처리 합니다.

- 1) SELECT 문 사용시 한개 이상의 ROW 가 검색되면 Oracle8 Server 는 미리 정해진 EXCEPTION 인 TOO_MANY_ROWS 라고 부르는 에러 번호 -1422 를 발생한다.
- 2) SELECT 문 사용시 아무런 ROW 도 검색되지 않으면 Oracle8 Server 는 미리 정해진 EXCEPTION 인 NO_DATA_FOUND 라고 부르는 에러 번호 +1403 인 발생한다.

문제 1) 이름을 입력받아 급여와 입사일을 출력하는 SCRIPT 를 작성하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_name PROMPT '사원의 이름을 입력하시오 : '
DECLARE
    v_name          emp.ename%TYPE := UPPER('&p_name');
    v_sal           emp.sal%TYPE;
    v_hiredate     emp.hiredate%TYPE;
BEGIN
    SELECT sal,hiredate
        INTO v_sal,v_hiredate
        FROM emp
       WHERE ename = v_name;
    DBMS_OUTPUT.PUT_LINE('급 여 : ' || LTRIM(TO_CHAR(v_sal,'$999,999')));
    DBMS_OUTPUT.PUT_LINE('입사일 : ' || TO_CHAR(v_hiredate,'YYYY-MM-DD'));
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 자료가 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 자료가 여러행이 존재합니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러가 발생했습니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

♣ 참고

위 문제에서 SET VERIFY OFF 은 old 와 new 값을 출력하지 않고, SET SERVEROUTPUT ON 은 DBMS_OUTPUT 이라는 PACKAGE 내의 PUT_LINE 함수를 사용한다. 이 함수를 SQL*Plus 에서 사용하려면 환경 변수를 사용하기 위하여 사용하였다.

문제 2) 부서번호를 입력받아 급여의 합을 출력하는 SCRIPT를 작성하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_deptno PROMPT '부서번호를 입력하시오(급여의 합을 구함) : '
DECLARE
    v_sal_total      NUMBER;
BEGIN
    SELECT SUM(sal)
        INTO v_sal_total
        FROM emp
        WHERE deptno = &p_deptno;
    DBMS_OUTPUT.PUT_LINE(&p_deptno || '번 부서 급여의 합 : ' || 
                         LTRIM(TO_CHAR(v_sal_total,'$99,999,999')));
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

☞ Guidelines

- 1) 세미콜론(;)으로 개별 SQL 문장을 종료한다.
- 2) SELECT 문장이 PL/SQL에 내장될 때 INTO 절을 사용한다.
- 3) WHERE 절은 선택적이며 입력변수, 상수, 리터럴, 또는 PL/SQL의 표현식을 지정하기 위해 사용될 수 있다.
- 4) SELECT 절에서의 데이터베이스 열과 INTO 절에서의 출력 변수의 수를 동일하게 좌측부터 1 대 1 대응되게 지정해야 한다.
- 5) 식별자의 데이터형과 열의 데이터형을 갖도록 보증하기 위해 %TYPE 속성을 사용합니다. INTO 절의 변수의 데이터형과 변수의 수는 SELECT에서 기술한 Column의 수와 일치하야 합니다.
- 6) 그룹 함수는 테이블의 행 그룹에 적용되기 때문에 SUM 같은 그룹 함수는 SQL에서 사용합니다.

1.3 PL/SQL을 이용한 데이터 조작

DML 명령어를 사용하여 데이터베이스 테이블에 대한 내용을 변경할 수 있다.

- 1) INSERT 문장은 테이블에 데이터의 새로운 행을 추가한다.
- 2) UPDATE 문장은 테이블에 존재하는 행을 수정한다.
- 3) DELETE 문장은 테이블에서 원치 않는 행을 제거한다.

1.3.1 데이터 삽입

- 1) USER와 SYSDATE 같은 SQL 함수를 사용합니다.
- 2) 데이터베이스 시퀀스를 사용하여 기본키 값을 생성합니다.

3) PL/SQL 블록에서 값을 얻거나 DEFAULT 값을 이용합니다.

문제 3) 초기값이 8000 부터 9999 까지 1 씩 증가하는 SEQUENCE(EMPNO_SEQUENCE)를 생성하여 EMP 테이블에 등록하는 SCRIPT 를 작성하여라. 단 이름은 JONG, 업무는 MANAGER, 부서번호는 10 이다.

```
SQL> CREATE SEQUENCE empno_sequence
  2  INCREMENT BY 1
  3  START WITH 8000
  4  MAXVALUE 9999
  5  NOCYCLE
  6  NOCACHE;

Sequence created.

SQL> DECLARE
  2    v_empno      emp.empno%TYPE;
  3  BEGIN
  4    SELECT empno_sequence.NEXTVAL
  5      INTO v_empno
  6      FROM dual;
  7    INSERT INTO emp(empno,ename,job,deptno)
  8      VALUES (v_empno,'JONG','MANAGER',10);
  9  END;
10 /

PL/SQL procedure successfully completed.
```

1.3.2 데이터 간접

- 1) 지정 연산자 좌측에 있는 식별자는 항상 데이터베이스 열이지만 오른쪽에 있는 식별자도 데이터베이스 열 또는 PL/SQL 에서 사용되는 변수도 기술 가능하다.
- 2) PL/SQL 에서의 SELECT 문장과 달리 수정된 행이 없으면 예러가 발생하지 않는다.

문제 4) 사원번호가 7369 인 사원의 급여에 1000 을 더하여 간접하여라.

```
DECLARE
  v_sal      emp.sal%TYPE := 1000;
BEGIN
  UPDATE emp
    SET sal = sal + v_sal
   WHERE empno = 7369;
END;
/
```

1.3.3 데이터 삭제

PL/SQL 에서 SQL 의 DELETE 문장을 사용하여 필요 없는 자료를 삭제할 수 있다.

문제 5) 사원번호가 7654 인 사원의 정보를 삭제하여라.

```
DECLARE
    v_empno emp.empno%TYPE := 7654;
BEGIN
    DELETE emp
        WHERE empno = v_empno;
END;
/
```

1.4 이름 지정 규약

- 1) WHERE 절에서 모호성을 피하기 위해 이름 지정 규약을 사용한다.
- 2) 데이터베이스 열과 식별자는 다른 이름을 가져야 한다.
- 3) PL/SQL 이 테이블의 열을 첫번째로 조사하기 때문에 구문 오류가 발생할 수도 있다.

1.5 COMMIT 과 ROLLBACK 문장

COMMIT 또는 ROLLBACK SQL 문장으로 트랜잭션 논리를 제어 함으로써 데이터베이스를 영구적으로 변경하게 합니다. ORACLE SERVER 에서와 마찬가지로 DML 트랜잭션은 COMMIT 또는 ROLLBACK 을 수행한 다음에 시작하고 성공적인 COMMIT 또는 ROLLBACK 다음에 종료합니다.

```
DECLARE
    v_empno emp.empno%TYPE := 7934;
BEGIN
    DELETE emp
        WHERE empno = v_empno;
    COMMIT;
END;
/
```

1.6 SQL CURSOR

SQL 문장을 실행할 때마다 ORACLE SERVER 은 명령이 분석되고 실행되는 곳에서 메모리 영역을 개방합니다. 이 영역을 CURSOR 라 합니다. 블록의 실행 부분이 SQL 문장을 실행할 때 PL/SQL 은 SQL 식별자를 가지는 암시적 CURSOR 를 생성합니다. PL/SQL 은 자동적으로 이 CURSOR 를 관리합니다. 명시적 CURSOR 는 명시적으로 선언되고 프로그래머에 의해 명명됩니다.

- 1) CURSOR 는 개별 SQL 작업 영역입니다.
- 2) CURSOR 에는 임시적 커서와 명시적 커서가 있습니다.
- 3) ORACLE SERVER 은 SQL 문장을 분석하고 실행하기 위해 암시적 커서를 사용합니다.
- 4) 명시적 커서는 프로그램에 의해 명시적으로 선언 됩니다.

1.6.1 CURSOR 의 속성

SQL CURSOR 의 속성을 사용하여 SQL 문장의 결과를 테스트할 수 있다.

종 류	설 명
SQL%ROWCOUNT	가장 최근의 SQL 문장에 의해 영향을 받은 행의 수
SQL%FOUND	가장 최근의 SQL 문장이 하나 또는 그 이상의 행에 영향을 미친다면 TRUE로 평가한다.
SQL%NOTFOUND	가장 최근의 SQL 문장이 어떤 행에도 영향을 미치지 않았다면 TRUE로 평가한다.
SQL%ISOPEN	PL/SQL 이 실행된 후에 즉시 암시적 커서를 닫기 때문에 항상 FALSE로 평가된다.

문제 6) ITEM 테이블에서 OR DID 가 605 인 자료를 모두 삭제하여라.

```
VARIABLE rows_deleted VARCHAR2(60)
DECLARE
    v_ordid      NUMBER := 605;
BEGIN
    DELETE FROM item
        WHERE ordid = v_ordid;
    IF SQL%FOUND THEN
        :rows_deleted := SQL%ROWCOUNT || ' rows deleted.';
    ELSE
        :rows_deleted := '삭제한 자료가 없습니다.';
    END IF;
END;
/
PRINT rows_deleted
```

◆ 연습문제 ◆

1. EMP 테이블에서 급여가 최고인 사원의 이름, 급여를 출력하는 SCRIPT 를 생성하여라. 단 SQL*Plus 변수를 이용하여 화면에 출력한다.
2. DEPT 테이블에 등록하는 SCRIPT 를 작성하여라. 단 아래의 표를 참고하여라.

```
부서번호를 입력하시오. : 50  
부서명을 입력하시오. : 현대교육센터  
근무지를 입력하시오. : 마북리
```

```
PL/SQL procedure successfully completed.
```

3. DEPT 테이블에 등록되어 있는 자료 중 근무지를 변경하는 SCRIPT 를 작성하여라. 단 아래의 표를 참고하여라

```
부서번호를 입력하시오. : 50  
근무지를 입력하시오. : 강남
```

```
PL/SQL procedure successfully completed.
```

4. EMP 테이블에서 사원번호를 입력받아 자료를 삭제하는 SCRIPT 를 작성하여라.

5. EMP 테이블에서 하나의 자료를 입력할 수 있는 SCRIPT 를 작성하여라. 단 사원번호는 SEQUENCE 를 생성하여 입력한다.

6. SQL%ROWCOUNT 란 무엇을 의미하는가.

7. SQL CURSOR 란 ?

1. 개요

여러 가지 제어 구조를 이용하여 PL/SQL 블럭에 있는 문장들의 논리적 흐름을 변경할 수 있다. 조건에 의해 분기하는 IF 문을 이용한 조건 구조와 LOOPING 구조(조건 없이 반복하는 BASIC 루프, 계수를 이용하여 반복을 하는 FOR 루프, 문장이 TRUE 인 동안에 반복을 하는 WHILE 루프, 루프를 종료하는 EXIT 문)가 있다.

1.1 IF 문

PL/SQL 의 IF 문장은 다른 언어의 IF 문장과 거의 유사하다. 즉 일치하는 조건(TRUE,FALSE, NULL)에 따라 선택적으로 작업을 수행할 수 있게 해준다. TRUE 면 THEN 과 ELSE 사이의 문장을 수행하고 FALSE 나 NULL 이면 ELSE 와 END IF 사이의 문장을 수행한다.

1.1.1 Syntax

```
IF condition THEN  
    statements;  
  
[ELSIF condition THEN]  
    statements;  
  
[ELSE  
    statements;]  
  
END IF;
```

condition BOOLEAN 변수 또는 표현식을 기술할 수 있다.(TRUE,FALSE,NULL)

statements 하나 이상의 PL/SQL 또는 SQL 문장을 기술한다.

ELSIF 처음식이 FALSE 또는 NULL 일 경우 추가적인 조건이 필요한 경우에 사용

1.1.2 단순 IF 문장

조건이 TRUE 이면 THEN 이하의 문장을 실행하고 조건이 FALSE 나 NULL 이면 END IF 다음 문장을 수행한다.

가) Syntax

```
IF condition THEN  
    statements;  
END IF;
```

문제 1) 이름, 급여, 부서번호를 입력받아 EMP 테이블에 자료를 등록하는 SCRIPT 를 작성하여라. 단 10 번 부서일 경우 입력한 급여의 20%를 추가하고 초기값이 9000 부터 9999 까지 1 씩 증가하는 SEQUENCE(EMP_EMPNO_SEQ)작성하여 사용하고 아래의 표를 참고하여라.

이 름: 홍길동
급 여: 2000
부서번호: 10

```
SET VERIFY OFF
ACCEPT p_name PROMPT '이 름: '
ACCEPT p_sal PROMPT '급 여: '
ACCEPT p_deptno PROMPT '부서번호: '
DECLARE
    v_name          VARCHAR2(10) := UPPER('&p_name');
    v_sal           NUMBER(7,2) := &p_sal;
    v_deptno NUMBER(2) := &p_deptno;
BEGIN
    IF v_deptno = 10 THEN
        v_sal := v_sal * 1.2;
    END IF;
    INSERT INTO emp(empno,ename,sal,deptno)
        VALUES (empno_sequence.NEXTVAL,v_name,v_sal,v_deptno);
    COMMIT;
END;
/
SET VERIFY ON
```

☞ Guidelines

- 1) 충족하는 조건에 따라 선택적으로 작업을 수행할 수 있다.
- 2) 코드를 사용할 때 키워드의 철자를 바르게 기술하시오.(ELSIF, END IF)
- 3) 제어의 조건들이 TRUE 이면 THEN 과 END IF 사이의 관련된 문장들이 수행됩니다. 그러나 FALSE 나 NULL 이면 END IF 다음의 문장으로 제어가 넘어 갑니다.
- 4) ELSE 절은 한번만 사용 가능 합니다.
- 5) 명확성을 위해 조건적으로 실행되는 문장을 들여쓰기 하는 것이 좋습니다.

1.1.3 IF - THEN - ELSE 문장의 실행 흐름

조건이 TRUE 이면 THEN 부터 ELSE 사이의 문장을 수행하고 제어는 END IF 다음 문장으로 넘어가고 FALSE 나 NULL 이면 ELSE 부터 END IF 사이의 문장을 수행하고 제어는 END IF 다음의 문장으로 이동 된다.

가) Syntax

```
IF condition THEN
    statements;
ELSE
    statements;
END IF;
```

문제 2) 이름을 입력받아 그 사람의 업무가 'MANAGER', 'ANALYST'이면 급여의 50%를 가산하여 갱신하고 업무가 'MANAGER', 'ANALYST'이 아니면 20%를 가산하는 SCRIPT를 작성하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_name PROMPT '이름: '
DECLARE
    v_empno emp.empno%TYPE;
    v_name emp.ename%TYPE := UPPER('&p_name');
    v_sal emp.sal%TYPE;
    v_job emp.job%TYPE;
BEGIN
    SELECT empno, job
        INTO v_empno, v_job
        FROM emp
        WHERE ename = v_name;
    IF v_job IN ('MANAGER', 'ANALYST') THEN
        v_sal := v_sal * 1.5;
    ELSE
        v_sal := v_sal * 1.2;
    END IF;
    UPDATE emp
        SET sal = v_sal
        WHERE empno = v_empno;
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || '개의 행이 갱신되었습니다.');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 자료가 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 동명 이인입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러가 발생 했습니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.1.4 IF-THEN-ELSIF 문장의 실행 흐름

가능하면 중첩 IF 문장 대신 ELSIF 절을 사용하여라. 코드를 읽고 이해하기가 더 쉬우며 로직을 정확하게 식별됩니다. ELSE 절 안의 작업이 순수하게 다른 IF 문으로 구성된다면 이것은 ELSIF 절을 사용하는 것이 더욱 편리합니다. 조건과 수행이 각각 종료 시에 중첩 END IF에 대해 일일이 요구하지 않음으로써 코드를 더 명확하게 만들어 줍니다.

가) Syntax

```
IF condition THEN
    statements;
ELSIF condition THEN
    statements;
ELSIF condition THEN
    statements;
ELSE
    statements;
END IF;
```

문제 3) 이름을 입력받아 업무를 조회하여 업무별로 급여를 갱신하는 SCRIPT 를 작성하여라.

