

ORACLE

넥스트 아이티 교육센터

데이터 베이스 시스템 개요

- 데이터

- 현실 세계로부터 단순한 관찰이나 측정을 통해서 수집된 사실이나 값

- 정보

- 데이터를 처리해서 얻어진 결과자료
- 상황에 따른 적절한 의사 결정을 할 수 있게 하는 지식으로
데이터의 유효한 해석이나 데이터 상호간의 관계

- 데이터 베이스

- 조직체 및 기업이 지속적으로 유지관리 해야 할 데이터의 집합 저장소
- 통합, 공유, 저장 데이터(**Integrated, shared, stored data**)
- 의사 결정 데이터 (**decision-making data**)

I. SQL 구조

1. SQL의 개요

SQL



- IBM 연구소에서 개발
- DB Server와 Tool 사이에 통신할 수 있는 언어
- **Structured Query Language** : 관계형데이터베이스의 표준언어
- ANSI에서 보완 RDB 시스템의 표준언어로 정함
(American National Standards Institute)
- 자료의 관계 모델에 따라 정보를 관리
- 고급언어에 삽입 가능
- 대화식 수행
- DB내에서 자료 및 조건 수행을 서술하는 문과 절로 구성

1.1 SQL의 기능

1. SQL문의 해석과 SQL문을 수행하기 위한 지원 기능
2. 호스트 언어에 삽입된 SQL문의 사전 컴파일을 지원
3. SQL문을 작성하고 수행하기 위한 대화식 인터페이스 제공



1.2 SQL 용어

관계형 데이터베이스 개념

- 1970년 **Dr.E.F.Codd**에 의해 관계형 모델이 제시
- 관계형 데이터베이스는 관계형 데이터베이스 관리 시스템(**RDBMS**)으로 이루어 진다.

항 목	내 용
개념요소	<ul style="list-style-type: none"> - Object나 Relation의 집합 - Relation에 작용할 Operation의 집합 - 데이터의 정확성과 일관성을 위한 완전성 요구
구성요소	<ul style="list-style-type: none"> ● 2차원의 Table로 구성 - Table은 한 개 또는 여러 개의 Column, 0개 또는 여러 개의 Row로 구성 - Row는 Column들의 집합. Record라고도 칭한다 - Column은 동일 종류의 데이터를 나타낸다.

SQL 역사

년 도	비 고
1973	SQUARE(Structured Queries As Relational Express)
1974	System/R ⅹ SEQUEL (Structured English QUERY Language)
1976	SEQUEL-2
1980	SQL(Structured Query Language)로 명칭 변경
1986	QL-86, 최초의 SQL 표준안
1988	ANSI, ISO 국제표준 인정
1989	SQL-1(SQL/89) 표준안 제정
1992	SQL-2(SQL/92) 표준안 제정
1999	SQL-3(SQL/99) 표준안 제정

데이터베이스 개체

- 데이터를 저장하거나 이와 상호 작용하는 다른 정보를 저장

항 목	내 용
테이블 Table	열과 행의 집합으로 2차원 배열의 데이터 저장 기본 단위
행 Row	일련의 열이 들어 있는 테이블의 수평부분
열 Colum	한 가지 자료 유형으로 된 테이블의 수직 부분
가상표 (VIEW)	하나 이상으로 된 테이블로부터 데이터의 부분을 선택하는 논리적 개념
저장 프로시저	미리 컴파일된 SQL 문장들의 집합 (stored procedure)
트리거 (trigger)	사용자가 테이블의 데이터를 수정할 때 자동으로 수행되는 저장된 프로시저

데이터베이스 개체

항 목	내 용
디폴트 (default)	사용자가 값을 지정하지 않았을 때 자동으로 입력하는 값
규칙(rule)	어떤 값이 테이블에 들어가야 하는지를 명시하고 제어
색인 (Indexes), sysindexes	<ul style="list-style-type: none">- 키 값에 의해 논리적으로 정렬된 포인터들의 집합- 키 값을 통해 테이블의 행에 있는 데이터를 접근- Query의 속도를 향상 시킴
사용자 정의 데이터형	<ul style="list-style-type: none">- 컬럼이 가지는 정보의 종류와 저장 방식을 알려 주는 구분자
제약사항 (Constraints)	<ul style="list-style-type: none">- 데이터 무결성을 유지하기 위한 구성요소

개발자의 역할

● 데이터베이스 설계자

항 목	내 용
논리적 DB설계	<ul style="list-style-type: none"> - 데이터베이스 스키마 생성 저장할 정보가 무엇인지 결정하여 데이터 구조를 설계한다. (테이블 정의 및 컬럼(column)정의)
논리적 DB구현	<ul style="list-style-type: none"> - 저장장치 - 테이블 생성, 주키 / 참조키 등의 생성 - 실질적으로 DDL(Data Definition Language)로 구성
데이터무결성 요구사항의 설계 및구현	<ul style="list-style-type: none"> - 특정 데이터에 대한 무결성을 강화하는 디폴트와 규칙을 구현한다.
데이터베이스 서버 프로그래밍	<ul style="list-style-type: none"> - 관리 기능 자동화 및 서버 작업 관리를 위한 프로그램들을 작성한다. Batch, script, 트리거, 저장 프로시저 - 서버 외부와 통신하기위한 확장된 저장 프로시저 (extended stored procedure)들을 작성한다.

SQL명령어

항 목	내 용
Query	- SELECT , 조회문
DML data manipulation language	- INSERT - UPDATE - DELETE - COMMIT, ROLLBACK 등
DDL data definition language	- CREATE - ALTER - DROP 등
DCL data control language	- GRANT - REVOKE

SQL 용어

Entity	관리대상, 객체	Table
Attribute	관리대상의 구체적 내용 데이터 타입, 길이, 제약조건 등	Column, Field
Identifier	Entity를 대표하는 Attribute	Primary Key
Tuple	여러 개의 Attribute의 집합	Row, Record
Relation	Entity와 Entity의 관계	Foreign Key

SQL	RDBMS를 사용하기 위해 ANSI에서 채택한 표준언어로 DB서버와 통신하기 위한 명령언어
SQL*Plus	Oracle DBMS를 운영하기 위해 Oracle사에서 제공하는 interface tool
PL/SQL	비즈니스 로직을 처리할 수 있도록 SQL을 확장한 절차적 언어(Procedural Language)

SQL*Plus Edit Commands

Command	Description
*A[PPEND] text	현재 line의 마지막 문장 뒤에 text를 붙인다.
*C[HANGE] /old/new	현재 line의 old text를 new text로 바꾼다.
CL[EAR] BUFF[ER]	buffer의 내용을 모두 지운다.
*DEL	현재 line을 지운다.
DEL n	n번째 line을 지운다.
DEL m n	m ~ n번째 line을 지운다.
*I[NPUT]	현재 line 다음에 line이 추가된다.
*I[NPUT] text	현재 line 다음에 line이 추가 되면서 text가 들어간다.
L[IST]	buffer전체를 보여준다.
L[IST] n	m ~ n번째 line을 보여준다
L[IST] m n	m ~ n번째 line을 보여준다.

SQL*Plus Edit Commands (cont.)

Command	Description
R[UN] or /	SQL, PL/SQL 문장을 실행
n	n번째 line을 display하면서 Editing 상태로 해준다.
n text	n번째 line이 text로 바뀐다.
0 text	1번째 line이 추가되면서 text가 1번째 line으로 들어간다.
* 표시된 명령어는 line번호를 먼저 수행한 후 실행해야 한다.	

SQL*Plus File Commands

Command	Description
SAVE filename	buffer의 내용을 filename.sql로 저장한다
GET filename	filename.sql을 buffer로 불러온다..
START filename	filename.sql을 실행한다.
@ filename	START filename과 같다.
ED[IT]	buffer의 내용을 edit program으로 실행한다.
ED[IT] filename	filename.sql을 edit program으로 실행한다
SPOOL filename	retrieve data를 filename.lst로 저장한다.
SPOOL OFF	SPOOL을 끝낸다.
EXIT	SQL*Plus를 종료한다.

SQL*Plus Variable

Command	Description
&	임시적인 사용을 위한 변수
&&	해당 세션 내에서 CLEAR하기 전 까지 사용
DEFINE	현재 선언된 사항들을 보여준다.
DEFINE <i>variable</i>	변수의 상태를 보여주거나 설정할 수 있다.
UNDEFINE <i>variable</i>	지정한 변수를 CLEAR한다.
ACCEPT	<p>ACCEPT <i>variable</i> [<i>datatype</i>] [FORMAT <i>format</i>] [PROMPT <i>prompt</i>] [HIDE]</p> <ul style="list-style-type: none"> - <i>datatype</i> : char, number, date 데이터 타입 중 하나 - <i>format</i> : 각 형식에 맞는 출력 형태 지정 - <i>prompt</i> : 입력 받기 위한 문자열을 보여줌 - HIDE : 입력하는 값이 화면에 숨겨짐

사용자 생성

현재 DB에서 작업하기 위해 사용자를 등록하도록 합니다.

사용자를 만들려면 사용자를 먼저 USER를 생성하기 위해서는 USER생성 권한이 있는 사용자로 접속해야 합니다.

● 계정 생성 구문

```
CREATE USER 사용자명  
IDENTIFIED BY 패스워드  
[ DEFAULT TABLESPACE 테이블스페이스명 ]  
[ TEMPORARY TABLESPACE 테이블스페이스명 ]  
[ QUOTA {integer [K|M] | UNLIMITED} ON 테이블스페이스명 ]  
[ ACCOUNT {LOCK | UNLOCK} ]  
[ PROFILE 프로파일명 ]
```


사용자 생성

사용자 생성 및 권한에 대한 상세 내용은 다음에 다루도록 하겠다.

- 본인의 컴퓨터명에 맞게 계정생성

```
SQL> CREATE USER pc99  
IDENTIFIED BY java;
```

```
SQL> GRANT connect, resource TO pc99;
```

```
SQL> CONNECT pc99/java;
```

```
SQL> SELECT * FROM tab;
```

데이터TYPE의 분류

TYPE	설명
VARCHAR2 (n [byte char])	가변 길이 문자열 저장 (1 ~ 4000bytes)
CHAR (n [byte char])	고정 길이 문자열 저장 (1 ~ 2000bytes)
NUMBER [(p [,s])]	숫자 데이터를 저장 (p-정체길이, s-소수점이하) -10 ¹³⁰ ~ 10 ¹²⁶ 범위까지, p=1~ 38, s=-84 ~ 127
DATE	날짜 데이터형 (BC 4712-01-01 부터 AD 9999-12-31)
TIMESTAMP	Date 확장형, 1/1000초까지
LONG	가변 길이 데이터(1 - 2G), 8i 부터 Deprecated
NCHAR, NVARCHAR2	유티코드 문자로 저장(N=National)
CLOB/BLOB/NCLOB	대용량 문자, 이진, 유티코드 데이터(1 - 4Gb) (4Gb - 1) * (database block size)

테이블

- 행과 열로 구성된 개체들을 표현하기 위한 데이터베이스 개체

항 목	내 용
특징	<ul style="list-style-type: none"> - 개체를 구분하는 고유 이름을 가진다. - 행(row, record)과 열(column, field)들로 구성 - 행과 열을 순서를 가지지 않는다. - 하나의 행은 하나의 테이블에 속한다
제약 사항	<ul style="list-style-type: none"> - 한 데이터베이스 내에서의 테이블 이름은 고유 - 한 테이블 내에서의 컬럼 이름들은 고유 - 한 테이블 내에서 열들을 고유 - 같은 열을 두번 나타낼 수는 없다

테이블의 생성

항 목	내 용
지정항목	<ul style="list-style-type: none"> - 테이블 이름 - 컬럼들 : 컬럼이름, 자료형, 제약사항 - 기본 키 및 참조키의 설정

테이블의 생성

항 목	내 용
컬럼 자료형 결정	- 시스템 자료형 또는 사용자 정의 자료형을 사용 가능
컬럼의 제약	<ul style="list-style-type: none"> - NOT NULL (NN) : 해당 컬럼에 반드시 값을 입력 - NO DUPLICATES(ND) : 테이블 내에서 해당 컬럼의 모든 값들이 달라야 함을 의미 기본 키 제약(primary key constraint) 유일제약(unique constraint) 유일색인(unique index) - NO CHANGES(NC) : 당 컬럼의 값이 변경될 수 없음을 의미
기본 키 Primary Key	<ul style="list-style-type: none"> - 테이블의 각 행이 유일함을 보장함으로써 개체 무결성을 유지하는 하나의 컬럼 또는 컬럼 들의 집합 - 모든 테이블은 기본 키를 가져야 한다 - 모든 테이블은 단 하나의 기본 키를 가질수 있다 - 기본 키는 NULL이나 중복을 허용하지 않아야 한다
참조 키 Foreign Key	<ul style="list-style-type: none"> - 다른 테이블의 기본 키를 참조하는 하나의 컬럼 또는 컬럼 들의 집합 - 두 테이블을 연결시킨다. - 참조 무결성(referential integrity)의 강화

1.3 SQL문의 유형

항 목	내 용
SQL 데이터 정의문 (DDL)	데이터베이스를 생성하거나 구조를 수정 하기 위하여 사용되는 언어 (Data Definition Language 라고도 함)
SQL 데이터 조작성문 (DML)	데이터베이스에 저장되어있는 정보를 처리하고 조작하기 위하여 사용자와 DBMS 사이에서 인터페이스 역할을 수행한다. (Data Manipulation Language) OPEN, CLOSE 등과 같은 조작성문

2. SQL 구문

CREATE TABLE

● CREATE TABLE <테이블명>

(필드명1 TYPE [NOT NULL | NULL],

필드명2 TYPE [NOT NULL | NULL],

.

.

.

Constraint 인덱스키명 **Primary Key** (필드명1 [,필드명2])

Constraint 외부키명 **Foreign Key** (필드명2)

References 외부테이블명(외부필드명))

항 목	내 용
CREATE TABLE	모든 상응하는 컬럼과 자신의 데이터타입을 갖는 새로운 테이블을 만든다.

CREATE TABLE

항 목	내 용
필드 명	사용자 정의어로 임의의 변수
TYPE	NUMBER [(p, s)] : 정수 및 실수 CHAR(n) : 고정 문자열 VARCHAR2(n) : 가변 문자열 DATE : 일시
NULL NOT NULL	NULL : 널 값 허용 NOT NULL : 널 값 허용하지 않음
CONSTRAINT	명시적인 이름 할당 (예)인덱스 명
키 명	사용자 정의어로 임의의 변수

CREATE TABLE

항 목	내 용
Primary Key	주 KEY 로써 Unique 한 하나 또는 조합된 필드
Foreign Key	관계되는 외부 KEY
References	외부 KEY 지정시 사용
외부테이블	관계가 형성되는 항목이 존재하는 테이블
외부필드명	관계가 형성되는 외부테이블의 한 항목

쉬어 가는 페이지

피곤하죠...



아~~~ 우
쉬었다가 해요

✉ 항상 준비하는 자만이 기회를 얻는다.

상품분류정보테이블 생성

```

● CREATE TABLE lprod
(
  lprod_id number(5) NOT NULL,      -- 순
  lprod_gu char(4) NOT NULL,        -- 상품분류코드
  lprod_nm varchar2(40) NOT NULL,    -- 상품분류명
  Constraint pk_lprod Primary Key (lprod_gu)
);

```

INSERT

```

● INSERT INTO 테이블 명 (필드명1, 필드명2...)
VALUES (값1, 값2...)

```

항 목	내 용
INSERT INTO	INSERT 문은 데이터를 삽입한다. INTO 는 삽입될 테이블을 지정한다.
VALUES	삽입하고자 하는 값들 을 지정 즉, 상수나 변수를 지정함.

```
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (1, 'P101', '컴퓨터제품') ;
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (2, 'P102', '전자제품');
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (3, 'P201', '여성캐주얼')
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (4, 'P202', '남성캐주얼');
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (5, 'P301', '피혁잡화');
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (6, 'P302', '화장품');
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (7, 'P401', '음반/CD');
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (8, 'P402', '도서');
INSERT INTO lprod (lprod_id, lprod_gu, lprod_nm)
                VALUES (9, 'P403', '문구류') ;
```

SELECT

- **SELECT [DISTINCT]** <데이터 목록>
FROM <테이블목록>
WHERE <검색 조건>
[GROUP BY <열 목록>]
[HAVING <검색 조건>]
[ORDER BY <열 목록>]

항 목	내 용
SELECT	검색을 원하는 COLUMN 명들을 기술
FROM	SELECT 절에서 기술된 COLUMN 명들이 포함된 table 명을 기술
DISTINCT	중복되는 행이 제거
데이터목록	조회할 필드 명(전체 *)

SELECT

항 목	내 용
WHERE	원하는 데이터의 검색 조건을 기술
GROUP BY	선택조건에 해당 그룹을 지정
HAVING	반드시 GROUP BY 절과 함께 사용됨이 바람직하며 지정된 그룹의 적용될 조건을 기술 (예) Having COUNT(*) < 4
ORDER BY	- 결과 값을 정렬 ASC DESC - DESC 가 명시되지 않으면 오름차순

- **SELECT * FROM lprod**
- **SELECT lprod_gu, lprod_nm
FROM lprod
WHERE lprod_gu > 'P102'**
- **SELECT * FROM lprod
WHERE lprod_gu > 'P201'**

UPDATE

- **UPDATE** 테이블 명 **SET** 필드명1 = 값1,
필드명2 = 값2. . .,
WHERE <검색 조건>

항 목	내 용
UPDATE	갱신하고자 하는 테이블을 지정
SET	새로운 데이터로 갱신
WHERE	갱신하기 원하는 데이터의 검색 조건

- **UPDATE** lprod
 SET lprod_nm = '향수'
 WHERE lprod_gu = 'P102'

DELETE

- **DELETE FROM** 테이블 명
WHERE <검색 조건>

항 목	내 용
DELETE	삭제하고자 하는 자료가 있는 테이블을 지정
WHERE	갱신하기 원하는 데이터의 검색 조건

- **DELETE FROM** lprod
WHERE lprod_gu = ' P202 '

※Record단위로 삭제됨



아 ! **SQL** 데이터 조작은
입력, 수정, 삭제가 기본

거래처정보테이블 생성

● CREATE TABLE buyer

(buyer_id	char(6)	NOT NULL,	-- 거래처코드
buyer_name	varchar2(40)	NOT NULL,	-- 거래처명
buyer_lgu	char(4)	NOT NULL,	-- 취급상품 대분류
buyer_bank	varchar2(60),		-- 은행
buyer_bankno	varchar2(60),		-- 계좌번호
buyer_bankname	varchar2(15),		-- 예금주
buyer_zip	char(7),		-- 우편번호
buyer_add1	varchar2(100),		-- 주소1
buyer_add2	varchar2(70),		-- 주소2
buyer_comtel	varchar2(14)	NOT NULL,	-- 전화번호
buyer_fax	varchar2(20)	NOT NULL)	-- FAX번호

ALTER

● ALTER TABLE <테이블명>

ADD (필드명 TYPE [DEFAULT value] , ...)

MODIFY (필드명 TYPE [NOT NULL] [DEFAULT value] , ...)

DROP COLUMN 필드명

ALTER

● ALTER TABLE <테이블명>

ADD (Constraint 인덱스키명 Primary Key (필드명1,필드명2),

Constraint 외부키명 Foreign key (필드명2)

References 외부테이블명(외부필드명));

항목	내 용
ALTER TABLE	1. FIELD 및 변경 2. INDEX or FOREIGN Key 변경 3. CHECK Option변경
ADD	추가 : FIELD, KEY, CHECK
MODIFY	변경 : FIELD속성
DROP	삭제 : FIELD, KEY, CHECK

거래처정보테이블 생성

거래처테이블 항목 및 Key추가

Field명	자릿수	데이터 Type	설 명
buyer_mail	60	varchar2	E-mail주소 NOT NULL
buyer_charger	20	varchar2	담당자
buyer_telext	2	varchar2	구내전화번호
buyer_name	60	varchar2	거래처명 변경 NOT NULL

PRIMARY KEY명	FIELD 명
pk_buyer	buyer_id

FOREIGN KEY명	FIELD 명
fr_buyer_prod	buyer_lgu REFERENCES lprod (lprod_gu)

- **ALTER TABLE buyer ADD (buyer_mail varchar2(60) NOT NULL,
buyer_charger varchar2(20),
buyer_telex varchar2(2));**
- **ALTER TABLE buyer
MODIFY (buyer_name varchar(60));**
- **ALTER TABLE buyer
ADD (Constraint pk_buyer Primary Key (buyer_id),
Constraint fr_buyer_lprod Foreign key (buyer_lgu)
References lprod(lprod_gu));**
- **ALTER TABLE buyer
ADD Constraint Check_Phone
CHECK (buyer_telex LIKE '[0-9][0-9]');**
- **CREATE INDEX idx_buyer ON buyer (buyer_name, buyer_id);**
- **DROP INDEX buyer_inx1;**

INSERT

```
● INSERT INTO buyer (buyer_id,buyer_name,buyer_lgu,buyer_bank,
                      buyer_bankno,buyer_bankname,buyer_zip,
                      buyer_add1,buyer_add2,buyer_comtel,buyer_fax,
                      buyer_mail, buyer_charger)
VALUES ('P10101','삼성컴퓨터','P101','주택은행','123-456-7890', '이건상',
        '135-972','서울 강남구 도곡2동 현대비전21', '1125호',
        '02-522-7890','02-522-7891', 'samcom@samcom.co.kr','송동구')
VALUES ('P10102','삼보컴퓨터','P101','제일은행','732-702-195670', '김현우',
        '142-726','서울 강북구 미아6동 행원빌딩', '2712호',
        '02-632-5690','02-632-5699', 'sambo@sambo.co.kr','김서구')
VALUES ('P10103','현주컴퓨터','P101','국민은행','112-650-397811', '심현주',
        '404-260','인천 서구 마전동', '157-899번지', '032-233-7832',
        '032-233-7833', 'hyunju@hyunju.com','강남구')
VALUES ('P10201','대우전자','P102','농협','222-333-567890', '강대우',
        '702-864','대구 북구 태전동', '232번지', '053-780-2356',
        '053-780-2357', 'daewoo@daewoo.co.kr','성대우')
VALUES ('P10202','삼성전자','P102','외환은행','989-323-567898', '박삼성',
        '614-728','부산 부산진구 부전1동 동아빌딩', '1708호',
        '051-567-5312','051-567-5313', 'samsung@samsung.com','김인우')
VALUES ('P20101','대현','P201','국민은행','688-323-567898', '신대현',
        '306-785','대전 대덕구 오정동 윤암빌딩', '508호', '042-332-5123',
        '042-332-5125', 'daehyun@daehyun.com','전대영')
```

INSERT

- VALUES ('P20102', '마르조', 'P201', '주백은행', '123-777-7890', '이마루',
'135-972', '서울 강남구 도곡2동 현대비전21', '1211호', '02-533-7890',
'02-533-7891', 'mar@marjo.co.kr', '조현상')
- VALUES ('P20201', 'LG패션', 'P202', '제일은행', '732-702-556677', '김애자',
'142-726', '서울 강북구 미아6동 행천빌딩', '5011호', '02-332-5690',
'02-332-5699', 'lgfashion@lgfashion.co.kr', '남지수')
- VALUES ('P20202', '캠브리지', 'P202', '국민은행', '112-888-397811', '안불이주',
'404-260', '인천 서구 마천동', '535-899번지', '032-255-7832',
'032-255-7833', 'cambrige@cambrige.com', '신일수')
- VALUES ('P30101', '가파치', 'P301', '농협', '211-333-511890', '김선아',
'702-864', '대구 북구 매천동', '555-66번지', '053-535-2356',
'053-535-2357', 'gapachi@gapachi.co.kr', '이수나')
- VALUES ('P30201', '한국화장품', 'P302', '외환은행', '333-355-568898', '박한국',
'614-728', '부산 부산진구 부전1동 동아빌딩', '309호',
'051-212-5312', '051-212-5313', 'hangook@hangook.com', '김사우')
- VALUES ('P30202', '피어리스', 'P302', '국민은행', '677-888-569998', '신상우',
'306-785', '대전 대덕구 오정동 윤암빌딩', '612호',
'042-222-5123', '042-222-5125', 'pieoris@pieoris.com', '이진영')
- VALUES ('P30203', '참존', 'P302', '주백은행', '555-777-567778', '오참존',
'306-785', '대전 대덕구 오정동 윤암빌딩', '1007호',
'042-622-5123', '042-622-5125', 'chamjon@chamjon.com', '성애만')

상품정보테이블 생성

● **CREATE TABLE prod**

(prod_id	varchar2(10) NOT NULL,	-- 상품코드
prod_name	varchar2(40) NOT NULL,	-- 상품명
prod_lgu	char(4) NOT NULL,	-- 상품분류
prod_buyer	char(6) NOT NULL,	-- 공급업체(코드)
prod_cost	number(10) NOT NULL,	-- 매입가
prod_price	number(10) NOT NULL,	-- 소비자가
prod_sale	number(10) NOT NULL,	-- 판매가
prod_outline	varchar2(100) NOT NULL,	-- 상품개략설명
prod_detail	clob,	-- 상품상세설명
prod_img	varchar2(40) NOT NULL,	-- 이미지(소)
prod_totalstock	number(10) NOT NULL,	-- 재고수량
prod_insdte	date,	-- 신규일자(등록일)
prod_properstock	number(10) NOT NULL,	-- 안전재고수량
prod_size	varchar2(20),	-- 크기
prod_color	varchar2(20),	-- 색상
prod_delivery	varchar2(255),	-- 배달특기사항
prod_unit	varchar2(6),	-- 단위(수량)

회원정보테이블 생성

● CREATE TABLE member

(mem_id	varchar2(15) NOT NULL,	-- 회원ID
mem_pass	varchar2(15) NOT NULL,	-- 비밀번호
mem_name	varchar2(20) NOT NULL,	-- 성명
mem_regno1	char(6) NOT NULL,	-- 주민등록번호앞6자리
mem_regno2	char(7) NOT NULL,	-- 주민등록번호뒤7자리
mem_bir	date,	-- 생일
mem_zip	char(7) NOT NULL,	-- 우편번호
mem_add1	varchar2(100) NOT NULL,	-- 주소1
mem_add2	varchar2(80) NOT NULL,	-- 주소2
mem_hometel	varchar2(14) NOT NULL,	-- 집전화번호
mem_comtel	varchar2(14) NOT NULL,	-- 회사전화번호
mem_hp	varchar2(15),	-- 이동전화
mem_mail	varchar2(60) NOT NULL,	-- E-mail주소
mem_job	varchar2(40),	-- 직업
mem_like	varchar2(40),	-- 취미
mem_memorial	varchar2(40),	-- 기념일명
mem_memorialday	date,	-- 기념일날짜
mem_mileage	number(10),	-- 마일리지
mem_delete	varchar2(1),	-- 삭제여부
Constraint pk_member Primary Key (mem_id))		



장바구니정보테이블 생성

● **CREATE TABLE cart**
(
 cart_member **varchar2(15)** **NOT NULL**, -- 회원ID
 cart_no **char(13)** **NOT NULL**, -- 주문번호
 cart_prod **varchar2(10)** **NOT NULL**, -- 상품코드
 cart_qty **number(8)** **NOT NULL**, -- 수량
 Constraint pk_cart **Primary Key** (**cart_no**,**cart_prod**),
 Constraint fr_cart_member **Foreign Key** (**cart_member**)
 References **member(mem_id)**,
 Constraint fr_cart_prod **Foreign Key** (**cart_prod**)
 References **prod(prod_id)**
)

우편번호정보테이블 생성

```
CREATE TABLE ziptb
( zipcode char(7) NOT NULL,      -- 우편번호
  sido    varchar2(2 char) NOT NULL, -- 특별시, 광역시, 도
  gugun   varchar2(10 char) NOT NULL, -- 시,군,구
  dong    varchar2(30 char) NOT NULL, -- 읍,면,동,리,건물명
  bunji   varchar2(10 char),      -- 번지,아파트동,호수
  seq     number(5) NOT NULL );   -- 자료순서
```

```
CREATE INDEX idx_ziptb_zipcode ON ziptb(zipcode);
```

SELECT 확인

전체 자료의 검색

- 테이블의 모든 **row**와 **column**을 검색
- **SELECT * FROM** 테이블명
 - 상품 테이블로부터 모든 **row**와 **column**을 검색하시오
SELECT * FROM prod
 - 회원 테이블로부터 모든 **row**와 **column**을 검색하시오 ?

