Chapter 03. SQL 기본

3-1. 관계형 데이터베이스 개요

➢ 데이터베이스

① 넓은 의미에서의 데이터베이스는 일상적인 정보들을 모아 놓은 것 자체를 의미한다.
② 일반적으로 데이터베이스라고 말할 때는 특정 기업이나 조직 또는 개인이 필요에 의해(ex: 부가가치가 발생하는) 데이터를 일정한 형태로 저장해 놓은 것을 의미한다.
③ 사용자들은 보다 효율적인 데이터의 관리 뿐만 아니라 예기치 못한 사건으로 인한 데이터의 손상을 피하고, 필요시 필요한 데이터를 복구하기 위한 강력한 기능의 소프트웨어를 필요로 하게 되었고 이러한 기본적인 요구사항을 만족시켜주는 시스템을 DBMS(Database Management System)라고 한다.

➢ 관계형 데이터베이스(Relational Database)

① 관계형 데이터베이스는 정규화를 통한 합리적인 테이블 모델링을 통해 이상(ANOMALY) 현상을 제거하고 데이터 중복을 피할 수 있으며, 동시성 관리, 병행 제어를 통해 많은 사용자들이 동시에 데이터를 공유 및 조작할 수 있는 기능을 제공
② 관계형 데이터베이스는 메타 데이터를 총괄 관리할 수 있기 때문에 데이터의 성격, 속성 또는 표현 방법 등을 체계화할 수 있고, 데이터 표준화를 통한 데이터 품질을 확보할 수 있는 장점을 가지고 있음
③ DBMS는 인증된 사용자만이 참조할 수 있도록 보안 기능을 제공하고 있다. 테이블 생성 시에 사용할 수 있는 다양한 제약조건을 이용하여 사용자가 실수로 조건에 위배되는 데이터를 입력 한다든지, 관계를 연결하는 중요 데이터를 삭제하는 것을 방지하여 데이터 무결성(Integrity)을 보장
④ DBMS는 시스템의 갑작스런 장애로부터 사용자가 입력, 수정, 삭제하던 데이터가 제대로 반영될 수 있도록 보장해주는 기능과, 시스템 다운, 재해 등의 상황에서도 데이터를 회복/복구할 수 있는 기능을 제공

➢ SQL(Structured Query Language)

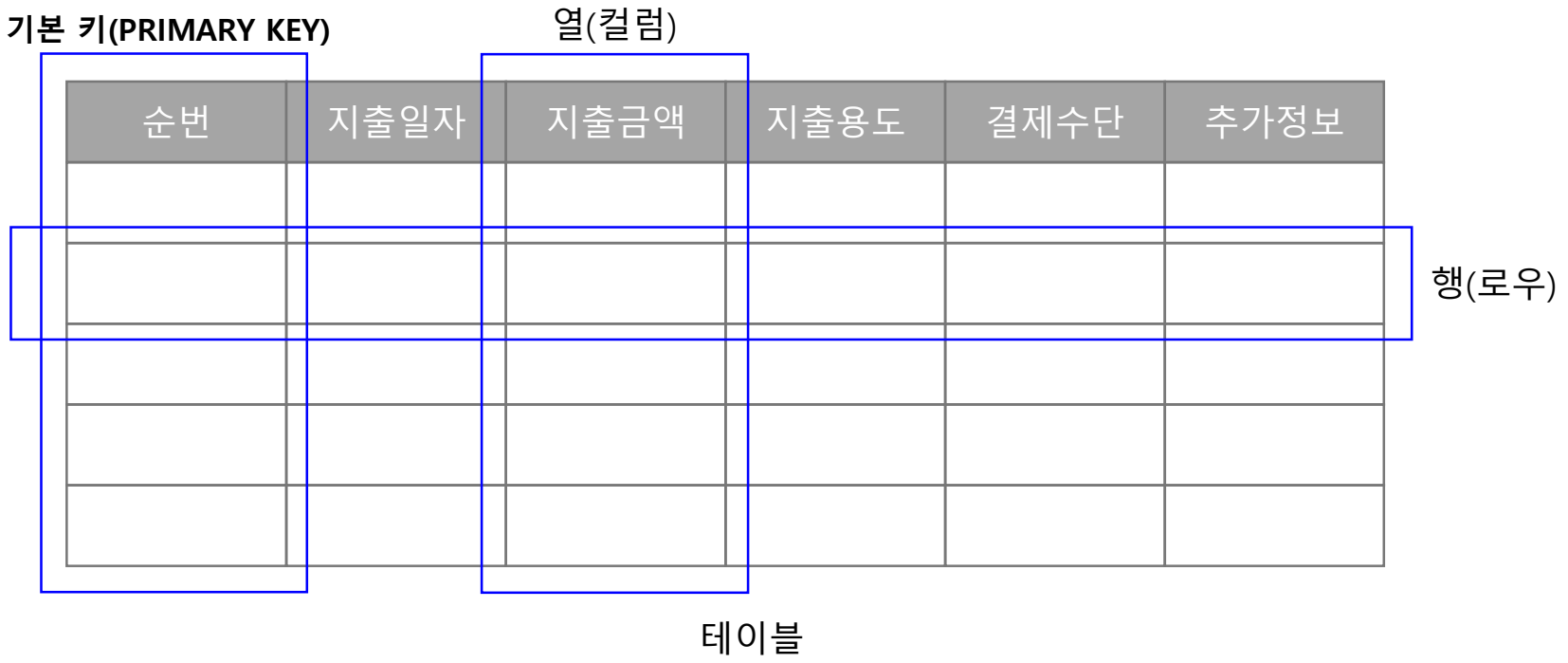
① SQL(Structured Query Language)은 관계형 데이터베이스에서 데이터 정의, 데이터 조작, 데이터 제어를 하기 위해 사용하는 언어
② 특정 데이터들의 집합에서 필요로 하는 데이터를 꺼내서 조회하고 새로운 데이터를 입력/수정/삭제하는 행위를 통해서 사용자는 데이터베이스와 대화하게 됨