단 PRESIDENT:10%, MANAGER:20%, ANALYST:30%, SALESMAN:40%, CLERK:50%를 적용한다.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_name PROMPT '이름: '
DECLARE
    v_empno emp.empno%TYPE;
    v_name emp.ename%TYPE := UPPER('&p_name');
    v_sal emp.sal%TYPE;
    v_job emp.job%TYPE;
BEGIN
    SELECT empno, job
        INTO v_empno, v_job
        FROM emp
        WHERE ename = v_name;
    IF v_job = 'PRESIDENT' THEN
        v_sal := v_sal * 1.1;
    ELSIF v_job = 'MANAGER' THEN
        v_sal := v_sal * 1.2;
    ELSIF v_job = 'ANALYST' THEN
        v_sal := v_sal * 1.3;
    ELSIF v_job = 'SALESMAN' THEN
        v_sal := v_sal * 1.4;
    ELSIF v_job = 'CLERK' THEN
        v_sal := v_sal * 1.5;
    ELSE
        v_sal := NULL;
    END IF;
    UPDATE emp
```

```

        SET sal = v_sal
        WHERE empno = v_empno;
        DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || '개의 행이 갱신되었습니다.');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 자료가 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(v_name || '는 동명 이인입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러가 발생 했습니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

1.2 논리적 조건 설정

비교 연산자를 써서 숫자, 문자 또는 날짜 식을 결합한 간단한 논리 조건을 만든다. 일 반적으로 IS NULL 연산자로 NULL 값을 처리할 수 있다.

1.2.1 식과 비교에서 널(NULL) 논리적 조건 설정

널 값을 공 문자열로 처리하는 연결식(Concatenation)은 예외이지만 기타 다른 널 값을 포함하는 식은 널 값을 return하고 IS NULL 비교의 결과는 TRUE나 FALSE로 return된다.

1.2.2 논리 테이블

논리연산자 AND, OR, NOT 을 가지고 단순한 BOOLEAN 조건을 조합함으로써 복잡한 BOOLEAN 조건을 구축할 수 있다.

AND	TRUE	FALUE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALUE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

1.3 LOOP 문

LOOP 문은 일련의 문장(SQL,PL/SQL)들을 여러 번 반복하기 위해 많은 편의를 제공한다.

- 1) 조건 없이 반복 작업을 제공하기 위한 BASIC LOOP 문
- 2) COUNT 를 기본으로 작업의 반복 제어를 제공하는 FOR LOOP 문
- 3) 조건을 기본으로 작업의 반복 제어를 제공하기 위한 WHILE LOOP 문
- 4) LOOP 를 종료하기 위한 EXIT 문

1.4 BASIC LOOP 문

가장 간단한 루프는 구분 문자인 LOOP 와 END LOOP 사이에 반복되는 문장 부분들로 이루어져 있다. 실행상의 흐름이 END LOOP 에 도달할 때마다 그와 짹을 이루는 LOOP 문으로 제어가 되돌아간다. 이러한 루프를 무한 루프라 하며, 여기서 빠져나가려면 EXIT 문을 사용한다. 기본 LOOP 는 LOOP 에 들어갈 때 조건이 이미 일치했다 할지라도 적어도 한번은 문장이 실행된다.

1.4.1) Syntax

```
LOOP
    statement1;
    statement2;
    . . .
    EXIT [WHERE condition];
END LOOP;
```

1.5 EXIT 문

EXIT 문을 이용하면 END LOOP 문 다음 문으로 제어를 보내기 때문에 루프를 종료할 수 있다. EXIT 는 IF 문 내의 처리 작업으로서 또는 루프 내의 독립적인 문장으로서도 사용할 수 있다. 조건에 따라 루프를 종료할 수 있도록 WHEN 절을 덧붙일 수 있다. EXIT 문에 직면하게 되면 조건이 평가 됩니다. 조건이 TRUE 를 RETURN 하면 LOOP 을 끝내고 LOOP 후의 다음 문장으로 제어를 전달합니다. 기본 LOOP 는 여러 개의 EXIT 문장을 포함할 수 있다.

1.5.1) Syntax

```
EXIT [WHEN condition];
```

문제 4) LOOP 문으로 아래와 같이 출력하는 SCRIPT 를 작성하여라.

```
*****
PL/SQL procedure successfully completed.

SET SERVEROUTPUT ON
DECLARE
    v_cnt    NUMBER := 1;
    v_str    VARCHAR2(20) := NULL;
BEGIN
    LOOP
        v_str := v_str || '*';
```

```

        DBMS_OUTPUT.PUT_LINE(v_str);
        v_cnt := v_cnt + 1;
        IF v_cnt >= 20 THEN
            EXIT;
        END IF;
    END LOOP;
END;
/
SET SERVEROUTPUT OFF

```

문제 5) EVEN_ODD(ID:NUMBER(4) GUBUN:VARCHAR2(4)) 테이블을 작성하여 START 숫자와 END 숫자를 입력 받아 그사이의 숫자를 ID 에 ID 의 숫자가 짝수이면 GUBUN 에 “짝수”를 홀수이면 GUBUN 에 “홀수”라고 입력하는 SCRIPT 를 LOOP 문으로 작성하여라.

```

DROP TABLE even_odd;
CREATE TABLE even_odd(
    id      NUMBER(4) CONSTRAINT even_odd_id_pk PRIMARY KEY,
    gubun   VARCHAR2(4));
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_start PROMPT ' START 숫자를 입력하시오 : '
ACCEPT p_end   PROMPT ' END 숫자를 입력하시오 : '
DECLARE
    v_start even_odd.id%TYPE := &p_start;
    v_end   even_odd.id%TYPE := &p_end;
BEGIN
    IF &p_start > &p_end THEN
        DBMS_OUTPUT.PUT_LINE('START 가 END 보다 큽니다.');
    ELSE
        DELETE FROM even_odd;
        LOOP
            IF MOD(v_start,2) = 0 THEN
                INSERT INTO even_odd
                    VALUES (v_start,'짝수');
            ELSE
                INSERT INTO even_odd
                    VALUES (v_start,'홀수');
            END IF;
            v_start := v_start + 1;
            EXIT WHEN v_start > v_end;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(&p_start || '부터 ' || &p_end || '까지 ' ||
                            TO_CHAR(&p_end - &p_start + 1) ||
                            '간의 자료가 입력되었습니다.');
    END IF;
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

1.6 FOR LOOP 문

FOR LOOP는 기본 LOOP와 동일한 일반 구조를 가집니다. 그리고 PL/SQL이 수행되는 수를 정하기 위해 LOOP 키 워드 앞에 제어문을 기술합니다. FOR LOOP 문에서 사용되는 인덱스는 정수로 자동 선언되므로 따로 선언할 필요가 없다. FOR LOOP 문은 LOOP을 반복할 때마다 자동적으로 1씩 증가 또는 감소한다. REVERSE는 1씩 감소함을 의미한다.

1.6.1) Syntax

```
FOR index_counter IN [REVERSE] lower_bound..upper_bound LOOP
    statement1;
    statement2;
    .
    .
    .
END LOOP;
```

index_counter upper_bound나 lower_bound에 도달할 때까지 LOOP를 반복함으로써 1씩 자동적으로 증가하거나 감소되는 값을 가진 암시적으로 선언된 정수입니다.

REVERSE upper_bound에서 lower_bound 까지 반복함으로써 인덱스가 1씩 감소되도록 합니다.

lower_bound index_counter 값의 범위에 대한 하단 바운드값을 지정한다.

upper_bound index_counter 값의 범위에 대한 상단 바운드값을 지정한다.

☞ Guidelines

- 1) 일련의 문장들은 두 바운드에 의해 카운트가 결정되고 증가될 때마다 실행 됩니다.
- 2) 루프 범위의 하단 바운드와 상단 바운드는 리터럴, 변수, 표현식이 가능하지만 정수로 기술되어야 합니다.
- 3) 루프 범위의 하단 바운드가 상단 바운드보다 더 큰 값이 기술되면 일련의 문장들은 수행되지 않습니다.
- 4) 루프 내에서만 인덱스 카운터를 참조할 수 있다. 즉 루프 밖에서는 정의되지 않는다.
- 5) 인덱스 카운터의 값을 참조하기 위해서 표현식을 사용한다.
- 6) := 좌측에 인덱스 카운터를 기술할 수 없다.

문제 6) FOR 문으로 아래와 같이 출력하는 SCRIPT를 작성하여라.

```
*
**
.
.
.
*****
*****
```

```
PL/SQL procedure successfully completed.
```

```

SET SERVEROUTPUT ON
DECLARE
    v_str      VARCHAR2(10) := NULL;
BEGIN
    FOR i_idx IN 1..10 LOOP
        v_str := v_str || '*';
        DBMS_OUTPUT.PUT_LINE(v_str);
    END LOOP;
END;
/
SET SERVEROUTPUT OFF

```

문제 7) EVEN_ODD(ID:NUMBER(4) GUBUN:VARCHAR2(4)) 테이블을 작성하여 START 숫자와 END 숫자를 입력 받아 그사이의 숫자를 ID에 ID의 숫자가 짝수이면 GUBUN에 “짝수”를 훌수이면 GUBUN에 “홀수”라고 입력하는 SCRIPT를 FOR 문으로 작성하여라.

```

DROP TABLE even_odd;
CREATE TABLE even_odd(
    id      NUMBER(4) CONSTRAINT even_odd_id_pk PRIMARY KEY,
    gubun   VARCHAR2(4));
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_start PROMPT ' START 숫자를 입력하시오 : '
ACCEPT p_end   PROMPT ' END 숫자를 입력하시오 : '
DECLARE
BEGIN
    IF &p_start > &p_end THEN
        DBMS_OUTPUT.PUT_LINE('START 가 END 보다 큽니다.');
    ELSE
        DELETE FROM even_odd;
        FOR i_idx IN &p_start .. &p_end LOOP
            IF MOD(i_idx,2) = 0 THEN
                INSERT INTO even_odd
                    VALUES (i_idx,'짝수');
            ELSE
                INSERT INTO even_odd
                    VALUES (i_idx,'홀수');
            END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(&p_start || '부터 ' || &p_end || '까지 ' ||
                            TO_CHAR(&p_end - &p_start + 1) ||
                            '간의 자료가 입력되었습니다.');
    END IF;
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

1.7 WHILE LOOP 문

제어 조건이 TRUE 인 동안만 일련의 문장을 반복하기 위해 WHILE LOOP 문장을 사용한다. 조건은 반복이 시작될 때 체크하게 되어 LOOP 내의 문장이 한번도 수행되지 않을 경우도 있다. LOOP을 시작할 때 조건이 FALSE 이면 반복 문장을 탈출하게 된다.

1.7.1) Syntax

```
WHILE condition LOOP
    statement1;
    statement2;
    . . . .
END LOOP;
```

condition BOOLEAN 변수 또는 표현식을 기술(TRUE,FALSE,NULL)

문제 8) WHILE 문으로 아래와 같이 출력하는 SCRIPT 를 작성하여라.

```
*
**
. . . . .
*****
PL/SQL procedure successfully completed.
```

```
SET SERVEROUTPUT ON
DECLARE
    v_cnt    NUMBER := 1;
    v_str    VARCHAR2(10) := NULL;
BEGIN
    WHILE v_cnt <= 10 LOOP
        v_str := v_str || '*';
        DBMS_OUTPUT.PUT_LINE(v_str);
        v_cnt := v_cnt + 1;
    END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

문제 9) EVEN_ODD(ID:NUMBER(4) GUBUN:VARCHAR2(4)) 테이블을 작성하여 START 숫자와 END 숫자를 입력 받아 그사이의 숫자를 ID에 ID의 숫자가 짝수이면 GUBUN에 “짝수”를 홀수이면 GUBUN에 “홀수”라고 입력하는 SCRIPT를 WHILE 문으로 작성하여라.

```
DROP TABLE even_odd;
CREATE TABLE even_odd(
    id      NUMBER(4) CONSTRAINT even_odd_id_pk PRIMARY KEY,
    gubun   VARCHAR2(4));
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_start PROMPT 'START 숫자를 입력하시오 : '
ACCEPT p_end   PROMPT 'END 숫자를 입력하시오 : '
DECLARE
    v_start      even_odd.id%TYPE := &p_start;
    v_end        even_odd.id%TYPE := &p_end;
BEGIN
    IF v_start > v_end THEN
        DBMS_OUTPUT.PUT_LINE('START 가 END 보다 큽니다.');
    ELSE
        DELETE FROM even_odd;
        WHILE v_start <= v_end LOOP
            IF MOD(v_start,2) = 0 THEN
                INSERT INTO even_odd
                    VALUES (v_start, '짝수');
            ELSE
                INSERT INTO even_odd
                    VALUES (v_start, '홀수');
            END IF;
            v_start := v_start + 1;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(&p_start || '부터 ' || &p_end || '까지 ' ||
            TO_CHAR(&p_end - &p_start + 1) || |
            '건의 자료가 입력되었습니다.');
    END IF;
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.8 중첩 LOOP 와 레이블

여러 단계로 루프를 중첩할 수 있습니다. WHILE 루프 내에서 FOR 루프를, FOR 루프 내에서 WHILE 루프를 주첩할 수 있습니다. 대개 중첩 루프가 종결되면 예외가 발생하지 않는 한 둘 러싸는 루프가 종결되지 않습니다. 레이블은 같은 라인 또는 다음 라인에서 문장 앞에 위치됩니다. 레이블 구분 문자 안에 LOOP라는 글자 앞에 레이블을 위치 시킴으로써 루프를 레이블 시킵니다. 루프가 레이블이 되면 END LOOP 문장 후에 루프 이름을 선택적으로 쓸 수 있습니다.

1.8.1) 사용 예

```
BEGIN
    <<outer_loop>>
    LOOP
        v_count := v_count + 1;
        EXIT WHEN v_counter > 10;
        <<inner_loop>>
        LOOP
            . . .
            EXIT outer_loop WHEN total_done = 'YES';
            . . .
            EXIT WHEN inner_done = 'YES';
            . . .
        END LOOP inner_loop;
        . . .
    END LOOP outer_loop;
END;
```

◆ 연습문제 ◆

1. TEST 테이블을 생성하여 1부터 10 사이의 값을 입력하는데 3,6,7 을 제외하고 등록하여라. 단 TEST 테이블에는 ID(NUMBER(4)) column 만 존재한다.

2. EMP 테이블에 등록하는 SCRIPT를 작성하는데 아래의 조건을 만족하도록 한다.
 - ① SQL*Plus 치환 변수를 사용하여 사원번호, 이름, 급여, 부서번호를 입력받는다.
 - ② 급여가 0~999 이면 40%, 1000~1499 이면 30%, 1500~1999 이면 20%, 2000~2999 이면 10%, 3000 이상이면 5%의 보너스를 받는 것으로 한다.

3. EMP 테이블에서 이름을 입력받아 급여를 200 으로 나눈 숫자만큼 “*”를 출력하여야라.

4. LOOP 문장의 종류를 설명하여라.

5. START 숫자와 END 를 입력받아 그 안에 있는 숫자를 출력하여야라.

6. 중첩 LOOP 를 설명하여라.

7. LOOP 와 WHILE LOOP 의 차이점을 설명하여라.

1. 커서의 개념

ORACLE SERVER 은 SQL 문장을 실행하 위하여 Private SQL Area 이라 불리는 작업 영역을 사용합니다. Private SQL Area 에 이름을 붙이고 저장된 정보를 액세스하기 위해 PL/SQL CURSOR 를 사용한다. 블록의 실행 부분이 SQL 문장을 실행할 때 PL/SQL 은 SQL 식별자를 가지는 암시적 CURSOR 를 생성하고 자동적으로 이 CURSOR 를 관리합니다. 명시적 CURSOR 는 명시적으로 선언되고 프로그래머에 의해 명명됩니다.

1.1 CURSOR 의 종류

1.1.1 암시적 CURSOR

ORACLE SERVER 은 명시적으로 선언된 CURSOR 와 관련 없는 각 SQL 문장을 수행하기 위해 CURSOR 를 암시적으로 생성하여 사용한다. PL/SQL 은 SQL CURSOR 로써 가장 최근의 암시적 CURSOR 를 참조할 수 있도록 해 줍니다. SQL CURSOR 를 제어하기 위해 OPEN, FETCH, CLOSE 를 사용할 수 없지만 가장 최근에 실행된 SQL 문장에 대한 정보를 얻기 위한 CURSOR 속성을 사용할 수 있다.(SQL%ROWCOUNT, SQL%FOUND, SQL%NOTFOUND, SQL%ISOPEN 등)

1.1.2 명시적 CURSOR

다중 행 SELECT 문장에 의해 RETURN 되는 각 행을 개별적으로 처리하기 위해 명시적 CURSOR 를 사용합니다. 다중 행 질의에 의해 RETURN 된 행의 집합은 result set 이라 불립니다. 그것의 크기는 검색 조건에 일치하는 행의 수입니다.

가) 명시적 CURSOR 의 함수

- 1) 질의에 의해 RETURN 된 첫번째 행부터 행 하나씩 처리할 수 있다.
- 2) 현재 처리되는 행의 트랙을 유지 합니다.
- 3) 프로그래머가 PL/SQL 블록에서 수동으로 제어할 수 있습니다.