특정 **COLUMN**의 검색

- 선택한 **column**만 검색
- **SELECT** 컬럼명, 컬럼명, 컬럼명, ... **FROM** 테이블명
 - 회원 테이블로부터 회원**ID**와 성명을 검색하시오
SELECT mem_id, mem_name FROM member
 - 상품 테이블로부터 상품코드와 상품명을 검색하시오 ?

SELECT 확인

산술식을 사용한 검색

- 산술연산자를 사용하여 검색되는 자료값 변경
- 산술연산식은 **COLUMN**명, 상수값, 산술연산자로 구성
- 산술연산자는 **+, -, *, /, ()** 로구성
- **SELECT** 산술연산식 **FROM** 테이블명
- 회원 테이블의 마일리지를 **12**로 나눈 값을 검색하시오

SELECT mem_mileage, mem_mileage / 12 FROM member

- 상품 테이블의 상품코드, 상품명, 판매금액을 검색 하시오?
판매금액은 = 판매단가 * **55** 로 계산한다.

COLUMN alias

- 기본적으로 **Column Heading**은 **Column명**이 출력
- 검색결과 출력되는 **COLUMN명**을 변경
- **SELECT Column명 alias, Column명 "alias", Column명 AS alias, FROM 테이블명**
 - 회원 테이블의 마일리지를 **12**로 나눈 값을 월평균으로 변경 검색
SELECT mem_mileage / 12 AS 월평균 FROM member
SELECT mem_mileage / 12 "월평균 " FROM member
SELECT mem_mileage / 12 월평균 FROM member
 - 상품테이블에서 **prod_id, prod_name, prod_buyer**를 검색하시오 ?
단, **Column Alias**는 상품코드, 상품명, 거래처코드로 정의하시오

SELECT 확인

중복 ROW의 제거

- **SELECT** 결과값에 중복된 값이 있을때 중복을 피하고 **Unique**하게 검색
- 중복된 **ROW**를 제거
- **SELECT DISTINCT Column명, Column명, Column명,**
FROM 테이블명
 - 상품 테이블의 상품분류를 중복되지 않게 검색
SELECT prod_lgu 상품분류 FROM prod
SELECT DISTINCT prod_lgu 상품분류 FROM prod
 - 상품 테이블의 거래처코드를 중복되지 않게 검색하시오 ?
(**Alias**는 거래처)

데이터의 정렬

- Row를 Sort하고자 하면 **ORDER BY** 절을 사용
 - 역순으로 Sort는 **Column**명 뒤에 **DESC**
 - **Column**명 대신 **Alias** 또는 **Select**한 **Column**의 순서로 지정 가능
 - **Default Sort**순서는 **Ascending**

항목	내 용
숫자	0 ~ 99999 순으로 정렬
날짜	1990-01-01 ~ 2001-12-31 순으로 정렬
문자	a ~ z 순으로 정렬 (좌측 1자리 부터 비교)
NULL	Ascending 에서는 앞에, Descending 에서는 뒤에 정렬

SELECT 확인

- 회원테이블에서 회원ID,회원명,생일,마일리지 검색

```
SELECT mem_id, mem_name, mem_bir, mem_mileage  
FROM member  
ORDER BY mem_id
```

```
SELECT mem_id 회원ID, mem_name 성명,  
       mem_bir 생일, mem_mileage 마일리지  
FROM member  
ORDER BY 성명
```

```
SELECT mem_id, mem_name, mem_bir, mem_mileage  
FROM member  
ORDER BY 3
```

```
SELECT mem_id, mem_name, mem_bir, mem_mileage  
FROM member  
ORDER BY mem_mileage, 1
```

- 상기 검색 조회를 역순으로 하시오

WHERE 절**비교연산자**

항목	내 용
A = B	같다
A <> B, A != B	같지않다
A < B, A > B	소 < 대 비교(미만 또는 초과)
A <= B, A >= B	소 < 대 비교(이상 또는 이하)

- 상품 중 판매가가 **170,000원**인 상품 조회

```

SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale = 170000

```

비교연산자

- 상품 중 판매가가 **170,000**원이 아닌 상품 조회
SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale <> 170000

SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale != 170000

- 상품 중 판매가가 **170,000**원초과 또는 미만인 상품 조회
SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale < 170000

SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale > 170000

비교연산자

- 상품 중 판매가가 **170,000원**이상 또는 이하인 상품 조회

```
SELECT prod_name 상품, prod_sale 판매가  
FROM prod  
WHERE prod_sale <= 170000
```

```
SELECT prod_name 상품, prod_sale 판매가  
FROM prod  
WHERE prod_sale >= 170000
```

- 상품 중 매입가가 **200,000원** 이하인 상품을 검색하시오 ?
(**Alias**는 상품코드, 상품명, 매입가)

- 회원 중 **76년도 1월 1일** 이후에 태어난 회원을 검색하시오 ?
단, 주민등록번호 앞자리로 비교
(**Alias**는 회원ID, 회원 명, 주민등록번호 앞자리)

WHERE 절

논리연산자

항목	내 용
(조건1) AND (조건2)	조건1도 참이고 조건2도 참인 경우
(조건1) OR (조건2)	조건1이 참이거나 조건2가 참인 경우
NOT(조건1)	조건1이 아닌경우 참
우선순위	(), NOT, AND, OR

- 상품 중 상품분류가 **P201(여성 캐주얼)**이고 판매가가 **170,000원**인 상품 조회

```
SELECT prod_name 상품, prod_lgu 상품분류, prod_sale 판매가
FROM prod
WHERE prod_lgu = 'P201'
      AND prod_sale = 170000
```

- 상품 중 상품분류가 **P201(여성 캐주얼)**이거나 판매가가 **170,000원**인 상품 조회

```
SELECT prod_name 상품, prod_lgu 상품분류, prod_sale 판매가  
FROM prod  
WHERE prod_lgu = 'P201'  
OR prod_sale = 170000
```

- 상품 중 상품분류가 **P201(여성 캐주얼)**도 아니고 판매가가 **170,000원**도 아닌 상품 조회

```
SELECT prod_name 상품, prod_lgu 상품분류, prod_sale 판매가  
FROM prod  
WHERE NOT(prod_lgu = 'P201' OR prod_sale = 170000)
```

- 상품 중 판매가가 **300,000원** 이상, **500,000원** 이하인 상품을 검색 하시오 ?
(**Alias**는 상품코드, 상품명, 판매가)

WHERE 절

기타연산자

IN

질의 탐색을 위해 사용될 둘이상의 표현식을 지정 (**NOT**연산자와 함께 사용 가능)

- 상품 중 판매가가 150,000원, 170,000원, 330,000원인 상품 조회

```
SELECT prod_name 상품, prod_sale 판매가  
FROM prod  
WHERE prod_sale IN (150000, 170000, 330000)
```

- 회원테이블에서 회원ID가 C001, F001, W001 인 회원만
검색하시오 ?
(Alias는 회원ID, 회원명)

- 상품 분류테이블에서 현재 상품테이블에 존재하는 분류만 검색
(분류코드, 분류명)

```
SELECT lprod_gu 분류코드, lprod_nm 분류명  
FROM lprod  
WHERE lprod_gu IN (SELECT prod_lgu FROM prod )
```

- 상품 분류테이블에서 현재 상품테이블에 존재하지 않는 분류만 검색
하시오 ?
(**Alias**는 분류코드, 분류명)

WHERE 절

기타연산자

BETWEEN

- 범위내의 모든 값을 탐색
- 두 범위의 한계 값을 포함

- 상품 중 판매가가 **100,000원** 부터 **300,000원** 사이의 상품 조회

```
SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale BETWEEN 100000 AND 300000
```

```
SELECT prod_name 상품, prod_sale 판매가
FROM prod
WHERE prod_sale >= 100000
AND prod_sale <= 300000
```

- 회원 중 생일이 **1975-01-01**에서 **1976-12-31**사이에 태어난 회원을
검색하시오 ?
(**Alias**는 회원ID, 회원 명, 생일)

- 상품 중 매입가가 **300,000~1,500,000**이고
판매가가 **800,000~2,000,000** 인 상품을 검색하시오 ?
(**Alias**는 상품명, 매입가, 판매가)

- 회원 중 생일이 **1975**년도 생이 아닌 회원을 검색하시오 ?
(**Alias**는 회원ID, 회원 명, 생일)

WHERE 절

기타연산자

LIKE

컬럼 값을 지정된 패턴과 비교하여 문자형태가 같은 Row를 검색. Wildcard를 사용 문자의 형태 지정.
% : 여러 문자 _ : 한 문자
"%" 나 "_" 을 검색하기 위해서는 ESCAPE 사용

- LIKE ' 형태 ' 또는 NOT LIKE ' 형태 '

```
SELECT prod_id 상품코드, prod_name 상품명 FROM prod
WHERE prod_name LIKE '삼%'
```

```
SELECT prod_id 상품코드, prod_name 상품명 FROM prod
WHERE prod_name LIKE '_성%'
```

```
SELECT prod_id 상품코드, prod_name 상품명 FROM prod
WHERE prod_name LIKE '%치'
```

- **SELECT prod_id 상품코드, prod_name 상품명 FROM prod
WHERE prod_name NOT LIKE '%치'**

**SELECT prod_id 상품코드, prod_name 상품명 FROM prod
WHERE prod_name LIKE '%여름%'**

**SELECT lprod_gu 분류코드, lprod_nm 분류명 FROM lprod
WHERE lprod_nm LIKE '%홍\%' ESCAPE '\'**

- 회원테이블에서 김씨 성을 가진 회원을 검색하시오 ?
(**Alias**는 회원ID, 성명)

- 회원테이블의 주민등록번호 앞자리를 검색하여 **1975**년생을 제외한 회원을 검색하시오 ?
(**Alias**는 회원ID, 성명, 주민등록번호)

함수 개요

● SQL 함수

- 컬럼의 값이나 데이터 타입을 변경할 경우
- 숫자 또는 날짜 데이터의 출력형식을 변경할 경우
- 하나 이상의 행에 대한 집계를 수행하는 경우

● SQL 함수 유형

• 단일행(Single-row) 함수

- 테이블에 저장되어있는 개별 행을 대상으로 함수를 적용하여 하나의 결과를 반환한다.
- 문자, 숫자, 날짜 등의 처리함수와 각 데이터 타입을 변환하기 위한 변환함수가 있다.
- **SELECT, WHERE, ORDER BY** 절에서 사용
- 함수를 중첩(nested) 사용 할 수 있다.

• 복수행(Multiple-row) 함수

- 여러 행을 그룹화하여 그룹별로 결과를 처리하여 하나의 결과 반환
- 그룹화하고자 하는 경우 **GROUP BY** 절을 사용

함수(문자열)

C || C

둘이상의 문자열을 연결하는 결합 연산자

- **SELECT 'a' || 'bcde' FROM dual**
SELECT mem_id || ' name is ' || mem_name FROM member

CONCAT

두 문자열을 연결하여 반환

- **SELECT CONCAT('My Name is ', mem_name) FROM member**

CHR, ASCII

ASCII값을 문자로, 문자를 ASCII값으로 반환

SELECT CHR(65) "CHR", ASCII('ABC') "ASCII" FROM dual
SELECT ASCII(CHR(65)) RESULT FROM dual
SELECT CHR(75) "CHR", ASCII('K') "ASCII" FROM dual

- 회원테이블의 회원ID Column의 ASCII값을 검색하시오 ?

SELECT ASCII(mem_id) 회원ASCII,
CHR(ASCII(mem_id)) 회원CHR
FROM member

함수(문자열)

LOWER, UPPER, INITCAP

해당 문자나 문자열을 소문자로 반환,
대문자로 반환,
첫 글자를 대문자로 나머지는 소문자로 반환

● **SELECT LOWER('DATA manipulation Language') "LOWER",
UPPER('DATA manipulation Language') "UPPER",
INITCAP('DATA manipulation Language') "INITCAP"
FROM dual**

- 회원테이블의 회원ID를 대문자로 변환하여 검색하시오 ?
(Alias명은 변환 전ID, 변환 후ID)

함수(문자열)

**LPAD,
RPAD**
(c1, n, [c2])

지정된 길이 **n**에서 **c1**을 채우고 남은 공간을 **c2**로 채워서 반환한다.

● **SELECT LPAD ('Java', 10, '*') "LPAD",
RPAD ('Flex', 12, '^') "RPAD"**
FROM dual

- 상품테이블의 소비자가격과 소비자가격을 치환하여 다음과 같이 출력되게 하시오 ?

prod_price prod_result

```
-----
290000      ****290000
390000      ****390000
490000      ****490000
1780000     ***1780000
2780000     ***2780000
...         ...
```

함수(문자열)

LTRIM, RTRIM (c1, [c2])	LTRIM 은 좌측, RTRIM 은 우측의 공백문자를 제거 c2 문자가 있는 경우 일치하는 문자를 제거
--	--

- **SELECT '<' || LTRIM(' AAA ') || '>' "LTRIM1",
'<' || LTRIM('Hello World', 'He') || '>' "LTRIM2"**
FROM dual
- **SELECT '<' || RTRIM(' AAA ') || '>' "RTRIM1",
'<' || RTRIM('Hello World', 'ld') || '>' "RTRIM2"**
FROM dual

TRIM	LTRIM, RTRIM 함수를 조합한 형태 ([[LEADING TRAILING BOTH] [c1] FROM] source)
-------------	---

- **SELECT '<' || TRIM(' AAA ') || '>' TRIM1,
'<' || TRIM(LEADING 'a' FROM 'aaAaBaAaa') || '>' TRIM2,
'<' || TRIM('a' FROM 'aaAaBaAaa') || '>' TRIM3,**
FROM dual

함수(문자열)

SUBSTR **(c, m, [n])**

문자열의 일부분을 선택
c문자열의 **m**위치부터 길이 **n**만큼의 문자 리턴
m이 **0** 또는 **1**이면 첫 글자를 의미
m이 음수이면 뒤쪽에서부터 처리

```
● SELECT SUBSTR('SQL PROJECT', 1, 3) RESULT1,  
        SUBSTR('SQL PROJECT', 5) RESULT2,  
        SUBSTR('SQL PROJECT', -7, 3) RESULT3  
FROM dual
```

-회원테이블의 성씨 조회

```
SELECT mem_id, SUBSTR(mem_name, 1, 1) 성씨  
FROM member
```

- 상품테이블의 상품명의 4째 자리부터 2글자가 '칼라' 인 상품의
상품코드, 상품명을 검색하시오 ?
(**Alias**명은 상품코드, 상품명)

함수(문자열)

- 상품테이블의 상품코드에서 왼쪽**4**자리, 오른쪽**6**자리를 검색하시오 ?
(**Alias**명은 상품코드, 대분류, 순번)

TRANSLATE (c1, c2, c3)

c1문자열에 포함된 문자 중 **c2**에 지정된 문자가 **c3**문자로 각각 변경
c3 문자가 **c2**보다 적은 경우 해당 문자는 제거

● SELECT

TRANSLATE('2009-02-28', '0123456789-', 'ABCDEFGHIJK') RESULT
FROM dual

- 회원테이블의 회원아이디에서 숫자를 제거하여 출력하시오?
(**Alias**명은 회원아이디, 변환아이디)

함수(문자열)

REPLACE (c1, c2, [c3])

문자나 문자열을 치환
c1에 포함된 c2문자를 c3값으로 치환,
c3가 없는 경우 찾은 문자를 제거한다.

- **SELECT REPLACE('SQL Project', 'SQL', 'SSQQLL') 문자치환1,
REPLACE('Java Flex Via', 'a') 문자치환2
FROM dual**

- 거래처 테이블의 거래처명중 '삼' --> '육' 으로 치환

**SELECT buyer_name, REPLACE(buyer_name, '삼', '육') "삼->육 "
FROM buyer**

- 회원테이블의 회원성명 중 '이' --> '리' 로 치환 검색하시오 ?
(Alias명은 회원명, 회원명치환)

함수(문자열)

INSTR
(c1, c2, [m, [n]])

c1문자열에서 c2문자가 처음 나타나는 위치를 리턴
m은 시작 위치, n은 n번째

- **SELECT INSTR('hello heidi', 'he') RESULT1,
INSTR('hello heidi', 'he', 3) RESULT2
FROM dual**

LENGTH

문자열의 길이를 돌려준다.

- **SELECT LENGTH('SQL 프로젝트') "LENGTH",
LENGTHB('SQL 프로젝트') "LENGTHB"**

-거래처테이블의 거래명 조회

**SELECT buyer_id, LENGTH(buyer_id) 거래처코드길이,
buyer_name, LENGTH(buyer_name) 거래처명길이,
LENGTHB(buyer_name) 거래처명byte수
FROM buyer**

함수(숫자열)

함수	내용
ABS(n)	절대 값
SIGN(n)	양수, 음수, 0 을 구분. 각 1, -1, 0 리턴 음수인지를 판단하거나 비교시 사용
POWER(n, y)	승수 값 (n^y)
SQRT(n)	n의 제곱근

- **SELECT ABS(-365) FROM dual**
- **SELECT SIGN(12), SIGN(0), SIGN(-55)
FROM dual**
- **SELECT POWER(3, 2) , POWER(2, 10)
FROM dual**
- **SELECT SQRT(2) , SQRT(9) FROM dual**

함수(숫자열)

**GREATEST,
LEAST
(m [,n1 ...])**

열거된 항목 중 가장 큰 또는 작은 항목을 리턴
m의 데이터타입에 따라 n의 항목도 판단

- **SELECT GREATEST(10, 20, 30) "큰값",
LEAST(10,20,30) "작은값" FROM dual**

- **SELECT GREATEST('강아지', 256, '송아지') "큰값",
LEAST('강아지', 256, '송아지') "작은값"
FROM dual**

- 회원 테이블에서 회원이름, 마일리지를 출력하시오
(단, 마일리지가 **1000**보다 작은 경우 **1000**으로 변경)

함수(숫자열)

ROUND(n, I)	지정된 자릿수(I) 밑에서 반올림 - 숫자의 반올림 : ROUND(Column명, 위치)
TRUNC(n, I)	ROUND 와 동일. 단, 반올림이 아닌 절삭

- **Column값 1234.567 일때**
위치 : -3, -2, -1, 0, 1, 2, 3

SELECT ROUND(345.123, 0) 결과 FROM dual

**SELECT ROUND(345.123, -1) 결과1,
TRUNC(345.123, -1) 결과2 FROM dual**

- 회원 테이블의 마일리지를 **12**로 나눈 값을 검색
(소수3째자리 반올림, 절삭)

**SELECT mem_mileage,
ROUND(mem_mileage / 12, 2),
TRUNC(mem_mileage / 12, 2)
FROM member**

함수(숫자열)

- 상품테이블의 상품명, 원가율(매입가 / 판매가)을 비율(%)로 (반올림 없는 것과 소수 첫째자리 반올림 비교) 검색하시오 ?
(Alias는 상품명, 원가율1, 원가율2)

MOD(c, n)

n으로 나눈 나머지

- **SELECT MOD(10,3) FROM dual**

함수(숫자열)

FLOOR(n)	n과 같거나 작은 수 중에 가장 큰 정수
CEIL(n)	n과 같거나 큰 수 중에 가장 작은 정수 소수점 이하의 값이 존재하면 무조건 올림하는 함수로, 급여, 세금과 같은 금액관련 계산 중에 자주 사용된다.
COS, SIN, TAN, LOG 등 수학과관련 함수	

- **SELECT FLOOR(1332.69), CEIL(1332.69) FROM dual**
- **SELECT FLOOR(-1332.69), CEIL(-1332.69) FROM dual**

함수(숫자열)

REMAINDER (c, n)

n으로 나눈 나머지, MOD 함수와 유사

MOD 함수, 나머지 값 구하기 함수 구현방식

$$\begin{aligned}\text{나머지} &= (\text{원값}) - (\text{나눌값} \times \text{나눌값을 소수 첫째자리에서 버림한 값}) \\ &= 10 - 3 * \text{FLOOR}(10 / 3)\end{aligned}$$

REMAINDER 함수, 나머지 값 구하기 함수 구현방식

$$\begin{aligned}\text{나머지} &= (\text{원값}) - (\text{나눌값} \times \text{나눌값을 소수 첫째자리에서 반올림한 값}) \\ &= 10 - 3 * \text{ROUND}(10 / 3)\end{aligned}$$

- **SELECT MOD(10, 3), REMAINDER(10, 3) FROM dual**
- **SELECT MOD(10, 3.7) , REMAINDER(10, 3.7) FROM dual**

함수(숫자열)

WIDTH_BUCKET
(c, min, max, b)

min에서 **max**의 범위로 설정하고 **b**구간으로 나누어 **c**가 어느 구간에 속하는지 리턴

● **SELECT WIDTH_BUCKET(88, 0, 100, 10) FROM dual**

- 회원 테이블에서 회원이름, 마일리지, 등급을 출력하시오
(단, 등급은 마일리지를 **500** 부터 **3000**까지 **5**등급으로 한다.)

함수(숫자열)

숫자의 절사 (버림함수구현)

지정된 자리 수까지 나타내고 그 밑은 버림(**TRUNC**)

- 버림함수 구현

버림할 위치의 값에 5를 뺀 후에 반올림하면 버림의 효과를 얻는다.

```
SELECT ROUND(345.123 - 0.05, 1) RESULT1,  
       ROUND(345.123 - 0.5, 0) RESULT2,  
       ROUND(345.123 - 5, -1) RESULT3
```

- 100 / 9 결과 값을 십의 자리까지 나타내고 일의 자리는 버리시오

```
SELECT ROUND(100 / 9 - 5, -1) RESULT
```

- **FLOOR** 함수 활용

```
SELECT 10000 / 7 FROM dual  
SELECT FLOOR(10000 / 7 * 1000) / 1000 FROM dual  
SELECT FLOOR(10000 / 7 * 100) / 100 FROM dual  
SELECT FLOOR(10000 / 7 * 10) / 10 FROM dual  
SELECT FLOOR(10000 / 7) FROM dual  
SELECT FLOOR(10000 / 7 / 10) * 10 FROM dual  
SELECT FLOOR(10000 / 7 / 100) * 100 FROM dual  
SELECT FLOOR(10000 / 7 / 1000) * 1000 FROM dual
```

함수(숫자열)

- 상품테이블의 상품명, 원가율(매입가 / 판매가)을 비율(%)로 (가공 없는 원래 결과와 소수 첫째자리까지 나타내고 둘째자리 이하는 버린 결과 비교) 검색 ?
(**TRUNC**함수를 사용하면 쉽지만 여기서는 **FLOOR** 사용)

함수(날짜열)

SYSDATE

시스템에서 제공하는 현재 날짜와 시간 값

- **date + NUMBER** : 숫자만큼 일수가 더해진 날짜 **Return**
- **date - NUMBER** : 숫자만큼 일수가 빠진 날짜 **Return**
- **date - date** : 두 날짜 사이의 일수 **Return**
- **date + 1 / 24** : 한시간을 더한 날짜 **Return**

```
SELECT SYSDATE "현재시간",  
       SYSDATE - 1 "어제 이시간",  
       SYSDATE + 1 "내일 이시간"  
FROM dual
```

- 회원테이블의 생일과 **12000**일째 되는 날을 검색하시오 ?
(**Alias**는 회원명, 생일, **12000**일째)

함수(날짜열)

ADD_MONTHS (date, n)	date에 월을 더한 날짜
--------------------------------	----------------

- SELECT ADD_MONTHS(SYSDATE, 5) FROM dual

NEXT_DAY (date, char)	해당 날짜 이후의 가장 빠른 요일의 날짜 char : 월, 월요일, ...
LAST_DAY (date)	월의 마지막 날짜

- SELECT NEXT_DAY(SYSDATE, '월요일'),
LAST_DAY(SYSDATE)
FROM dual

- 이번달이 며칠이 남았는지 검색하시오?

함수(날짜열)

ROUND (date [, fmt])	char 기준으로 날짜를 반올림한 날짜 fmt : YEAR, MONTH, DAY, ...
TRUNC (date [,fmt])	fmt 기준으로 날짜를 버림한 날짜

- **SELECT ROUND(SYSDATE,'MM'),
 TRUNC(SYSDATE, 'MM')
FROM dual**
- **SELECT ROUND(SYSDATE,'YEAR'),
 TRUNC(SYSDATE, 'YEAR')
FROM dual**

함수(날짜열)

● ROUND/TRUNC에서 사용하는 포맷

의 미	FORMAT
년도	CC, YEAR, YYYY, YYYY, YY, Y
분기	Q
월	MONTH, MON, MM, RM
주차, 일차, 요일(숫자)	WW, IW, W
일	DDD, DD, J
주의 시작일	DAY, DY, D
시간	HH, HH12, HH24
분	MI

함수(날짜열)

MONTHS_BETWEEN
(date1, date2)

두 날짜 사이의 달수를 숫자로 리턴

- **SELECT MONTHS_BETWEEN(SYSDATE, '2000-01-01')**
FROM dual

함수(날짜열)

EXTRACT
(fmt FROM date)

날짜에서 필요한 부분만 추출

(fmt=YEAR, MONTH, DAY, HOUR, MINUTE, SECOND)

- **SELECT EXTRACT(YEAR FROM SYSDATE) "년",
EXTRACT(MONTH FROM SYSDATE) "월",
EXTRACT(DAY FROM SYSDATE) "일"
FROM dual**

- 생일이 3월인 회원을 검색하시오?

함수(Conversion)

항목	내 용
CAST (expr AS type)	명시적으로 형 변환

- **SELECT '[' || CAST('Hello' AS CHAR(30)) || ']' "형변환"**
FROM dual
- **SELECT CAST('1997/12/25' AS DATE) FROM dual**

TO_CHAR	숫자, 문자, 날짜를 지정한 형식의 문자열 반환
TO_NUMBER	숫자형식의 문자열을 숫자로 반환
TO_DATE	날짜형식의 문자열을 날짜로 반환

위 3가지 변환함수는 자주 사용함.