➢ SQL문의 종류

종류	명령어	설명
데이터 조작어 (DML : Data Manipulation Language)	SELECT	• 데이터베이스에 들어있는 데이터를 조회하거나 검색하기 위한 명령어
	INSERT, UPDATE, DELETE	• 데이터베이스에 들어있는 데이터에 변형을 가하는데 사용되는 명령어
데이터 정의어 (DDL : Data Definition Language)	CREATE, ALTER, DROP, RENAME	• 테이블과 같은 데이터 구조를 정의하는데 사용되는 명령어들로 그러한 구조를 생성하거나 변경하거나 삭제하거나 이름을 바꾸는 데이터 구조와 관련된 명령어들을 DDL이라고 부른다.
데이터 제어어 (DCL : Data Control Language)	GRANT, REVOKE	• 데이터베이스에 접근하고 객체들을 사용하도록 권한을 부여 및 회수하는 명령어
트랜잭션 제어어 (TCL : Transaction Control Language)	COMMIT, ROLLBACK	• 논리적인 작업 단위를 묶어서 DML에 의해 조작된 결과를 작업 단위 별로 적용 및 취소하는 명령어

➢ 테이블(Table)

① 데이터는 관계형 데이터베이스의 기본 단위인 테이블 형태로 저장된다. 모든 자료는 테이블에 등록이 되고, 우리는 테이블로부터 원하는 자료를 꺼내 올 수 있다.
② 테이블(TABLE)은 데이터를 저장하는 객체(Object)로서 관계형 데이터베이스의 기본 단위이다.
③ 관계형 데이터베이스에서는 모든 데이터를 컬럼과 행의 2차원 구조로 나타낸다. 세로 방향을 컬럼(Column), 가로 방향을 행(Row)이라고 하고, 컬럼과 행이 겹치는 하나의 공간을 필드(Field)라고 한다.