♣ 참고

암시적 CURSOR 에 대한 인출(FETCH)은 배열 인출(FETCH)이며, 두번째 행의 존재는 여전히 TOO_MANY_ROWS 예외가 발생합니다. 그러므로 다중 인출을 수행하고 작업 영역에서 구문 분석된 질의를 재실행하기 위해 명시적 CURSOR 를 사용할 수 있습니다.

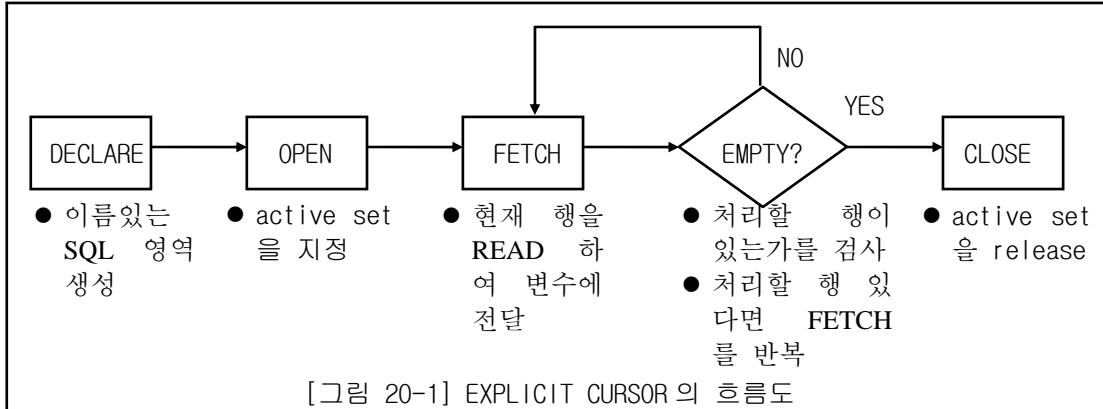
1.2 명시적 CURSOR 의 제어

명시적 CURSOR 를 사용하기 위해서는 4 가지 단계를 거쳐야 한다.

- 1) 수행되기 위한 질의의 구조를 정의하고 이름을 지정함으로써 CURSOR 를 선언한다.
- 2) CURSOR 를 OPEN 한다. OPEN 문장은 질의를 실행하고 참조되는 임의의 변수를 바인드 합니다. 질의에 의해 식별된 행을 active set 이라 불리고 인출(FETCH) 가능합니다.
- 3) CURSOR 에서 데이터를 인출(FETCH)합니다. FETCH 문장은 CURSOR 에서 변수로 현재 행

을 로드합니다. 각 인출(FETCH)은 활성 셋(active set)에서 다음 행으로 그 포인터를 이동하도록 합니다.

- 4) CURSOR 를 CLOSE 합니다. CLOSE 문장은 행의 활성 셋(active set)을 해제 합니다. 이제 새로운 활성 셋(active set)을 생성하기 위해 CURSOR 를 다시 OPEN 할 수 있습니다.



1.3 DECLARE CURSOR

명시적으로 CURSOR 를 선언하기 위해 CURSOR 문장을 사용한다. 질의 내에서 변수를 참조 할 수 있지만 CURSOR 문장 전에 선언되어야 한다.

1.3.1 Syntax

```
CURSOR cursor_name IS
    select_statement;
```

cursor_name PL/SQL 식별자

select_statement INTO 절이 없는 SELECT 문장

1.4 OPEN CURSOR

질의를 수행하고 검색 조건을 충족하는 모든 행으로 구성된 결과 셋을 생성하기 위해 CURSOR 를 OPEN 한다. CURSOR 는 이제 결과 셋에서 첫번째 행을 가리킵니다.

1.4.1 Syntax

```
OPEN cursor_name;
```

나) OPEN 문장은 다음의 작업을 수행한다.

- 1) 문맥 영역에 대해 동적으로 메모리를 할당하여 중요한 프로세싱 정보를 포함 합니다.
- 2) SELECT 문장을 구문 분석합니다.
- 3) 입력 변수를 바인드 합니다.
- 4) 결과 셋을 식별합니다. 즉 검색 조건을 충족시키는 행의 집합입니다. OPEN 문장이 실

행될 때 결과 셋에 있는 행을 변수로 읽어 들이지 않습니다. 그대신 FETCH 문장이 행을 읽습니다.

- 5) 포인터는 활성 셋에서 첫번째 행에 위치합니다.

1.5 FETCH CURSOR

FETCH 문장은 결과 셋에서 하나의 행을 읽어 들입니다. 각 인출(FETCH) 후에 CURSOR 는 결과 셋에서 다음 행으로 이동한다.

1.5.1 Syntax

```
FETCH cursor_name INTO {variable1[,variable2, . . .] | record_name};
```

☞ Guidelines

- 1) SELECT 문장의 열과 같은 개수의 변수를 FETCH 문장의 INTO 절에 포함시켜 좌측부터 1 대 1 대응 되도록 데이터형과 길이가 같아야 합니다.
- 2) CURSOR 에 대한 레코드를 정의하고 FETCH INTO 절에서 레코드를 참조할 수 있습니다.
- 3) CURSOR 가 RETURN 할 행을 포함하는지 테스트합니다. FETCH 시 아무 값도 읽지 않아도 즉 활성 셋에서 프로세스할 남겨진 행이 없는 경우에도 오류가 발생되지 않습니다.
- 4) FETCH 문장은 다음 작업을 수행합니다.
 - ① 활성 셋에서 다음 행으로 포인터를 이동합니다.
 - ② 현재 행에 대한 정보를 출력 PL/SQL 변수로 읽어 들입니다.
 - ③ 포인터가 활성 셋의 끝에 위치하게 되면 CURSOR 는 FOR LOOP 를 탈출 합니다.

1.6 CLOSE CURSOR

CLOSE 문장은 CURSOR 를 사용할 수 없게 하고 결과 셋의 정의를 해제합니다. SELECT 문장이 다 처리된 완성 후에는 CURSOR 를 닫습니다. 필요하다면 CURSOR 를 다시 열수도 있습니다. 그러므로 활성 셋을 여러 번 설정할 수 있다.

1.6.1 Syntax

```
CLOSE cursor_name;
```

☞ Guidelines

CLOSE 문장은 context area 를 해제 합니다. 커서를 닫지 않고 PL/SQL 블록을 종료하는 것 이 가능하다 할 지라도 리소스를 다시 사용 가능하게 하기 위해 명시적으로 선언된 임의의 커서를 닫는 습관을 들여야 합니다. 데이터베이스 매개변수(initial parameter file)에서 OPEN_CURSORS 매개변수에 의해 결정되는 사용자마다 해당하는 커서의 수에는 최대 한계가 있습니다. 디폴트로 OPEN_CURSORS=50 입니다.

1.7 명시적 CURSOR 의 속성

명시적 CURSOR 로 CURSOR 에 대해 상태 정보를 얻기 위한 4 가지 속성이 있습니다.

속 성	타 입	설 명
%ISOPEN	BOOLEAN	CURSOR 가 열리면 TRUE
%NOTFOUND	BOOLEAN	가장 최근의 인출(FETCH)이 행을 RETURN 하지 않으면 TRUE
%FOUND	BOOLEAN	가장 최근의 인출(FETCH)이 행을 RETURN 하면 TRUE
%ROWCOUNT	NUMBER	지금까지 RETURN 된 행의 총 수

1.8 복수 인출(FETCH) 제어

명시적 CURSOR 에서 여러 행을 처리하기 위해서 반복적으로 인출(FETCH)을 수행하는 루프를 정의합니다. 결과적으로 활성 셋의 모든 행은 처리되고 인출(FETCH)이 실패하면 %NOTFOUND 속성을 TRUE 로 설정한다. CURSOR 에 대해 참조하기 전에 각 인출(FETCH)의 성공을 테스트하기 위해 명시적 CURSOR 를 사용합니다.

문제 1) 부서번호를 입력받아 사원번호, 이름, 급여를 출력하는 SCRIPT 를 작성하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_deptno PROMPT '부서번호를 입력하시오 : '
DECLARE
    v_deptno emp.deptno%TYPE := &p_deptno;
    v_empno      emp.empno%TYPE;
    v_ename      emp.ename%TYPE;
    v_sal        emp.sal%TYPE;
    v_sal_total  NUMBER(10,2) := 0;
    CURSOR emp_cursor IS
        SELECT empno,ename,sal
        FROM emp
        WHERE deptno = v_deptno
        ORDER BY empno;
BEGIN
    OPEN emp_cursor;
    DBMS_OUTPUT.PUT_LINE('사번      이름          급     여');
    DBMS_OUTPUT.PUT_LINE('----  -----  -----');
    LOOP
        FETCH emp_cursor INTO v_empno,v_ename,v_sal;
        EXIT WHEN emp_cursor%NOTFOUND;
        v_sal_total := v_sal_total + v_sal;
        DBMS_OUTPUT.PUT_LINE(RPAD(v_empno,6) ||
                             RPAD(v_ename,12) || LPAD(TO_CHAR(v_sal,'$99,999,990.00'),16));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(v_deptno),2) || '번 부서의 합   ' ||
                         LPAD(TO_CHAR(v_sal_total,'$99,999,990.00'),16));
    CLOSE emp_cursor;

```

```
END;  
/  
SET VERIFY ON  
SET SERVEROUTPUT OFF
```

1.9 CURSOR 와 RECORD

테이블에서 열의 구조를 사용하기 위해 RECORD 를 정의할 수 있다. 또한 명시적_CURSOR에서의 열 목록을 기초로 하여 RECORD 를 정의할 수 있습니다. 이것은 단순히 인출할 수 있기 때문에 활성 셋의 행을 처리하기가 편리하다. 그러므로 행 값은 RECORD 의 해당 필드 안으로 직접 LOAD 된다.

문제 2) DEPT 테이블의 내용을 조회하는 SCRIPT 을 작성하여라. 단 %ROWTYPE 을 사용하여라.

```
SET SERVEROUTPUT ON  
DECLARE  
    dept_record      dept%ROWTYPE;  
    CURSOR dept_cursor IS  
        SELECT *  
            FROM dept  
            ORDER BY deptno;  
  
BEGIN  
    OPEN dept_cursor;  
    DBMS_OUTPUT.PUT_LINE('부서번호      부서명      위치');  
    DBMS_OUTPUT.PUT_LINE('-----  -----  -----');  
    LOOP  
        FETCH dept_cursor INTO dept_record;  
        EXIT WHEN dept_cursor%NOTFOUND;  
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.deptno,2) || '  
                           || RPAD(dept_record.dname,15) || RPAD(dept_record.loc,12));  
    END LOOP;  
    CLOSE dept_cursor;  
END;  
/  
SET SERVEROUTPUT OFF
```

문제 3) DEPT 테이블의 내용을 조회하는 SCRIPT 을 작성하여라. 단 RECORD TYPE 을 선언하여 사용하여라.

```
SET SERVEROUTPUT ON
DECLARE
    TYPE dept_record_type IS RECORD
        (v_deptno      dept.deptno%TYPE,
         v_dname       dept.dname%TYPE,
         v_loc         dept.loc%TYPE);
    dept_record    dept_record_type;
    CURSOR dept_cursor IS
        SELECT *
            FROM dept
            ORDER BY deptno;
BEGIN
    OPEN dept_cursor;
    DBMS_OUTPUT.PUT_LINE('부서번호    부서명    위치');
    DBMS_OUTPUT.PUT_LINE('-----  -----  -----');
    LOOP
        FETCH dept_cursor INTO dept_record;
        EXIT WHEN dept_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.v_deptno,2) ||
                             || RPAD(dept_record.v_dname,15) ||
                             RPAD(dept_record.v_loc,12));
    END LOOP;
    CLOSE dept_cursor;
END;
/
SET SERVEROUTPUT OFF
```

1.10 CURSOR 와 FOR LOOP

CURSOR FOR LOOP 는 명시적 CURSOR 에서 행을 처리합니다. LOOP 에서 각 반복마다 CURSOR 를 열고 행을 인출(FETCH)하고 모든 행이 처리되면 자동으로 CURSOR 가 CLOSE 되므로 사용하기가 편리합니다.

1.10.1 Syntax

```
FOR record_name IN cursor_name LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

record_name 암시적으로 선언된 RECORD 이름

cursor_name 선언되어 있는 CURSOR 의 이름

☞ Guidelines

- 1) LOOP 를 채어 하는 RECORD 를 선언하지 마십시오.
- 2) 필요하다면 LOOP 내에서 CURSOR 의 속성을 이용하십시오.
- 3) 필요하다면 FOR 문 안에서 CURSOR 이름 다음에 괄호로 CURSOR 에 대한 매개변수를 묶어 사용하십시오.
- 4) CURSOR 작업이 수동으로 처리되어야 할 때는 FOR LOOP 를 사용하지 마십시오.
- 5) LOOP 가 시작될 때 질의를 정의할 수 있습니다. 질의 표현식은 SELECT 부속문장이라 불리고 CURSOR 는 FOR LOOP 내에서만 사용할 수 있습니다. 이름을 가지고 CURSOR 가 선언되지 않기 때문에 그 속성을 사용할 수는 없습니다.

문제 4) CURSOR FOR LOOP 를 사용하여 DEPT 테이블의 자료를 조회하여라.

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR dept_cursor IS
        SELECT *
            FROM dept
            ORDER BY deptno;
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호      부 서 명      위 치');
    DBMS_OUTPUT.PUT_LINE('-----  -----  -----');
    FOR dept_record IN dept_cursor LOOP
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.deptno,2) || ' '
                           || RPAD(dept_record.dname,15) || RPAD(dept_record.loc,12));
    END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

1.11 SUBQUERY 를 사용한 CURSOR FOR LOOP

PL/SQL 의 SUBQUERY 를 치환 하도록 하기 때문에 CURSOR 는 선언할 필요가 없다.

1.11.1 Syntax

```
FOR record_name IN (subquery) LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

subquery SELECT 문장을 기술

문제 5) SUBQUERY 를 사용한 CURSOR FOR LOOP 를 이용하여 DEPT 테이블의 내용을 조회하여 라.

```
SET SERVEROUTPUT ON
DECLARE
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호    부 서 명      위 치');
    DBMS_OUTPUT.PUT_LINE('-----  -----  -----');
    FOR dept_record IN (SELECT * FROM dept ORDER BY deptno) LOOP
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.deptno,2) || ' '
                           || RPAD(dept_record.dname,15) || RPAD(dept_record.loc,12));
    END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

◆ 연습문제 ◆

1. CURSOR 란?
2. CURSOR의 종류를 설명하여라.
3. 담당 업무를 입력받아 이름,업무,급여,부서명,근무지를 출력하여라. 단 두개의 CURSOR(이름,업무,급여:EMP_CURSOR, 부서명,근무지:DEPT_CURSOR)를 이용하여 작성하여라.
4. EMP_DEPTNO_TOTAL(deptno:NUMBER(2), total:NUMBER(10,2))이라는 테이블을 생성한다.
5. 2번에서 생성한 테이블에 각 부서별 급여의 합을 입력하여라. 단 CURSOR를 이용한다.
6. SQL*Plus 치환 변수로 급여가 많은 사람을 출력하기 위한 인원수를 입력받아 인원수에 해당하는 사원의 정보를 이름,업무,급여를 출력하여라.
7. 동일한 급여를 받는 사원의 정보를 이름,업무,급여를 출력하여라.

1. 매개변수와 CURSOR

CURSOR 가 열릴 때 CURSOR 로 매개변수 값을 전달하고, CURSOR 가 실행될 때 질의에서 그 값이 사용될 수 있습니다. 이것은 각 경우마다 다른 활성 셋(set)을 생성하는 블록에서 여러 번 명시적 CURSOR 를 열고 닫을 수 있음을 의미합니다. CURSOR 선언 시 각각 형식적인 (formal) 매개변수는 OPEN 문장에서 실제 해당 매개변수를 가져야 합니다. 매개변수 데이터형은 스칼라 변수의 데이터형과 동일하지만 크기는 주지 않습니다. 매개변수 명은 CURSOR 의 질의 표현식에서 참조하기 위한 것입니다.

1.1 Syntax

```
CURSOR cursor_name [(parameter_name1 datatype, . . . .)]
IS
select_statement;
```

cursor_name 앞에 선언된 CURSOR 대한 PL/SQL 식별자입니다.

parameter_name 매개변수 이름입니다. 매개변수는 아래의 구문을 따릅니다.

```
Cursor_parameter_name [IN] datatype [{ := | DEFAULT} expression]
```

datatype 매개변수의 스칼라 데이터형입니다.

select_statement INTO 절이 없는 SELECT 문장을 기술합니다.