함수(Conversion)

의 미	날짜 및 시간 형식의 FORMAT
세기 및 년도	AD, BC, CC, YEAR, YYYY, YYY, YY, Y
분기	Q
월	MONTH, MON, MM, RM
주차	W, WW, IW
일, 일차	DD, DDD, J
주의 요일	DAY, DY, D
오전/오후	AM, PM, A.M., P.M.
시간	HH, HH12, HH24
분	MI
초	SS, SSSSS(자정 이후의 초 0 - 86399)
기타	" " (문자열 직접출력)

함수(Conversion)

TO_CHAR	숫자, 문자, 날짜를 지정한 형식의 문자열 변환
(char)	CHAR, CLOB 타입을 VARCHAR2로 변환
(date [, fmt])	날짜를 특정 형식의 문자열로 변환
(number [, fmt])	숫자를 특정 형식의 문자열로 변환

- **SELECT TO_CHAR(SYSDATE, 'AD YYYY, CC"세기" ')
FROM dual**
- **SELECT
TO_CHAR(CAST('2008-12-25' AS DATE) ,
'YYYY.MM.DD HH24:MI')
FROM dual**

함수(Conversion)

- 상품테이블에서 상품입고일을 '**2008-09-28**' 형식으로 나오게 검색하시오 (**Alias** 상품명, 상품판매가, 입고일)

- 회원이름과 생일로 다음처럼 출력되게 작성하시오.

김은대님은 1976년 1월 출생이고 태어난 요일은 목요일
이별이님은 1974년 1월 출생이고 태어난 요일은 월요일
신용환님은 1974년 1월 출생이고 태어난 요일은 목요일

...

함수(Conversion)

숫자 FORMAT	내 용
9	출력형식의 자리, 유효한 숫자인 경우 출력
0	출력형식의 자리, 무효한 숫자인 경우 0출력
\$, L	달러 및 지역 화폐기호
MI	음수인 경우 우측에 마이너스 표시, 우측에 표시
PR	음수인 경우 "< >" 괄호로 묶는다, 우측에 표시
,(comma) .(dot)	해당위치에 ", " 표시, 소수점 기준
X	해당 숫자를 16진수로 출력. 단독으로 사용

- `SELECT TO_CHAR(1234.6, '99,999.00')`
`FROM dual`
- `SELECT TO_CHAR(-1234.6, 'L9999.00PR')`
`FROM dual`
- `SELECT TO_CHAR(255, 'XXX') FROM dual`

함수(Conversion)

- 상품테이블에서 상품코드, 상품명, 매입가격, 소비자가격, 판매가격을 출력하시오. (단, 가격은 천단위 구분 및 원화표시)

TO_NUMBER	숫자형식의 문자열을 숫자로 반환
(char [, fmt])	fmt는 TO_CHAR에서 사용했던 숫자형식과 동일

- SELECT TO_NUMBER('3.1415')
FROM dual
- SELECT TO_NUMBER('₩1,200')
FROM dual

- 두번째 실행문장은 무엇을 해야 할까요?

함수(Conversion)

- 회원테이블에서 이쁜이회원의 회원ID 2~4 문자열을 숫자형으로 치환한 후 10을 더하여 새로운 회원ID로 조합하시오 ?
(Alias는 회원ID, 조합회원ID)

회원ID	조합회원ID

b001	b011

(1 row(s) affected)

함수(Conversion)

TO_DATE	날짜형식의 문자열을 DATE형으로 반환
(char [, fmt])	fmt는 TO_CHAR에서 사용했던 날짜형식과 동일

- **SELECT TO_DATE('2009-03-05') + 3
FROM dual**
- **SELECT TO_DATE('200803101234', 'YYYYMMDDHH24MI')
FROM dual**
- 두번째 실행문장은 무엇을 해야 할까요?

함수(Conversion)

- 회원테이블에서 주민등록번호1을 날짜로 치환한 후 검색하시오
(**Alias**는 회원명, 주민등록번호1, 치환날짜)

함수(GROUP)

항목	내 용
AVG(column)	<p>조회 범위 내 해당 컬럼 들의 평균값</p> <ul style="list-style-type: none"> - DISTINCT : 중복된 값은 제외 - ALL : Default로써 모든 값을 포함 - Column명 : NULL값은 제외 * : NULL값도 포함(COUNT함수만 사용)

- **SELECT AVG(DISTINCT prod_cost), AVG(All prod_cost),
AVG(prod_cost) 매입가평균
FROM prod**
- **상품테이블의 상품분류별 매입가격 평균 값**
**SELECT prod_lgu,
ROUND(AVG(prod_cost), 2) "분류별 매입가격 평균"
FROM prod
GROUP BY prod_lgu**

함수(GROUP)

- 상품테이블의 총 판매가격 평균 값을 구하시오 ?
(Alias는 상품총판매가격평균)
- 상품테이블의 상품분류별 판매가격 평균 값을 구하시오 ?
(Alias는 상품분류, 상품분류별판매가격평균)

함수(GROUP)

COUNT(col)
COUNT(*)

조회 범위 내 해당 컬럼 들의 자료 수
선택된 자료의 수

- **SELECT COUNT(DISTINCT prod_cost), COUNT(All prod_cost),
COUNT(prod_cost), COUNT(*)
FROM prod**

- 상품 테이블의 자료 수

**SELECT COUNT(*) RESULT1, COUNT(prod_lgu) RESULT2
FROM prod**

상품테이블의 상품분류별 자료의 수

**SELECT prod_lgu, COUNT(*) "상품 분류별 자료의 수"
FROM prod
GROUP BY prod_lgu**

- 거래처테이블의 담당자를 컬럼으로 하여 **COUNT** 집계 하시오 ?
(**Alias**는 자료수(**DISTINCT**), 자료수, 자료수(*****))

함수(GROUP)

- 회원테이블의 취미종류수를 **COUNT** 집계 하시오 ?
(**Alias**는 취미종류수)

```
SELECT COUNT(DISTINCT mem_like) 취미종류수  
FROM member
```

- 회원테이블의 취미별 **COUNT** 집계 하시오 ?
(**Alias**는 취미, 자료수, 자료수(*))

```
SELECT mem_like 취미,  
       COUNT(mem_like) 자료수, COUNT(*) "자료수(*)"  
FROM member  
GROUP BY mem_like
```

- 회원테이블의 직업종류수를 **COUNT** 집계 하시오 ?
(**Alias**는 직업종류수)

함수(GROUP)

- 회원테이블의 직업별 **COUNT**집계 하시오 ?
(**Alias**는 직업, 자료수, 자료수(*))

```
SELECT mem_job 직업,  
       COUNT(mem_job) 자료수, COUNT(*) "자료수(*)"  
FROM member  
GROUP BY mem_job
```

- 장바구니테이블의 회원별 **COUNT**집계 하시오 ?
(**Alias**는 회원ID, 자료수(**DISTINCT**), 자료수, 자료수(*))

함수(GROUP)

**MAX(col),
MIN(col)**

조회 범위 내 해당 컬럼 들 중 최대 값과 최소 값

- **SELECT MAX(DISTINCT prod_cost), MAX(prod_cost),
MIN(DISTINCT prod_cost), MIN(prod_cost)
FROM prod**
- **상품 중 최고판매가격과 최저판매가격**
SELECT MAX(prod_sale) 최고판매가,
MIN(prod_sale) 최저판매가
FROM prod
- **상품 중 거래처별 최고매입가격과 최저매입가격**
SELECT prod_buyer 거래처,
MAX(prod_cost) 최고매입가,
MIN(prod_cost) 최저매입가
FROM prod
GROUP BY prod_buyer

함수(GROUP)

- 장바구니테이블의 회원별 최대구매수량을 검색 하시오 ?
(**Alias**는 회원ID, 최대수량, 최소수량)

```
SELECT cart_member 회원ID,  
       MAX(DISTINCT cart_qty) "최대수량(DISTINCT)",  
       MAX(cart_qty) 최대수량,  
       MIN(DISTINCT cart_qty) "최소수량(DISTINCT)",  
       MIN(cart_qty) 최소수량  
FROM cart  
GROUP BY cart_member
```

- 오늘이 2000년도7월11일이라 가정하고 장바구니테이블에 발생될
추가주문번호를 검색 하시오 ?
(**Alias**는 최고치주문번호, 추가주문번호)

함수(GROUP)

SUM(column)	조회 범위 내 해당 컬럼 들의 합계
--------------------	---------------------

- 상품테이블의 매입가의 총합계 값
SELECT SUM(DISTINCT prod_cost), SUM(prod_cost)
FROM prod
- 상품테이블의 판매가의 총합계 값
SELECT SUM(prod_sale) "상품 판매가 총합계"
FROM prod
- 상품테이블의 상품분류별 판매가 합계 값
SELECT prod_lgu, SUM(prod_sale) "분류별 판매가 합계"
FROM prod
GROUP BY prod_lgu
- 상품입고테이블의 상품별 입고수량의 합계 값
SELECT buy_prod 상품, SUM(buy_qty) "입고수량합계"
FROM buyprod
GROUP BY buy_prod

함수(GROUP)

- 장바구니테이블의 상품분류별 판매수량의 합계 값
(**Alias**는 상품, 판매수량합계)

```
SELECT SUBSTR(cart_prod, 0, 4) 상품분류,  
       SUM(cart_qty) "판매수량 합계"  
FROM cart  
GROUP BY SUBSTR(cart_prod, 0, 4)
```

- 회원테이블의 회원전체의 마일리지 평균, 마일리지 합계,
최고 마일리지, 최소 마일리지,인원수를 검색하시오 ?
(**Alias**는 마일리지평균, 마일리지합계,최고마일리지,
최소마일리지,인원수)

함수(GROUP)

- 상품테이블에서 판매가전체의 평균,합계,최고 값,최저 값,자료 수를 검색하시오 ? (**Alias**는 평균, 합계, 최고값, 최저값, 자료수)

함수(소 GROUP 분리)

소 GROUP

- 집계함수를 제외한 **select**절에 기술된 **column**명들은 모두 **GROUP BY**절에 기술
- **GROUP BY**절에 기술된 **column**명들은 **select**절에 기술되지 않아도 무방
- 하지만 결과를 파악하기 위해서는 **select**절에 기술해 주는 것이 타당
- **GROUP BY**절을 기술하면 **GROUP BY**절에 기술된 **column**값으로 1개의 **Table**이 소**GROUP**으로 나뉜다.

- 상품테이블에서 거래처, 상품분류별로 최고판매가, 최소판매가, 자료 수를 검색하시오 ?

```
SELECT prod_buyer 거래처, prod_lgu 상품분류,  
       MAX(prod_sale) 최고판매가,  
       MIN(prod_sale) 최소판매가,  
       COUNT(prod_sale) 자료수  
FROM prod  
GROUP BY prod_buyer, prod_lgu  
ORDER BY prod_buyer, prod_lgu
```

함수(소 GROUP 분리)

- 장바구니테이블에서 회원, 상품분류별로 구매수량평균, 구매수량합계, 자료수를 검색하시오 ?
(**Alias**는 회원ID, 상품분류, 구매수량평균, 구매수량합계, 자료수)
(회원ID, 상품분류 순으로 **SORT**하시오)

```
SELECT cart_member 회원ID,  
       SUBSTR(cart_prod, 1, 4) 상품분류,  
       AVG(cart_qty) 구매수량평균,  
       SUM(cart_qty) 구매수량합계,  
       COUNT(cart_qty) 자료수  
FROM cart  
GROUP BY cart_member, SUBSTR(cart_prod, 1, 4)  
ORDER BY cart_member, SUBSTR(cart_prod, 1, 4)
```


함수(소 GROUP 분리)

- 회원테이블에서 지역(주소1의 2자리),생일년도별로 마일리지평균, 마일리지합계, 최고마일리지, 최소마일리지, 자료수를 검색하시오 ?
(**Alias**는 지역,생일연도, 마일리지평균, 마일리지합계, 최고마일리지,최소마일리지, 자료수)

함수(NULL)

- 데이터를 처리할 때 **NULL** 값의 사용은 최대한 줄여야 하지만 사용해야 할 경우가 있다.
- **NULL** 사용 예
 - 학생에 대한 정보를 입력할 때 '전화번호' 속성은 전화번호가 없는 학생이 있을 수 있다. 이런 경우에는 **NULL** 값을 사용한다.
- **NULL** 값은 0, 1과 같은 특정한 값이 아니고 아무 것도 없는 것을 뜻한다.
- **SQL**에서 **NULL** 값을 허용하지 않는 속성에 대해 **NULL** 값으로 수정하려 한다면 에러가 발생한다.
- **Oracle**은 빈 공백(**White Space**)도 **Null** 로 인식한다.
(**MySQL**, **SQL Server** 인 경우 빈 공백은 빈 공백으로 처리)

함수(NULL)

--거래처테이블에서 거래처명, 담당자 조회

```
SELECT buyer_name 거래처, buyer_charger 담당자  
FROM buyer
```

--거래처 담당자 성씨가 '김 ' 이면 **NULL**로 갱신

```
SELECT buyer_name 거래처, buyer_charger 담당자  
FROM buyer  
WHERE buyer_charger LIKE '김%'
```

```
UPDATE buyer SET buyer_charger = NULL  
WHERE buyer_charger LIKE '김%'
```

--거래처 담당자 성씨가 '성 ' 이면 **White Space**로 갱신

```
SELECT buyer_name 거래처, buyer_charger 담당자  
FROM buyer  
WHERE buyer_charger LIKE '성%'
```

```
UPDATE buyer SET buyer_charger = "  
WHERE buyer_charger LIKE '성%'
```

함수(NULL관련)

항목	내용
IS NULL, IS NOT NULL	NULL값인지 아닌지 비교
NVL(c, r)	c가 Null이 아니면 c값으로, Null이면 r 반환
NVL2(c, r1, r2)	c가 Null이 아니면 r1값으로, Null이면 r2 반환
NULLIF(c, d)	c와 d를 비교하여 같으면 NULL을 다르면 c값을 돌려준다.
COALESCE (p [, p ...])	파라미터중 Null이 아닌 첫 번째 파라미터 반환

- 해당 컬럼이 NULL값 비교 조회

- NULL을 이용 NULL값 비교

```
SELECT buyer_name 거래처, buyer_charger 담당자
FROM buyer
WHERE buyer_charger = NULL
```

함수(NULL관련)

- **SELECT buyer_name 거래처, buyer_charger 담당자**
FROM buyer
WHERE buyer_charger IS NULL
- **SELECT buyer_name 거래처, buyer_charger 담당자**
FROM buyer
WHERE buyer_charger IS NOT NULL
- 해당 컬럼이 **NULL**일 경우에 대신할 문자나 숫자 치환
 - 1) **NULL**이 존재하는 상태로 조회
SELECT buyer_name 거래처, buyer_charger 담당자
FROM buyer
 - 2) **NVL**을 이용 **NULL**값일 경우만 '없다' 로 치환
SELECT buyer_name 거래처,
NVL(buyer_charger, '없다') 담당자
FROM buyer

함수(NULL관련)

- 전체회원 마일리지에 **100**을 더한 수치를 검색하시오
(**Alias**는 성명, 마일리지, 변경마일리지)

```
SELECT mem_name 성명, mem_mileage 마일리지,  
       mem_mileage+100 변경마일리지  
FROM member
```

- 회원 성씨가 '바' 을 포함하면 마일리지를 **NULL**로 갱신

```
SELECT mem_name 성명, mem_mileage 마일리지  
FROM member  
WHERE mem_name >= '바' AND mem_name <= '빌'
```

```
UPDATE member SET mem_mileage = NULL  
WHERE mem_name >= '바' AND mem_name <= '빌'
```

3) 정확한 연산을 위한 **NVL** 이용

```
SELECT NULL + 10 덧셈, 10 * NULL 곱셈 FROM dual
```

```
SELECT NVL(NULL,0) + 10 덧셈,  
       10 * NVL(NULL,0) 곱셈 FROM dual
```

함수(NULL관련)

- 회원 마일리지에 **100**을 더한 수치를 검색하시오 ?
(**NVL** 사용, **Alias**는 성명, 마일리지, 변경마일리지)

- 회원 마일리지에 있으면 '정상 회원', **Null**이면 '비정상 회원' 으로
검색하시오 ?
(**NVL2** 사용, **Alias**는 성명, 마일리지, 회원상태)

함수(NULL관련)

- **SELECT NULLIF(123, 123) RESULT1,
NULLIF(123, 1234) RESULT3,
NULLIF('A', 'B') RESULT4
FROM dual**
- **SELECT COALESCE(Null, Null, 'Hello', Null, 'World')
FROM dual**

함수(Miscellaneous)

항목	내용
DECODE	IF문과 같은 기능을 함 (expr {[,search, result]} [, default])
CASE WHEN	연속적인 조건 문 (표준) CASE WHEN ~ THEN ~ ELSE ~ END

DECODE 함수와 CASE 문은 실무에서 자주 사용함.

- **SELECT DECODE(9, 10, 'A', 9, 'B', 8, 'C', 'D')**
FROM dual

- **SELECT DECODE(SUBSTR(prod_lgu, 1,2),**
 'P1', '컴퓨터/전자 제품',
 'P2', '의류',
 'P3', '잡화', '기타')
FROM prod

함수(Miscellaneous)

-상품 분류중 앞의 두 글자가 'P1' 이면 판매가를 10%인상하고 'P2' 이면 판매가를 15%인상하고, 나머지는 동일 판매가로 검색하시오 ?
(DECODE 함수 사용, Alias는 상품명, 판매가, 변경판매가)

```

● SELECT CASE WHEN '나' = '나' THEN '맞다'
              ELSE '아니다' END RESULT
FROM dual

```

함수(Miscellaneous)

- **SELECT CASE '나' WHEN '철호' THEN '아니다'**
WHEN '너' THEN '아니다'
WHEN '나' THEN '맞다'
ELSE '모르겠다' END RESULT

FROM dual

- **SELECT prod_name 상품, prod_lgu 분류,**
CASE WHEN prod_lgu = 'P101' THEN '컴퓨터제품'
WHEN prod_lgu = 'P102' THEN '전자제품'
WHEN prod_lgu = 'P201' THEN '여성캐주얼'
WHEN prod_lgu = 'P202' THEN '남성캐주얼'
WHEN prod_lgu = 'P301' THEN '피혁잡화'
WHEN prod_lgu = 'P302' THEN '화장품'
WHEN prod_lgu = 'P401' THEN '음반/CD'
WHEN prod_lgu = 'P402' THEN '도서'
WHEN prod_lgu = 'P403' THEN '문구류'
ELSE '미등록분류'

END "상품 분류"

FROM prod

함수(Miscellaneous)

- 10만원 초과 상품 판매가 가격대를 검색하시오

```
SELECT prod_name 상품, prod_price 판매가,  
       CASE WHEN (100000 - prod_price) > 0 THEN '10만원 미만'  
            WHEN (200000 - prod_price) > 0 THEN '10만원대'  
            WHEN (300000 - prod_price) > 0 THEN '20만원대'  
            WHEN (400000 - prod_price) > 0 THEN '30만원대'  
            WHEN (500000 - prod_price) > 0 THEN '40만원대'  
            WHEN (600000 - prod_price) > 0 THEN '50만원대'  
            WHEN (700000 - prod_price) > 0 THEN '60만원대'  
            WHEN (800000 - prod_price) > 0 THEN '70만원대'  
            WHEN (900000 - prod_price) > 0 THEN '80만원대'  
            WHEN (1000000 - prod_price) > 0 THEN '90만원대'  
            ELSE '100만원 이상'  
       END "가격대"  
FROM prod  
WHERE prod_price > 100000
```

함수(Miscellaneous)

- 회원정보테이블의 주민등록 뒷자리(7자리 중 첫째자리)에서
성별 구분을 검색하시오 ?
(**CASE** 구문 사용, **Alias**는 회원명,주민등록번호(주민1-주민2), 성별)

함수(Regular Expression)

● Regular Expression 함수는 10g에서 추가됨

REGEXP_LIKE	패턴을 사용하여 문자열을 검증 (WHERE 절)
REGEXP_INSTR	패턴에 일치하는 위치 반환
REGEXP_SUBSTR	패턴에 일치하는 하위 문자열 반환
REGEXP_REPLACE	패턴에 일치하는 문자열 변환

패턴	의 미	패턴	의 미
^	시작 문자열		또는
\$	종결 문자열	\	제외 (패턴문자)
?	앞의 문자/식 0, 1	[]	문자의 범위 [0-9]
+	앞의 문자/식 1, ∞	{ }	반복 {n}, {n,}, {n,m}
*	앞의 문자/식 0, 1, ∞	()	식 (작은 그룹)
.	어떠한(any) 문자	\n, \t, ..	개행, 탭 등
\d, \D	[0-9], [^0-9]	\w, \W	문자 및 숫자 [0-9a-zA-Z_]

함수(Regular Expression)

REGEXP_LIKE	패턴을 사용하여 str을 검증 (str, pattern [, opt])
[opt] c: 대소문자 구분, i: 대소문자 구분 안함, m: 다중 행 검색	

-회원 중에 성이 '김' 이고, 성 다음에 '성' 또는 '형' 이 있는 회원을 검색

● SELECT mem_id, mem_name
FROM member
WHERE REGEXP_LIKE(mem_name, '^김(성|형)')

-상품 이름 중에 '삼성' 이라는 말이 있고, 숫자 두 개가 같이 있는 상품의
상품코드, 상품명, 판매가를 검색하시오?

함수(Regular Expression)

REGEXP_SUBSTR	<p>패턴에 맞는 문자열을 반환</p> <p>(str, pattern [, pos [, occur [, opt]]])</p> <p>pos : 시작위치, occur : 몇번째</p>
----------------------	--

- SELECT REGEXP_SUBSTR('Java Flex Oracle', '[^]+')
FROM dual
- SELECT REGEXP_SUBSTR('Java Flex Oracle', '[^]+', 1, 3)
FROM dual

_회원 테이블에서 이메일주소를 근거로 이메일아이디, 이메일서버로
구분하여 검색하시오?
(Alias는 회원이름, 이메일, 이메일아이디, 이메일서버)

함수(Regular Expression)

REGEXP_REPLACE

패턴에 맞는 문자열을 다른 문자로 변환

패턴에 맞지 않는 경우 원본을 리턴

(str, pattern [,replace [, pos [, occur [,opt]]]])

replace : 바꿀 문자열

- SELECT REGEXP_REPLACE('Java Flex Oracle', '[^]+', 'C++')
FROM dual

- SELECT
REGEXP_REPLACE('Java Flex Oracle', '[^]+', 'C++', 1, 2)
FROM dual

_회원 테이블에서 주민번호 1 컬럼을 기준으로 XX월 XX일 형식으로
조회하시오. (회원명, 주민번호1, 생일)

함수(Regular Expression)

REGEXP_INSTR	<p>패턴에 맞는 문자열의 위치 반환</p> <p>패턴을 찾지 못한 경우 0 반환</p> <p>(str, pattern [, pos [, occur [,reopt [, opt]]]])</p> <p>[reopt] 0 : 시작위치, 1 : 일치하는 마지막위치</p>
---------------------	--

- SELECT
REGEXP_INSTR('JAVA Flex Oracle','[ae]') RESULT
FROM dual
- SELECT
REGEXP_INSTR('JAVA Flex Oracle','[ae]', 1, 1, 0, 'i') RESULT
FROM dual

- 상품명에 숫자가 들어 있지 않은 것을 출력하시오?
(상품코드, 상품명, 판매가)

한 개의 **TABLE** 조회

- 회원테이블 전체를 조회

```
SELECT * FROM member
```

- 회원테이블에서 회원**ID**,회원명만 선택 조회

- 간단한 **SQL**문이다.

'*'을 사용하여 조회할 수도 있고, 각각의 컬럼을 지정하여 조회 할 수도 있다.

```
SELECT mem_id, mem_name FROM member
```

- 기본 컬럼명으로 조회

-회원테이블의 회원**ID**, 주민등록번호, 성명, 생일 조회

```
SELECT mem_id, mem_regno1, mem_regno2,  
       mem_name, mem_bir  
FROM member  
ORDER BY mem_regno1, mem_regno2
```

한 개의 **TABLE** 조회

- 주민등록번호 편집 및 생일구분 치환
- 타이틀 변경
- 사원테이블에서 사번과 주민등록번호, 성명, 생년월일을 주민등록번호 순으로 조회
- 같은 내용의 조회를 약간의 함수와 더불어 좀더 다른 표현의 조회가 가능하다는 것을 알 수 있다.

-회원테이블의 회원**ID**, 주민등록번호(치환: **A-B**), 성명, 생일 (**Conversion**) 조회

```
SELECT mem_id 회원ID,  
       mem_regno1 || '-' || mem_regno2 "주민등록번호",  
       mem_name AS 성명,  
       TO_CHAR(mem_bir, 'YYYY-MM-DD') 생일  
FROM member  
ORDER BY mem_regno1, mem_regno2
```

한 개의 **TABLE** 조회

- 상품 중에 분류가 피혁 잡화인 상품을 조회
 - 피혁잡화의 분류코드는 '**P301**'
 - 상품분류 테이블은 **lprod**
-
- 조회 시점에서 조건을 주는 경우
 - 첫번째 방법의 조회면 충분하지만, 조회가 가능한 타 방법을 열거
 - **IN** 이나 **EXISTS** 는 해당 컬럼에 여러 조건을 만족시켜야 할 때 주로 사용
 - **EXISTS** 함수는 부 질의를 매개변수로 받아서, 만약 이 부질의가 하나 이상의 행들을 돌려주면 **true**를 돌려준다

```
SELECT prod_id, prod_name, prod_lgu  
FROM prod  
WHERE prod_lgu = 'P301'
```

한 개의 TABLE 조회

- **SELECT prod_id, prod_name, prod_lgu
FROM prod
WHERE prod_lgu IN ('P301')**
- **SELECT prod_id, prod_name, prod_lgu
FROM prod
WHERE prod_lgu IN (SELECT lprod_gu
FROM lprod
WHERE lprod_nm = '피혁잡화')**
- **SELECT prod_id, prod_name, prod_lgu
FROM prod
WHERE EXISTS (SELECT lprod_gu
FROM lprod
WHERE lprod_gu = prod.prod_lgu
AND lprod_gu = 'P301')**

한 개의 **TABLE** 조회

- '대흥동'의 우편번호를 조회
- 우편번호 테이블은 **ziptb**
- **LIKE**문은 중간글자나 한두 글자만으로 해당 내용을 찾는 조건으로 사용

```
SELECT zipcode 우편번호,  
          sido || ' ' || gugun || ' ' dong 주소,  
          bunji 번지  
FROM ziptb  
WHERE dong LIKE '%대흥동%'
```

- 본인의 주거지 우편번호 확인

한 개의 **TABLE** 조회

- 상품마스터에서 판매가가 **200,000원**에서 **400,000원** 사이에 있는 상품을 판매가가 높은 순서로 조회

```
SELECT prod_id, prod_name, prod_sale  
FROM prod  
WHERE prod_sale >= 200000  
      AND prod_sale <= 400000  
ORDER BY prod_sale DESC
```

- 어떤 범위의 조건시에 보통 부등호를 자주 사용한다.
여러 경우 중 범위사이의 조건이라면, **BETWEEN ~ AND**를 사용함으로써 **SQL**문이 간결해진다.

```
SELECT prod_id, prod_name, prod_sale  
FROM prod  
WHERE prod_sale BETWEEN 200000 AND 400000  
ORDER BY prod_sale DESC
```


TABLE JOIN

- RDB의 핵심
- 관계형DB의 가장 큰 장점은 많은 TABLE을 JOIN하여 원하는 결과를 도출하는데 있다

Join 종류	설명
Cartesian Product	모든 가능한 행들의 조합
Equi Join	조건이 일치하는 컬럼을 매칭(주로 PK 와 FK) (Simple Join)
Non-Equi Join	조건이 일치하는 컬럼이 없지만 다른 조건을 사용하여 Join 한다.
Outer Join	조건이 일치하지 않더라도 모든 행들을 검색하고자 할 때 사용, (+) 로 표시한다.
Self Join	한 테이블 내에서 Join 하는 경우

TABLE JOIN

● ANSI JOIN

- SQL/86 표준안 Oracle 9i 이전 버전
- SQL/92 표준안 Oracle 9i 부터 지원
- From 절에 Join 문을 사용하며 On 다음에 조건을 기술한다.
- 표준이므로 타 DBMS에서도 적용된다.

Join 종류	설명
Cross Join	Cartesian Product 와 동등
Natural Join	각 테이블에 동일한 이름의 컬럼이 존재 할 때 자동으로 조건이 적용된다.
Inner Join	Equi Join 과 동등
Outer Join	(Left/ Right/ Full Outer Join)

TABLE JOIN

Cartesian Product / Cross Join

- 다수개의 **Table**로부터 조합된 결과가 발생한다. ($n * m$)
- 모든 행, 모든 컬럼이 조합된다.
- 특별한 목적 이외에는 사용되는 거의 없다.
- 조인조건을 잘못 주었을 때도 **Cartesian Product** 가 발생한다.

● **SELECT * FROM lprod, prod**

● **SELECT COUNT(*)
FROM lprod, prod, buyer**

● **ANSI 형식**

- **SELECT * FROM lprod CROSS JOIN prod**

- **SELECT COUNT(*)
FROM lprod CROSS JOIN prod CROSS JOIN buyer**

TABLE JOIN

Equi - Join (Simple - Join)

- 다수개의 **Table**로부터 정보를 검색하려면, **Select**문장의 **From**절에 **Table**명들을 적고 **Where**절에 각 **Table**의 **Row**들을 연결시킬 조건식을 기술한다.
- 각 **Table**의 **Column**명이 중복될 때는 반드시 **Column**명 앞에 **Table**명을 붙여야 한다.
- 중복되지 않을 때에는 붙이지 않아도 되지만 명확성을 위해서나 **Access**를 위해서 붙이는 것이 좋다.
- **N**개의 **Table**을 **Join**할 때는 최소한 **n-1**개의 조건식이 필요하다.
- **ANSI** 표준에서는 **Inner Join** 을 사용할 것을 권고한다.

```
SELECT table명.column명, table명.column명 .....  
FROM table1명, table2명  
WHERE table1명.column명 = table2명.column명
```

TABLE JOIN

- 상품테이블에서 상품코드, 상품명, 분류명을 조회.

상품테이블 : **prod**

분류테이블 : **lprod**

```
SELECT prod.prod_id    "상품코드",  
       prod.prod_name  "상품명",  
       lprod.lprod_nm  "분류명"  
FROM prod, lprod  
WHERE prod.prod_lgu = lprod.lprod_gu
```

- **ANSI 형식**

```
- SELECT prod.prod_id    "상품코드",  
       prod.prod_name  "상품명",  
       lprod.lprod_nm  "분류명"  
FROM prod  
      INNER JOIN lprod ON ( prod.prod_lgu = lprod.lprod_gu )
```

TABLE JOIN

TABLE Alias

- 긴 Table명 대신 Alias를 지정하여 사용
SELECT alias명.column명, alias명.column명
FROM table1명 alias1명, table2명 alias2명
WHERE alias1명.column명 = alias2명.column명
- 상품테이블에서 상품코드, 상품명, 분류명, 거래처 명을 조회.
1) 테이블 명을 그대로 사용
SELECT prod.prod_id "상품코드",
prod.prod_name "상품명",
lprod.lprod_nm "분류명",
buyer.buyer_name "거래처명"
FROM prod, lprod, buyer
WHERE prod.prod_lgu = lprod.lprod_gu
AND prod.prod_buyer = buyer.buyer_id

TABLE JOIN

2) Alias를 사용한 방법

```
SELECT A.prod_id    "상품코드",  
       A.prod_name  "상품명",  
       B.lprod_nm   "분류명",  
       C.buyer_name "거래처명"  
FROM prod A, lprod B, buyer C  
WHERE A.prod_lgu = B.lprod_lgu  
      AND A.prod_buyer = C.buyer_id
```

● ANSI 형식

```
- SELECT A.prod_id    "상품코드",  
       A.prod_name  "상품명",  
       B.lprod_nm   "분류명",  
       C.buyer_name "거래처명"  
FROM prod A  
     INNER JOIN lprod B ON (A.prod_lgu = B.lprod_lgu )  
     INNER JOIN buyer C ON ( A.prod_buyer = C.buyer_id )
```

TABLE JOIN

특정 Row의 Join

- Join문장을 기술할 때 Join조건식 이외에 다른 조건식을 And로 연결

```
SELECT table명.column명, table명.column명 .....  
FROM table1명, table2명  
WHERE table1명.column명 = table2명.column명  
AND Condition1 문  
AND Condition2 문
```

- 상품테이블에서 거래처가 '삼성전자' 인 자료의 상품코드, 상품명, 거래처 명을 조회

```
SELECT prod_id    상품코드,  
       prod_name  상품명,  
       buyer_name 거래처명  
FROM prod, buyer  
WHERE prod_buyer = buyer_id  
AND buyer_name LIKE '삼성전자%'
```


TABLE JOIN

- **Inner Join**에서는 **Join**조건식은 **ON** 다음에 기술하고 그 이외의 조건은 **WHERE** 절에 기술한다.
- 상품테이블에서 거래처가 '삼성전자' 인 자료의 상품코드, 상품명, 거래처 명을 조회

```
SELECT prod_id    상품코드,  
        prod_name  상품명,  
        buyer_name 거래처명  
FROM prod  
        INNER JOIN buyer ON ( prod_buyer = buyer_id )  
WHERE buyer_name LIKE '삼성전자%'
```

TABLE JOIN

- 상품테이블에서 상품코드, 상품명, 분류명, 거래처명, 거래처주소를 조회.

- 1) 판매가격이 10만원 이하 이고
- 2) 거래처 주소가 부산인 경우만 조회

```
SELECT A.prod_id    "상품코드",  
       A.prod_name  "상품명",  
       B.lprod_nm   "분류명",  
       C.buyer_name "거래처명",  
       C.buyer_add1 "주소"  
FROM prod A, lprod B, buyer C  
WHERE A.prod_sale <= 100000  
      AND A.prod_lgu = B.lprod_gu  
      AND A.prod_buyer = C.buyer_id  
      AND C.buyer_add1 LIKE '%부산%'
```

TABLE JOIN

- 상품 분류가 전자제품(P102)인 상품의 상품코드, 상품명, 분류명, 거래처 명을 조회.

```
SELECT A.prod_id    "상품코드",  
       A.prod_name  "상품명",  
       B.lprod_nm   "분류명",  
       C.buyer_name "거래처명"  
FROM prod A, lprod B, buyer C  
WHERE A.prod_lgu = 'P102'  
      AND A.prod_lgu = B.lprod_gu  
      AND A.prod_buyer = C.buyer_id
```

TABLE JOIN

- 상품입고테이블의 **2005년도 1월**의 거래처별 매입금액을 검색 하시오 ? (매입금액 = 매입수량 * 매입단가)
(**Alias**는 거래처코드, 거래처명, 매입금액)

```
SELECT buyer_id 거래처코드,  
       buyer_name 거래처명,  
       SUM(buy_qty * buy_cost) 매입금액  
FROM buyprod, prod, buyer  
WHERE buy_date BETWEEN '2005-01-01' AND '2005-01-31'  
      AND buy_prod = prod_id  
      AND prod_buyer = buyer_id  
GROUP BY buyer_id, buyer_name  
ORDER BY buyer_id, buyer_name
```

TABLE JOIN

- Inner Join으로 작성

```
SELECT buyer_id 거래처코드,  
       buyer_name 거래처명,  
       SUM(buy_qty * buy_cost) 매출금액  
FROM buyprod  
     INNER JOIN prod ON (buy_prod = prod_id)  
     INNER JOIN buyer ON (prod_buyer = buyer_id)  
WHERE buy_date BETWEEN '2005-01-01' AND '2005-01-31'  
GROUP BY buyer_id, buyer_name  
ORDER BY buyer_id, buyer_name
```

TABLE JOIN

- 장바구니테이블의 **2005년도 5월**의 회원별 구매금액을 검색 하시오 ? (구매금액 = 구매수량 * 판매가)
(**Alias**는 회원**ID**, 회원명, 구매금액)
(**Equi Join** 방식과 **Inner Join** 방식 중 선택)

OUTER JOIN

- 두 **Table**을 **Join**할 때 **Join**조건식을 만족시키지 못하는 **Row**는 검색에서 빠지게 되는데, 이러한 누락된 **Row**들이 검색되도록 하는 방법
- 조인에서 부족한 쪽에 **"(+)"** 연산자 기호를 사용한다.
- **(+)** 연산자는 **NULL** 행을 생성하여 조인하게 한다.
- **주의 :-** **NULL**만큼이나 실수하기 쉽다.
 - 안전하다고 너무 많이 사용할 경우 처리속도가 저하
- 조건식을 만족시키지 못하는 데이터도 검색
SELECT table명.column명, table명.column명
FROM table1명, table2명
WHERE table1명.column명 = table2명.column명(+)

OUTER JOIN

- 전체 분류의 상품자료 수 를 검색 조회
(**Alias**는 분류코드, 분류명, 상품자료수)

-1. 분류테이블 조회

```
SELECT * FROM lprod
```

-2. 일반 JOIN

```
SELECT lprod_gu 분류코드,  
       lprod_nm 분류명,  
       COUNT(prod_lgu) 상품자료수  
FROM lprod, prod  
WHERE lprod_gu = prod_lgu  
GROUP BY lprod_gu, lprod_nm
```

-3. OUTER JOIN 사용 확인

```
SELECT lprod_gu 분류코드, lprod_nm 분류명,  
       COUNT(prod_lgu) 상품자료수  
FROM lprod, prod  
WHERE lprod_gu = prod_lgu(+)  
GROUP BY lprod_gu, lprod_nm  
ORDER BY lprod_gu
```


OUTER JOIN

● ANSI 형식

- 모든 행이 검색되어야 할 테이블의 위치를 기준으로 한다.
- 위치에 따라서 **LEFT** , **RIGHT**, **FULL** 로 나눈다.
- **"(+)"** 보다 명확하게 결과가 나온다.
- **"(+)"** 가 지원하지 못하는 **FULL OUTER JOIN** 이 된다.

-4. ANSI OUTER JOIN 사용 확인

```
SELECT lprod_gu 분류코드, lprod_nm 분류명,  
       COUNT(prod_lgu) 상품자투수  
FROM lprod  
     LEFT OUTER JOIN prod ON ( lprod_gu = prod_lgu )  
GROUP BY lprod_gu, lprod_nm  
ORDER BY lprod_gu
```

OUTER JOIN

- 전체상품의 2005년 1월 입고수량을 검색 조회
(**Alias**는 상품코드, 상품명, 입고수량)

-1. 일반 JOIN

```
SELECT prod_id 상품코드,  
       prod_name 상품명,  
       SUM(buy_qty) 입고수량  
FROM prod, buyprod  
WHERE prod_id = buy_prod  
      AND buy_date BETWEEN '2005-01-01' AND '2005-01-31'  
GROUP BY prod_id, prod_name
```

-2. OUTER JOIN 사용 확인

```
SELECT prod_id 상품코드,  
       prod_name 상품명,  
       SUM(buy_qty) 입고수량  
FROM prod, buyprod  
WHERE prod_id = buy_prod(+)  
      AND buy_date BETWEEN '2005-01-01' AND '2005-01-31'  
GROUP BY prod_id, prod_name  
ORDER BY prod_id, prod_name
```

OUTER JOIN

- 2번 쿼리는 전체상품의 결과가 조회되지 않았다.
 - 서브쿼리를 사용하거나 **ANSI** 조인을 사용하여 해결 해야 한다.

-3. ANSI OUTER JOIN

```
SELECT prod_id 상품코드,  
       prod_name 상품명,  
       SUM(buy_qty) 입고수량  
FROM prod LEFT OUTER JOIN buyprod  
       ON ( prod_id = buy_prod  
           AND buy_date BETWEEN '2005-01-01' AND '2005-01-31' )  
GROUP BY prod_id, prod_name  
ORDER BY prod_id, prod_name
```

OUTER JOIN

- 전체상품의 2005년 1월 입고수량을 검색 조회
(Alias는 상품코드, 상품명, 입고수량)

-4. OUTER JOIN 사용 확인(NULL값 제거)

```
SELECT prod_id 상품코드,  
       prod_name 상품명,  
       SUM( NVL(buy_qty, 0) ) 입고수량  
FROM prod LEFT OUTER JOIN buyprod  
      ON ( prod_id = buy_prod  
          AND buy_date BETWEEN '2005-01-01' AND '2005-01-31' )  
GROUP BY prod_id, prod_name  
ORDER BY prod_id, prod_name
```

OUTER JOIN

- 전체 회원의 **2005년도 4월**의 구매현황 조회
(회원ID, 성명, 구매수량의 합)

- 일반 JOIN

```
SELECT mem_id    "회원ID",  
       mem_name  "성명",  
       SUM(cart_qty) "구매수량"  
FROM member, cart  
WHERE mem_id = cart_member  
      AND SUBSTR(cart_no,1, 6) = '200504'  
GROUP BY mem_id, mem_name  
ORDER BY mem_id, mem_name
```

- **OUTER JOIN** 사용 확인(NULL값 제거) 검색하시오 ?

OUTER JOIN

- 전체 상품의 2005년도 5월 5일의 입고.출고현황 조회
(상품코드, 상품명, 입고수량의 합, 판매수량의 합)

- 입고 확인(Equi Join)

```
SELECT prod_id 상품코드, prod_name 상품, SUM(buy_qty) 입고수량  
FROM prod, buyprod  
WHERE prod_id = buy_prod  
      AND buy_date = '20050505'  
GROUP BY prod_id, prod_name
```

- 판매 확인(Inner Join)

```
SELECT prod_id 상품코드, prod_name 상품, SUM(cart_qty) 판매수량  
FROM prod INNER JOIN cart  
      ON ( prod_id = cart_prod )  
WHERE cart_no LIKE '200500505%'  
GROUP BY prod_id, prod_name
```

OUTER JOIN

- 전체 상품의 2005년도 5월 5일의 입고.출고현황 조회
(상품코드, 상품명, 입고수량의 합, 판매수량의 합)

- 일반 JOIN

```
SELECT prod_id 상품코드, prod_name 상품,  
       SUM(buy_qty) 입고수량, SUM(cart_qty) 판매수량  
FROM prod, buyprod, cart  
WHERE prod_id = buy_prod  
      AND prod_id = cart_prod  
      AND buy_date = '200500505'  
      AND SUBSTR(cart_no,1,8) = '20050505'  
GROUP BY prod_id, prod_name
```

-위 Equi Join은 결과가 나오지 않는다.

OUTER JOIN 사용 확인(NULL값 제거) 검색하시오 ?

SELF JOIN

- **Table Alias**를 사용하여 마치 **2개의 Table**처럼 자신의 **Table**과 자신의 **Table**을 **Join**하여 검색

- 회원ID가 'h001(라준호)'인 고객의 마일리지 점수보다
이상인 회원만 검색 조회

(Alias는 회원ID, 성명, 마일리지)

```
SELECT B.mem_id "회원ID",  
       B.mem_name "성명",  
       B.mem_mileage "마일리지"  
FROM member A, member B  
WHERE A.mem_id = 'h001'  
      AND A.mem_mileage <= B.mem_mileage
```


SELF JOIN

- 거래처코드가 'P30203(참존)' 과 동일지역에 속한 거래처만
검색 조회

(Alias는 거래처코드, 거래처, 주소)

```
SELECT B.buyer_id 거래처코드,  
       B.buyer_name 거래처명,  
       B.buyer_add1 || ' ' || B.buyer_add2 주소  
FROM buyer A, buyer B  
WHERE A.buyer_id = 'P30203'  
      AND SUBSTR(A.buyer_add1, 1, 2) = SUBSTR(B.buyer_add1, 1, 2)
```

JOIN GROUP BY

- 주로 통계, 분석자료를 검색하기 위해 자주 활용
- 회원ID가 'r001'인 고객의 2005년도 월별 구매현황 조회
(회원ID, 성명, 주민등록번호, 구매월, 구매총액)

```
SELECT mem_id      "회원ID",  
       mem_name    "성명",  
       mem_regno1 || '-' || mem_regno2 "주민등록번호",  
       SUBSTR(cart_no, 1, 6) "구매월",  
       SUM(cart_qty * prod_sale) "구매총액"  
FROM member, cart, prod  
WHERE mem_id = 'r001'  
      AND mem_id = cart_member  
      AND cart_no LIKE '2005%'  
      AND cart_prod = prod_id  
GROUP BY mem_id, mem_name,  
         mem_regno1 || '-' || mem_regno2, SUBSTR(cart_no, 1, 6)
```

JOIN GROUP BY

- 거래처의 **2005년도 1월**의 거래처별 일자별 매입현황 조회
(거래처명, 매입일자, 매입금액)

```
SELECT buyer_name "거래처명",  
       buy_date    "매입일자",  
       SUM(buy_qty * buy_cost) "매입금액"  
FROM buyprod, buyer, prod  
WHERE buy_date BETWEEN '2005-01-01' AND '2005-01-31'  
      AND buy_prod = prod_id  
      AND prod_buyer = buyer_id  
GROUP BY buyer_name, buy_date
```

JOIN GROUP BY

- 2005년도 상품의 판매 총합계 현황을 조회
(상품코드, 상품명, 판매수량, 판매금액)

1. 문자열 비교 검색

```
SELECT prod_id 상품코드, prod_name 상품명,  
       SUM(cart_qty) 판매수량,  
       SUM(cart_qty * prod_sale) 판매금액  
FROM prod, cart  
WHERE prod_id = cart_prod  
      AND SUBSTR(cart_no, 1, 4) = '2005'  
GROUP BY prod_id, prod_name
```

2. LIKE문 비교 검색

```
SELECT prod_id 상품코드, prod_name 상품명,  
       SUM(cart_qty) 판매수량,  
       SUM(cart_qty * prod_sale) 판매금액  
FROM prod, cart  
WHERE prod_id = cart_prod  
      AND cart_no LIKE '2005%'  
GROUP BY prod_id, prod_name
```

JOIN GROUP BY

- 2005년도 월별 매입 현황을 검색하시오 ?

(Alias는 매입월, 매입수량, 매입금액(매입수량*상품테이블의 매입가))

```
SELECT TO_CHAR(buy_date,'MM') 매입월,  
       SUM(buy_qty) 매입수량,  
       TO_CHAR(SUM(buy_qty * prod_cost),'L999,999,999') 매입금액  
FROM buyprod, prod  
WHERE buy_prod = prod_id  
      AND EXTRACT(YEAR FROM buy_date) = 2005  
GROUP BY TO_CHAR(buy_date,'MM')  
ORDER BY 매입월 ASC
```

- 2005년도 월별 판매 현황을 검색하시오 ?

(Alias는 판매월, 판매수량, 판매금액(판매수량*상품테이블의 판매가))

JOIN GROUP BY

❖ HAVING절 정리

- **GROUP BY**에서만 사용 가능하고, **COUNT(*)**, **SUM()**등의 **GROUP**함수에 조건을 줄 수 있다.

- **Having절** 기술 시 처리 순서

1. Row들이 **Group**되어 진다
2. **Group**에 대해 **Group Function**이 적용된다
3. **Having절**을 만족하는 **Group**을 선택한다

SELECT column1명, column2명..., group함수(column명)
FROM table1명...
GROUP BY column1명, column2명...
HAVING 그룹조건식

- 2005년도 판매된 상품 중에 5회 이상의 판매회수가 있는 상품 조회 (상품코드, 상품명, 판매횟수)

- **GROUP BY**와 **HAVING**을 이용하여 해당 조회

```
SELECT prod_id 상품코드 , prod_name 상품명,  
       COUNT(*) 판매횟수
```

```
FROM prod, cart
```

```
WHERE prod_id = cart_prod
```

```
AND SUBSTR(cart_no, 1, 4) = '2005'
```

```
GROUP BY prod_id, prod_name
```

```
HAVING COUNT(*) >= 5
```

```
SELECT A.prod_id 상품코드 , A.prod_name 상품명,  
       PAN_COUNT 판매횟수
```

```
FROM ( SELECT prod_id, prod_name,  
             COUNT(*) PAN_COUNT
```

```
FROM prod, cart
```

```
WHERE prod_id = cart_prod
```

```
AND SUBSTR(cart_no, 1, 4) = '2005'
```

```
GROUP BY prod_id, prod_name ) A
```

```
WHERE A.PAN_COUNT >= 5
```

❖ **HAVING**절 정리

- 상품분류가 컴퓨터제품('P101')인 상품의 **2005**년도 일자별 판매 조회
(판매일, 판매금액(**5,000,000**초과의 경우만), 판매수량)

- **HAVING**을 이용하여 해당 조회

```
SELECT SUBSTR(cart_no,1,8)      "판매일",  
       SUM(cart_qty * prod_sale) "판매금액",  
       SUM(cart_qty)   "판매수량"  
FROM prod, cart  
WHERE cart_no LIKE '2005%'  
      AND prod_id = cart_prod  
      AND prod_lgu = 'P101'  
GROUP BY SUBSTR(cart_no,1,8)  
HAVING SUM(cart_qty * prod_sale) > 5000000  
ORDER BY SUBSTR(cart_no,1,8)
```


❖ HAVING절 정리

- 2005년도 판매일자, 판매총액 (5,000,000초과의 경우만), 판매수량(50초과 의 경우만), 판매상품종류가 8개이상인 판매일자만 조회

- HAVING을 이용하여 해당 조회

```
SELECT SUBSTR(cart_no,1,8) "판매일",  
       SUM(cart_qty * prod_sale) "판매금액",  
       SUM(cart_qty) "판매수량",  
       COUNT(*) "판매횟수"  
FROM prod INNER JOIN cart  
       ON ( prod_id = cart_prod )  
WHERE cart_no LIKE '2005%'  
GROUP BY SUBSTR(cart_no,1,8)  
HAVING SUM(cart_qty * prod_sale) > 5000000  
       AND SUM(cart_qty) > 50  
       AND COUNT(*) >= 8  
ORDER BY SUBSTR(cart_no,1,8)
```

서브쿼리(Subquery)

- **SQL** 구문 안에 또 다른 **Select** 구문이 있는 것을 말한다.
- **Subquery** 가 없다면 **SQL** 구문은 너무 많은 **Join**을 해야 하거나 구문이 복잡해진다.
- **Subquery** 는 괄호로 묶는다
- 연산자와 사용할 경우 오른쪽에 배치한다.
- **FROM** 절에 사용하는 경우 **View**와 같이 독립된 테이블처럼 활용되어 **inline view** 라고 부른다.
- **Main query** 와 **Sub query** 사이의 참조성 여부에 따라 연관(**Correlated**) 또는 비연관 (**Noncorrelated**) 서브쿼리로 구분
- 반환하는 행의 수, 컬럼수에 따라 단일행/다중행, 단일컬럼/다중컬럼으로 구분하며 대체적으로 연산자의 특성을 이해하면 쉽다.

서브쿼리(Subquery)

- 상품코드, 상품명, 분류명을 조회
 - 실제 상기 내용의 경우는 **Subquery**가 필요치는 않다.