♣ 참고

매개변수 표기법은 더 많은 기능성을 제공하지 않습니다. 단순히 입력 값을 명확하고 쉽게 지정할 수 있도록 해 줍니다. 이것은 동일한 CURSOR 가 반복적으로 참조될 때 특히 유용합니다.

문제 1) DEPT 테이블의 내용을 조회하는 SCRIPT 을 작성하여라. 단 매개변수를 이용하여라

```
SET SERVEROUTPUT ON
DECLARE
    CURSOR dept_cursor (v_deptno NUMBER) IS
        SELECT *
        FROM dept
        WHERE deptno = v_deptno;
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호      부 서 명      위 치');
    DBMS_OUTPUT.PUT_LINE('----- ----- -----');
    FOR dept_record IN dept_cursor(10) LOOP
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.deptno,2) || ' '
        || RPAD(dept_record.dname,15) || RPAD(dept_record.loc,12));
    END LOOP;
    FOR dept_record IN dept_cursor(20) LOOP
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.deptno,2) || ' '
        || RPAD(dept_record.dname,15) || RPAD(dept_record.loc,12));
    END LOOP;
END;
```

```

        END LOOP;
END;
/
SET SERVEROUTPUT OFF

```

문제 2) EMP 테이블에서 부서번호와 업무를 입력 받아 사원번호, 이름, 급여를 출력하는 SCRIPT를 작성하여라. 단 매개변수를 이용하여라.

```

SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_deptno PROMPT '부서번호를 입력하시오 : '
ACCEPT p_job    PROMPT '담당업무를 입력하시오 : '
DECLARE
    v_sal_total      NUMBER(10,2) := 0;
    CURSOR emp_cursor(v_deptno      emp.deptno%TYPE,
                       v_job        VARCHAR2) IS
        SELECT empno,ename,sal
          FROM emp
         WHERE deptno = v_deptno AND job = v_job
           ORDER BY empno;
BEGIN
    DBMS_OUTPUT.PUT_LINE('사번      이름      급     여');
    DBMS_OUTPUT.PUT_LINE('----  -----  -----');
    FOR emp_record IN emp_cursor(&p_deptno,UPPER('&p_job')) LOOP
        v_sal_total := v_sal_total + emp_record.sal;
        DBMS_OUTPUT.PUT_LINE(RPAD(emp_record.empno,6) ||
                             RPAD(emp_record.ename,12) ||
                             LPAD(TO_CHAR(emp_record.sal,'$99,999,990.00'),16));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(&p_deptno),2) || '번 부서의 합' || 
                         LPAD(TO_CHAR(v_sal_total,'$99,999,990.00'),16));
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

문제 3) EMP 테이블에서 부서번호와 업무를 입력 받아 사원번호, 이름, 급여를 출력하는 SCRIPT 를 작성하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_deptno PROMPT '부서번호를 입력하시오 : '
ACCEPT p_job      PROMPT '담당업무를 입력하시오 : '
DECLARE
    TYPE emp_record_type IS RECORD(
        v_empno      emp.empno%TYPE,
        v_ename       emp.ename%TYPE,
        v_sal         emp.sal%TYPE);
    emp_record     emp_record_type;
    v_sal_total    NUMBER(10,2) := 0;
    CURSOR emp_cursor (v_deptno      emp.deptno%TYPE,
                        v_job        VARCHAR2) IS
        SELECT empno,ename,sal
            FROM emp
            WHERE deptno = v_deptno AND job = v_job
            ORDER BY empno;
BEGIN
    DBMS_OUTPUT.PUT_LINE('사번      이름      급    여');
    DBMS_OUTPUT.PUT_LINE('----- ----- -----');
    OPEN emp_cursor(&p_deptno,UPPER('&p_job'));
    LOOP
        FETCH emp_cursor INTO emp_record;
        EXIT WHEN emp_cursor%NOTFOUND;
        v_sal_total := v_sal_total + emp_record.v_sal;
        DBMS_OUTPUT.PUT_LINE(RPAD(emp_record.v_empno,6) ||
                             RPAD(emp_record.v_ename,12) ||
                             LPAD(TO_CHAR(emp_record.v_sal,'$99,999,990.00'),16));
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE(RPAD(TO_CHAR(&p_deptno),2) || '번 부서의 합');
    || LPAD(TO_CHAR(v_sal_total,'$99,999,990.00'),16));
    CLOSE emp_cursor;
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.2 FOR UPDATE 절

행을 갱신하거나 삭제하기 전에 행을 잠글 수 있습니다. CURSOR 가 열릴 때 영향을 미치는 행을 잠그기 위해 CURSOR 질의에서 FOR UPDATE 절을 추가합니다. ORACLE SERVER 은 TRANSACTION 이 종료할 때 잠금(locking)을 해제하기 때문에 FOR UPDATE 가 사용된다면, 명시적 CURSOR 에서 인출(fetch)한 후에 바로 COMMIT 해서는 안됩니다. FOR UPDATE 절은, ORDER BY 절이 있다 해도, SELECT 문장에서 마지막 절이 됩니다. 다중 테이블을 질의할 때, 특정 테이블에 대해서만 행을 잠그기 위해 FOR UPDATE 절을 사용할 수 있습니다. 테이블의 행은 FOR UPDATE 절이 그 테이블의 열을 참조할 때만 잠겨집니다. 독점적인(exclusive) 행 잠금(locking)은 FOR UPDATE 절이 사용될 때, OPEN 하기 전에 결과 셋(set)에 행해집니다.

1.2.1 Syntax

```
CURSOR cursor_name IS  
  SELECT . . .  
    FOR UPDATE [NOWAIT] [OF column1[,column2,. . . .]];
```

♣ 주의

ORACLE SERVER 은 SELECT FOR UPDATE 에서 필요로 하는 행의 잠금을 얻을 수 없다면, 막연하게 기다립니다. SELECT FOR UPDATE 문장에서 NOWAIT 절을 사용할 수 있고 류프에서 잠금을 얻는데 실패하여 생기는 오류 코드를 테스트할 수 있습니다. 그러므로 PL/SQL 블록을 종료하기 전에 CURSOR OPEN 을 n 번 다시 시도할 수 있습니다. 대형 테이블이라면, 테이블의 모든 행을 잠그기 위해 LOCK TABLE 문장을 사용함으로써 더 나은 성능을 얻을 수도 있습니다. 그러나 LOCK TABLE 을 사용할 때, WHERE CURRENT OF 절을 사용할 수 없고 WHERE column = identifier 를 사용해야 합니다. FOR UPDATE OF 절이 열을 참조 하는 것이 필수적이지는 않지만 이것은 더 쉽게 읽고 유지할 수 있게 하기 위해 추천되어집니다.

1.3 WHERE CURRENT OF 절

명시적 CURSOR 에서 현재 참조할 때 WHERE CURRENT OF 절을 사용합니다. 이를 통해서 명시적으로 ROWID 를 참조하지 않고 현재 처리 중인 행을 갱신하고 삭제할 수 있게 해 줍니다. 행을 OPEN 시에 참기게 하기 위해서 CURSOR 의 SELECT 문에서 FOR UPDATE 절을 포함해야 합니다. CURSOR 에서 일정 조건에 따라 행을 갱신 할 수 있고 또한 FETCH 문장에 의해 가장 최근에 프로세스된 행을 참조하기 위해 WHERE CURRENT OF cursor_name 절이 있는 DELETE 또는 UPDATE 문장을 쓸 수 있습니다. WHERE CURRENT OF 절을 사용할 때 참조되는 CURSOR 는 CURSOR 질의에서 FOR UPDATE 절을 포함해야 하고 존재해야 합니다. 그렇지 않으면 에러를 일으키게 됩니다. 이 절은 ROWID pseudocolumn 을 명시적으로 참조할 필요 없이 현재 처리된 행에 대해 갱신과 삭제를 할 수 있도록 해 줍니다.

1.3.1 Syntax

```
DECLARE
    . . .
    CURSOR cursor_name IS
        SELECT . . .
            FOR UPDATE [NOWAIT] [OF column1[,column2,. . .]];
BEGIN
    OPEN cursor_name;
    LOOP
        . . .
        UPDATE . . .
            WHERE CURRENT OF cursor_name;
        . . .
    END LOOP;
    COMMIT;
    CLOSE cursor_name;
END;
```

문제 4) EMP 테이블에서 다음의 조건을 만족하는 SCRIPT 를 작성하여라. 10 번부서는 급여의 25%, 20 번부서는 급여의 15%, 30 번부서는 급여의 20%를 인상하고 10 번 부서는 20, 20 번 부서는 30 번,30 번 부서는 10 번 부서로 바꾸어라..

```
DECLARE
    CURSOR emp_cursor IS
        SELECT sal,deptno
            FROM emp
            ORDER BY deptno
            FOR UPDATE OF sal,deptno;
BEGIN
    FOR emp_record IN emp_cursor LOOP
        IF emp_record.deptno = 10 THEN
            UPDATE emp
            SET sal = TRUNC(emp_record.sal * 1.25,-1),deptno = 20
                WHERE CURRENT OF emp_cursor;
        ELSIF emp_record.deptno = 20 THEN
            UPDATE emp
            SET sal = TRUNC(emp_record.sal * 1.15,-1),deptno = 30
                WHERE CURRENT OF emp_cursor;
        ELSIF emp_record.deptno = 30 THEN
            UPDATE emp
            SET sal = TRUNC(emp_record.sal * 1.20,-1),deptno = 10
                WHERE CURRENT OF emp_cursor;
        END IF;
    END LOOP;
    COMMIT;
END;
/
```

1.4 SUBQUERY

SUBQUERY은 다른 SQL 데이터 조작 문장 속에 있는 질의(일반적으로 괄호로 둘러쌈)입니다. SUBQUERY는 수행되면 값 또는 값의 집합을 RETURN합니다. SUBQUERY은 SELECT문장의 WHERE절에서 주로 사용됩니다. 또한 FROM절에서도 사용될 수 있습니다. SUBQUERY 또는 상호관련 질의(correlated subquery)가 사용됩니다.

문제 5) CURSOR에서 SUBQUERY를 이용하여 부서번호,부서명,인원수를 출력하여라.

```
SET SERVEROUTPUT ON
DECLARE
    v_cnt    NUMBER;
    CURSOR dept_cursor IS
        SELECT d.deptno,d.dname
        FROM dept d
        WHERE 5 <= (SELECT count(*)
                     FROM emp
                     WHERE deptno = d.deptno);
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호    부 서 명    인원수');
    DBMS_OUTPUT.PUT_LINE('-----  -----  -----');
    FOR dept_record IN dept_cursor LOOP
        SELECT COUNT(*)
        INTO v_cnt
        FROM emp
        WHERE deptno = dept_record.deptno;
        DBMS_OUTPUT.PUT_LINE(LPAD(dept_record.deptno,2) || ' ' ||
                             RPAD(dept_record.dname,15) || LPAD(v_cnt,4));
    END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

◆ 연습문제 ◆

1. EMP_MESS(E_MESS VARXHAR2(100)) 테이블을 생성한다.
2. DEPT TABLE 과 EMP TABLE 을 각각 CURSOR 로 OPEN 하여 EMP_MESS TABLE 에 다음과 같이 등록하여라.

KING - ACCOUNTING
CLARK - ACCOUNTING
MILLER - ACCOUNTING
JONES - RESEARCH
FORD - RESEARCH
3. CURSOR 처리에서 FOR UPDATE 와 WHERE CURRENT OF 의 절을 설명하여라.
4. EMP TABLE 에서 10 번부서는 급여의 10%를 가산하고, 20 번부서는 20%, 30 번부서는 15%를 가산하여라.
5. EMP TABLE 에서 급여가 1000~2000 사이의 사람만 현재의 급여에 30%를 가산하여라.

1. PL/SQL로 예외 처리

PL/SQL 코드를 실행할 때 error 발생하는 경우가 있다. Error는 예외(Exception)를 발생시켜 PL/SQL 블록을 중지시키고 예외 처리기 부분으로 제어가 이동한다. Exception handler는 Exception을 검출하고 조건에 따라 조치 작업을 할 수 있다.

1.1 예외 처리란

예외는 PL/SQL 블록의 실행 중에 발생하여 블록의 주요 부분을 중단 시킨다. 항상 PL/SQL 예외가 발생할 때 블록은 항상 종료되지만 마지막 조치 작업을 수행하도록 예외 처리 부분을 작성할 수 있다.

1) 예외란 무엇인가 ?

◆ PL/SQL을 실행 동안에 발생하는 error 처리를 의미한다.

2) 어떻게 발생되는가 ?

◆ Oracle 오류가 발생할 때
◆ 사용자가 직접 발생시킬 수 있다.

3) 처리하는 방법은 무엇인가 ?

◆ 처리기를 이용한다.
◆ 실행 환경에 전달한다.

1.2 예외를 발생시키는 두 가지 방법

- 1) Oracle 오류가 발생하면 관련된 예외가 자동적으로 발생한다. 예를 들어, 오류 ORA-014-3 는 데이터베이스에서 검색된 행이 전혀 없을 때 발생하며 PL/SQL 은 NO_DATA_FOUND라는 예외를 발생시킨다.
- 2) 블록에 RAISE 문을 써서 명시적으로 예외를 발생시킬 수 있다. 발생하는 예외를 사용자가 정의한 것일 수도 있고 미리 정의된 것일 수도 있다.

1.3 예외 처리

1.3.1 예외 트랩(trap)

만일 예외가 블록의 실행 가능한 섹션에서 발생한다면, 처리는 블록의 예외 섹션에서 해당 예외 처리기로 제어가 넘어갑니다. PL/SQL 블록이 성공적으로 예외를 처리한다면 이 때 예외는 둘러싸는 블록이나 환경으로 전달되지 않는다.

1.3.2 예외 전달

예외를 처리하는 다른 방법은 실행 환경으로 예외를 전달하도록 하는 것이다. 예외가 블록의 실행부에서 발생하여 해당 예외 처리기가 없다면, PL/SQL 블록의 나머지 부분은 수행되지 못하고 종료된다.

1.3.3 예외 검출

예외가 블록의 실행부에서 발생하면 블록의 예외부에 있는 해당 예외 처리부로 제어가 넘어간다.

1.4 예외의 유형

실행 중에 ERROR 가 발생하면 프로그램이 중단되지 않고 예외에 대한 프로그램을 할 수 있다.

예 외	설 명	처 리 용 지 시
정의된 ORACLE SERVER ERROR	PL/SQL 코드에서 자주 발생하는 ERROR 을 미리 정의함	선언할 수 없고 ORACLE SERVER 이 암시적으로 발생
정의되지 않은 ORACLE SERVER ERROR	기타 표준 ORACLE SERVER ERROR	사용자가 선언하고 ORACLE SERVER 이 그것을 암시적으로 발생
사용자 정의 ERROR	프로그래머가 정한 조건이 만족되지 않을 경우 발생	사용자가 선언하고 명시적으로 발생 한다.

♣ 참고

Developer/2000 에서는 PL/SQL 문이 있는 곳에서는 독자적으로 예외를 가진다.

1.5 예외 정의

PL/SQL 블록의 예외 섹션 내에서 해당 루틴을 포함하므로 모든 에러를 처리할 수 있다. 각각의 에러 처리기는 WHERE 절로 구성되는데 그 곳에 에러를 명시하고 WHERE 절 뒤에는 예외가 발생했을 때 처리할 문장을 기술한다.

1.5.1 Syntax

```
EXCEPTION
  WHEN exception1 [OR exception2, . . .] THEN
    statement1;
    statement2;
    . . . .
  [WHEN exception2 [OR exception3, . . .] THEN
    statement3;
    statement4;
    . . . .]
  [WHEN OTHERS THEN
    statement5;
    statement6;
    . . . .]
END;
```

exception	선언섹션에서 선언된 미리 정의된 예외의 표준 이름 이거나 사용자 정의예외의 이름입니다.
Statement	하나 이상의 PL/SQL 또는 SQL 문장입니다.
OTHERS	명시적으로 선언되지 않은 모든 예외를 트랩하는 예외 처리 절입니다.

1.5.2 WHEN OTHERS 예외 처리기

예외 처리 섹션은 지정된 예외만 트랩(trap)합니다. OTHERS 예외 처리기를 사용하지 않으면 다른 예외들은 트랩(trap) 되지 않습니다. 이것은 아직 처리되지 않은 모든 예외를 트랩합니다. 그러므로 OTHERS 는 마지막에 정의되는 예외 처리기입니다. 일부 ORACLE 툴들은 어플리케이션에서 이벤트를 발생시키기 위해 일으키는 개별적인 미리 정의된 예외들을 가지고 있습니다. OTHERS 는 또한 이 예외들도 트랩합니다.

☞ Guidelines

- 1) EXCEPTION 키워드로 블록의 예외 처리 섹션을 시작합니다.
- 2) 블록에서 개별적인 작업에 대해 여러 예외 처리기를 정의합니다.
- 3) 예외가 발생할 때 블록 종료 전에 PL/SQL 은 하나의 처리기만 프로세스 합니다. 다른 모든 예외 처리 절 후에 OTHERS 절을 넣습니다.
- 4) 최대 하나의 OTHERS 절을 가질 수 있습니다.
- 5) 예외는 지정(assignment) 문장 또는 SQL 문장에서 쓰일 수 없습니다.

1.6 미리 정의된 ORACLE SERVER 에러

해당 예외 처리 루틴에서 표준 이름을 참조함으로써 미리 정의된 ORACLE SERVER 에러를 트랩(trap)합니다.

예외 이름	에러 번호	설 명
ACCESS_INTO_NULL	ORA-06530	초기화 되지않은 객체의 속성에 대해 값을 지정하는 것을 시도합니다.
COLLECTION_IS_NULL	ORA-06531	초기화되지 않은 종첨 테이블 대해 EXISTS 를 제외한 메쏘드 모음의 적용을 시도합니다.
CURSOR_ALREADY_OPEN	ORA-06511	이미 열린 CURSOR 의 열기를 시도합니다.
DUP_VAL_ON_INDEX	ORA-00001	중복 값의 삽입을 시도합니다.
INVALID_CURSOR	ORA-01001	잘못된 CURSOR 연산이 발생합니다.
INVALID_NUMBER	ORA-01722	수의 문자열 전환은 실패입니다.
LOGIN_DENIED	ORA-01017	잘못된 사용자명과 비밀 번호로 ORACLE 에 로그온합니다.
NO_DATA_FOUND	ORA-01403	데이터를 RETURN 하지 않는 SELECT 문장
NOT_LOGGED_ON	ORA-01012	PL/SQL 프로그램은 ORACLE 에 연결하지 않고

		데이터베이스 호출을 발생합니다.
PROGRAM_ERROR	ORA-06501	PL/SQL 은 내부 문제를 가지고 있습니다.
ROWTYPE_MISMATCH	ORA-06504	지정문에 포함된 호스트 CURSOR 변수와 PL/SQL CURSOR 변수는 RETURN 유형이 다릅니다.
STORAGE_ERROR	ORA-06500	PL/SQL 이 메모리를 다 써버리거나 또는 메모리가 훼손되었습니다.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	모음의 요소 개수보다 더 큰 인덱스 개수를 사용하는 중첩 테이블 참조합니다.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	범위 밖의 인덱스 번호를 사용하여 중첩 테이블 참조 합니다.
TIMEOUT_ON_RESOURCE	ORA-00051	ORACLE 이 리소스를 대기하는 동안 시간 초과가 발생합니다.
TOO_MANY_ROWS	ORA-01422	단일 행 SELECT 는 하나 이상의 행을 RETURN 합니다.
VALUE_ERROR	ORA-06502	계산, 변환, 절단, 또는 크기 제약 오류가 발생합니다.
ZERO_DIVIDE	ORA-01476	0 으로 배분을 시도합니다.

문제 1) 이름을 입력받아 부서번호에 따라 급여를 갱신한다. 단 10 번이면 25%, 20 번 이면 20%, 30 번이면 15%를 적용한다.

```

SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_ename PROMPT ' 이름을 입력하시오 : '
DECLARE
    TYPE emp_record_type IS RECORD(
        v_empno      emp.empno%TYPE,
        vENAME      emp.ename%TYPE,
        v_sal       emp.sal%TYPE,
        v_deptno    emp.deptno%TYPE);
    emp_record    emp_record_type;
    g_ename       emp.ename%TYPE := UPPER('&p_ename');
BEGIN
    SELECT empno,ename,sal,deptno
        INTO emp_record
        FROM emp
        WHERE ename = g_ename;
    IF emp_record.v_deptno = 10 THEN
        UPDATE emp
            SET sal = TRUNC(emp_record.v_sal * 1.25,-1)
            WHERE empno = emp_record.v_empno;
    ELSIF emp_record.v_deptno = 20 THEN
        UPDATE emp
            SET sal = TRUNC(emp_record.v_sal * 1.15,-1)
            WHERE empno = emp_record.v_empno;
    END IF;
END;
  
```

```

        SET sal = TRUNC(emp_record.v_sal * 1.20,-1)
        WHERE empno = emp_record.v_empno;
    ELSIF emp_record.v_deptno = 30 THEN
        UPDATE emp
            SET sal = TRUNC(emp_record.v_sal * 1.15,-1)
            WHERE empno = emp_record.v_empno;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '는 자료가 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '는 자료가 여러개 있습니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF

```

1.7 미리 정의되지 않은 ORACLE SERVER 에러

우선 에러를 선언하고 나서 OTHERS에서 미리 정의되지 않은 ORACLE SERVER 에러를 처리(에러 번호 확인)합니다. 선언된 예외는 암시적으로 발생합니다. PL/SQL에서 PARAGMA와 EXCEPTION_INIT는 ORACLE 에러 번호와 예외 이름을 관련시키기 위해 컴파일러에게 알려줍니다. PARAGMA는 PL/SQL 블록이 실행될 때 처리되지 않는 컴파일러 명령문임을 의미하는 키워드입니다. 블록 내에서 예외 이름이 발생되면 그것을 관련된 ORACLE SERVER 에러번호로 해독하기 위해 PL/SQL 컴파일러에게 지시합니다.

1.7.1 선언 절차

가) 선언 부분에서 예외 이름을 선언

```
exception_name EXCEPTION;
```

exception_name 예외 이름을 정의한다.

나) PRAGMA EXCEPTION_INIT 문장을 사용하여 표준 에러 번호와 선언된 예외를 연결한다.

```
PRAGMA EXCEPTION_INIT(exception_name, error_number);
```

exception_name 앞에서 선언 된 예외 이름을 기술한다.

error_number 표준 ORACLE SERVER의 에러 번호를 기술한다.

다) 해당 예외 처리 부분에서 선언된 예외를 참조한다.

문제 2) 삭제하고자 하는 사원의 이름을 입력하여 자료를 삭제하여라. EXCEPTION 절을 이용하여 각종 예외를 처리하여라.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_ename PROMPT '삭제하고자 하는 사원의 이름을 입력하시오 : '
DECLARE
    v_ename      emp.ename%TYPE := '&p_ename';
    v_empno     emp.empno%TYPE;
    emp_constraint EXCEPTION;
    PRAGMA EXCEPTION_INIT (emp_constraint, -2292);
BEGIN
    SELECT empno
        INTO v_empno
        FROM emp
        WHERE ename = UPPER(v_ename);
    DELETE emp
        WHERE empno = v_empno;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '는 자료가 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '는 자료가 여러개 있습니다.');
    WHEN emp_constraint THEN
        DBMS_OUTPUT.PUT_LINE('&p_ename' || '는 삭제할 수 없습니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.8 사용자 정의 예외

PL/SQL에서는 개별적으로 예외를 정의할 수 있습니다. 사용자 정의 PL/SQL 예외는 PL/SQL 블록의 선언 섹션에서 선언하고 RAISE 문장으로 명시적으로 발생시킨다.

1.8.1 선언 절차

가) 선언 섹션에서 사용자가 선언한다.

```
Exception_name EXCEPTION;
```

Exception_name 예외 이름을 정의

나) 실행 섹션에서 명시적으로 예외를 발생하기 위해 RAISE 문장을 사용한다.

```
RAISE exception_name;
```

exception_name 앞에서 선언된 예외 이름을 기술한다.

다) 해당 예외 처리기 안에 선언된 예외를 참조한다.

문제 3) 조회하고자 하는 부서번호를 입력받아 사원번호, 이름, 담당업무, 급여를 출력하여라.

단 가능한 모든 에러를 EXCEPTION에서 처리한다.

```
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_deptno PROMPT '조회하고자 하는 부서번호를 입력하시오 : '
DECLARE
    v_deptno emp.deptno%TYPE := &p_deptno;
    CURSOR emp_cursor IS
        SELECT empno,ename,job,sal
        FROM emp
        WHERE deptno = v_deptno;
    emp_deptno_ck EXCEPTION;
BEGIN
    IF v_deptno NOT IN (10,20,30) THEN
        RAISE emp_deptno_ck;
    ELSE
        DBMS_OUTPUT.PUT_LINE('사번      이름      담당업무      급      여');
        DBMS_OUTPUT.PUT_LINE('----- ----- ----- ----- -----');
        FOR emp_record IN emp_cursor LOOP
            DBMS_OUTPUT.PUT_LINE(RPAD(emp_record.empno,4) || ' ' ||
                RPAD(emp_record.ename,11) || RPAD(emp_record.job,10) ||
                RPAD(TO_CHAR(emp_record.sal,'$999,990.00'),12));
        END LOOP;
    END IF;
EXCEPTION
    WHEN emp_deptno_ck THEN
        DBMS_OUTPUT.PUT_LINE('&p_deptno' || '는 자료가 없습니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.9 예외 트래핑 함수

에러가 발생 했을 때 두 함수를 사용하여 관련된 에러 코드 또는 메시지를 확인할 수 있습니다. 코드 또는 메시지에 따라 에러에 대해 취할 작업을 정할 수 있습니다.

함수	설명
SQLCODE	에러 코드에 대한 숫자를 RETURN 한다.
SQLERRM	에러 번호에 해당하는 MESSAGE를 RETURN 한다.

1.9.1 SQL CODE 값

SQL CODE 값	설명
0	예외가 없습니다.(NO ERROR)
1	사용자 정의 ERROR NUMBER
+100	NO_DATA_FOUND 예외
양의 정수	표준 에러 번호

문제 4) 삭제하고자 하는 사원의 이름을 입력하여 삭제하여라. 단 가능한 모든 에러를 처리 하여라.

```
SET VERIFY OFF
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_ename PROMPT '삭제하고자 하는 사원의 이름을 입력하시오 : '
DECLARE
    v_ename      emp.ename%TYPE := '&p_ename';
    v_empno     emp.empno%TYPE;
    v_err_code   NUMBER;
    v_err_msg    VARCHAR2(255);
BEGIN
    SELECT empno
        INTO v_empno
        FROM emp
       WHERE ename = UPPER(v_ename);
    DELETE emp
       WHERE empno = v_empno;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        v_err_code := SQLCODE;
        v_err_msg := SQLERRM;
        DBMS_OUTPUT.PUT_LINE('에러 번호 : ' || TO_CHAR(v_err_code));
        DBMS_OUTPUT.PUT_LINE('에러 내용 : ' || v_err_msg);
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

1.10 RAISE_APPLICATION_ERROR

표준화 되지 않은 에러 코드와 에러 MESSAGE 를 RETURN 하는 RAISE_APPLICATION_ERROR 프로시저를 사용합니다. RAISE_APPLICATION_ERROR 로 어플리케이션에 대한 에러를 제어할 수 있고 처리되지 않은 에러가 RETURN 되지 않도록 합니다.

1.10.1 Syntax

```
raise_application_error (error_number, message[,{TRUE|FALSE}]);
```

error_number -20000 과 20999 사이의 예외에 대해 지정된 번호

message 예외에 대한 사용자 지정 MESSAGE

TRUE|FALSE 선택적 BOOLEAN 매개변수로 TRUE 면 에러는 이전의 에러 스택에
의치하고 FALSE(DEFAULT)면 에러는 모든 이전의 에러를 대체합니다.

문제 5) 삭제하고자 하는 사원의 이름을 입력하여 삭제하여라. 단 가능한 모든 에러를 처리
하여라.

```
SET VERIFY OFF
SET VERIFY OFF
SET SERVEROUTPUT ON
ACCEPT p_ename PROMPT '삭제하고자 하는 사원의 이름을 입력하시오 : '
DECLARE
    v_ename      emp.ename%TYPE := '&p_ename';
    v_err_code   NUMBER;
    v_err_msg    VARCHAR2(255);
BEGIN
    DELETE emp
        WHERE ename = v_ename;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20100,'no data found');
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        v_err_code := SQLCODE;
        v_err_msg := SQLERRM;
        DBMS_OUTPUT.PUT_LINE('에러 번호 : ' || TO_CHAR(v_err_code));
        DBMS_OUTPUT.PUT_LINE('에러 내용 : ' || v_err_msg);
END;
/
SET VERIFY ON
SET SERVEROUTPUT OFF
```

◆ 연습문제 ◆

1. 급여를 입력 받아 하나 이상의 행을 RETURN 하면 예외 처리기에서 “사원이 한명 이상입니다”를 출력하고, 행이 없으면 “사원이 없습니다”를 출력하고 한명 있으면 이름,업무,급여를 출력하여라.
2. 급여를 입력 받아 -100 부터 +100 사이의 모든 사원을 출력하여라.
3. 사용자가 필요한 에러를 정의하여 EXCEPTION에서 사용하는 절차를 설명하여라.
4. PRAGMA는 언제 사용하는가.
5. DEPT TABLE에 행을 삽입하는 SCRIPT를 작성하여라. 단 ERROR 발생시 SQL ERROR CODE와 SQL ERROR MESSAGE를 출력하여라.
6. WHEN OTHERS 절을 설명하여라.

1. SUBPROGRAM

PL/SQL 을 지원하는 어떤 툴이나 언어에서도 SUBPROGRAM(프로시저와 함수)를 실행할 수 있다. PL/SQL 내부에서 식의 일부로서 함수를 실행할 수 있다. EXECUTE 는 명령 다음에 입력되는 Stored Procedure 를 실행한다.

1.1 SUBPROGRAM 의 개요

PL/SQL 프로시저와 함수는 3GL 의 프로시저 및 함수와 매우 비슷하게 동작된다. 모듈화를 통해 관리가 용이하고 적절히 논리적 단위로 나누어진 프로그래밍을 할 수 있다. 즉, 잘 정의된 논리적인 단위로 코드를 분할할 수 있다. PL/SQL 에서 이들 단위를 단위 프로그램 또는 SUBPROGRAM 이라 부른다. PL/SQL 에는 프로시저와 함수라는 두 가지 유형의 SUBPROGRAM 이 있다. SUBPROGRAM 은 컴파일된 상태로 데이터베이스에 저장되어 있어 Performance 가 향상된다.

1.2 SUBPROGRAM 작성 단계

1.2.1 구문 작성

TEXT 편집기를 이용하여 SCRIPT FILE 에 CREATE PROCEDURE 나 CREATE FUNCTION 문을 작성 한다.

```
SQL> ed emp_up
CREATE OR REPLACE PROCEDURE emp_sal_update(
    p_emphno IN emp.empno%TYPE, p_sal IN emp.sal%TYPE)
IS
BEGIN
    UPDATE emp
        SET sal = p_sal
        WHERE empno = p_emphno;
    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(p_emphno) ||
                           '는 없는 사원번호입니다.');
    ELSE
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(SQL%ROWCOUNT) ||
                           '명의 자료를 수정하였습니다.');
    END IF;
END emp_sal_update;
/
```

1.2.2 코드 컴파일

SCRIPT FILE 을 실행 시켜 컴파일하여 컴파일된 코드를 데이터베이스에 저장한다.

```
SQL> @emp_up
```

```
Procedure created.
```

1.2.3 에러 수정

코드 컴파일 시 에러가 발생하면 에러를 확인하고 수정하여 코드를 다시 컴파일한다.

```
SQL> @emp_up
Warning: Procedure created with compilation errors.

SQL> ed emp_up
      -- emp_up 를 수정한 후 저장하고 종료한다.
SQL> @emp_up
Procedure created.
```

1.2.4 실행

SQL*Plus에서 EXECUTE 명령으로 SUBPROGRAM을 실행한다.

```
SQL> EXECUTE emp_sal_update(7788,3500)
PL/SQL procedure successfully completed.

SQL> SELECT empno,ename,job,sal
  2  FROM emp
  3 WHERE empno = 7788;

EMPNO ENAME      JOB          SAL
----- ----- -----
 7788 SCOTT       ANALYST     3500
```

♣ 참고

변경된 내용을 확인하기 위해 SELECT문을 사용할 수 있다.