(Inner Join)

```
SELECT p.prod_id, p.prod_name, l.lprod_nm
FROM prod p INNER JOIN lprod l
      ON ( p.prod_lgu = l.lprod_gu )
```

(Subquery)

```
SELECT prod_id, prod_name,
      (SELECT lprod_nm FROM lprod WHERE prod_lgu = lprod_gu)
FROM prod
```

서브쿼리(Subquery)

- 상품**Table**에서 판매가가 상품평균판매가 보다 큰 상품을 검색하시오 ? (**Alias**는 상품명,판매가,평균판매가)

```
SELECT A.prod_name 상품명,  
       TO_CHAR(A.prod_sale,'999,999,999') 판매가,  
       TO_CHAR(B.AVG_sale,'999,999,999') 평균판매가  
FROM prod A, (SELECT AVG(prod_sale) AVG_sale  
              FROM prod) B  
WHERE A.prod_sale > B.AVG_sale
```

서브쿼리(Subquery)

- 회원 **Table**에서 마일리지가 평균마일리지 보다 큰 회원을 검색하시오 ? (**Alias**는 회원명,마일리지,평균마일리지)

```
SELECT M.mem_name 회원명, M.mem_mileage 마일리지  
      A.AVG_mileage 평균마일리지  
FROM member M,  
      (SELECT AVG(mem_mileage) AVG_mileage  
      FROM member ) A  
WHERE M.mem_mileage > A.AVG_mileage
```

- 위 평균마일리지는 **2**명의 정보가 빠져있다. 수정하시오.
(마일리지는 천단위 구분, 평균마일리지는 소숫점이하 **2**자리로)

서브쿼리(Subquery)

- 장바구니**Table**에서 회원별 최고의 구매수량을 가진 자료의 회원, 주문번호, 상품, 수량에 대해 모두 검색하시오 ?
(**Alias**는 회원, 주문번호, 상품, 수량)

(결과)

회원	주문번호	상품	수량
a001	2005040100001	P103000018	16
b001	2005042800002	P103000016	15
b001	2005072800004	P201000016	15
c001	2005050100002	P103000015	7
c001	2005061200001	P106000001	7
.....			
v001	2005052800001	P103000009	3
w001	2005040100003	P103000020	21
x001	2005052900002	P104000021	8

(30 rows selected)

```
SELECT cart_member 회원, cart_no 주문번호, cart_prod 상품,
       cart_qty 수량 FROM cart
WHERE cart_qty = ( SELECT MAX(cart_qty) FROM cart c
                   WHERE cart.cart_member = c.cart_member)
ORDER BY cart_member ASC
```

서브쿼리(Subquery)

- 모든 거래처의 **2005**년도 거래처별 매입금액 합계를 조회

```
SELECT A.buyer_id 거래처코드, A.buyer_name 거래처명,  
       NVL(B.IN_AMT, 0) 매입금액합계  
FROM ( SELECT DISTINCT buyer_id, buyer_name  
        FROM buyer ) A,  
     ( SELECT buyer_id,  
          SUM( buy_cost * buy_qty) IN_AMT  
        FROM buyprod, buyer, prod  
        WHERE buy_date Between '2005-01-01' AND '2005-12-31'  
          AND buy_prod = prod_id  
          AND buyer_id = prod_buyer  
        GROUP BY buyer_id ) B  
WHERE A.buyer_id = B.buyer_id(+)  
ORDER BY A.buyer_name
```

- 1) From절의 **A**는 거래처테이블의 자료중 거래처코드와 거래처명
- 2) **B**는 거래처별 매입 총액
- 3) 거래처코드로 **JOIN** 해당거래처의 매입 총액 **READ**
- 4) 복잡해 보이거나, 실상 내용을 잘게 나눈 후 다시 **JOIN**하여 원하는 자료 도출

서브쿼리(Subquery)

- 모든 거래처의 **2005**년도 거래처별 매출금액합계를 검색하시오 ?
(**Alias**는 거래처코드, 거래처명, 매출금액합계 거래처명 순)
(**cart** 테이블 이용, 매출금액은 **prod_sale * cart_qty**)

서브쿼리(Subquery)

- 모든 거래처의 **2005**년도 거래처별 매입금액합계, 매출금액합계를 검색하시오 ?
(**Alias**는 거래처코드, 거래처명, 매입금액합계, 매출금액합계 거래처명 순으로 검색)
Subquery와 **OUTER JOIN**을 이용하여 거래처 모두 조회

서브쿼리(Subquery)

- Subquery의 결과가 1개의 Row로 나오는 것을 말한다.
- 사용가능 연산자 : =, <>, >, >=, <, <=

```
SELECT column명, column1명...  
FROM table명  
WHERE column명 연산자 ( SELECT column2명...  
                        FROM table명  
                        WHERE 조건식 )
```

- 상품Table에서 판매가가 상품평균판매가 보다 큰 상품을 검색하시오 ? (Alias는 상품명,판매가)

```
SELECT prod_name 상품명,  
       prod_sale 판매가  
FROM prod  
WHERE prod_sale > (SELECT AVG(prod_sale) FROM prod)
```

서브쿼리(Subquery)

- Subquery의 결과가 여러 Row로 나오는 것을 말한다.
- 사용가능 연산자 : IN, ANY, ALL, EXISTS

```
SELECT column명, column1명...  
FROM table명  
WHERE column명 IN ( SELECT column2명...  
                     FROM table명  
                     WHERE 조건식 )
```

- 회원Table에서 회원주소 지역이 거래처주소 지역중 하나이면 선택
검색하시오 ? (Alias는 회원명,지역)
SELECT mem_name 회원명,
 mem_add1 지역
FROM member
WHERE SUBSTR(mem_add1, 1, 2)
 IN (SELECT DISTINCT SUBSTR(buyer_add1, 1, 2)
 FROM buyer)

서브쿼리(Subquery)

- ANY, ALL 은 비교 연산자와 조합된다.(=, <>, >, >=, <, <=)
 - ANY는 OR의 개념, 어떤 것이라도 맞으면 TRUE
 - ALL은 AND의 개념, 모두 만족해야만 TRUE
 - 비교 연산자 다음에 ANY 또는 ALL을 기술했고 서브쿼리 사용
-
- 직업이 '공무원'인 사람들의 마일리지를 검색하여 최소한 그들 중 어느 한사람보다는 마일리지가 큰 사람들을 출력하시오.
단, 직업이 '공무원'인 사람은 제외하고 검색하시오 ?
(Alias는 회원명, 직업, 마일리지)

```
SELECT mem_name    회원명,   mem_job 직업,  
       mem_mileage 마일리지  
FROM member  
WHERE mem_job <> '공무원'  
      AND mem_mileage > ANY (SELECT mem_mileage  
                              FROM member  
                              WHERE mem_job = '공무원')
```

서브쿼리(Subquery)

- 직업이 '공무원'인 사람들의 마일리지를 검색하여 최소한 그들 보다는 마일리지가 큰 사람들을 출력하시오.
단, 직업이 '공무원'인 사람은 제외하고 검색하시오 ?
(Alias는 회원명, 직업, 마일리지)

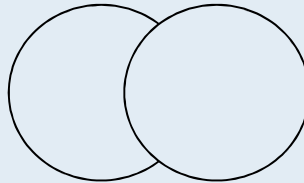
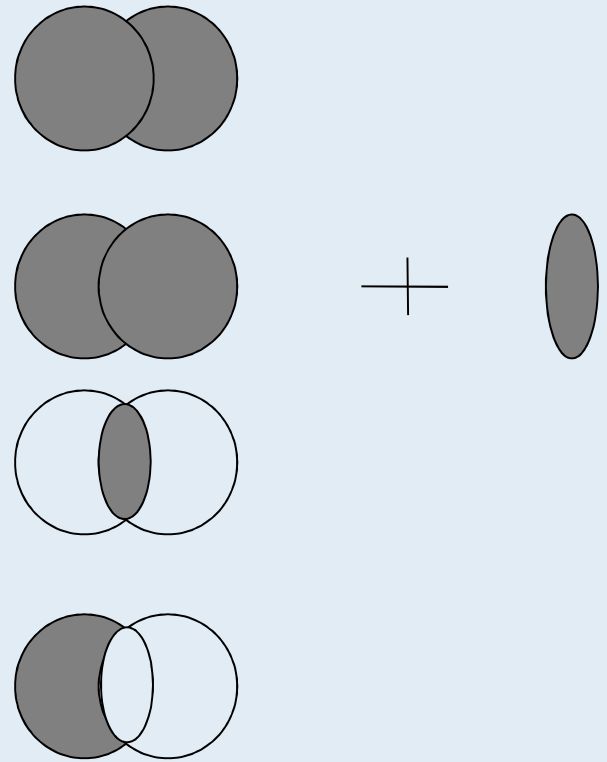
```
SELECT mem_name    회원명,   mem_job 직업,  
       mem_mileage 마일리지  
FROM member  
WHERE mem_job <> '공무원'  
      AND mem_mileage > ALL (SELECT mem_mileage  
                             FROM member  
                             WHERE mem_job = '공무원')
```

SET (집합)

- 여러 테이블의 내용을 한 테이블처럼 조회하는 기능

UNION / UNION ALL / INTERSECT / MINUS

중요 : Select문의 컬럼들의 DATA TYPE과 컬럼수는 동일

<div data-bbox="313 558 481 614">A B</div> 	<div data-bbox="750 614 1008 654">A Union B</div> <div data-bbox="772 678 1108 774">(모든 자료조회지만 중복자료는 한번만)</div> <div data-bbox="750 837 1086 877">A Union All B</div> <div data-bbox="784 893 1064 933">(모든 자료 조회)</div> <div data-bbox="750 997 1075 1037">A Intersect B</div> <div data-bbox="750 1053 1108 1093">(동일한 자료만 조회)</div> <div data-bbox="750 1220 1019 1260">A Minus B</div> <div data-bbox="772 1284 1097 1380">(동일자료를 제외한 자료 조회)</div>	
<div data-bbox="257 893 571 1189"> 1) 일반 2) UNION 3) UNION ALL 4) INTERCEPT 5) MINUS </div>		

SET (집합)

1. 일반적인 조회

```
SELECT mem_name, mem_job, mem_mileage  
FROM member  
WHERE mem_mileage > 4000
```

```
SELECT mem_name, mem_job, mem_mileage  
FROM member  
WHERE mem_job = '자영업'
```

2. UNION

```
SELECT mem_name, mem_job, mem_mileage  
FROM member  
WHERE mem_mileage > 4000  
UNION  
SELECT mem_name, mem_job, mem_mileage  
FROM member  
WHERE mem_job = '자영업'
```

SET (집합)

3. UNION ALL

```
SELECT mem_name, mem_job, mem_mileage
  FROM member
 WHERE mem_mileage > 4000
UNION ALL
SELECT mem_name, mem_job, mem_mileage
  FROM member
 WHERE mem_job = '자영업'
ORDER BY mem_name ASC
```

- 처음 **SELECT** 구문에 적용된 컬럼의 타입, 개수, 이름이 기준이 된다.
- **CLOB, BLOB, BFILE, VARRAY** 타입은 사용불가
- **ORDER BY**는 마지막에 기술한다.

SET (집합)

4. INTERSECT

```
SELECT mem_name, mem_job, mem_mileage
  FROM member
 WHERE mem_mileage > 4000
INTERSECT
SELECT mem_name, mem_job, mem_mileage
  FROM member
 WHERE mem_job = '자영업'
ORDER BY 1 ASC
```

SET (집합)

5. MINUS

```
SELECT mem_name, mem_job, mem_mileage
FROM member
WHERE mem_mileage > 4000
MINUS
SELECT mem_name, mem_job, mem_mileage
FROM member
WHERE mem_job = '자영업'
ORDER BY mem_name ASC
```

SET (집합)

- 상품분류테이블, 상품테이블, 회원테이블의 자료수를 조회
 - **UNION** 을 이용하여 조회

```
SELECT 'lprod' TABLE_ID, '상품분류테이블' TABLE_NAME,  
       COUNT(*) DATA_COUNT FROM lprod
```

```
UNION
```

```
SELECT 'prod' TABLE_ID, '상품테이블' TABLE_NAME,  
       COUNT(*) DATA_COUNT FROM prod
```

```
UNION
```

```
SELECT 'member' TABLE_ID, '회원테이블' TABLE_NAME,  
       COUNT(*) DATA_COUNT FROM member
```

EXISTS

- **EXISTS** 연산자를 통하여 **SET** 연산자의 **INTERSECT, MINUS** 구현함

● 장바구니 테이블에서

A) 2005년도 4월에 판매된 상품과

```
SELECT DISTINCT c.cart_prod 판매상품, p.prod_name 상품명  
FROM cart c, prod p  
WHERE c.cart_prod = p.prod_id  
AND SUBSTR(c.cart_no,1,8) BETWEEN '20050401' AND '20050430'
```

B) 2005년도 6월에 판매된 상품 중,

```
SELECT DISTINCT c.cart_prod 판매상품, p.prod_name 상품명  
FROM cart c, prod p  
WHERE c.cart_prod = p.prod_id  
AND c.cart_no LIKE '200506%'
```

EXISTS

1) A),B)모두에 해당되는 상품 (**EXIST**를 이용 **INTERSECT**구현)

```
SELECT DISTINCT c.cart_prod 판매상품, p.prod_name 상품명
FROM cart c, prod p
WHERE c.cart_prod = p.prod_id
      AND SUBSTR(c.cart_no,1,8) BETWEEN '20050401' AND '20050430'
      AND EXISTS
        ( SELECT DISTINCT cart_prod
          FROM cart
          WHERE cart.cart_prod = c.cart_prod
            AND cart.cart_no LIKE '200506%' )
```

2) A)에는 있고 B)에는 없는 상품 (**EXIST**를 이용 **MINUS**구현)

```
SELECT DISTINCT c.cart_prod 판매상품, p.prod_name 상품명
FROM cart c, prod p
WHERE c.cart_prod = p.prod_id
      AND SUBSTR(c.cart_no,1,8) BETWEEN '20050401' AND '20050430'
      AND NOT EXISTS
        ( SELECT DISTINCT cart_prod
          FROM cart
          WHERE cart.cart_prod = c.cart_prod
            AND cart.cart_no LIKE '200506%' )
```

EXISTS

3) B)에는 있고 A)에는 없는 상품을 상품 제외(EXIST를 이용 MINUS구현)

```
SELECT DISTINCT c.cart_prod 판매상품, p.prod_name 상품명
FROM cart c, prod p
WHERE c.cart_prod = p.prod_id
AND c.cart_no LIKE '200506%'
AND NOT EXISTS
    ( SELECT DISTINCT cart_prod
      FROM cart
      WHERE cart.cart_prod = c.cart_prod
        AND SUBSTR(cart.cart_no,1, 6) = '200504')
```

EXISTS

- 2005년도 구매금액 2천만 이상 우수고객으로 지정하여 검색하시오 ?
(Alias는 회원ID, 회원명, '우수고객')

JOIN ORDER BY

- 2005년도 상품의 매입.매출현황을 조회(**UNION**문 사용)

- 일자, 상품명 순

```
SELECT TO_CHAR(buy_date,'YYYY/MM/DD') 일자,  
       prod_name 상품명, buy_qty 수량, '매입' 구분  
FROM buyprod, prod  
WHERE buy_prod = prod_id  
      AND buy_date BETWEEN '2005-01-01' AND '2005-12-31'  
UNION  
SELECT TO_CHAR( TO_DATE(SUBSTR(cart_no,1,8), 'YYYYMMDD')  
            , 'YYYY/MM/DD') 일자,  
       prod_name 상품명, cart_qty 수량, '매출' 구분  
FROM cart, prod  
WHERE cart_prod = prod_id  
      AND cart_no LIKE '2005%'  
ORDER BY 1, 상품명
```


INSERT

- 1) 프로그램을 통한 입력
- 2) 데이터베이스에서 제공하는 함수를 사용하여 입력
- 3) 데이터베이스에 이미 존재하는 자료를 활용하여 입력

```
● CREATE TABLE remain
( remain_year    CHAR(04) NOT NULL,           -- 해당 년도
  remain_prod    VARCHAR2(10) NOT NULL,        -- 상품 코드
  remain_j_00    NUMBER(5),                   -- 전년 재고
  remain_i       NUMBER(5),                   -- 입고
  remain_o       NUMBER(5),                   -- 출고
  remain_j_99    NUMBER(5),                   -- 현재 재고
  remain_date    DATE,                        -- 처리일자
  CONSTRAINT pk_remain PRIMARY KEY
              (remain_year, remain_prod),
  Constraint fr_remain_prod Foreign Key (remain_prod)
              References prod(prod_id) )
```

INSERT

1. 단순 데이터 입력

- 테이블의 전체 컬럼에 한 건의 자료를 입력하는 것.
- 단순히 자료를 추가할 때 자주 사용
- 수불테이블 자료추가 : 값을 지정하는 순서는 **Table**의 **column**순서, 각 **column**의 자료형식에 맞추어 지정
- **Table**전체 컬럼에 값을 입력
INSERT INTO remain
VALUES ('2003','P101000001',20,10,12,18,'2004-01-01')
- 상기 방법보다는 **Column**명을 기술하여 입력하는 방법 권장
INSERT INTO remain(remain_year,remain_prod,remain_j_00,
remain_i,remain_o,remain_j_99,remain_date)
VALUES ('2003','P101000002',11,7,6,12,'2004-01-02')

INSERT

- **INSERT**문의 가장 기본적인 방법으로 **1건씩**의 자료가 입력 테이블의 **NOT NULL**과 **UNIQUE**컬럼은 반드시 자료를 넣어 주어야 한다.

2. 특정 **Column**에 데이터 입력

- 데이터를 입력하고자하는 **Column**을 선택하여 입력
- 수불 테이블 특정 **Column**지정 **INSERT**
 - 해당년도 : **2003년** - 상품코드 : **P102000007** (대우 **VTR 6**헤드)
 - 입고 : **10**
- **INSERT INTO remain (remain_year, remain_prod, remain_i)**
VALUES('2003', 'P102000007', 10)
- **SELECT * FROM remain**

INSERT

3. 특수 Value, Null 값 입력

- Column값에 Null값을 지정하는 방법
 - Insert문장의 **Column List**에서 생략
 - Insert문장의 **Value**절에서 **Null** 또는 빈공백("")으로 지정

- 수불 테이블 특정 Column에 NULL값 INSERT

```
INSERT INTO remain ( remain_year, remain_prod, remain_j_00,  
                    remain_i, remain_j_99, remain_date)  
VALUES( '2003', 'P102000001', 31, 21, 41, '2003-12-31' )
```

```
INSERT INTO remain (remain_year,remain_prod,remain_j_00,  
                    remain_i,remain_o,remain_j_99,remain_date)  
VALUES( '2003', 'P102000002', 31, 21, NULL, 41, '2003-12-31' )
```

```
INSERT INTO remain (remain_year,remain_prod,remain_j_00,  
                    remain_i,remain_o,remain_j_99,remain_date)  
VALUES( '2003', 'P102000003', 31, 21, 11, 41, SYSDATE )
```

4. 다른 Table로부터 데이터 입력

- 테이블에 추가 시 타 테이블의 내용을 활용하여 자료 추가
- 모든 상품에 대한 재고 수불 파일 생성 (2003년도 재고 수불 마감)
 - 해당년도 : **2004년**
 - 상품코드 : 전 상품
 - 기초(전년)재고 : 상품코드의 우측 **2**자리를 숫자로 **Conversion**하여 처리 (원칙은 전년도말 재고가 되어야 함)
 - 입고 : **10**으로 일괄 처리
 - 출고 : **7**으로 일괄 처리
 - 현재고 : 전년재고 + 입고 - 출고
- ```
INSERT INTO remain (remain_year, remain_prod, remain_j_00,
 remain_i, remain_o, remain_j_99, remain_date)
SELECT '2004', prod_id, TO_NUMBER(SUBSTR(prod_id,-2)),
 10, 7, TO_NUMBER(SUBSTR(prod_id,-2))+10-7,
 SYSDATE
FROM prod
```
- ```
SELECT * FROM remain
```

INSERT

- 2004년도 재고수불 마감 작업을 수행하시오 ?
 - 모든 상품에 대한 재고 수불 파일 생성
 - 해당년도 : 2005년
 - 상품코드 : 2004년도 수불 상품
 - 전년재고 : 2004년도 말의 현재고
 - 현재고 : 2004년도 말의 현재고

```
SELECT remain_year 재고년도, remain_prod 상품코드, remain_j_99 현재고
FROM remain
WHERE remain_year = '2004'
```

```
-----
INSERT INTO remain (remain_year, remain_prod,
                    remain_j_00, remain_j_99, remain_date)
```

- **SELECT문 INSERT 사용과 2004년도 연말 재고를 조회한 후에 한 개씩 단순 INSERT사용 (자료가 수십만 건이라면 ?)**

UPDATE

1. 단순 UPDATE

- 이미 존재하는 **Column**의 값을 수정
- **WHERE** 조건이 없으면 테이블의 해당 **Column** 전체의 자료 수정
- **WHERE** 조건으로 수정하는 자료 지정

- '이'와 '김'씨 성을 가진 회원의 휴대폰 컬럼에
'011-111-1111'로 갱신

- 조건이 있으므로 **WHERE**조건에 **OR** 사용

1. 회원의 성명 및 휴대폰 조회

```
SELECT mem_id 회원ID, mem_name 성명, mem_hp 휴대폰  
FROM member
```

```
SELECT mem_id 회원ID, mem_name 성명, mem_hp 휴대폰  
FROM member  
WHERE mem_name LIKE '김%' OR mem_name LIKE '이%'
```

UPDATE

2. 휴대폰 컬럼에 '011-111-1111'로 갱신

UPDATE member

SET mem_hp = '011-111-1111'

WHERE mem_name LIKE '김%' OR mem_name LIKE '이%'

3. JOB 1번 수행 확인

- 회원테이블에서 회원ID가 'a001' 인 회원의 데이터를 다음과 같이 수정하시오 ?

(취미 : 독서, 직업 : 군인)

SELECT mem_id, mem_job, mem_like

FROM member

WHERE mem_id = 'a001'

UPDATE

- '이' 와 '김' 씨 성을 가진 회원 중 회원ID가 'a001', 'j001'인 회원을 제외하고 휴대폰 컬럼에 '099-999-9999'로 갱신

1. 회원의 성명 및 휴대폰 조회

```
SELECT mem_id 회원ID, mem_name 성명, mem_hp 휴대폰  
FROM member  
WHERE (mem_name LIKE '김%' OR mem_name LIKE '이%')  
AND mem_id NOT IN ('a001', 'j001')
```

2. 휴대폰 컬럼에 '099-999-9999'로 갱신

```
UPDATE member  
SET mem_hp = '099-999-9999'  
WHERE (mem_name LIKE '김%' OR mem_name LIKE '이%')  
AND mem_id NOT IN ('a001', 'j001')
```

3. JOB 1번 수행 확인

UPDATE

- 회원테이블에서 모든 회원의 마일리지 **Column**값을 **10%** 인상되게 수정하시오 ?

```
SELECT mem_mileage, mem_mileage * 1.1  
FROM member
```

UPDATE member

SET mem_mileage = mem_mileage * 1.1

- 재고수불 테이블에서 해당연도가 '**2003**' 년이고, 상품분류가 '**P102**' 이면서 전년재고가 **NULL**인 상품의 수량을 **0**으로 수정하시오 ?

```
SELECT remain_year, remain_prod, remain_j_00  
FROM remain  
WHERE remain_year = '2003'  
AND remain_prod LIKE 'P102%'  
AND remain_j_00 IS NULL
```

UPDATE

2. EXISTS UPDATE

- 단순 **UPDATE**에 비해 수정할 자료를 좀더 세밀하게 제어
- **WHERE**절에서 **IN**이나 **EXISTS**를 활용
- 2005년도 판매금액이 8천만 이상인 거래처의 담당자 컬럼에 '우수거래처'로 갱신 (**WHERE**절에 **EXISTS**문 사용)

1. 거래처테이블의 담당자 조회

```
SELECT buyer_id, buyer_name, buyer_charger  
FROM buyer
```

2. 판매금액 8천만 이상 우수거래처로 지정 (조회)

```
SELECT buyer_id, buyer_name, buyer_charger, '우수거래처'  
FROM buyer  
WHERE EXISTS ( SELECT SUM(cart_qty * prod_sale)  
                FROM prod, cart  
                WHERE cart_no LIKE '2005%'  
                  AND cart_prod = prod_id  
                  AND prod_buyer = buyer_id  
                HAVING SUM(cart_qty * prod_sale) > 80000000 )
```

UPDATE

- 2005년도 판매금액이 8천만 이상인 거래처의 담당자 컬럼에 '우수거래처'로 갱신 (**WHERE절에 EXISTS문 사용**)

(갱신)

```
UPDATE buyer SET buyer_charger = '우수거래처'
WHERE EXISTS ( SELECT SUM(cart_qty * prod_sale)
                FROM prod, cart
                WHERE cart_no LIKE '2005%'
                AND cart_prod = prod_id
                AND prod_buyer = buyer_id
                HAVING SUM(cart_qty * prod_sale) > 80000000)
```

3. JOB 1번 수행 확인

UPDATE

- **2005년도 구매금액이 3천만 이상인 회원의 마일리지 점수를 20만으로 하여 수정하시오 ?**

- 1. 회원테이블의 마일리지 조회

```
SELECT mem_id, mem_name, mem_mileage FROM member
```

- 2. 구매금액 3천만 이상 마일리지 조회

```
SELECT mem_id, mem_name, mem_mileage  
FROM member
```

```
WHERE EXISTS ( SELECT SUM(cart_qty * prod_sale)  
                FROM prod, cart  
                WHERE cart_no LIKE '2005%'  
                AND cart_prod = prod_id  
                AND cart_member = mem_id  
                HAVING SUM(cart_qty * prod_sale) > 30000000 )
```

- 3. 구매금액 3천만 이상 마일리지 20만으로 갱신

3. JOIN UPDATE

- 단순 **UPDATE** 비해 대량의 수정할 자료를 처리
- 예) 월 마감처리 등

- 상품 모두의 **2005**년도 입고수량 을 합산하여 상품테이블의 총 입고수량 **Column** 갱신

1. 상품테이블의 총 입고수량 조회

```
SELECT prod_id 상품코드,  
       SUM(NVL(prod_qtyin,0)) 총입고수량  
FROM prod  
GROUP BY prod_id
```

2. 상품입고 테이블에서 **2005**년도 상품별 입고수량 합산하여 조회

```
SELECT prod_id 상품코드,  
       SUM(NVL(buy_qty,0)) 매입수량  
FROM prod LEFT OUTER JOIN buyprod  
       ON ( prod_id = buy_prod  
           AND buy_date BETWEEN '2005-01-01' AND '2005-12-31')  
GROUP BY prod_id
```

3. JOIN UPDATE

- 상품 모두의 **2005**년도 입고수량 을 합산하여 상품테이블의 총 입고수량 **Column** 갱신

3. 2005년도 상품별 입고수량 합산하여 갱신

```
UPDATE prod
  SET prod_qtyin =
    ( SELECT SUM(buy_qty)
      FROM buyprod
     WHERE prod_id = buy_prod
       AND buy_date BETWEEN
         '2005-01-01' AND '2005-12-31' )
```

4. JOB 1번 수행 확인

(●주목)

1. 대량의 자료를 갱신할때 **JOIN**을 활용한 일괄처리를 모를 경우 황당
2. 일괄 갱신의 의미를 파악

UPDATE

- 상품 모두의 **2005**년도 판매수량을 합산하여 상품테이블의 총판매수량 **Column** 갱신

1. 상품테이블의 총판매수량 조회

```
SELECT prod_id 상품코드,  
       SUM(NVL(prod_qty,0)) 총판매수량  
FROM prod  
GROUP BY prod_id
```

2. 장바구니 테이블에서 **2005**년도 상품별 판매수량 합산하여 조회

```
SELECT prod_id 상품코드,  
       SUM(NVL(cart_qty,0)) 매출수량  
FROM prod LEFT OUTER JOIN cart  
       ON ( prod_id = cart_prod  
           AND cart_no LIKE '2005%' )  
GROUP BY prod_id
```


UPDATE

3. 2005년도 상품별 판매수량 합산하여 갱신 하시오 ?

4. 수행 결과 확인(검증)

```
SELECT SUM( NVL( prod_qtyin, 0 ) ),  
        SUM( NVL( prod_qtysale, 0 ) )  
FROM prod
```

```
SELECT SUM(buy_qty)  
FROM buyprod  
WHERE buy_date BETWEEN '2005-01-01' AND '2005-12-31'
```

```
SELECT SUM(cart_qty)  
FROM cart  
WHERE cart_no LIKE '2005%'
```

UPDATE

- 상품 모두의 **2005**년도 입고수량, 판매수량을 합산하여 재고수불 (**remain**)테이블의 입고, 출고, 현재고 **Column**을 수정하시오 ?

1. 2005년도 입고수량 및 판매수량을 서브쿼리로 조회

```
SELECT prod_id,  
       ( SELECT SUM(NVL(buy_qty,0))  
         FROM buyprod  
        WHERE prod_id = buy_prod  
              AND TO_CHAR(buy_date,'yyyy') = '2005' ) in_amt,  
       ( SELECT SUM(NVL(cart_qty,0))  
         FROM cart  
        WHERE prod_id = cart_prod  
              AND cart_no LIKE '2005%' ) out_amt  
FROM prod
```

UPDATE

2. 1) 번에의 쿼리를 **Inline View** 로 적용하여 재고수불 테이블 수정
(다중행, 다중열 서브쿼리는 자주 사용하지는 않지만 **Update** 할 때
유용하게 사용할 수 있음.)

UPDATE remain

```
SET (remain_i, remain_o, remain_j_99)
= ( SELECT in_amt, out_amt, remain_j_00 + in_amt - out_amt
    FROM (
        SELECT prod_id,
        ( SELECT SUM(NVL(buy_qty,0))
          FROM buyprod
         WHERE prod_id = buy_prod
               AND TO_CHAR(buy_date,'yyyy') = '2005' ) in_amt,
        ( SELECT SUM(NVL(cart_qty,0))
          FROM cart
         WHERE prod_id = cart_prod
               AND cart_no LIKE '2005%' ) out_amt
        FROM prod
      ) A
    WHERE remain_prod = A.prod_id
  )
```

WHERE remain_year = '2005'

UPDATE

3. 수행 결과 확인(검증)

```
SELECT * FROM remain WHERE remain_year = '2005'
```

```
SELECT SUM( NVL( remain_i, 0 ) ),  
        SUM( NVL( remain_o, 0 ) )  
FROM remain  
WHERE remain_year = '2005'
```

```
SELECT SUM(buy_qty)  
FROM buyprod  
WHERE buy_date BETWEEN '2005-01-01' AND '2005-12-31'
```

```
SELECT SUM(cart_qty)  
FROM cart  
WHERE cart_no LIKE '2005%'
```

DELETE

Subquery 로 Table 생성

- 기존의 테이블을 복사하여 테이블 생성
- 특정열, 특정행만 추출하여 사용자가 원하는 테이블 생성
- 단순백업, 테이블 수정등의 용도로 사용
- - 인덱스는 복사되지 않는다.
 - 제약사항 중 **NULL** 속성만 복사
- **SubQuery** 로 Table 생성 구문
CREATE TABLE *table_name* [column_list...]
AS
Subquery
- 재고 수불 테이블을 복사하여 **remain2** 테이블을 생성하시오
CREATE TABLE remain2
AS
SELECT * FROM remain

DELETE

1. 단순 DELETE

- **WHERE** 조건이 없으면 테이블의 해당 컬럼 전체의 자료 삭제
- **WHERE** 조건으로 삭제하는 자료 지정
- 상품분류테이블의 내용을 모두 삭제
 - 1) 전체 **RECORD** 삭제
 - **DELETE FROM *table_name***
(ROLLBACK 가능, 느림)
 - **TRUNCATE TABLE *table_name***
(자동 **COMMIT** (복구 불가), 빠름)
 - 2) **TABLE** 제거
 - 자료가 수십만 건 이상일 경우 또는 빠른 시간 내에 자료전체를 조건 없이 삭제해야 하는 경우 사용가능
 - 단 **TABLE**의 모든 이력이 사라지므로 **TABLE CREATE, PRIMARY KEY, INDEX**등을 다시 생성하여 주어야 한다.

DROP TABLE *table_name*

DELETE

- 재고수불2 테이블의 내용을 모두 삭제
 - 1) **DELETE**로 전체 레코드 삭제
DELETE FROM remain2
 - 2) 작업 취소
ROLLBACK
 - 3) **TRUNCATE** 로 전체 레코드 삭제
TRUNCATE TABLE remain2
 - 4) 작업 취소
ROLLBACK
 - 5) **TABLE** 제거
DROP TABLE remain2

DELETE

- 재고수불 테이블에서 **2003**년도 자료 중 출고수량이 **6**개 또는 **11**개인 자료를 삭제
 - 조건이 있으므로 **WHERE**조건에 **OR** 사용
SELECT remain_year, remain_prod, remain_o
FROM remain
WHERE remain_year = '2003'
AND (NVL(remain_o, 0) = 6 OR NVL(remain_o,0)= 11)

DELETE FROM remain
WHERE remain_year = '2003'
AND (NVL(remain_o, 0) = 6 OR NVL(remain_o,0) =11)
 - 조건에 해당되는 컬럼이 숫자 값 일 경우 **NULL**이 있으면 자료연산을 하지 않기 때문에, 모두 **NULL Checking**을 하여 **DELETE**
(단순 연산조건을 통한 삭제 시 권장 : 자료누락의 가능성을 배제)

DELETE

- 재고수불 테이블에서 **2003**년도 자료 중 입고수량 + 출고수량이 **20**개 이상인 자료를 삭제 하시오 ?
(조건이 있으므로 **WHERE**조건에 **OR** 사용)
- 1. 조회
(**Alias**는 연도, 상품, 입고량, 출고량 ,합계수량)
SELECT remain_year 연도, remain_prod 상품, remain_i 입고량,
remain_o 출고량, remain_i + remain_o 합계수량
FROM remain
WHERE remain_year = '2003'
AND (NVL(remain_i,0) + NVL(remain_o,0)) >= 20
- 2. 삭제

2. IN / EXISTS DELETE

- **WHERE** 조건에 **IN**이나 **EXISTS**를 사용함으로써 삭제할 자료를 좀 더 세밀하게 제어
- 장바구니 2005년도 5월 자료 중 회원ID='p001' (오성순)인 자료 조회

```
SELECT mem_id, mem_name, cart_no, cart_prod,  
       cart_member, cart_qty  
FROM member, cart  
WHERE mem_id = 'p001'  
      AND cart_no LIKE '200505%'  
      AND cart_member = mem_id
```
- 상기 자료 중 상품= 'P202000013' 인 자료 삭제 (**EXISTS**문 사용)

```
DELETE FROM cart  
WHERE EXISTS (SELECT mem_id  
               FROM member  
               WHERE mem_id = 'p001'  
                  AND cart_member = mem_id)  
      AND cart_no LIKE '200505%'  
      AND cart_prod = 'P202000013 '
```

2. IN / EXISTS DELETE

- 장바구니 2005년도 5월 자료 중 회원ID='p001' (오성순)인 자료 조회

-상기 자료 중 상품='P302000005'인 자료 삭제 (**IN**문 사용)

```
DELETE FROM cart
WHERE cart_member IN (SELECT mem_id
                      FROM member
                      WHERE mem_id = 'p001'
                      AND cart_member = mem_id)
AND cart_no LIKE '200505%'
AND cart_prod = 'P302000005'
```

- 1) 다수의 테이블과 연관관계에 있어서 **EXISTS**는 **IN**에 비해 용이.
- 2) 실습 2처럼 **IN**으로 가능하지만 한계가 있다.
(**EXISTS**절 사용을 권장)

2. IN / EXISTS DELETE

- 장바구니테이블 자료추가 : 기존 삭제한 자료

```
INSERT INTO cart(cart_member, cart_no, cart_prod, cart_qty)
VALUES ('p001', '20050052100002', 'P202000013', 7)
```

```
INSERT INTO cart(cart_member, cart_no, cart_prod, cart_qty)
VALUES ('p001', '2005052100002', 'P302000005', 3)
```

- 상품입고테이블에서 거래처가 'P20101(대현)'인 자료를 조회하시오 ?
(Alias는 입고일자, 상품, 수량, 거래처코드, 거래처명)

```
SELECT buy_date 입고일자, buy_prod 상품, buy_qty 수량,
       buyer_id 거래처코드, buyer_name 거래처명
FROM buyprod, buyer, prod
WHERE buy_prod = prod_id
      AND prod_buyer = buyer_id
      AND buyer_id = 'P20101'
```

2. IN / EXISTS DELETE

- 상기 조회 자료 중 입고일자가 **2005-01-28**이고,
상품= '**P201000007**' 인 자료를 삭제 하시오 ? (**EXISTS**문 사용)

VIEW 객체

- View는 Table과 유사한 객체이다.
- View는 기존의 테이블이나 다른 View 객체를 통하여 새로운 SELECT문의 결과를 테이블처럼 사용한다.(가상테이블)
 - View는 SELECT문에 귀속되는 것이 아니고,
독립적인 테이블처럼 존재
- View를 이용하는 경우
 - 필요한 정보가 한 개의 테이블에 있지 않고, 여러 개의 테이블에 분산되어 있는 경우
 - 테이블에 들어 있는 자료의 일부분만 필요하고 자료의 전체 row나 column이 필요하지 않은 경우
 - 특정 자료에 대한 접근을 제한하고자 할 경우(보안)

VIEW 객체

● View 구문

```
CREATE [OR REPLACE] [FORCE|NOFORCE ] VIEW view_name
  [ columnName_list ... ]
AS
  select-statement
[WITH CHECK OPTION [CONSTRAINT constraint ] ]
[WITH READ ONLY]
```

항목	내용
OR REPLACE	존재하는 경우 변경
FORCE	기본 테이블이 없어도 View 생성 가능 옵션 default는 NOFORCE 이다.
view_name	View의 객체명
select-statement	View를 생성하기 위한 서브쿼리
WITH CHECK OPTION	View에 보일 수 있을 때만 추가, 수정 가능
WITH READ ONLY	읽기 전용 View, (DML 실행 금지)

VIEW 객체

● VIEW 사용시 주의할 점

- View를 생성할 때 제약조건(WITH)이 있는 경우 **ORDER BY** 절 불가.
- View가 집계 함수, **GROUP BY**절, **DISTINCT**를 사용하여 만들어진 경우 **INSERT, UPDATE, DELETE** 구문을 사용할 수 없다.
- 어느 컬럼이 표현식, 일반 함수를 통하여 만들어진 경우 해당 컬럼의 추가 및 수정 불가능
- **CURRVAL, NEXTVAL** 의사컬럼(pseudo column) 사용불가
- **ROWID, ROWNUM, LEVEL** 의사컬럼을 사용할 경우 **alias**사용

● view_prod1 생성 : 상품분류, 상품수

```
CREATE VIEW view_prod1 (분류, 상품수)
AS
  SELECT prod_lgu, COUNT(*) FROM prod
  GROUP BY prod_lgu
```

```
SELECT * FROM view_prod1
```


VIEW 객체

- **view_member** 생성: 회원명, 아이디, 성명, 마일리지, 생일, 직업

```
CREATE or REPLACE VIEW view_member
AS
  SELECT mem_name "회원명", mem_id "아이디",
         mem_mileage "마일리지",
         TO_CHAR(mem_bir,'yyyy-mm-dd') "생일",
         mem_job "직업"
  FROM member
 WHERE mem_mileage > 1000
 WITH CHECK OPTION
```

```
SELECT * FROM view_member
```

```
UPDATE view_member SET 회원명 = '홍길동'
WHERE 아이디 = 'e001'
```

```
UPDATE view_member SET 마일리지 = 800
WHERE 아이디 = 'e001'
```

VIEW 객체

- 회원아이디, 회원명, 나이, 주문번호, 주문 상품명, 가격, 수량, 금액을 보여주는 읽기전용 **View** 를 생성하시오?

(객체명 : **view_cart** , 판매금액 = 수량 * 상품판매가)

- 'a001', 'f001' 회원의 주문내역을 모두 조회하시오?

SEQUENCE 객체

- **Sequence**는 연속, 열거, 순서의 의미.
- **Sequence**객체는 자동적으로 번호를 생성하기 위한 객체.
- **Sequence**객체는 테이블과 독립적이므로 여러 곳에서 사용가능.
- **Sequence**를 이용하는 경우
 - **Primary Key**를 설정할 후보키가 없거나 **PK**를 특별히 의미 있게 만들지 않아도 되는 경우
 - 자동으로 순서적인 번호가 필요한 경우

- 상품 분류테이블에 '**P103**', '**USB** 제품'을 등록하시오?
(단, **lprod_id** 컬럼의 값은 최대값을 구하여 1을 더한 서브쿼리를 작성하여 입력한다.)

```
SELECT NVL( MAX(lprod_id), 0) FROM lprod
```

SEQUENCE 객체

● Sequence 구문

CREATE SEQUENCE *sequence_name*

[START WITH *n*]

[INCREMENT BY *n*]

[MAXVALUE *n* | NOMAXVALUE]

[MINVALUE *n* | NOMINVALUE]

[CYCLE | NOCYCLE]

[CACHE *n* | NOCACHE]

[ORDER | NOORDER]

Pseudo Column	내 용
시퀀스명.NEXTVAL	시퀀스객체의 다음 값(Next Value) 리턴
시퀀스명.CURRVAL	시퀀스객체의 현재 값(Current Value) 리턴

- **CURRVAL**은 해당 세션에서 **NEXTVAL**이 최소한 한번 이상 호출 되었을 때만 사용 가능하다.

SEQUENCE 객체

항 목	내 용
<code>sequence_name</code>	sequence 객체명
<code>START WITH n</code>	시작값 설정, 기본은 <code>MIN_VALUE</code>
<code>INCREMENT BY n</code>	자동 증가값 기본은 1, 음수 사용가능
<code>MAXVALUE n</code> <code>NOMAXVALUE</code>	시퀀스의 최대값 설정, default는 <code>NOMAXVALUE</code> 이며 10^{27} 까지
<code>MINVALUE n</code> <code>NOMINVALUE</code>	시퀀스의 최소값 설정, default는 <code>NOMINVALUE</code> 이며 1이다
<code>CYCLE</code> <code>NOCYCLE</code>	최대값 또는 최소값 이후 다시 시작할지 여부 default는 <code>NOCYCLE</code>
<code>CACHE n</code> <code>NOCACHE</code>	메모리에 미리 할당할 값, default는 <code>CACHE 20</code>
<code>ORDER</code> <code>NOORDER</code>	요청순서대로 생성을 보증, default는 <code>NOORDER</code>

- `INCREMENT BY`가 음수로 지정되면 내림차순의 시퀀스가 되면서 `NOMAXVALUE` 는 -1로 , `NOMINVALUE` 는 -10^{27} 이 된다.
`START WITH`도 `MIN_VALUE`가 아닌 `MAX_VALUE`로 설정된다.

SEQUENCE 객체

- **lprod_seq** 시퀀스 생성

```
CREATE SEQUENCE lprod_seq  
INCREMENT BY 1  
START WITH 15
```

- 상품 분류테이블에 'P203', '아동복' 을 등록하시오?

```
INSERT INTO lprod ( lprod_id, lprod_gu, lprod_nm )  
VALUES ( lprod_seq.NEXTVAL, 'P203', '아동복' )
```

```
SELECT lprod_seq.NEXTVAL FROM dual
```

```
SELECT lprod_seq.CURRVAL FROM dual
```

SEQUENCE 객체

- **Sequence**가 사용되는 경우
 - **SELECT**문의 **SELECT** 절 (단, **subquery**, **VIEW**가 아닌)
 - **INSERT** 문의 **SELECT** 절
 - **INSERT** 문의 **VALUES** 절
 - **UPDATE**문의 **SET** 절
- **Sequence**가 제한되는 경우
 - **SELECT**, **DELETE**, **UPDATE** 에서의 서브쿼리
 - **VIEW**의 **query** 구문
 - **DISTINCT** 가 있는 **SELECT** 문
 - **GROUP BY**, **ORDER BY** 가 있는 **SELECT** 문
 - **SET** 연산자(**UNION**, **INTERSECT**, **MINUS**)가 있는 **SELECT** 문
 - **SELECT** 문의 **WHERE** 절
 - **CREATE / ALTER TABLE** 에서 컬럼의 **DEFAULT** 값
 - **CHECK** 제약조건

SEQUENCE 객체

- Sequence 변경

- **START WITH**는 변경할 수 없다.

```
ALTER SEQUENCE lprod_seq  
INCREMENT BY 2  
MAXVALUE 999
```

- Sequence 제거

```
DROP SEQUENCE lprod_seq
```

- 다음 요건을 만족하는 시퀀스를 생성하시오?

객체명 : **cart_seq**, **증감값** : 1, **최소값** : 10000, **최대값** : 99999, 순환가능

```
SELECT cart_seq.NEXTVAL FROM dual
```


SYNONYM 객체

- **Synonym**은 동의어, 별칭의 의미.
- **Synonym**은 객체에 대한 다른 이름으로 대체한다.
- **Synonym**을 이용하는 경우
 - 다른 소유자의 객체를 접근하는 경우 "스키마명.객체명"으로 접근하는데 이를 편하게 한다.
 - 긴 이름의 객체명을 쉬운 이름으로 대체하고자 할 때

● Sequence 구문

**CREATE [or REPLACE] [PUBLIC] SYNONYM *synonym_name*
FOR *object_name***

항 목	내 용
or REPLACE	존재하는 경우 변경
PUBLIC	공개 시노ним을 생성할 경우 기술한다.
<i>synonym_name</i>	synonym 객체명
<i>object_name</i>	동의어에 해당하는 객체명

SYNONYM 객체

- **CREATE SYNONYM mem
FOR member**
- **SELECT * FROM mem**
- **CREATE SYNONYM mydual
FOR sys.dual**
- **SELECT 'Hello World' FROM mydual**
- **DROP SYNONYM mydual**

INDEX 객체

항 목	내 용
정의	DB Server 의 성능을 가장 많이 좌우하게 되는 요소 중 하나로써 특정 데이터를 빨리 찾기 위해 사용한다
용도	<ul style="list-style-type: none"> - SELECT 문과 INSERT 문 혹은 UPDATE 문에서 WHERE 절을 사용하게 되는 경우에 인덱스를 사용 - 또한 SELECT한 데이터를 SORT(ORDER BY)하거나 GROUP BY를 사용하여 그룹별로 묶을 때에도 인덱스를 사용하게 되면 속도 향상에 도움 - DBMS의 부하를 줄여 전체 성능을 향상 시킨다.
단점	<ul style="list-style-type: none"> - 인덱스를 사용하게 되면 인덱스를 만드는 데 많은 저장 공간과 시간이 소요된다. - 이는 처음 만드는 경우에만 적용되는 것이 아니라 데이터가 지속적으로 삽입, 수정, 삭제의 과정을 거칠 때마다 요구되는 사항이며, - 데이터를 수정하는데 있어서 더 많은 시간이 소요된다. - 따라서 인덱스는 꼭 필요한 곳에만 적절하게 사용

INDEX 객체

항 목	내 용
INDEX필요 컬럼	<ul style="list-style-type: none"> - 자주 검색해야 하는 컬럼 - WHERE절에서 '='을 사용하여 특정 값을 찾는 경우 - WHERE절에서 범위를 주고 검색하는 경우 - Primary Key - Foreign Key : Join시에 속도를 향상 시킨다. - Sort(Order by)를 자주 사용하는 컬럼 - Join을 자주 사용하는 컬럼
INDEX불필요 컬럼	<ul style="list-style-type: none"> - 검색을 잘 안하는 컬럼 - 같은 값이 많이 입력되는 컬럼(예: 성별, 나이등) : - WHERE절을 사용했을 때 반환되는 결과가 전체 테이블에서 많은 양을 차지하는 경우 - SELECT 속도보다 INSERT나 UPDATE, DELETE등의 속도가 더 중요한 경우

INDEX 객체

분 류	내 용
Unique / Non-Unique	<ul style="list-style-type: none"> - Unique 인덱스는 중복된 값을 허용하지 않음 - Non-Unique 인덱스는 중복 허용 - 인덱스는 Null 값도 사용할 수 있으나, Key(Primary 혹은 Foreign)에서는 사용할 수 없다. - Null을 허용하는 컬럼에 대해서 Unique 인덱스를 설정하게 되면 Null 값도 하나만 인정한다.
Single / Composite	<ul style="list-style-type: none"> - 단일인덱스는 구성하는 컬럼이 한 개 - 복합인덱스는 두 개 이상의 컬럼으로 구성 - 복합인덱스의 경우 검색할 때 조건절에 두 개가 포함 될 때 효과가 크다. - 복합인덱스의 경우 구성컬럼의 순서와 조건절의 순서도 맞추는 것이 좋다.
자동 / 수동 Index	<ul style="list-style-type: none"> - 테이블을 생성 할 때 PK, Unique 제약조건을 주면 오토컬이 자동으로 Unique 인덱스를 생성 (제약조건과 Index는 별개임) - 수동인덱스는 사용자가 직접 생성하는 인덱스

INDEX 객체

타입	내용
Normal Index (B-tree Index)	<ul style="list-style-type: none"> - Default 인덱스이다. - 트리구조이므로 검색횟수는 모두 동일 - 컬럼의 값과 ROWID(물리적인 위치정보)를 기반으로 저장된다.
Bitmap Index	<ul style="list-style-type: none"> - Cardinality가 적은 경우 효율적 (성별, 결혼, 탈퇴여부) - 추가, 수정, 삭제가 많은 경우 비효율적 - ROWID와 컬럼의 값을 이진(0,1)으로 조합하여 저장
Function-based Index	<ul style="list-style-type: none"> - 조건절에 함수를 사용하여 검색하는 경우가 많다면 인덱스를 생성 할 때 함수를 사용하여 만든다.
이외에도 Domain index, Partitioned index 등 다양	

INDEX 객체

● Index 구문

```
CREATE [ UNIQUE | BITMAP ] INDEX index_name
ON table_name (column1 [column2 ...] [ASC | DESC])
```

항 목	내 용
UNIQUE BITMAP	기본은 NONUNIQUE
<i>index_name</i>	인덱스 객체명
<i>table_name</i>	인덱스의 대상이 되는 테이블명
ASC DESC	오름차순, 내림차순으로 인덱스 생성, 기본은 ASC

● Index 구문은 형식에 따라 옵션도 다르고 복잡하다.

Oracle사에서 제공하는 **SQL Reference**를 참조하기 바람.

INDEX 객체

- 회원 생일이 조건절에 자주 사용되어 **Index**를 생성

```
CREATE INDEX idx_member_bir  
ON member ( mem_bir )
```

```
SELECT mem_id, mem_name, mem_job, mem_bir  
FROM member  
WHERE TO_CHAR( mem_bir, 'YYYY') = '1975'
```

- 회원 생일에서 년도만 분리하여 인덱스를 생성(Function-based Index)

```
CREATE INDEX idx_member_bir_year  
ON member ( TO_CHAR( mem_bir, 'YYYY' ) )
```

```
SELECT mem_id, mem_name, mem_job, mem_bir  
FROM member  
WHERE TO_CHAR( mem_bir, 'yyyy' ) = '1975'
```

- Index Scan? Full Scan?

INDEX 객체

● Index 삭제

DROP INDEX *index_name*

- 인덱스는 수정할 수 없다.
- 수정을 원하는 경우 삭제하고 다시 생성

● Index 재생성

ALTER INDEX *index_name* REBUILD

- 인덱스를 다른 테이블 스페이스로 이동한 후(기준 테이블포함)
- 삭제된 행이 많아서 인덱스를 재 구성해야 할 때

- **idx_member_bir** 인덱스는 삭제하고,
idx_member_bir_year 인덱스는 **rebuild** 하시오?

INDEX 객체

- **INDEX KEY Column**에 변형을 막는 **Query**문 사용 권장

- ```
SELECT buy_date, buy_prod, buy_qty
FROM buyprod
WHERE buy_date - 10 = '2005-02-20'
```

==> 재구성

```
SELECT buy_date, buy_prod, buy_qty
FROM buyprod
WHERE buy_date = TO_DATE('2005-02-20') + 10
```

## INDEX 객체

- **SELECT cart\_no, cart\_prod, cart\_qty  
FROM cart  
WHERE SUBSTR(cart\_no,1,8) = '20050405'**

==> 재구성

**SELECT cart\_no, cart\_prod, cart\_qty  
FROM cart  
WHERE cart\_no LIKE '20050405%'**

==> 뚫는

**SELECT cart\_no, cart\_prod, cart\_qty  
FROM cart  
WHERE cart\_no > '2005040500000'  
AND cart\_no < '2005040599999'**

## Data Dictionary

- **DataBase**를 위한 **Meta 정보**를 관리한다.
- **Data Dictionary**는 **SYS user**의 소유이며 **Oracle** 서버의 중요한 구성 요소 중 하나이며 데이터베이스에 대한 모든 정보를 가진다.
- 대부분 읽기 전용으로 제공되는 **Table**과 **View**의 집합이다.
- 데이터베이스에 대한 작업이 수행되면 자동으로 **Data Dictionary**를 갱신하고 유지 보수한다.

### ● **Data Dictionary**의 주요 정보

- 모든 사용자에 대한 정보
- 사용자에게 허가된 권한
- 데이터베이스의 모든 객체 정보(**Table, View, Index, Procedure** 등)
- 모든 객체들의 저장정보
- 테이블의 컬럼에 대한 **Default** 값 및 제약 조건
- 감사(**Audit**) 정보
- 기타 데이터베이스에 대한 정보

## Data Dictionary

- **Base Table**이란 데이터베이스에 관련된 정보들이 저장된 테이블이며 해당 테이블에 제어하는 것은 불가능하다.
- **Base Table**의 일부 정보를 **View**를 통해서 얻을 수 있게 했다.  
이를 **System View**라 한다.

| 접두어   | 설 명                                                                |
|-------|--------------------------------------------------------------------|
| ALL_  | 사용자가 접근 가능한 모든 스키마의 정보를 가진 뷰                                       |
| USER_ | 현재 사용자의 스키마 정보만을 가진 뷰                                              |
| DBA_  | 관리자용 뷰, DBA권한의 사용자만 접근 가능                                          |
| V\$   | Dynamic Performance View, 현재 DataBase의 상태에 관한 정보를 가진, 대부분 DBA권한 필요 |

- 현재 세션의 사용 가능 **Data Dictionary** 정보는 **Dictionary View**에서 제공

## Data Dictionary

- **ALL\_OBJECTS**의 모든 컬럼 상세

DESC all\_objects

| 이름             | 널        | 유형           |
|----------------|----------|--------------|
| -----          | -----    | -----        |
| OWNER          | NOT NULL | VARCHAR2(30) |
| OBJECT_NAME    | NOT NULL | VARCHAR2(30) |
| SUBOBJECT_NAME |          | VARCHAR2(30) |
| OBJECT_ID      | NOT NULL | NUMBER       |
| DATA_OBJECT_ID |          | NUMBER       |
| OBJECT_TYPE    |          | VARCHAR2(19) |
| CREATED        | NOT NULL | DATE         |
| LAST_DDL_TIME  | NOT NULL | DATE         |
| TIMESTAMP      |          | VARCHAR2(19) |
| STATUS         |          | VARCHAR2(7)  |
| TEMPORARY      |          | VARCHAR2(1)  |
| GENERATED      |          | VARCHAR2(1)  |
| SECONDARY      |          | VARCHAR2(1)  |

## Data Dictionary

- Dictionary 뷰에서 'ALL\_'로 시작 하는 모든 테이블 조회

```
SELECT table_name, comments
FROM dictionary
WHERE table_name LIKE 'ALL_%';
```

- 현재 로그인한 사용자가 만든 모든 객체 정보를 출력

```
SELECT object_name, object_type, created
FROM all_objects
WHERE owner = 'PC99'
ORDER BY object_type ASC
```

## Data Dictionary

### ● 자주 사용되는 **USER** 뷰

| 뷰명                | 설명                    |
|-------------------|-----------------------|
| USER_OBJECTS      | 해당 스키마의 모든 객체들        |
| USER_TABLES       | 테이블 정보                |
| USER_TAB_COLUMNS  | 테이블의 컬럼 정보            |
| USER_VIEWS        | 뷰 정보                  |
| USER_SEQUENCES    | 시퀀스 정보                |
| USER_CONSTRAINTS  | 제약 조건                 |
| USER_CONS_COLUMNS | 제약조건을 가진 컬럼 정보        |
| USER_INDEXES      | 인덱스 정보                |
| USER_IND_COLUMNS  | 인덱스 컬럼 정보             |
| USER_TRIGGERS     | 트리거 정보                |
| USER PROCEDURES   | 프로시저, 함수, 패키지 정보      |
| USER_SOURCE       | 프로시저, 함수, 패키지, 트리거 소스 |
| USER_SYS_PRIVS    | 시스템 권한 정보             |



## Data Dictionary

- **USER\_TABLES**의 컬럼 상세를 확인하고 각 테이블 전체 레코드 개수를 출력. (테이블명, 레코드 수)

**DESC user\_tables**

**SELECT table\_name, num\_rows  
FROM user\_tables**

- **USER\_CONSTRAINTS, USER\_CONS\_COLUMNS**의 컬럼 상세를 확인하고 상품 테이블의 제약조건을 출력하시오?  
(컬럼명, 제약명, 타입, 제약내용)