1.3 PROCEDURE 생성

나중에 실행할 일련의 동작을 저장하기 위해 PL/SQL 프로시저를 작성한다. 프로시저는 실행할 때 사용하는 Parameter 가 없거나 여러 개를 가질 수도 있다. 프로시저에서는 DECLARE 절이 생략되고 IS와 BEGIN 사이에 필요한 변수를 선언하여 사용한다

1.3.1 Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(argument1 [mode1] datatype [{:= | DEFAULT} expression]
    [,argument2 [mode2] datatype [{:= | DEFAULT} expression], . . .)])
{IS | AS}
BEGIN
  pl/sql_block;
END;
```

OR REPLACE procedure_name 이 존재할 경우 PROCEDURE 의 내용을 지우고 다시 생성
procedure_name PROCEDURE 명
argument 매개변수의 이름
mode 3 가지가 있다
 IN : 입력 매개변수로 사용
 OUT : 출력 매개변수로 사용
 IN OUT : 입력, 출력 매개변수로 사용
pl/sql_block PROCEDURE 를 구성하는 코드를 구성하는 PL/SQL 의 블록

☞ Guidelines

- 1) SQL*Plus 에서 프로시저를 작성할 때 CREATE OR REPLACE 를 사용하시오.
- 2) 어떠한 Parameter 라도 사용 가능하다.
- 3) IS 로 PL/SQL 블록을 시작하시오.
- 4) Local 변수 선언은 IS 와 BEGIN 사이에 선언 하시오.

1.3.2 PROCEDURE 실행

PL/SQL 을 지원하는 어떤 틀이나 언어에서도 프로시저를 실행할 수 있다. SQL*Plus 에서 프로시저 호출은 Stored Procedure 를 참조하는 PL/SQL 문을 실행하기 위해 EXECUTE 명령을 사용할 수 있다. EXECUTE 는 명령 다음에 입력되는 Stored Procedure 를 실행한다.

가) Syntax

```
procedure_name[(argument1[,argument2, . . . .])]
```

나) SQL*Plus에서 프로시저 실행

EXECUTE 는 명령 다음에 입력되는 Stored Procedure 를 실행한다.

```
SQL> EXECUTE emp_sal_update(7902,4000)
```

```
PL/SQL procedure successfully completed.
```

문제 1) EMP 테이블에 새로운 사원의 정보를 이름,업무,매니저,급여를 입력받아 등록하는 프로시저를 생성하여라. 단 부서 번호는 매니저의 부서 번호와 동일하게 하고 보너스는 SALESMAN 은 0 을 그 외는 NULL 을 입력하여라.

```
CREATE OR REPLACE PROCEDURE emp_input(
    v_name    IN      emp.ename %TYPE,
    v_job     IN      emp.job %TYPE,
    v_mgr     IN      emp.mgr %TYPE,
    v_sal     IN      emp.sal %TYPE)
IS
    v_comm        emp.comm%TYPE;
    v_deptno     emp.deptno%TYPE;
    manager_error EXCEPTION;
BEGIN
    IF UPPER(v_job) NOT IN ('PRESIDENT','MANAGER','ANALYST',
                            'SALESMAN','CLERK') THEN
        RAISE manager_error;
    ELSIF UPPER(v_job) = 'SALESMAN' THEN
        v_comm := 0;
    ELSE
        v_comm := NULL;
    END IF;
    SELECT deptno
        INTO v_deptno
        FROM emp
        WHERE empno = v_mgr;
    INSERT INTO emp
        VALUES (empno_sequence.NEXTVAL,v_name,UPPER(v_job),
                v_mgr,SYSDATE,v_sal,v_comm,v_deptno);
EXCEPTION
    WHEN manager_error THEN
        DBMS_OUTPUT.PUT_LINE('담당 업무가 잘못 입력되었습니다.');
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('입력한 MANAGER 는 없습니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
```

```
SQL> SET SERVEROUTPUT ON
SQL> EXECUTE emp_input('YOONJB','MANAGER',7788,2500)
```

문제 2) 이름을 입력받아 그 사원의 정보 중 부서명과 급여를 검색하는 프로시저를 생성하여라.

```
CREATE OR REPLACE PROCEDURE dname_sal_disp(
    v_ename IN      emp.ename%TYPE,
    v_dname OUT     dept.dname%TYPE,
    v_sal   OUT     emp.sal%TYPE)
IS
    v_deptno emp.deptno%TYPE;
BEGIN
    SELECT sal,deptno
        INTO v_sal,v_deptno
        FROM emp
        WHERE ename = UPPER(v_ename);
    SELECT dname
        INTO v_dname
        FROM dept
        WHERE deptno = v_deptno;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('입력한 MANAGER 는 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('자료가 2 건 이상입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
```

```
SQL> VAR g_dname VARCHAR2(14)
SQL> VAR g_sal NUMBER
SQL> EXECUTE dname_sal_disp('SCOTT',:g_dname,:g_sal);
PL/SQL procedure successfully completed.
SQL> PRINT g_dname
G_DNAME
-----
ACCOUNTING
SQL> PRINT g_sal
G_SAL
-----
3000
```

문제 3) 이름을 입력받아 그 사원의 정보 중 부서명과 급여를 검색하는 프로시저를 생성하여라.

```
CREATE OR REPLACE PROCEDURE tel(
    v_tel    IN OUT    VARCHAR2)
IS
BEGIN
    v_tel := SUBSTR(v_tel,1,3) || '-' || SUBSTR(v_tel,4);
    DBMS_OUTPUT.PUT_LINE('전화번호 : ' || v_tel);
END tel;
/
SQL> SET SERVEROUTPUT ON
SQL> VAR g_tel VARCHAR2(20)
SQL> BEGIN
  2  :g_tel := 1234567;
  3 END;
  4 /
PL/SQL procedure successfully completed.
SQL> EXECUTE tel(:g_tel)
전화번호 : 123-4567
PL/SQL procedure successfully completed.
SQL> PRINT g_tel
G_TEL
-----
123-4567
```

1.4 FUNCTION 생성

실행 환경에 반드시 하나의 값을 Return 하기 위해 PL/SQL 함수를 사용한다. 함수 선언에서 Datatype 이 있는 RETURN 절을 추가하고 PL/SQL 블록에 적어도 한 개의 이상의 RETURN 문을 포함한다. PARAMETER 에서 사용하는 IN,OUT,IN OUT 는 PROCEDURE 에서 사용한 것과 동일하게 사용 가능하나 대부분 IN 을 사용한다.

1.4.1 Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
    [(argument1 [mode1] datatype [{:= | DEFAULT} expression]
      [,argument2 [mode2] datatype [{:= | DEFAULT} expression], . . .])]

RETURN data_type
{IS | AS}
BEGIN
    pl/sql_block;
END;
```

OR REPLACE function_name이 존재할 경우 FUNCTION 의 내용을 지우고 다시 생성

function_name Function 의 이름은 표준 Oracle 명명법에 따른 함수이름

argument 매개변수의 이름

mode	3 가지가 있다
	IN : 입력 매개변수로 사용
	OUT : 출력 매개변수로 사용
	IN OUT : 입력, 출력 매개변수로 사용
data_type	반환되는 값의 datatype
pl/sql_block	FUNCTION 를 구성하는 코드를 구성하는 PL/SQL 의 블록

1.4.2 RETURN 문

- 1) PL/SQL 블록에는 RETURN 문이 있어야 한다.
- 2) 함수는 RETURN 절에 지정된 것과 동일한 datatype 으로 RETURN 값을 설정해야 한다.
- 3) 다중 RETURN 문은 사용할 수 있지만 한 번의 호출로는 한 개의 RETURN 문만 실행된다.
- 4) 일반적으로 다중 RETURN 문은 IF 문에서 사용한다.

1.4.3 FUNCTION 실행

PL/SQL 을 지원하는 어떤 틀이나 언어에서도 함수를 실행할 수 있고 PL/SQL 내부에서 식의 일부로서 함수를 실행할 수 있다. SQL*Plus 에서 FUNCTION 호출은 Stored Function 를 참조하는 PL/SQL 문을 실행하기 위해 EXECUTE 명령을 사용할 수 있다. EXECUTE 는 명령 다음에 입력되는 Stored Function 를 실행한다.

가) Syntax

```
output_variable := function_name[(argument1[, argument2, . . . . .])]
```

나) SQL*Plus에서 함수 실행

```
SQL> EXECUTE :g_deptno := ename_deptno('ALLEN')
```

```
PL/SQL procedure successfully completed.
```

문제 4) EMP 테이블에서 이름으로 부서 번호를 검색하는 함수를 작성하여라.

```
CREATE OR REPLACE FUNCTION ename_deptno(
    v_ename IN      emp.ename%TYPE)
RETURN NUMBER
IS
    v_deptno emp.deptno%TYPE;
BEGIN
    SELECT deptno
        INTO v_deptno
        FROM emp
       WHERE ename = UPPER(v_ename);
    DBMS_OUTPUT.PUT_LINE('부서번호 : ' || TO_CHAR(v_deptno));
    RETURN v_deptno;
```

```

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('입력한 MANAGER는 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('자료가 2건 이상입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/

```

```

SQL> SET SERVEROUTPUT ON
SQL> VAR g_deptno NUMBER
SQL> EXECUTE :g_deptno := ename_deptno('SCOTT')
부서번호 : 10

```

PL/SQL procedure successfully completed.

```
SQL> PRINT g_deptno
```

```

G_DEPTNO
-----
10

```

문제 5) EMP 테이블에서 이름을 입력 받아 부서번호, 부서명, 급여를 검색하는 FUNCTION 을 작성하여라. 단 부서번호를 RETURN에 사용하여라.

```

CREATE OR REPLACE FUNCTION emp_disp(
    v_ename IN      emp.ename%TYPE,
    v_dname OUT     dept.dname%TYPE,
    v_sal   OUT     emp.sal%TYPE)
RETURN NUMBER
IS
    v_deptno emp.deptno%TYPE;
    v_dname_temp      dept.dname%TYPE;
    v_sal_temp       emp.sal%TYPE;
BEGIN
    SELECT sal,deptno
        INTO v_sal_temp,v_deptno
        FROM emp
        WHERE ename = UPPER(v_ename);
    SELECT dname
        INTO v_dname_temp
        FROM dept
        WHERE deptno = v_deptno;
    v_dname := v_dname_temp;
    v_sal := v_sal_temp;
    DBMS_OUTPUT.PUT_LINE('성명 : ' || v_ename);
    DBMS_OUTPUT.PUT_LINE('부서번호 : ' || TO_CHAR(v_deptno));
    DBMS_OUTPUT.PUT_LINE('부서명 : ' || v_dname_temp);

```

```

        DBMS_OUTPUT.PUT_LINE('급여 : ' || TO_CHAR(v_sal_temp, '$999,999'));
        RETURN v_deptno;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('입력한 MANAGER는 없습니다.');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('자료가 2건 이상입니다.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('기타 에러입니다.');
END;
/
SQL> SET SERVEROUTPUT ON
SQL> VAR g_deptno NUMBER
SQL> VAR g_dname VARCHAR2(20)
SQL> VAR g_sal NUMBER
SQL> EXECUTE :g_deptno := emp_disp('scott', :g_dname, :g_sal)
성명 : scott
부서번호 : 10
부서명 : ACCOUNTING
급여 : $3,000
PL/SQL procedure successfully completed.
SQL> PRINT g_deptno
G_DEPTNO
-----
10
SQL> PRINT g_dname
G_DNAME
-----
ACCOUNTING
SQL> PRINT g_sal
G_SAL
-----
3000

```

1.5 함수와 프로시저 비교

프로시저	함수
PL/SQL 문으로서 실행	식의 일부로서 사용
RETURN Datatype이 없음	RETURN Datatype이 필수
값을 Return 할 수 있음	값을 Return 하는 것이 필수

♣ 참고

- 1) 프로시저는 parameter 리스트를 가질 수 있지만 값 반환이 필수적 이지는 않다.
- 2) 함수는 다음과 같은 두 가지 점에서 프로시저와 다르다.
 - ① 식(expression)의 일부로서 함수를 사용한다.
 - ② 함수는 값을 return 하는 것이 필수적이다.

1.6 TRIGGER

특정 테이블에 DML(INSERT,UPDATE,DELETE)문장이 수행되었을 때 데이터베이스에서 자동적으로 PL/SQL 블록을 수행 시키기 위해서 데이터베이스 TRIGGER를 사용한다. TRIGGER는 트리거링 이벤트가 일어날 때마다 암시적으로 실행된다. 트리거링 이벤트에는 데이터베이스 테이블에서 INSERT,UPDATE,DELETE 오퍼레이션이다.

1.6.1 TRIGGER가 사용되는 경우

- 1) 테이블 생성시 CONSTRAINT로 선언 제한이 불가능하고 복잡한 무결성 제한을 유지
- 2) DML 문장을 사용한 사람, 변경한 내용, 시간 등을 기록함으로써 정보를 AUDIT하기
- 3) 테이블을 변경할 때 일어나야 할 동작을 다른 테이블 또는 다른 프로그램들에게 자동적으로 신호하기

1.6.2 TRIGGER에 대한 제한

- 1) TRIGGER는 트랜잭션 제어 문(COMMIT,ROLLBACK,SAVEPOINT)장을 사용하지 못한다.
- 2) TRIGGER 주요부에 의해 호출되는 프로시저나 함수는 트랜잭션 제어 문장을 사용하지 못한다.
- 3) TRIGGER 주요부는 LONG 또는 LONG RAW 변수를 선언할 수 없다.
- 4) TRIGGER 주요부가 액세스하게 될 테이블에 대한 제한이 있다.

1.6.3 TRIGGER 생성

CREATE TRIGGER 문장에 의해 TRIGGER를 생성할 수 있다.

가) Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
    {BEFORE | AFTER} triggering_event [OF column1, . . .] ON table_name
    [FOR EACH ROW [WHEN trigger_condition]
trigger_body;
```

trigger_name	TRIGGER의 식별자
BEFORE AFTER	DML 문장이 실행되기 전에 TRIGGER를 실행할 것인지 실행된 후에 TRIGGER를 실행할 것인지를 정의
triggering_event	TRIGGER를 실행하는 DML(INSERT,UPDATE,DELETE)문을 기술한다.
OF column	TRIGGER가 실행되는 테이블에서 COLUMN명을 기술한다.
table_name	TRIGGER가 실행되는 테이블 이름
FOR EACH ROW	이 옵션을 사용하면 행 레벨 트리거가 되어 triggering 문장에 의해 영향받은 행에 대해 각각 한번씩 실행하고 사용하지 않으면 문장 레벨 트리거가 되어 DML 문장 당 한번만 실행된다.

나) TRIGGER에서 OLD 와 NEW

행 레벨 TRIGGER 에서만 사용할 수 있는 예약어로 트리거 내에서 현재 처리되고 있는 행을 액세스할 수 있다. 즉 두개의 의사 레코드를 통하여 이 작업을 수행할 수 있다. :OLD 는 INSERT 문에 의해 정의되지 않고 :NEW 는 DELETE 에 대해 정의되지 않는다. 그러나 UPDATE 는 :OLD 와 :NEW 를 모두 정의한다. 아래의 표는 OLD 와 NEW 값을 정의한 표이다.

문장	:OLD	:NEW
INSERT	모든 필드는 NULL 로 정의	문장이 완전할 때 삽입된 새로운 값
UPDATE	갱신하기 전의 원래 값	문장이 완전할 때 갱신된 새로운 값
DELETE	행이 삭제되기 전의 원래 값	모든 필드는 NULL 이다.

다) TRIGGER 솔루션 사용하기

트리거 내에서 오퍼레이션이 무엇인지를 결정하기 위해 사용할 수 있는 3 가지 BOOLEAN 함수가 있다.

솔루션	설명
INSERTING	트리거링 문장이 INSERT 이면 TRUE 를 그렇지 않으면 FALSE 를 RETURN
UPDATING	트리거링 문장이 UPDATE 이면 TRUE 를 그렇지 않으면 FALSE 를 RETURN
DELETING	트리거링 문장이 DELETE 이면 TRUE 를 그렇지 않으면 FALSE 를 RETURN

라) TRIGGER 삭제와 억제하기

DROP TRIGGER 명령어로 트리거를 삭제할 수 있고 TRIGGER 를 잠시 disable 할 수 있다.

```
DROP TRIGGER trigger_name;  
ALTER TRIGGER trigger_name {DISABLE | ENABLE};
```

마) TRIGGER 와 DATA DICTIONARY

TRIGGER 가 생성될 때 소스 코드는 데이터 사전 VIEW 인 user_triggers 에 저장된다. 이 VIEW 는 TRIGGER_BODY, WHERE 절, 트리거링 테이블, TRIGGER 태입을 포함 한다.

```
SQL> SELECT trigger_type,table_name,triggering_event  
2 FROM user_triggers;
```

trigger_type	table_name	triggering_event
AFTER STATEMENT	EMP	INSERT OR UPDATE OR DELETE
BEFORE STATEMENT	EMP	INSERT OR UPDATE OR DELETE
BEFORE EACH ROW	EMP	UPDATE

문제 6) EMP 테이블에서 급여를 수정시 현재의 값보다 적게 수정할 수 없으며 현재의 값보다 10% 이상 높게 수정할 수 없다. 이러한 조건을 만족하는 트리거를 작성하여라.