### 3. PL/SQL 의 개요

#### PL/SQL이란?

- PL/SQL은 Procedural Language/SQL
- 서버에서 절차적인 처리를 위해 표준 SQL을 확장한 절차적 언어
- 블록(block) 구조로 여러 SQL문을 한번에 실행
- 모듈화, 캡슐화가 가능

|        | 장 점                                             | 단 점                                                                                       |
|--------|-------------------------------------------------|-------------------------------------------------------------------------------------------|
| SQL    | 작성하기 쉽다.<br>쉽게 배우고 있다.<br>ANSI에 문법 표준화          | 변수 선언, 비교, 반복, 예외처리 불가<br>SQL 구문을 캡슐화 할 수 없다.<br>매번 구문 분석으로 성능저하<br>긴 구문 전송으로 네트워크 트래픽 증가 |
| PL/SQL | 변수, 비교, 반복, 예외처리<br>모듈화 및 캡슐화<br>서버에 저장되어 빠른 실행 | 문법에 대한 표준이 없다<br>각 DBMS에 종속적                                                              |

## PL/SQL로 할 수 있는 것?

| 구분               | 내용                                                               |
|------------------|------------------------------------------------------------------|
| Anonymous block  | 단순 스크립트에서 실행되는 블록<br>서버에 저장되지 않는다.                               |
| Stored Procedure | 자주 실행되거나, 복잡한 비즈니스 로직을 미리 작성하여 서버에 저장하여 사용한다.                    |
| User Function    | Procedure와 유사하며, 실행결과를 반환한다.                                     |
| Package          | 여러 Procedure, Function 및 변수 등을 하나로 묶는다.                          |
| Trigger          | 테이블이나 뷰에 INSERT, UPDATE, DELETE등이 수행 전 또는 수행 후 자동 실행되는 Procedure |

## PL/SQL Block Structure

**DECLARE**

**Declation Section**  
(선언 부분)

**BEGIN**

**Excutabl Section**  
(실행 부분)

**EXCEPTION**

**Exception Section**  
(예외 처리 부분)

**END;**

- **Declation Section : 옵션**

변수, 상수, **CURSOR** 와  
**USER\_DEFINE Exception** 등 선언

- **Excutabl Section : 필수**

처리할 명령문들을 절차적으로 기술  
**SQL**문, 반복문, 조건문  
**BEGIN**으로 시작 **END**로 끝남

- **Exception Section : 옵션**

오류 처리에 관한 명령문을 기술

## PL/SQL의 확장 요소

| 항 목                                                   | 내 용                  |
|-------------------------------------------------------|----------------------|
| DECLARE                                               | 지역변수와 커서, 사용자 예외를 선언 |
| :=<br>SELECT INTO<br>FETCH INTO                       | 변수할당                 |
| BEGIN...END                                           | 문장의 블록               |
| --<br>/*...*/                                         | 한라인 주석<br>여러 라인 주석   |
| IF...ELSIF...END IF<br>CASE...END CASE                | 분기문                  |
| WHILE...END LOOP<br>LOOP...END LOOP<br>FOR...END LOOP | 반복문                  |
| EXIT                                                  | 반복 블록을 빠져나간다         |
| GOTO                                                  | 처리 순서 변경하기           |

## 1. 변수의 종류

- **SCALAR** 변수

데이터 하나만을 저장하는 일반적인 변수

- **REFERENCES** 변수

해당 테이블의 **row** 나 **column**의 타입과 크기를 참조하는 변수

- **COMPOSITE** 변수

**PL/SQL**에서 사용하는 배열 변수

**RECORD TYPE**

**TABLE TYPE**

- **BIND** 변수

파라미터로 넘겨지는 **IN**, **INOUT**에서 사용되는 변수  
리턴되는 값을 전달받기 위해 선언되는 변수

## 1. 변수의 데이터 타입

### Sclar 변수에 자주 사용되는 데이터 형

| 데이터 형          | 설 명                                           |
|----------------|-----------------------------------------------|
| CHAR[(N)]      | 고정 길이 문자, 사이즈를 지정하지 않으면 1                     |
| BINARY_INTEGER | -2147483647에서 2147483647 사이의 정수               |
| PLS_INTEGER    | BINARY_INTEGER와 같지만 저장공간 및 속도에서 효율적           |
| NUMBER[(P, S)] | 고정 및 부동 소숫점 수에 대한 기본 유형                       |
| VARCHAR2(N)    | 가변 길이 문자열, 32767Byte까지                        |
| DATE           | 날짜와 시간<br>범위는 BC 4712년 1월1일 - AD 9999년 12월 31 |
| BOOLEAN        | 논리연산에 사용(TRUE, FALSE, NULL)                   |
| LONG, LONG RAW | Deprecated                                    |

## 1. 변수의 데이터 타입 (cont.)

### References 변수에 사용되는 데이터 타입

| 데이터 타입        | 설 명                       |
|---------------|---------------------------|
| 테이블명.컬럼명%TYPE | 해당 테이블의 해당컬럼의 타입과 동일하게 지정 |
| 테이블명%ROWTYPE  | 해당 테이블의 모든 컬럼과 동일하게 지정    |

**%TYPE** 변수나 또는 **%ROWTYPE** 변수를 사용하여 얻을 수 있는 장점은 무엇일까요?

**References** 변수 및 **Composite** 변수의 사용법은 제어문을 하면서 실습.



## 1. 변수의 선언

- 식별자 [**CONSTANT**] 데이터타입 [**NOT NULL**] [:= 초기값];
- 초기값을 지정하고자 할 때는 할당연산자(:=)를 사용한다.
- 식별자를 상수로 지정하고 하는 경우는 **CONSTANT**라는 **KEYWORD**를 명시하고 반드시 초기값을 지정한다.
- **NOT NULL**이 정의되어 있으면 초기값을 반드시 지정 한다.
- 초기값을 정의하지 않으면 변수는 **NULL**값을 가진다.

```
v_i NUMBER(9,2) ;
v_str VARCHAR2(20) := '홍길동';
c_pi CONSTANT NUMBER(8,6) := 3.141592;
v_flag BOOLEAN NOT NULL := TRUE;
v_date VARCHAR2(10) := TO_CHAR(SYSDATE, 'YYYY-MM-DD');
```

## 2. 출력

### ● DECLARE

```
v_i NUMBER(9,2) := 0 ;
```

```
v_name VARCHAR2(20);
```

```
c_pi CONSTANT NUMBER(8,6) := 3.141592;
```

```
v_flag BOOLEAN NOT NULL := true;
```

```
v_date VARCHAR2(10) := TO_CHAR(SYSDATE, 'YYYY-MM-DD');
```

### BEGIN

```
v_name := '홍길동';
```

```
DBMS_OUTPUT.ENABLE;
```

```
DBMS_OUTPUT.PUT_LINE('v_i : ' || v_i);
```

```
DBMS_OUTPUT.PUT_LINE('v_name : ' || v_name);
```

```
DBMS_OUTPUT.PUT_LINE('c_pi : ' || c_pi);
```

```
DBMS_OUTPUT.PUT_LINE('v_date : ' || v_date);
```

```
END;
```

```
/
```

## 2. 출력(cont.)

- **SQL> SET SERVEROUTPUT ON**

- **DBMS\_OUTPUT** 결과값을 화면에 출력하기 위해 환경설정 변수 변경
- 매 세션마다 초기화 되므로 매번 재설정해야 한다.
- **DBMS\_OUTPUT**은 오라클에서 입,출력을 위해 제공하는 패키지이다.

### 3. IF 문

- 조건이 **true**이면 이하 문장을 실행하고, 조건이 **false**이면 관련된 문장을 통과한다.
- **ELSIF**절은 여러 개가 가능하나, **ELSE**절은 한 개만 가능하다

```
DECLARE
 v_num NUMBER := 37;

BEGIN
 DBMS_OUTPUT.ENABLE;

 IF MOD(v_num, 2) = 0 THEN
 DBMS_OUTPUT.PUT_LINE(v_num || ' 는 짝수');
 ELSE
 DBMS_OUTPUT.PUT_LINE(v_num || ' 는 홀수');
 END IF;
END;
/
```

### 3. IF 문

- 조건에 따른 다중 **ELSIF**

```
DECLARE
 v_num NUMBER := 77;
BEGIN
 DBMS_OUTPUT.ENABLE;
 IF v_num > 90 THEN
 DBMS_OUTPUT.PUT_LINE('수');
 ELSIF v_num > 80 THEN
 DBMS_OUTPUT.PUT_LINE('우');
 ELSIF v_num > 70 THEN
 DBMS_OUTPUT.PUT_LINE('미');
 ELSE
 DBMS_OUTPUT.PUT_LINE('분발합시다.');
```

END IF;

END;

/

### 3. IF 문

- **SELECT INTO** 로 변수에 값을 할당

- **DECLARE**

```
v_avg_sale PROD.PROD_SALE%TYPE;
v_sale NUMBER := 500000;
BEGIN
 DBMS_OUTPUT.ENABLE;

 SELECT AVG(prod_sale) INTO v_avg_sale FROM prod;

 IF v_sale < v_avg_sale THEN
 DBMS_OUTPUT.PUT_LINE('평균 단가가 500000 초과입니다.');
```

```
 ELSE
```

```
 DBMS_OUTPUT.PUT_LINE('평균 단가가 500000 이하 입니다.');
```

```
 END IF;
```

```
END;
```

```
/
```

### 3. IF 문

- 회원테이블에서 아이디가 '**e001**' 인 회원의  
마일리지가 **5000**을 넘으면 '**VIP** 회원' 그렇지 않다면 '일반회원'으로  
출력하시오. (회원이름, 마일리지 포함)

## 4. CASE 문

- SQL 에서 사용하는 **CASE** 문과 동일하다.  
단, 차이점은 **END CASE** 로 마지막을 지정해야 한다.

```
DECLARE
 v_num NUMBER := 77;
BEGIN
 v_num := TRUNC(v_num / 10);
 CASE v_num
 WHEN 10 THEN
 DBMS_OUTPUT.PUT_LINE('수');
 WHEN 9 THEN
 DBMS_OUTPUT.PUT_LINE('수');
 WHEN 8 THEN
 DBMS_OUTPUT.PUT_LINE('우');
 WHEN 7 THEN
 DBMS_OUTPUT.PUT_LINE('미');
 ELSE
 DBMS_OUTPUT.PUT_LINE('분발합시다. ');
 END CASE;
END;
/
```



## 5. WHILE 문

- 반복될 때마다 조건을 확인하고 조건이 **TRUE**가 되어야 **loop**실행
  - 조건이 만족할 때까지 반복 처리
  - **EXIT**문은 **WHILE LOOP**를 벗어나게 한다
  - **CONTINUE**문은 11g부터 제공된다.

- 1부터 10까지 더하기

```
DECLARE
```

```
 v_sum NUMBER := 0;
```

```
 v_var NUMBER := 1;
```

```
BEGIN
```

```
 WHILE v_var <= 10 LOOP
```

```
 v_sum := v_sum + v_var;
```

```
 v_var := v_var + 1;
```

```
 END LOOP;
```

```
 DBMS_OUTPUT.PUT_LINE('1 부터 10 까지의 합 = ' || v_sum);
```

```
END;
```

```
/
```

## 5. WHILE 문

- **WHILE**문을 사용하여 \* 로 피라미드 만들기

```
DECLARE
 v_id NUMBER := 1;
BEGIN
 WHILE v_id < 20 LOOP
 DBMS_OUTPUT.PUT_LINE(RPAD('*', v_id, '*'));
 v_id := v_id + 2;
 END LOOP;
END;
/
```

```
*


```

## 5. WHILE 문

- WHILE문을 사용하여 다음 형태의 피라미드 만들기

```
 *


```

## 5. WHILE 문

### ● 다중 WHILE문을 사용하여 구구단 만들기

2 \* 1 = 2  
2 \* 2 = 4  
2 \* 3 = 6  
2 \* 4 = 8  
2 \* 5 = 10  
2 \* 6 = 12  
2 \* 7 = 14  
2 \* 8 = 16  
2 \* 9 = 18  
.....  
.....  
9 \* 4 = 36  
9 \* 5 = 45  
9 \* 6 = 54  
9 \* 7 = 63  
9 \* 8 = 72  
9 \* 9 = 81

```
DECLARE
 v_dan NUMBER := 2;
 v_i NUMBER := 1;

BEGIN
 WHILE v_dan < 10 LOOP
 WHILE v_i < 10 LOOP
 DBMS_OUTPUT.PUT_LINE(
 v_dan || ' * ' || v_i || ' = ' || (v_dan * v_i));
 v_i := v_i + 1;
 END LOOP;
 v_i := 2;
 v_dan := v_dan + 1;
 END LOOP;
END;
/
```

## 5. WHILE 문

- **Oracle Server**는 **SQL**문장을 실행할 때 **PL/SQL**은 **SQL**식별자를 가지는 암시적 커서를 생성합니다. 또한 **PL/SQL**은 자동적으로 이 커서를 관리합니다.

| 커서 속성        | 내 용                             |
|--------------|---------------------------------|
| SQL%ISOPEN   | 항상 FALSE, (암시적 커서는 바로 CLOSE가 됨) |
| SQL%NOTFOUND | SQL문장이 어떠한 영향을 미치지 않았다면 TRUE    |
| SQL%FOUND    | SQL문장이 하나 이상의 영향을 미쳤다면 TRUE     |
| SQL%ROWCOUNT | SQL 문장에 의해 영향을 받은 행의 수          |

```

DECLARE
 v_nm VARCHAR2(20);
BEGIN
 SELECT lprod_nm INTO v_nm FROM lprod WHERE lprod_gu = 'P201' ;
 IF SQL%FOUND THEN
 DBMS_OUTPUT.PUT_LINE('받은 값 =' || v_nm);
 DBMS_OUTPUT.PUT_LINE('행 수 =' || SQL%ROWCOUNT);
 END IF;
END;
```

## 5. WHILE 문

- 상품분류 테이블에 6개의 코드 증가

**DECLARE**

**v\_add** **NUMBER**(5) **:=** 1000;

**v\_code** **CHAR**(4) **:=** '';

**v\_id** **NUMBER**(5);

**BEGIN**

**SELECT** **MAX**(lprod\_id) **INTO** v\_id **FROM** lprod;

**WHILE** v\_add <= 1005 **LOOP**

**v\_add** **:=** v\_add + 1;

**v\_id** **:=** v\_id + 1;

**v\_code** **:=** 'TT' || **SUBSTR**(**TO\_CHAR**(v\_add),-2);

**INSERT** **INTO** lprod ( lprod\_id, lprod\_gu, lprod\_nm )  
**VALUES** ( v\_id , v\_code , 'LOOP TEST' );

**IF** **SQL%FOUND** **THEN**

**DBMS\_OUTPUT.PUT\_LINE** ('새 코드 ' || v\_code || '가 추가되었습니다');

**END IF**;

**END LOOP**;

**END**;

## 5. WHILE 문

-**WHILE**문을 사용하여 상기 **INSERT**된 데이터를 삭제하시오 ?  
(삭제가 되었는지 확인 메시지 출력)

## 6. GOTO

- 실행처리를 임의의 지점으로 이동

```
DECLARE
 v_sum INT := 0;
 v_var INT := 1;
BEGIN
 <<mylabel>>
 v_sum := v_sum + v_var;
 v_var := v_var + 1;
 IF v_var <= 10 THEN
 GOTO mylabel;
 END IF;
 DBMS_OUTPUT.PUT_LINE(v_sum);
 DBMS_OUTPUT.PUT_LINE(v_var);
END;
/
```



## 7. LOOP 문

- 조건이 없는 단순한 무한 반복문이다.
  - **EXIT** 문을 사용하여 반복문을 빠져나가게 해야 한다.

- **1부터 10까지 더하기**

**DECLARE**

**v\_sum NUMBER := 0;**

**v\_var NUMBER := 1;**

**BEGIN**

**LOOP**

**v\_sum := v\_sum + v\_var;**

**v\_var := v\_var + 1;**

**IF v\_var > 10 THEN**

**EXIT;**

**END IF;**

**END LOOP;**

**DBMS\_OUTPUT.PUT\_LINE('1 부터 10 까지의 합 = ' || v\_sum);**

**END;**

**/**

## 8. EXIT

- **EXIT [label] [WHEN 조건]**

- 반복문을 빠져 나간다.
- **WHEN** 을 사용하여 조건에 따라서 빠져나간다.

- **EXIT WHEN** 을 사용해 1부터 10까지 더하기

**DECLARE**

**v\_sum NUMBER := 0;**

**v\_var NUMBER := 1;**

**BEGIN**

**LOOP**

**v\_sum := v\_sum + v\_var;**

**v\_var := v\_var + 1;**

**EXIT WHEN v\_var > 10 ;**

**END LOOP;**

**DBMS\_OUTPUT.PUT\_LINE('1 부터 10 까지의 합 = ' || v\_sum);**

**END;**

**/**

## 9. FOR 문

### ● FOR index IN [REVERSE] 최소값 .. 최대값 LOOP

처리문장들;

END LOOP;

- **index**는 1씩 증가하는 자동선언 정수형 변수(최소, 최대값)
- **REVERSE** 가 사용될 경우 1씩 감소
- **IN** 다음에 **SELECT** 문, **CURSOR** 문이 올 수 있다.
- **SELECT, CURSOR** 를 사용한 경우 **index**는 레코드타입 변수

```
BEGIN
 FOR i IN 1..10 LOOP
 DBMS_OUTPUT.PUT_LINE('i = ' || i);
 END LOOP;
END;
/
```

## 10. COMPOSITE 데이터 타입

- 컬렉션 : 배열형태의 타입
  - **VARRAY** : 고정길이를 가진 배열, 첨자로 접근
  - **TABLE** : 가변길이를 가진 배열, 첨자 또는 키로 접근
  
- 컬렉션 타입은 사용자를 위해 다음과 같은 메서드를 제공  
**COUNT, DELETE, EXISTS, EXTEND, FIRST, LAST, LIMIT, NEXT, PRIOR, TRIM.**
  
- 레코드 : 테이블 형태의 타입
  - **RECORD** : 여러 데이터 형이 조합된 구조체형식
  - **%ROWTYPE** : 기존 테이블의 이름과 타입을 사용

## 10. COMPOSITE 데이터 타입

### ● VARRAY의 사용

**TYPE** 타입명 **IS** {**VARRAY** | **VARYING ARRAY**} (사이즈) **OF**  
데이터 타입 [**NOT NULL**];

**DECLARE**

**TYPE** starcraft **IS** **VARRAY**(20) **OF** **VARCHAR2**(10); -- Type 선언

**v\_star** starcraft;

**BEGIN**

**v\_star** := starcraft('Terran','Protos');

**v\_star.EXTEND**;

**v\_star(3)** := 'Zerg';

**DBMS\_OUTPUT.PUT\_LINE**('스타크래프트 종족 : ' || **v\_star.COUNT**);

**FOR** **i** **IN** **v\_star.FIRST**..**v\_star.LAST** **LOOP**

**DBMS\_OUTPUT.PUT\_LINE** ( **i** || '번째 종족 : ' || **v\_star(i)**);

**END LOOP**;

**END**;

## 10. COMPOSITE 데이터 타입

### ● TABLE 의 사용

SQL의 테이블이 아니며 배열에 가깝다.

가변적이므로 사이즈를 지정하지 않는다.

#### - Nested table (첨자 기반)

TYPE 타입명 IS TABLE OF 데이터 타입 [NOT NULL];

#### - Associative array(index-by table, 키 기반)

TYPE 타입명 IS TABLE OF 데이터 타입 [NOT NULL]

INDEX BY [PLS\_INTEGER | BINARY\_INTEGER | VARCHAR2(n)];

## 10. COMPOSITE 데이터 타입

- **TABLE Type**을 사용하여 분류테이블정보 담기

DECLARE

```
TYPE lprod_nm_table IS TABLE OF VARCHAR2(40)
INDEX BY PLS_INTEGER;
```

```
t_lprod_nm lprod_nm_table;
```

BEGIN

```
FOR l_list IN (SELECT lprod_id, lprod_nm FROM lprod) LOOP
 t_lprod_nm(l_list.lprod_id) := l_list.lprod_nm;
 DBMS_OUTPUT.PUT_LINE (l_list.lprod_id || ' = ' || l_list.lprod_nm);
END LOOP;
```

```
DBMS_OUTPUT.PUT_LINE ('갯수 = ' || t_lprod_nm.COUNT);
```

```
FOR i IN t_lprod_nm.FIRST..t_lprod_nm.LAST LOOP
```

```
 IF t_lprod_nm.EXISTS(i) THEN
```

```
 DBMS_OUTPUT.PUT_LINE (i || ' ' || t_lprod_nm(i));
```

```
 END IF;
```

```
END LOOP;
```

```
END;
```

## 11. EXCEPTION 처리

- **PL/SQL**에서 **Error**가 발생하면 **Error**는 **Exception**을 발생시켜 해당 블록을 중지하고 예외처리부분으로 이동한다.
- 예외는 **Oracle**내부에서 자동으로 발생 할 수 있으며 또한 사용자에게 의해 직접 발생시킬 수 있다.
- 예외 유형
  1. 정의된 **Oracle Server Error**
    - **PL/SQL**에서 자주 발생하는 **Error**를 미리 정의함
    - 선언할 필요가 없으며 서버에서 암시적으로 발생
  2. 정의 되지 않은 **Oracle Server Error**
    - 기타 표준 **Error**
    - 선언을 해야 하며 서버에서 암시적으로 발생
  3. 사용자 정의 **Error**
    - 프로그래머가 정한 조건에 만족하지 않을 경우 발생
    - 선언을 해야 하고, 명시적으로 **RAISE**문을 사용하여 발생



## 11. EXCEPTION 처리

### ● Exception 구문

#### EXCEPTION

WHEN *exception1* [OR *exception2* ...] THEN

*statement1*;

*statement2*;...

[ WHEN *exception2* [OR *exception4*...] THEN

*statement1*;

*statement2*;... ]

[ WHEN OTHERS THEN

*statement1*;

*statement2*;... ]

| 항 목                | 내 용                         |
|--------------------|-----------------------------|
| <i>exception n</i> | 미리 정의된 예외 또는 선언부에 선언된 예외 이름 |
| <i>statement n</i> | PL/SQL 또는 SQL 구문            |
| WHEN OTHERS        | 명시적으로 선언되지 않은 모든 예외를 처리     |

## 11. EXCEPTION 처리

| 정의된 예외 종류        | 내 용                             |
|------------------|---------------------------------|
| NO_DATA_FOUND    | 선택된 행이 없는 경우 (SELECT.. INTO)    |
| TOO_MANY_ROWS    | 여러 행이 리턴 되는 경우 (SELECT.. INTO)  |
| DUP_VAL_ON_INDEX | UNIQUE인덱스가 걸린 컬럼에 중복 데이터를 입력할 때 |
| VALUE_ERROR      | 값을 할당하거나 변환할 때 오류가 나는 경우        |
| INVALID_NUMBER   | 숫자로 변환이 되지 않는 경우(문자형 데이터)       |
| NOT_LOGGED_ON    | DB에 접속하지 않은 채 실행하는 경우           |
| LOGIN_DENIED     | 잘못된 사용자나 비밀번호로 로그인을 시도할 때       |
| ZERO_DIVIDE      | 0으로 나누려고 할 때                    |
| INVALID_CURSOR   | 허용되지 않은(열리지 않은) 커서에 접근하는 경우     |

## 11. EXCEPTION 처리

- 미리 정의된 예외인 경우

**DECLARE**

**v\_name varchar2(20);**

**BEGIN**

**SELECT lprod\_nm INTO v\_name FROM lprod WHERE lprod\_gu = 'P201';**

**DBMS\_OUTPUT.PUT\_LINE ('상품명 = ' || v\_name);**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**DBMS\_OUTPUT.PUT\_LINE ( '해당 정보가 없습니다. ');**

**WHEN TOO\_MANY\_ROWS THEN**

**DBMS\_OUTPUT.PUT\_LINE ( '한개 이상의 값이 나왔습니다. ');**

**WHEN OTHERS THEN**

**DBMS\_OUTPUT.PUT\_LINE ( '기타 에러 : ' || SQLERRM );**

**END;**

## 11. EXCEPTION 처리

### ● 정의되지 않은 예외인 경우

- 선언부에 예외의 이름을 지정한다.
- **PRAGMA** 를 기술하고 **EXCEPTION\_INIT** 으로 예외이름과 에러번호를 컴파일러에게 등록한다.  
(**Pragma** 는 실행될 때 처리되지 않는 명령문임을 알려주는 예약어)
- **EXCEPTION** 영역에 해당 예외 처리

```
DECLARE
 exp_reference EXCEPTION;
 PRAGMA EXCEPTION_INIT(exp_reference, -2292);
BEGIN
 DELETE FROM lprod WHERE lprod_gu = 'P101';
 DBMS_OUTPUT.PUT_LINE ('불가 삭제');
EXCEPTION
 WHEN exp_reference THEN
 DBMS_OUTPUT.PUT_LINE ('삭제 불가 :' || SQLERRM);
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE (SQLCODE || ' ' || SQLERRM);
END;
```

## 11. EXCEPTION 처리

### ● 사용자 정의 예외인 경우

- 선언부에 예외의 이름을 지정한다.
- 실행부에서 **RAISE**문장으로 명시적으로 발생
- **EXCEPTION** 영역에 해당 예외 처리

```
ACCEPT p_lgu PROMPT '등록하려는 분류코드 입력 :'
DECLARE
 exp_lprod_gu EXCEPTION;
 v_lgu VARCHAR2(10) := UPPER('&p_lgu');
BEGIN
 IF v_lgu IN ('P101','P102','P201','P202') THEN
 RAISE exp_lprod_gu;
 END IF;
 DBMS_OUTPUT.PUT_LINE (v_lgu || '는 등록 가능');
EXCEPTION
 WHEN exp_lprod_gu THEN
 DBMS_OUTPUT.PUT_LINE (v_lgu || '는 이미 등록된 코드입니다. ');
END;
```

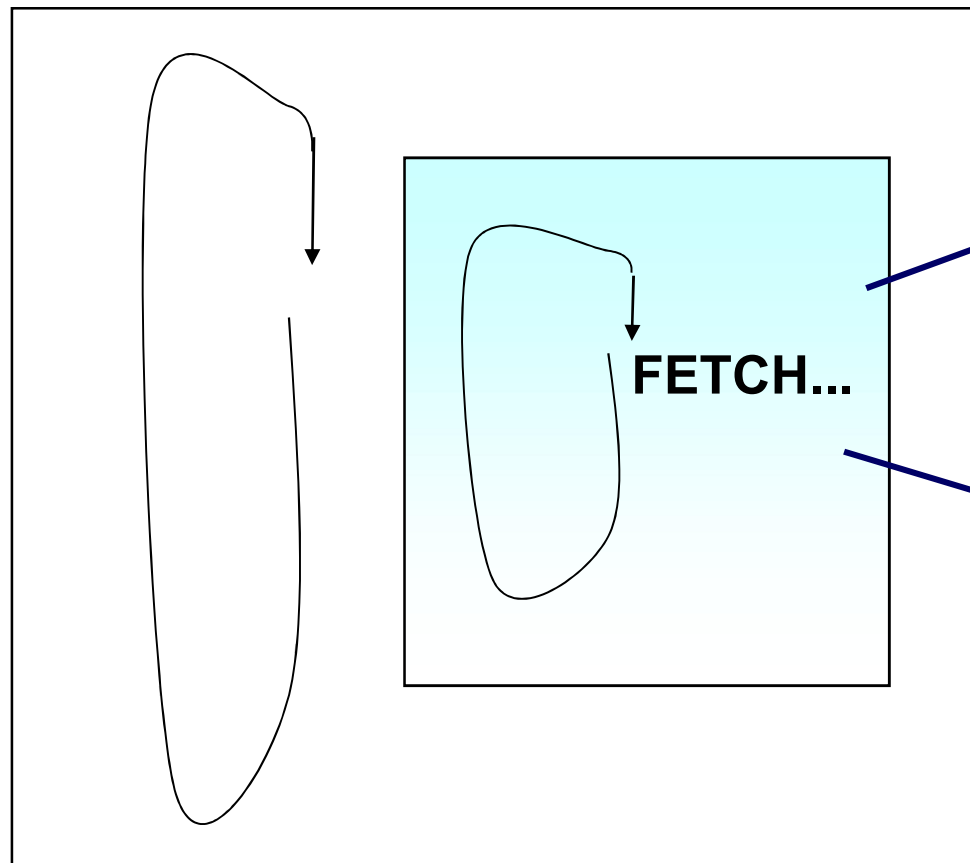
## CURSOR 문

- **SELECT** 문에서 생성된 결과 집합에 대해 개별적인 행 단위 작업을 가능하게 한다.
  - **Query**결과를 읽거나 수정, 삭제할 수 있도록 해주는 개념
  - **SELECT**문의 **Query**결과를 먼저 정의한 후 이를 바탕으로 첫레코드 부터 마지막 레코드까지 액세스
  - 선택된 행들은 서버상에서 개별적으로 처리된다
  - 개발자가 **PL/SQL** 블록에서 수동으로 제어할 수 있다.