```
CREATE OR REPLACE TRIGGER emp_sal_chk
BEFORE UPDATE OF sal ON emp
FOR EACH ROW WHEN (NEW.sal < OLD.sal
                    OR NEW.sal > OLD.sal * 1.1)
BEGIN
    raise_application_error(-20502,
                           'May not decrease salary. Increase must be < 10%');
END;
/
```

```
SQL> @emp_sal

Trigger created.

SQL> update emp
  2 set sal = 6000;
set sal = 6000
*
ERROR at line 2:
ORA-20502: May not decrease salary. Increase must be < 10%
ORA-06512: at "SCOTT.EMP_SAL_CHK", line 2
ORA-04088: error during execution of trigger 'SCOTT.EMP_SAL_CHK'
```

문제 7) EMP 테이블을 사용할 수 있는 시간은 월요일부터 금요일까지 09 시부터 18 시까지만 사용할 수 있도록 하는 트리거를 작성하여라.

```
CREATE OR REPLACE TRIGGER emp_resource
    BEFORE insert OR update OR delete ON emp
BEGIN
    IF TO_CHAR(SYSDATE, 'DY') IN ('SAT', 'SUN')
        OR TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'))
           NOT BETWEEN 9 AND 18 THEN
        raise_application_error(-20502,
                               '작업할 수 없는 시간입니다.');
    END IF;
END;
/
```

```
SQL> @tr1

Trigger created.

SQL> select to_char(sysdate,'hh24') from dual;
TO_CHAR(SYSDATE, 'HH24')
-----
19
SQL> update emp
```

```

2 set deptno = 10;
update emp
*
ERROR at line 1:
ORA-20502: 작업할 수 없는 시간입니다.
ORA-06512: at "SCOTT.EMP_RESOURCE", line 5
ORA-04088: error during execution of trigger 'SCOTT.EMP_RESOURCE'

```

문제 8) EMP 테이블에 INSERT,UPDATE,DELETE 문장이 하루에 몇 건 발생하는지 조사하려고 한다. 조사 내용은 EMP_AUDIT에 사용자 이름, 작업 구분, 작업 시간을 저장하는 트리거를 작성하여라.

```

CREATE SEQUENCE emp_audit_tr
    INCREMENT BY 1
    START WITH 1
    MAXVALUE 999999
    MINVALUE 1
    NOCYCLE
    NOCACHE;

CREATE TABLE emp_audit(
    e_id      NUMBER(6)
        CONSTRAINT emp_audit_pk PRIMARY KEY,
    e_name    VARCHAR2(30),
    e_gubun   VARCHAR2(10),
    e_date    DATE);

CREATE OR REPLACE TRIGGER emp_audit_tr
    AFTER insert OR update OR delete ON emp
BEGIN
    IF INSERTING THEN
        INSERT INTO emp_audit
            VALUES(emp_audit_tr.NEXTVAL,USER,'inserting',SYSDATE);
    ELSIF UPDATING THEN
        INSERT INTO emp_audit
            VALUES(emp_audit_tr.NEXTVAL,USER,'updating',SYSDATE);
    ELSIF DELETING THEN
        INSERT INTO emp_audit
            VALUES(emp_audit_tr.NEXTVAL,USER,'deleting',SYSDATE);
    END IF;
END;
/

```

```

SQL> UPDATE emp
2 SET deptno = 20
3 WHERE deptno = 10;
2 row updated.

SQL> SELECT * FROM emp_audit;
E_ID E_NAME          E_GUBUN     E_DATE
-----  -----
1 SCOTT             updating   23-FEB-99

```

문제 9) EMP 테이블에 INSERT,UPDATE,DELETE 문장이 하루에 몇 건의 ROW 가 발생되는지 조사하려고 한다. 조사 내용은 EMP_AUDIT_ROW 에 사용자 이름, 작업 구분, 작업 시간, 사원번호, 이전의 급여, 갱신된 급여를 저장하는 트리거를 작성하여라.

```

DROP SEQUENCE emp_row_seq;
CREATE SEQUENCE emp_row_seq
    INCREMENT BY 1
    START WITH 1
    MAXVALUE 999999
    MINVALUE 1
    NOCYCLE
    NOCACHE;

DROP TABLE emp_row_tab;
CREATE TABLE emp_row_tab(
    e_id          NUMBER(6)
                CONSTRAINT emp_row_pk PRIMARY KEY,
    e_name        VARCHAR2(30),
    e_gubun       VARCHAR2(10),
    e_date        DATE);

CREATE OR REPLACE TRIGGER emp_row_aud
    AFTER insert OR update OR delete ON emp
    FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO emp_row_tab
            VALUES(emp_row_seq.NEXTVAL,USER,'inserting',SYSDATE);
    ELSIF UPDATING THEN
        INSERT INTO emp_row_tab
            VALUES(emp_row_seq.NEXTVAL,USER,'updating',SYSDATE);
    ELSIF DELETING THEN
        INSERT INTO emp_row_tab
            VALUES(emp_row_seq.NEXTVAL,USER,'deleting',SYSDATE);
    END IF;
END;
/

```

```

SQL> UPDATE emp
  2 SET deptno = 40
  3 WHERE deptno = 10;

```

3 rows updated.

```
SQL> SELECT * FROM emp_row_tab;
```

E_ID	E_NAME	E_GUBUN	E_DATE
1	SCOTT	updating	23-FEB-99
2	SCOTT	updating	23-FEB-99
3	SCOTT	updating	23-FEB-99

◆ 연습문제 ◆

1. EMP TABLE에 이름, 사번, 급여, 부서번호를 전달받아 등록하는 PROCEDURE를 작성하여라.
2. 사원번호를 입력받아 급여를 수정하는 PROCEDURE를 작성하여라.
3. 최고의 월급을 받는 사원의 사번을 구하여 출력하는 PROCEDURE를 작성하여라.
4. 이름을 입력받아 부서명을 구하여 출력하는 PROCEDURE를 작성하여라.
5. 부서번호를 입력받아 그 부서의 최고 급여를 구하여 출력하는 FUNCTION을 작성하여라.
6. PROCEDURE와 FUNCTION의 차이점을 설명하여라.
7. TRIGGER란 ?
8. EMP_SAL_TOT(부서번호, 급여의 합) TABLE을 생성하여라.

```
SQL> CREATE TABLE emp_sal_tot AS
  2  SELECT deptno, SUM(sal) sal_tot
  3  FROM emp
  4  GROUP BY deptno;
```

```
Table created.
```

9. EMP_SAL_TOT TABLE의 내용은 EMP TABLE을 변경하면 EMP_SAL_TOT TABLE의 내용도 자동적으로 변경되어야 한다. 이를 반영한 TRIGGER를 작성하여라.

A. ORACLE DATA DICTIONARY

ORACLE DATA DICTIONARY 는 SYS USER 의 소유로써 ORACLE8 SERVER 의 가장 중요한 구성요소 중 하나로써 데이터베이스에 대한 모든 정보를 가지며, 대부분은 읽기 전용으로 제공되는 TABLE 과 VIEW 의 집합이다. 데이터베이스에 대한 작업, 주로 DDL, DCL 문장이 수행되면 ORACLE8 SERVER 는 DATA DICTIONARY 를 갱신하고 유지 보수한다.

1. ORACLE DATA DICTIONARY의 내용

- ① LOG ON 할 수 있는 사용자명(user name)
- ② 사용자에게 허가된 권한(system privilege, object privilege)
- ③ 데이터베이스 객체명(table, view, index, sequence, synonym 등)
- ④ 테이블 제약 조건
- ⑤ 감사(Audit) 정보

2. ORACLE DATA DICTIONARY의 내용 조회

ORACLE DBA 의 임무 중 하나가 DATA DICTIONARY 의 내용을 적절히 조회하여 최적의 상태로 Database 를 운영하는 것이다. 또한 일반 사용자도 사용자의 권한에 따라 QUERY 할 수 있는 DATA DICTIONARY 가 있다. 이들의 종류는 4 가지로 다음과 같다.

접두어	설명
USER_	SESSION 을 이루고 있는 사용자가 소유한 객체에 관한 정보
ALL_	SESSION 을 이루고 있는 사용자에게 액세스가 허용된 객체에 관한 정보
DBA_	DBA 권한을 가진 사용자가 액세스할 수 있는 정보
V\$	Dynamic Performance View 라고도 하며, 현재 Database 의 상태에 관한 정보로 주로 DBA 에게만 액세스가 허용되어 있다.

문제 1) 현재 SESSION 을 이루고 있는 사용자가 조회할 수 있는 DATA DICTIONARY 가 어떤 것이 있는가를 조회하여라.

SQL> COL table_name FORMAT a30
SQL> COL comments FORMAT a50
SQL> SELECT *
2 FROM dictionary;
TABLE_NAME COMMENTS
----- -----
-
ALL_ARGUMENTS Arguments in object accessible to the user
ALL_CATALOG All tables, views, synonyms, sequences accessible to the user
ALL_CLUSTERS Description of clusters accessible to the user

ALL_CLUSTER_HASH_EXPRESSIONS	Hash functions for all accessible clusters
ALL_COL_COMMENTS	Comments on columns of accessible tables and views

문제 2) 현재 SESSION을 이루고 있는 사용자가 소유하고 있는 OBJECT를 모두 조회하여라.

```
SQL> COL object_name FORMAT a30
SQL> SELECT object_name,object_type,created
  2  FROM user_objects;

OBJECT_NAME          OBJECT_TYPE      CREATED
-----              -----
BONUS                TABLE           23-FEB-99
CUSTID               SEQUENCE        23-FEB-99
CUSTOMER              TABLE           23-FEB-99
CUSTOMER_PRIMARY_KEY INDEX           23-FEB-99
DEPT                 TABLE           23-FEB-99
DEPT_PRIMARY_KEY     INDEX           23-FEB-99
DEPT_SUM              VIEW            18-FEB-99
DNAME_SAL             FUNCTION        23-FEB-99
DNAME_SAL_DISP        PROCEDURE       23-FEB-99
EMP_ROW_AUD           TRIGGER         23-FEB-99
.
.
.
47 rows selected.
```

문제 3) EMP 테이블에 대한 모든 제약 조건을 검색하시오.

```
SQL> COL constraint_name FORMAT a20
SQL> COL constraint_type FORMAT a1
SQL> COL search_condition FORMAT a20
SQL> COL r_constraint_name FORMAT a20
SQL> SELECT constraint_name,constraint_type,
  2  search_condition,r_constraint_name
  3  FROM user_constraints
  4  WHERE table_name = 'EMP';

CONSTRAINT_NAME      C SEARCH_CONDITION      R_CONSTRAINT_NAME
-----              -----
SYS_C00738           C EMPNO IS NOT NULL
SYS_C00739           C DEPTNO IS NOT NULL
EMP_PRIMARY_KEY       P
EMP_SELF_KEY          R                  EMP_PRIMARY_KEY
EMP_FOREIGN_KEY       R                  DEPT_PRIMARY_KEY
```

문제 4) EMP 테이블의 모든 열에 대한 제약 조건을 검색하시오.

```
SQL> COL constraint_name FORMAT a20
SQL> COL column_name FORMAT a20
SQL> SELECT constraint_name, column_name
  2  FROM user_cons_columns
  3 WHERE table_name = 'EMP';

CONSTRAINT_NAME      COLUMN_NAME
-----  -----
EMP_FOREIGN_KEY      DEPTNO
EMP_PRIMARY_KEY      EMPNO
EMP_SELF_KEY         MGR
SYS_C00738           EMPNO
SYS_C00739           DEPTNO
```

문제 5) SCOTT OI 소유하고 있는 테이블의 reference 관계에 대한 제약 조건을 검색하시오.

```
SQL> SET pagesize 24
SQL> SET linesize 132
SQL> COL fk_owner FORMAT a10
SQL> COL pk_owner FORMAT a10
SQL> COL fk_table FORMAT a15
SQL> COL pk_table FORMAT a15
SQL> COL fk_col FORMAT a15
SQL> COL pk_col FORMAT a15
SQL> SELECT fk.owner  fk_owner,
  2   fk.table_name  fk_table,
  3   fkc.column_name fk_col,
  4   pk.owner  pk_owner,
  5   pk.table_name pk_table,
  6   pkc.column_name pk_col
  7  FROM user_constraints fk, user_constraints pk,
  8  user_cons_columns fkc, user_cons_columns pkc
  9 WHERE fk.r_constraint_name = pk.constraint_name
10 AND fk.constraint_type = 'R'
11 AND pk.constraint_type = 'P'
12 AND fk.r_owner = pk.owner
13 AND fk.constraint_name = fkc.constraint_name
14 AND pk.constraint_name = pkc.constraint_name
15 AND fk.owner = fkc.owner
16 AND fk.table_name = fkc.table_name
17 AND pk.owner = pkc.owner
18 AND pk.table_name = pkc.table_name;
```

FK_OWNER	FK_TABLE	FK_COL	PK_OWNER	PK_TABLE	PK_COL
SCOTT	ORD	CUSTID	SCOTT	CUSTOMER	CUSTID
SCOTT	EMP	DEPTNO	SCOTT	DEPT	DEPTNO
SCOTT	EMP	MGR	SCOTT	EMP	EMPNO

SCOTT	ITEM	ORDID	SCOTT	ORD	ORDID

◆ 연습문제 ◆

1. 현재 SESSION을 이루고 있는 사용자의 시스템 권한을 확인하여라.
2. 현재 SESSION을 이루고 있는 사용자의 오브젝트 권한을 확인하여라.
3. 현재 SESSION을 이루고 있는 사용자가 사용할 수 있는 공간(free space)을 확인하여라.
4. 현재 SESSION을 이루고 있는 사용자가 작성한 프로시저와 함수를 조회하여라.
5. EMP 테이블에 관련된 TRIGGER를 모두 조회하여라.

B. 실습용 테이블을 생성하는 SCRIPT

1. 실습용 SCRIPT란 ?

이 데모용 SCRIPT 는 ORACLE 제품을 INSTALL 하면 INSTALL 시 생성되어지는 SCRIPT로 ORACLE 을 처음 사용하는 사용자가 SQL,SQL*Plus,PL/SQL 을 처음 배우고자 할 때 아주 유용하게 사용되는 SCRIPT 이다. 일반 사용자는 SQL*Plus 에서 SCOTT 사용자로 Log On 하여 사용하면 된다.

2. 실습용 TABLE 생성 절차

- 1) Personal Oracle7 을 Install 한다.
- 2) ORACLE_HOME\OBS\DEMOBLD.SQL FILE 을 NOTEPAD 로 OPEN 한다. ORACLE_HOME DIRECTORY 는 Personal Oracle7 을 Install 할 때 지정할 수 있다. 지금부터 ORACLE_HOME DIRECTORY 를 C:\ORAWIN95 로 사용한다.
- 3) 약 30 Line 에 있는 ENAME CHAR(10)을 ENAME VARCHAR2(10)로, JOB CHAR(9)을 JOB VARCHAR2(9)로 수정한 후 저장하고 종료하여라. 다음 PAGE 의 SCRIPT 에서 진하게 인쇄된 부분과 동일하게 수정 후 저장하고 종료하면 된다.
- 4) 시작 → 프로그램 → Oracle for Windows 95 → SQL Plus 3.3 을 선택한다.
- 5) User Name 에는 SCOTT 을 입력하고 Password 에는 TIGER 를 입력하고 Host String 에는 2: 또는 생략 가능하다(DEFAULT 는 LOCAL 변수에 정의되어있는 곳).
- 6) 날짜 형식을 DD-MON-YY 형식으로 SCRIPT 를 작성하였다. 날짜 형식이 DD-MON-YY 형식이 아닌 경우는 다음의 명령을 먼저 실행하여라. SQL PROMPT(SQL>)에서 ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YY'; 을 실행한다.
- 7) SQL PROMPT(SQL>)에서 @C:\ORAWIN95\OBS\DEMOBLD 를 입력하고 Enter 한다.

♣ 참고

기존의 SCRIPT 는 EMP TABLE 의 이름(ename)이 CHAR(10)로 되어 있고, 업무가 CHAR(9)로 되어 있다. 그러나 여러분의 이해를 돋기 위하여 이름은 VARCHAR2(10), 업무는 VARCHAR2(9) 바꾸었다.

3. 실습용 SCRIPT 의 내용

```
-- echo Building Oracle demonstration tables. Please wait.  
DROP TABLE EMP;  
DROP TABLE DEPT;  
DROP TABLE BONUS;  
DROP TABLE SALGRADE;  
DROP TABLE DUMMY;  
DROP TABLE ITEM;
```

```

DROP TABLE PRICE;
DROP TABLE PRODUCT;
DROP TABLE ORD;
DROP TABLE CUSTOMER;
DROP VIEW SALES;
DROP SEQUENCE ORDID;
DROP SEQUENCE CUSTID;
DROP SEQUENCE PRODID;

CREATE TABLE DEPT (
    DEPTNO      NUMBER(2) NOT NULL,
    DNAME       CHAR(14),
    LOC         CHAR(13),
    CONSTRAINT DEPT_PRIMARY_KEY PRIMARY KEY (DEPTNO));

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');

CREATE TABLE EMP (
    EMPNO      NUMBER(4) NOT NULL,
    ENAME      VARCHAR2(10),
    JOB        VARCHAR2(9),
    MGR        NUMBER(4) CONSTRAINT EMP_SELF_KEY REFERENCES EMP (EMPNO),
    HIREDATE   DATE,
    SAL         NUMBER(7,2),
    COMM        NUMBER(7,2),
    DEPTNO     NUMBER(2) NOT NULL,
    CONSTRAINT EMP_FOREIGN_KEY FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO),
    CONSTRAINT EMP_PRIMARY_KEY PRIMARY KEY (EMPNO));

INSERT INTO EMP VALUES (7839, 'KING', 'PRESIDENT', NULL, '17-NOV-81', 5000, NULL, 10);
INSERT INTO EMP VALUES (7698, 'BLAKE', 'MANAGER', 7839, '1-MAY-81', 2850, NULL, 30);
INSERT INTO EMP VALUES (7782, 'CLARK', 'MANAGER', 7839, '9-JUN-81', 2450, NULL, 10);
INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 7839, '2-APR-81', 2975, NULL, 20);
INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN', 7698, '28-SEP-81', 1250, 1400, 30);
INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698, '20-FEB-81', 1600, 300, 30);
INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN', 7698, '8-SEP-81', 1500, 0, 30);
INSERT INTO EMP VALUES (7900, 'JAMES', 'CLERK', 7698, '3-DEC-81', 950, NULL, 30);
INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698, '22-FEB-81', 1250, 500, 30);
INSERT INTO EMP VALUES (7902, 'FORD', 'ANALYST', 7566, '3-DEC-81', 3000, NULL, 20);
INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902, '17-DEC-80', 800, NULL, 20);

```

```
INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST',7566,'09-DEC-82',3000,NULL,20);
INSERT INTO EMP VALUES (7876,'ADAMS','CLERK',7788,'12-JAN-83',1100,NULL,20);
INSERT INTO EMP VALUES (7934,'MILLER','CLERK',7782,'23-JAN-82',1300,NULL,10);
```

```
CREATE TABLE BONUS (
    ENAME      CHAR(10),
    JOB        CHAR(9),
    SAL         NUMBER,
    COMM        NUMBER);
```

```
CREATE TABLE SALGRADE (
    GRADE      NUMBER,
    LOSAL      NUMBER,
    HISAL      NUMBER);
```

```
INSERT INTO SALGRADE VALUES (1,700,1200);
INSERT INTO SALGRADE VALUES (2,1201,1400);
INSERT INTO SALGRADE VALUES (3,1401,2000);
INSERT INTO SALGRADE VALUES (4,2001,3000);
INSERT INTO SALGRADE VALUES (5,3001,9999);
```

```
CREATE TABLE DUMMY (
    DUMMY      NUMBER );
```

```
INSERT INTO DUMMY VALUES (0);
```

```
CREATE TABLE CUSTOMER (
    CUSTID      NUMBER (6) NOT NULL,
    NAME        CHAR (45),
    ADDRESS     CHAR (40),
    CITY        CHAR (30),
    STATE       CHAR (2),
    ZIP         CHAR (9),
    AREA        NUMBER (3),
    PHONE       CHAR (9),
    REPID       NUMBER (4) NOT NULL,
    CREDITLIMIT NUMBER (9,2),
    COMMENTS    LONG,
    CONSTRAINT CUSTOMER_PRIMARY_KEY PRIMARY KEY (CUSTID),
    CONSTRAINT CUSTID_ZERO CHECK (CUSTID > 0));
```

```
CREATE TABLE ORD (
    ORDID      NUMBER (4) NOT NULL,
```

```

ORDERDATE      DATE,
COMMPLAN      CHAR (1),
CUSTID        NUMBER (6) NOT NULL,
SHIPDATE      DATE,
TOTAL         NUMBER (8,2) CONSTRAINT TOTAL_ZERO CHECK (TOTAL >= 0),
CONSTRAINT ORD_FOREIGN_KEY FOREIGN KEY (CUSTID) REFERENCES CUSTOMER (CUSTID),
CONSTRAINT ORD_PRIMARY_KEY PRIMARY KEY (ORDID));

CREATE TABLE ITEM (
ORDID          NUMBER (4) NOT NULL,
ITEMID         NUMBER (4) NOT NULL,
PRODID         NUMBER (6),
ACTUALPRICE    NUMBER (8,2),
QTY            NUMBER (8),
ITEMTOT        NUMBER (8,2),
CONSTRAINT ITEM_FOREIGN_KEY FOREIGN KEY (ORDID) REFERENCES ORD (ORDID),
CONSTRAINT ITEM_PRIMARY_KEY PRIMARY KEY (ORDID,ITEMID));

CREATE TABLE PRODUCT (
PRODID         NUMBER (6) CONSTRAINT PRODUCT_PRIMARY_KEY PRIMARY KEY,
DESCRIP        CHAR (30));

CREATE TABLE PRICE (
PRODID         NUMBER (6) NOT NULL,
STDPRICE       NUMBER (8,2),
MINPRICE       NUMBER (8,2),
STARTDATE      DATE,
ENDDATE        DATE);

INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('96711', 'CA', '7844', '598-6609',
'JOCKSPORTS',
'100', '5000', 'BELMONT', '415', '345 VIEWRIDGE',
'Very friendly people to work with -- sales rep likes to be called Mike.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('94061', 'CA', '7521', '368-1223',
'TKB SPORT SHOP',
'101', '10000', 'REDWOOD CITY', '415', '490 BOLI RD.',
'Rep called 5/8 about change in order - contact shipping.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)

```

```

VALUES ('95133', 'CA', '7654', '644-3341',
'VOLLYRITE',
'102', '7000', 'BURLINGAME', '415', '9722 HAMILTON',
'Company doing heavy promotion beginning 10/89. Prepare for large orders during
winter.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('97544', 'CA', '7521', '677-9312',
'JUST TENNIS',
'103', '3000', 'BURLINGAME', '415', 'HILLVIEW MALL',
>Contact rep about new line of tennis rackets.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('93301', 'CA', '7499', '996-2323',
'EVERY MOUNTAIN',
'104', '10000', 'CUPERTINO', '408', '574 SURRY RD.',
'Customer with high market share (23%) due to aggressive advertising.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('91003', 'CA', '7844', '376-9966',
'K + T SPORTS',
'105', '5000', 'SANTA CLARA', '408', '3476 EL PASEO',
'Tends to order large amounts of merchandise at once. Accounting is considering
raising their credit limit. Usually pays on time.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('94301', 'CA', '7521', '364-9777',
'SHAPE UP',
'106', '6000', 'PALO ALTO', '415', '908 SEQUOIA',
'Support intensive. Orders small amounts (< 800) of merchandise at a time.');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('93301', 'CA', '7499', '967-4398',
'WOMENS SPORTS',
'107', '10000', 'SUNNYVALE', '408', 'VALCO VILLAGE',
'First sporting goods store geared exclusively towards women. Unusual promotion
al style and very willing to take chances towards new products!');
INSERT INTO CUSTOMER (ZIP, STATE, REPID, PHONE, NAME, CUSTID, CREDITLIMIT,
CITY, AREA, ADDRESS, COMMENTS)
VALUES ('55649', 'MN', '7844', '566-9123',
'NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER',
'108', '8000', 'HIBBING', '612', '98 LONE PINE WAY', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDDID, ORDERDATE, CUSTID, COMMPLAN)

```

```

VALUES ('101.4', '08-JAN-87', '610', '07-JAN-87', '101', 'A');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('45', '11-JAN-87', '611', '11-JAN-87', '102', 'B');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('5860', '20-JAN-87', '612', '15-JAN-87', '104', 'C');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('2.4', '30-MAY-86', '601', '01-MAY-86', '106', 'A');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('56', '20-JUN-86', '602', '05-JUN-86', '102', 'B');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('698', '30-JUN-86', '604', '15-JUN-86', '106', 'A');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('8324', '30-JUL-86', '605', '14-JUL-86', '106', 'A');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('3.4', '30-JUL-86', '606', '14-JUL-86', '100', 'A');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('97.5', '15-AUG-86', '609', '01-AUG-86', '100', 'B');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('5.6', '18-JUL-86', '607', '18-JUL-86', '104', 'C');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('35.2', '25-JUL-86', '608', '25-JUL-86', '104', 'C');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('224', '05-JUN-86', '603', '05-JUN-86', '102', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('4450', '12-MAR-87', '620', '12-MAR-87', '100', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('6400', '01-FEB-87', '613', '01-FEB-87', '108', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('23940', '05-FEB-87', '614', '01-FEB-87', '102', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('764', '10-FEB-87', '616', '03-FEB-87', '103', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('1260', '04-FEB-87', '619', '22-FEB-87', '104', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('46370', '03-MAR-87', '617', '05-FEB-87', '105', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('710', '06-FEB-87', '615', '01-FEB-87', '107', '');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('3510.5', '06-MAR-87', '618', '15-FEB-87', '102', 'A');
INSERT INTO ORD (TOTAL, SHIPDATE, ORDID, ORDERDATE, CUSTID, COMMPLAN)
VALUES ('730', '01-JAN-87', '621', '15-MAR-87', '100', 'A');
INSERT INTO ITEM (QTY, PRODID, ORDID, ITEMTOT, ITEMID, ACTUALPRICE)
VALUES ('1', '100890', '610', '58', '3', '58');

```

```

INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '1' , '100861' , '611' , '45' , '1' , '45' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '100' , '100860' , '612' , '3000' , '1' , '30' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '1' , '200376' , '601' , '2.4' , '1' , '2.4' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '20' , '100870' , '602' , '56' , '1' , '2.8' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '3' , '100890' , '604' , '174' , '1' , '58' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '2' , '100861' , '604' , '84' , '2' , '42' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '10' , '100860' , '604' , '440' , '3' , '44' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '4' , '100860' , '603' , '224' , '2' , '56' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '1' , '100860' , '610' , '35' , '1' , '35' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '3' , '100870' , '610' , '8.4' , '2' , '2.8' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '200' , '200376' , '613' , '440' , '4' , '2.2' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '444' , '100860' , '614' , '15540' , '1' , '35' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '1000' , '100870' , '614' , '2800' , '2' , '2.8' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '20' , '100861' , '612' , '810' , '2' , '40.5' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '150' , '101863' , '612' , '1500' , '3' , '10' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '10' , '100860' , '620' , '350' , '1' , '35' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '1000' , '200376' , '620' , '2400' , '2' , '2.4' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '500' , '102130' , '620' , '1700' , '3' , '3.4' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '100' , '100871' , '613' , '560' , '1' , '5.6' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '200' , '101860' , '613' , '4800' , '2' , '24' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ( '150' , '200380' , '613' , '600' , '3' , '4' );
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)

```

```

VALUES ('100', '102130', '619', '340', '3', '3.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('50', '100860', '617', '1750', '1', '35');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100', '100861', '617', '4500', '2', '45');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('1000', '100871', '614', '5600', '3', '5.6');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('10', '100861', '616', '450', '1', '45');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('50', '100870', '616', '140', '2', '2.8');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('2', '100890', '616', '116', '3', '58');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('10', '102130', '616', '34', '4', '3.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('10', '200376', '616', '24', '5', '2.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100', '200380', '619', '400', '1', '4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100', '200376', '619', '240', '2', '2.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('4', '100861', '615', '180', '1', '45');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('1', '100871', '607', '5.6', '1', '5.6');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100', '100870', '615', '280', '2', '2.8');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('500', '100870', '617', '1400', '3', '2.8');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('500', '100871', '617', '2800', '4', '5.6');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('500', '100890', '617', '29000', '5', '58');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100', '101860', '617', '2400', '6', '24');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('200', '101863', '617', '2500', '7', '12.5');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100', '102130', '617', '340', '8', '3.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('200', '200376', '617', '480', '9', '2.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('300', '200380', '617', '1200', '10', '4');

```

```

INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('5' , '100870' , '609' , '12.5' , '2' , '2.5');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('1' , '100890' , '609' , '50' , '3' , '50');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('23' , '100860' , '618' , '805' , '1' , '35');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('50' , '100861' , '618' , '2255.5' , '2' , '45.11');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('10' , '100870' , '618' , '450' , '3' , '45');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('10' , '100861' , '621' , '450' , '1' , '45');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100' , '100870' , '621' , '280' , '2' , '2.8');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('50' , '100871' , '615' , '250' , '3' , '5');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('1' , '101860' , '608' , '24' , '1' , '24');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('2' , '100871' , '608' , '11.2' , '2' , '5.6');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('1' , '100861' , '609' , '35' , '1' , '35');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('1' , '102130' , '606' , '3.4' , '1' , '3.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100' , '100861' , '605' , '4500' , '1' , '45');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('500' , '100870' , '605' , '1400' , '2' , '2.8');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('5' , '100890' , '605' , '290' , '3' , '58');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('50' , '101860' , '605' , '1200' , '4' , '24');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100' , '101863' , '605' , '900' , '5' , '9');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('10' , '102130' , '605' , '34' , '6' , '3.4');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('100' , '100871' , '612' , '550' , '4' , '5.5');
INSERT INTO ITEM ( QTY , PRODID , ORDID , ITEMTOT , ITEMID , ACTUALPRICE)
VALUES ('50' , '100871' , '619' , '280' , '4' , '5.6');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('4.8' , '01-JAN-85' , '100871' , '3.2' , '01-DEC-85');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)

```

```

VALUES ('58', '01-JAN-85', '100890', '46.4', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('54', '01-JUN-84', '100890', '40.5', '31-MAY-84');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('35', '01-JUN-86', '100860', '28', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('32', '01-JAN-86', '100860', '25.6', '31-MAY-86');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('30', '01-JAN-85', '100860', '24', '31-DEC-85');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('45', '01-JUN-86', '100861', '36', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('42', '01-JAN-86', '100861', '33.6', '31-MAY-86');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('39', '01-JAN-85', '100861', '31.2', '31-DEC-85');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('2.8', '01-JAN-86', '100870', '2.4', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('2.4', '01-JAN-85', '100870', '1.9', '01-DEC-85');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('5.6', '01-JAN-86', '100871', '4.8', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('24', '15-FEB-85', '101860', '18', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('12.5', '15-FEB-85', '101863', '9.4', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('3.4', '18-AUG-85', '102130', '2.8', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('2.4', '15-NOV-86', '200376', '1.75', '');
INSERT INTO PRICE (STDPRICE, STARTDATE, PRODID, MINPRICE, ENDDATE)
VALUES ('4', '15-NOV-86', '200380', '3.2', '');
CREATE INDEX PRICE_INDEX ON PRICE(PRODID, STARTDATE);
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('100860', 'ACE TENNIS RACKET I');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('100861', 'ACE TENNIS RACKET II');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('100870', 'ACE TENNIS BALLS-3 PACK');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('100871', 'ACE TENNIS BALLS-6 PACK');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('100890', 'ACE TENNIS NET');
INSERT INTO PRODUCT (PRODID, DESCRIPT)

```

```

VALUES ('101860', 'SP TENNIS RACKET');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('101863', 'SP JUNIOR RACKET');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('102130', 'RH: "GUIDE TO TENNIS"');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('200376', 'SB ENERGY BAR-6 PACK');
INSERT INTO PRODUCT (PRODID, DESCRIPT)
VALUES ('200380', 'SB VITA SNACK-6 PACK');

CREATE SEQUENCE ORDID
INCREMENT BY 1
START WITH 622
NOCACHE;

CREATE SEQUENCE PRODID
INCREMENT BY 1
START WITH 200381
NOCACHE;

CREATE SEQUENCE CUSTID
INCREMENT BY 1
START WITH 109
NOCACHE;

CREATE VIEW SALES AS
SELECT REPID, ORD.CUSTID, CUSTOMER.NAME CUSTNAME, PRODUCT.PRODID,
DESCRIPT PRODNAME, SUM(ITEMTOT) AMOUNT
FROM ORD, ITEM, CUSTOMER, PRODUCT
WHERE ORD.ORDID = ITEM.ORDID
AND ORD.CUSTID = CUSTOMER.CUSTID
AND ITEM.PRODID = PRODUCT.PRODID
GROUP BY REPID, ORD.CUSTID, NAME, PRODUCT.PRODID, DESCRIPT;

```