## CURSOR 문

**DECLARE... CURSOR...**

**OPEN**



| ID   |
|------|
| 성명   |
| 전화   |
| 우편번호 |
| 주소   |
| 취미   |
| 직업   |
| 생일   |
| 성별   |
| 가입일  |

.  
. .  
. .  
. .

**CLOSE**

## CURSOR 문

| 항 목                       | 내 용                                                                                                                                                                           |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>DECLARE<br/>CURSOR</b> | <ul style="list-style-type: none"> <li>- 선언부에 커서를 선언한다.</li> </ul>                                                                                                            |
| <b>OPEN</b>               | <ul style="list-style-type: none"> <li>- 커서에 대한 메모리 할당 및 정보를 저장.</li> <li>- 입력변수에 대한 바인드작업.</li> </ul>                                                                        |
| <b>FETCH</b>              | <ul style="list-style-type: none"> <li>- 현재 행을 읽어 들인 후 변수에 저장</li> <li>- 커서 속성을 이용하여 행이 존재하는지 검사</li> <li>- 처리할 행이 있다면 <b>FETCH</b> 반복</li> </ul>                             |
| <b>CLOSE</b>              | <ul style="list-style-type: none"> <li>- 사용을 마친 커서는 반드시 닫아야 한다.</li> <li>- 커서를 닫고 버퍼용으로 사용중인 메모리를 반환</li> <li>- 커서를 다시 열수는 있지만 데이터를 <b>FETCH</b>하거나 갱신, 삭제할 수는 없다.</li> </ul> |



## CURSOR 문

### ● CURSOR 실행

선언부 에 커서 선언

```
CURSOR cursor_name IS
 select_statement;
```

실행 영역에서

```
OPEN cursor_name;
```

```
FETCH cursor_name INTO {variable_lists | record_name } ;
```

```
CLOSE cursor_name;
```

| 커서 속성     | 내용                           |
|-----------|------------------------------|
| %ISOPEN   | 커서가 열린 상태이면 TRUE             |
| %NOTFOUND | SQL문장이 어떠한 영향을 미치지 않았다면 TRUE |
| %FOUND    | SQL문장이 하나 이상의 영향을 미쳤다면 TRUE  |
| %ROWCOUNT | SQL 문장에 의해 영향을 받은 행의 수       |

## CURSOR 문

- 2005년도 및 상품별 총 입고수량을 출력하는 커서

DECLARE

v\_prod VARCHAR2(30);

v\_qty NUMBER(10,0);

CURSOR UpRemain\_cur IS

SELECT buy\_prod, SUM(buy\_qty) FROM buyprod  
WHERE EXTRACT(YEAR FROM buy\_date) = 2005  
GROUP BY buy\_prod ORDER BY buy\_prod ASC;

BEGIN

OPEN UpRemain\_cur;

FETCH UpRemain\_cur INTO v\_prod, v\_qty;

WHILE (UpRemain\_cur%FOUND) LOOP

DBMS\_OUTPUT.PUT\_LINE( UpRemain\_cur%ROWCOUNT || '번째 상품='  
|| v\_prod || ' 입고수량=' || v\_qty || '입니다.');

FETCH UpRemain\_cur INTO v\_prod, v\_qty;

END LOOP;

CLOSE UpRemain\_cur;

END;

## CURSOR 문

- 직업을 변수로 받아 이름, 회원명과 마일리지 출력하는 커서

DECLARE

v\_name VARCHAR2(30);

v\_mileage NUMBER(10);

CURSOR member\_cur ( v\_job VARCHAR2) IS

SELECT mem\_name, mem\_mileage FROM member

WHERE mem\_job = v\_job

ORDER BY mem\_name ASC;

BEGIN

OPEN member\_cur('주부');

LOOP

FETCH member\_cur INTO v\_name, v\_mileage;

EXIT WHEN member\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE( member\_cur%ROWCOUNT || '번째 ' || v\_name || ', ' || v\_mileage );

END LOOP;

CLOSE member\_cur;

END;

## CURSOR 문

### ● FOR LOOP를 이용하는 CURSOR

- 각 반복하는 동안 커서를 자동으로 **OPEN**하고 모든 행이 처리되면 자동으로 커서를 **CLOSE** 한다.

```
FOR record_name IN cursor_name LOOP
 statement;
 ...
END LOOP;
```

- *record\_name* 은 묵시적으로 자동 생성되므로 선언하지 않는다.

### ● Subquery를 이용한 FOR LOOP

- 서브쿼리를 사용하여 커서를 선언하지 않아도 동일한 결과

```
FOR record_name IN (subquery) LOOP
```

## CURSOR 문

- 직업을 입력받아서 **FOR LOOP**를 이용하는 **CURSOR**

```
ACCEPT p_job PROMPT '직업을 입력하세요 :'
```

```
DECLARE
```

```
 v_name VARCHAR2(30);
```

```
 v_mileage NUMBER(10);
```

```
 CURSOR member_cur IS
```

```
 SELECT mem_name, mem_mileage
 FROM member
```

```
 WHERE mem_job = '&p_job'
```

```
 ORDER BY mem_name ASC;
```

```
BEGIN
```

```
 FOR mem_rec IN member_cur LOOP
```

```
 DBMS_OUTPUT.PUT_LINE(member_cur%ROWCOUNT || '번째 '
 || mem_rec.mem_name || ', ' || mem_rec.mem_mileage);
```

```
 END LOOP;
```

```
END;
```

## CURSOR 문

- Subquery를 이용한 FOR LOOP

```
BEGIN
 FOR mem_rec IN (SELECT mem_name, mem_mileage
 FROM member ORDER BY mem_name ASC) LOOP
 DBMS_OUTPUT.PUT_LINE(mem_rec.mem_id || ', '
 || mem_rec.mem_name || ', '
 || mem_rec.mem_mileage);
 END LOOP;
END;
```

- 위 예시에서 마일리지가 3000이상인 사람만 출력하게 하시오.

## Stored Procedure

### ● 서버에 저장된 미리 컴파일 된 **SQL**문장들

- 저장 프로시저를 처음 수행될 때 문법을 검사하고 컴파일 된다.  
컴파일된 버전은 프로시저 캐시에 저장되므로 이후에 호출될 때 빠르게 수행될 수 있다.
- 클라이언트간 처리 루틴 공유  
모든 응용 프로그램에서 사용할 수 있도록 기능을 캡슐화 하므로 일관성 있는 데이터 변경을 보장한다.
- 데이터베이스 내부 구조 보안 : **VIEW**와 동일한 개념
- 서버 보호, 자료 무결성(**Integrity**)권한 구현  
**VIEW**사용 시 **Table**을 직접 **Access**하는 것이 아니라 **View**에 대한 권한만 주어 **Table**의 특정 **Column**과 **Record**만 **Access**하도록 제한할 수 있다.  
저장함수도 이와 같은 개념으로 서버 데이터를 보호하는데 사용 가능

## Stored Procedure

- 서버에 저장된 미리 컴파일 된 **SQL**문장들

- **Query**처리 속도 향상

저장함수 안에 지정된 모든 **SQL**구문을 하나의 **Batch**로 인식하여 한꺼번에 분석, 최적화 시키고 실행.

각각의 **SQL**문을 **Client**로부터 받아서 매번 분석, 최적화, 실행과정을 반복하는 것에 비해 처리속도가 현저히 향상된다

- **Network Traffic** 감소

**Client**에서 **Server**로 보내야 할 **SQL**구문을 **Server**가 미리 저장.

고로 **Client**에서 대량의 **SQL**구문을 보내는 대신에

**Stored Procedure**의 이름과 매개변수(**Parameter**)만 보내면 되므로 **Network Traffic** 이 그만큼 줄어드는 효과



## Stored Procedure

### ● Procedure 구문

```
CREATE [or REPLACE] PROCEDURE procedure_name
 [(argument [mode] [{:= | DEFAULT} expression]
 , ...)]
{ IS | AS }
BEGIN
 pl/sql_block;
END;
```

| 항 목                   | 내 용                                     |
|-----------------------|-----------------------------------------|
| or REPLACE            | 프로시저가 생성된 경우 다시 생성                      |
| <i>procedure_name</i> | 프로시저 개체명                                |
| <i>argument</i>       | 매개변수 이름                                 |
| mode                  | IN : 입력 전용, OUT : 출력 전용, IN OUT : 입, 출력 |
| <i>expression</i>     | 매개변수의 값이 없을 때 기본값 설정                    |
| <i>pl/sql_block</i>   | PL/SQL 블록                               |

## Stored Procedure

- 상품코드를 매개변수(**parameter**)로 하여 재고수량 **ADD**

```
CREATE OR REPLACE PROCEDURE usp_prod_totalstock_update
(v_prod_id IN prod.prod_id%TYPE,
 v_qty IN prod.prod_totalstock%TYPE)
IS
BEGIN
 UPDATE prod
 SET prod_totalstock = prod_totalstock + v_qty
 WHERE prod_id = v_prod_id;
 DBMS_OUTPUT.PUT_LINE('정상적으로 업데이트 되었습니다. ');
 COMMIT;

EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('예외 발생: ' || SQLERRM);
 ROLLBACK;
END;
```

## Stored Procedure

- **Procedure 실행**

**EXEC 또는 EXECUTE *procedure\_name* ( 매개변수, ...);**

- **SELECT prod\_id, prod\_totalstock  
FROM prod  
WHERE prod\_id = 'P102000006';**

- **EXECUTE usp\_prod\_totalstock\_update('P102000006', 500);**

## Stored Procedure

- **OUT 매개변수 예제 1**
  - 회원아이디를 입력받아 이름과 취미를 **OUT 매개변수**로 처리

```
CREATE OR REPLACE PROCEDURE usp_MemberID
(p_mem_id IN member.mem_id%TYPE,
 p_mem_name OUT member.mem_name%TYPE ,
 p_mem_like OUT member.mem_like%TYPE)
IS
BEGIN
 SELECT mem_name, mem_like
 INTO p_mem_name, p_mem_like
 FROM member
 WHERE mem_id = p_mem_id;
END;
```

## Stored Procedure

- OUT 매개변수의 간단 출력

**VAR** 문을 통하여 변수선언

변수를 사용할때는 ":" 을 사용

**PRINT** 문으로 변수의 값을 출력

- OUT 매개변수 예제 1 (cont.)

- 실행

```
SQL> VAR mem_name VARCHAR2(20)
```

```
SQL> VAR mem_like VARCHAR2(20)
```

```
SQL> EXECUTE usp_MemberID ('a001', :mem_name, :mem_like);
```

```
SQL> PRINT mem_name
```

```
SQL> PRINT mem_like
```

## Stored Procedure

### ● OUT 매개변수 예제 2

```
CREATE OR REPLACE PROCEDURE usp_MemberCartTop
(p_year IN VARCHAR2,
 p_amt OUT NUMBER ,
 p_mem_name OUT member.mem_name%TYPE)
IS
 v_year VARCHAR2(5);
BEGIN
 v_year := (p_year || '%');
 SELECT mem_name, mem_amt INTO p_mem_name, p_amt
 FROM (
 SELECT mem_name, SUM(prod_price * cart_qty) mem_amt
 FROM member, cart, prod
 WHERE cart_no LIKE v_year
 AND cart_member = mem_id
 AND cart_prod = prod_id
 GROUP BY mem_name
 ORDER BY SUM(prod_price * cart_qty) DESC
)
 WHERE ROWNUM <= 1 ;
END;
```

## Stored Procedure

- OUT 매개변수 예제 2 (cont.)

- 실행

```
SQL> VAR send_member VARCHAR2
```

```
SQL> VAR send_amt NUMBER
```

```
SQL> EXEC usp_MemberCartTop('2005', :send_amt, :send_member);
```

```
SQL> PRINT send_member
```

```
SQL> PRINT send_amt
```

## Stored Procedure

- 상품 코드와 월을 입력하면 해당 월에 대한 해당 상품의 입고, 출고를 처리해 화면에 출력하시오. (프로시저명 : **usp\_prod\_info**, 월 입력형식은 'YYYYMM' 이라 가정, 입고 및 출고는 **OUT** 매개변수로 처리.)



## User Defined Function

- **Function**은 **Procedure**가 갖는 장점은 동일하다.

- 반환값이 있다.  
즉, 일반 오라클 내장함수처럼 사용할 수 있다는 것이다.  
자주 반복되는 **subquery**, 복잡한 계산식을 사용자가 만들어서 일반 함수처럼 사용할 수 있다.
- 반환할 데이터 타입을 **RETURN** 으로 선언해야 한다.
- 실행영역에서 **RETURN** 문이 있어야 한다.

- **SELECT** cart\_no, cart\_prod, cart\_member,  
    (SELECT mem\_name FROM member WHERE mem\_id = cart\_member)  
    FROM cart  
    WHERE cart\_no = '2005040100001'

## User Defined Function

### ● Function 구문

```

CREATE [or REPLACE] FUNCTION function_name
 [(argument [mode] [{:= | DEFAULT} expression]
 , ...)]
RETURN data_type
{ IS | AS }
BEGIN
 pl/sql_block;
END;

```

| 항 목                  | 내 용                           |
|----------------------|-------------------------------|
| or REPLACE           | Function이 생성된 경우 다시 생성        |
| <i>function_name</i> | Function 이름                   |
| <i>argument</i>      | 매개변수 이름                       |
| mode                 | IN , OUT, IN OUT 이 있으며 기본은 IN |
| <i>data_type</i>     | 반환되는 값의 datatype              |
| <i>pl/sql_block</i>  | PL/SQL 블록                     |

## User Defined Function

- 회원 아이디를 받으면 해당 이름을 리턴하는 함수 만들기

```
CREATE OR REPLACE FUNCTION fn_memName
 (p_mem_id IN VARCHAR2)
 RETURN VARCHAR2
IS
 r_name VARCHAR2(30);
BEGIN
 SELECT mem_name INTO r_name FROM member
 WHERE mem_id = p_mem_id;
 RETURN r_name;

EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('예외 발생:' || SQLERRM);
 RETURN null;
END;
```

## User Defined Function

- fn\_memName 실행 테스트

```
SQL> VAR m_name VARCHAR2
```

```
SQL> EXECUTE :m_name := fn_memName('a001') ;
```

```
SQL> PRINT m_name
```

- 실제 함수처럼 SQL 구문에서 실행

```
SELECT cart_no, cart_prod, cart_member, fn_memName(cart_member)
FROM cart
```

```
WHERE cart_no = '2005040100001'
```

## User Defined Function

- 년도 및 상품코드를 입력 받으면 해당년도의 평균 판매 회수를 반환하는 함수

```
CREATE OR REPLACE FUNCTION fn_prodAvgQty
 (p_year IN NUMBER DEFAULT (EXTRACT(YEAR FROM SYSDATE)),
 p_prod_id IN VARCHAR2)
RETURN NUMBER
IS
 r_qty NUMBER(10);
 v_year VARCHAR2(5) := TO_CHAR(p_year) || '%';
BEGIN
 SELECT NVL(AVG(cart_qty),0) INTO r_qty FROM cart
 WHERE cart_prod = p_prod_id AND cart_no like v_year;
 RETURN r_qty;
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE('예외 발생:' || SQLERRM);
 RETURN 0;
END;
```

## User Defined Function

- **fn\_prodAvgQty 실행 테스트**

```
SQL> VAR qty NUMBER
```

```
SQL> EXEC :qty := fn_prodAvgQty(p_prod_id => 'P101000002');
```

```
SQL> PRINT qty
```

```
SQL> EXEC :qty := fn_prodAvgQty(2005, 'P101000002');
```

```
SQL> PRINT qty
```

- **실제 함수처럼 SQL 구문에서 실행**

```
SELECT prod_id, prod_name,
 fn_prodAvgQty(2004,prod_id) "2004년 평균 판매회수",
 fn_prodAvgQty(2005,prod_id) "2005년 평균 판매회수"
FROM prod
```

## Trigger

- **Oracle**에서는 절차적 무결성 제약을 강화하는 트리거(**Trigger**)라는 메커니즘을 제공
  - 자기가 종속된 특정 테이블에서 데이터의 변화가 발생했을 때 자동으로 **INSERT, UPDATE, DELETE**문이 자동으로 실행되는 아주 특별한 종류의 **Stored Procedure**이며 하나의 테이블과 연관되어서 만들어 진다
  - 주로 데이터 무결성(**Data integrity**)과 빠른 수행 성능을 보장하기 위해 사용되어 진다
- 주의 사항
  - **Transaction** 제어 문장(**COMMIT, ROLLBACK, SAVEPOINT**)을 사용할 수 없다.
  - **Trigger**에서 내부에서 사용되는 **Procedure, Function**은 트랜잭션 제어문장을 사용할 수 없다.
  - **LONG, LONG RAW** 변수를 선언할 수 없다.

## Trigger

### ● Trigger 주요 구문

```
CREATE [or REPLACE] TRIGGER trigger_name
 {BEFORE | AFTER } trigger_event [OF column1,...]
 ON table_name
 [FOR EACH ROW] [WHEN trigger_condition]
BEGIN
 pl/sql_block;
END;
```

| 항 목                          | 내 용                                  |
|------------------------------|--------------------------------------|
| <i>BEFORE</i>   <i>AFTER</i> | DML 문장이 실행되기 전, 후에 실행여부              |
| <i>trigger_event</i>         | INSERT, UPDATE, DELETE 이벤트 구분, 복수 가능 |
| OF <i>column1</i> ,...       | 해당 컬럼에 대한 이벤트 구분(UPDATE 에서)          |
| <i>table_name</i>            | 트리거의 대상이 되는 테이블                      |
| FOR EACH ROW                 | 행 기반의 트리거를 생성                        |
| <i>trigger_condition</i>     | 어떤 조건에 맞을 때 트리거 동작                   |



## Trigger

| 의사 레코드                                 | 내 용                                                                                 |
|----------------------------------------|-------------------------------------------------------------------------------------|
| <b>:NEW</b>                            | INSERT, UPDATE 에서 사용되며,<br>데이터가 삽입(갱신)될 때 들어온 새로운 값이다.<br>DELETE 시에는 모든 필드는 NULL이다. |
| <b>:OLD</b>                            | DELETE, UPDATE 에서 사용되며,<br>데이터가 삭제(갱신)될 때 이전의 값이다.<br>INSERT 시에는 모든 필드는 NULL이다.     |
| <b>:NEW, :OLD는 행 단위 TRIGGER에서 사용가능</b> |                                                                                     |

### ● 문장내에서 **DML**을 구분하기 위한 함수

| 함 수              | 내 용                        |
|------------------|----------------------------|
| <b>INSERTING</b> | 트리거된 문장이 INSERT 이면 TRUE 리턴 |
| <b>UPDATING</b>  | 트리거된 문장이 UPDATE 이면 TRUE 리턴 |
| <b>DELETING</b>  | 트리거된 문장이 DELETE 이면 TRUE 리턴 |

## Trigger

- 분류테이블에 추가되거나, 변경될 때 분류코드를 항상 대문자로 처리하는 트리거 예제

```
CREATE or REPLACE TRIGGER tg_lprod_upper
 BEFORE INSERT or UPDATE
 ON lprod
 FOR EACH ROW
BEGIN
 :NEW.lprod_gu := UPPER(:NEW.lprod_gu);
END;
/
```

- SELECT \* FROM lprod
- INSERT INTO lprod  
VALUES (lprod\_seq.NEXTVAL, 'tt07', '트리거 테스트');

## Trigger

- 장바구니 테이블에 입력이 발생할 때 재고 수불 테이블에 출고, 현재고를 변경하는 트리거 예제(2005년으로 예제 작성)

```
CREATE or REPLACE TRIGGER tg_cart_qty_change
 AFTER insert or update or delete ON cart
 FOR EACH ROW
 DECLARE
 v_qty NUMBER;
 v_prod VARCHAR2(20);
 BEGIN
 IF INSERTING THEN
 v_qty := NVL(:NEW.cart_qty,0);
 v_prod := :NEW.cart_prod;
 ELSIF UPDATING THEN
 v_qty := NVL(:NEW.cart_qty,0) - NVL(:OLD.cart_qty,0);
 v_prod := :NEW.cart_prod;
 ELSIF DELETING THEN
 v_qty := -(NVL(:OLD.cart_qty,0));
 v_prod := :OLD.cart_prod;
 END IF;
```

## Trigger

- 장바구니 테이블에 입력이 발생할 때 재고 수불 테이블에 출고, 현재고를 변경하는 트리거 예제 (2005년<sup>1월</sup> 예제 작성) (Cont.)

```
UPDATE remain SET remain_o = remain_o + v_qty,
 remain_j_99 = remain_j_99 - v_qty
WHERE remain_year = '2005' AND remain_prod = v_prod;
```

```
DBMS_OUTPUT.PUT_LINE('수량 :' || v_qty);
```

```
EXCEPTION
```

```
WHEN OTHERS THEN
```

```
 DBMS_OUTPUT.PUT_LINE('예외 발생:' || SQLERRM);
```

```
END;
```

```
/
```

```
●INSERT INTO cart VALUES ('a001','2005040100001','P101000002',7);
```

```
●SELECT * FROM remain
```

```
 WHERE remain_year='2005' AND remain_prod = 'P101000002';
```

## Trigger

- 위 작성된 트리거를 해당 회원의 마일리지도 변경되는 트리거로 변경하시오 ?  
단, 마일리지는 (수량 \* 상품판매가) 의 1%로 한다.

## Package

- **Package**는 꾸러미, 작은 짐, 포장한 상품 이라는 의미
- **업무적으로 관련 있는 것을 하나로 묶어서 사용한다.**
- **Package**는 여러 변수, 커서, 함수, 프로시저, 예외를 묶어 캡슐화 한다.
- **Package의 장점**
  - **Modularity, Easier Application Design, Information Hiding**  
모듈화를 함으로써 좀더 명확하게 정의되기 때문에 어플리케이션 개발이 용이하게 한다.
  - **Added Functionality**  
전역적인 변수, 커서 등을 서브프로그램에서 공유하여 사용가능
  - **Better Performance**  
관련있는 서브프로그램의 집합이므로 **disk I/O** 현상이 줄어든다.

## Package

- **Package** 는 선언부와 본문 두 부분으로 나눈다.
- **Package** 선언부는 패키지에 포함될 변수, 프로시저, 함수등을 선언

- **Package** 선언부 주요 구문

```
CREATE [or REPLACE] PACKAGE package_name
{ IS | AS }
 pl/sql_package_spec
END package_name;
```

| 항 목                        | 내 용                                    |
|----------------------------|----------------------------------------|
| or REPLACE                 | 패키지가 있는 경우 변경                          |
| <i>package_name</i>        | 패키지 명                                  |
| <i>pl/sql_package_spec</i> | 변수 선언, 커서 선언, 예외 선언,<br>프로시저 선언, 함수 선언 |

## Package

- **Package** 본문은 선언부에 정의된 부분의 실제 내용을 기술

- **Package** 본문 주요 구문

```
CREATE [or REPLACE] PACKAGE BODY package_name
{ IS | AS }
 pl/sql_package_body
END package_name;
```

| 항 목                        | 내 용                  |
|----------------------------|----------------------|
| or REPLACE                 | 패키지가 있는 경우 변경        |
| <i>package_name</i>        | 패키지 명                |
| <i>pl/sql_package_body</i> | 선언된 실제 프로시저, 펑션 등 기술 |



## Package

- 상품 목록과 관련한 간단한 패키지 예제
- **prod\_mgr** 패키지의 선언부

```
CREATE OR REPLACE PACKAGE prod_mgr
IS
 g_prod_lgu prod.prod_lgu%TYPE;
 PROCEDURE prod_list;
 PROCEDURE prod_list (p_prod_lgu IN prod.prod_lgu%TYPE);
 FUNCTION prod_count RETURN NUMBER;
 exp_no_prod_lgu EXCEPTION;
END;
```

## Package

### ● prod\_mgr 패키지의 본문

```
CREATE OR REPLACE PACKAGE BODY prod_mgr
IS
 CURSOR prod_cur (v_lgu VARCHAR2) IS
 SELECT prod_id, prod_name, to_char(prod_sale,'L999,999,999') prod_sale
 FROM prod
 WHERE prod_lgu = v_lgu;

 PROCEDURE prod_list IS
 BEGIN
 IF g_prod_lgu is null Then
 RAISE exp_no_prod_lgu;
 END IF;
 FOR prod_rec IN prod_cur (p_prod_lgu) LOOP
 DBMS_OUTPUT.PUT_LINE(prod_rec.prod_id || ', '
 || prod_rec.prod_name || ', ' || prod_rec.prod_sale);
 END LOOP;
 EXCEPTION
 WHEN exp_no_prod_lgu THEN
 DBMS_OUTPUT.PUT_LINE ('상품 분류가 없습니다.');
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE ('기타 에러 : ' || SQLERRM);
 END prod_list;
```

## Package

### ● prod\_mgr 패키지의 본문(Cont.)

```
PROCEDURE prod_list (p_prod_lgu IN prod.prod_lgu%TYPE)
IS
BEGIN
 FOR prod_rec IN prod_cur (p_prod_lgu) LOOP
 DBMS_OUTPUT.PUT_LINE(prod_rec.prod_id || ', '
 || prod_rec.prod_name || ', ' || prod_rec.prod_sale);
 END LOOP;
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.PUT_LINE ('기타 예외 : ' || SQLERRM);
END prod_list;

FUNCTION prod_count
RETURN NUMBER
IS
 v_cnt NUMBER;
BEGIN
 SELECT COUNT(*) INTO v_cnt FROM prod WHERE prod_lgu = g_prod_lgu;
 return v_cnt;
END prod_count;
END prod_mgr;
```