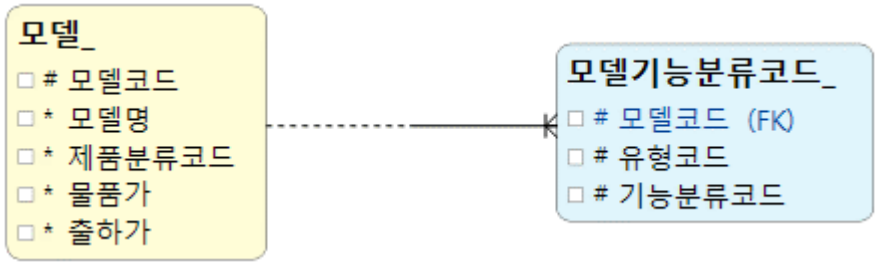


➢ 테이블(Table) 용어

종류	설명
테이블(Table)	• 행과 컬럼의 2차원 구조를 가진 데이터의 저장소
컬럼(Column)	• 테이블에서 세로방향으로 이루어진 하나하나의 속성(더이상 나눌 수 없는 것)
행(Row)	• 테이블에서 가로방향으로 이루어진 데이터
정규 형(Normalization)	• 테이블을 분할하여 데이터의 정합성을 확보하고, 불필요한 중복을 줄이는 프로세스
기본 키(Primary Key)	• 테이블에 존재하는 각 행을 한가지 의미로 특정할 수 있는 한 개 이상의 컬럼
외부 키(Foreign Key)	• 다른 테이블의 기본 키로 사용되고 있는 관계를 연결하는 컬럼

➢ ERD(Entity Relationship Diagram)

① 테이블 간 서로의 상관 관계를 그림으로 도식화한 것을 E-R 다이어그램이라고 하며, 간략히 ERD라고 함
② ERD의 구성 요소는 엔터티(Entity), 관계(Relationship), 속성(Attribute) 3가지이며 현실 세계의 데이터는 이 3가지 구성 요소로 모두 표현이 가능



Chapter 03. SQL 기본

3-2. DDL(DATA DEFINITION LANGUAGE)

➢ 주요 데이터 유형(타입) 정리

타입	설명
CHAR(L)	• 고정 길이 문자열, 고정 길이를 가지고 있으므로 할당된 변수의 값이 L값보다 작을 경우 그 차이만큼 공백으로 채워 짐
VARCHAR2(L)	• 가변 길이 문자열, L만큼의 최대 길이를 가짐, L값보다 작을 경우 해당 값만큼만 공간을 차지한다.
NUMBER(L, D)	• 정수,실수를 저장함, L값은 전체 자리 수, D값은 소수점 자리 수
DATE	• 날짜와 시각정보 "년월일시분초"를 표현함

❖ 데이터 유형은 데이터베이스의 테이블에 특정 자료를 입력할 때, 그 자료를 받아들일 공간을 자료의 유형별로 나누는 기준
❖ 선언한 유형이 아닌 다른 종류의 데이터가 들어오려고 하면 데이터베이스는 에러를 발생시킴

➢ CREATE TABLE

```
DROP TABLE SQLD.TB_DEPT_TEMP PURGE;

CREATE TABLE SQLD.TB_DEPT_TEMP
(
  DEPT_CD CHAR(6) NOT NULL
, DEPT_NM VARCHAR2(150) NOT NULL
, UPPER_DEPT_CD CHAR(6)
);

DROP TABLE SQLD.TB_DEPT_TEMP PURGE;
```

✓ DROP TABLE시 에러가 발생해도 바로 CREATE TABLE 실행하면됨

❖ 테이블 명은 단수형 권고
❖ 테이블 명은 같은 Owner내 다른 테이블과 중복되면 안됨
❖ 한 테이블내에서 컬럼 명이 중복되면 안됨
❖ 테이블 생성 문 끝은 ';'로 끝나야 함
❖ 데이터 유형은 반드시 지정 해야함
❖ 테이블 명과 컬럼 명은 반드시 문자로 시작해야함
❖ A-Z, a-z, 0-9, _, $, # 문자만 허용

➢ 제약조건

① 제약조건(CONSTRAINT)이란 사용자가 원하는 조건의 데이터만 유지하기 생성함
② 데이터의 무결성을 유지하기 위한 데이터베이스의 보편적인 방법으로 테이블의 특정 컬럼에 설정하는 제약

❖제약 조건의 종류

타입	설명
기본 키(Primary Key)	• 테이블에 저장된 행을 고유하게 식별하기 위함 하나의 테이블에 단 하나의 기본 키만 정의 가능 • 기본 키 생성 시 DBMS는 자동으로 UNIQUE 인덱스를 생성 • 기본 키 컬럼에는 NULL 입력 불가
고유 키(Unique Key)	• 테이블에 저장된 행 데이터를 고유하게 식별하기 위해 생성 • NULL은 입력 가능
NOT NULL	• NULL값의 입력을 금지 필수적으로 값이 들어가야하는 컬럼이 됨
CHECK	• 입력할 수 있는 값 종류 및 범위를 제한한다.
외래 키(Foreign Key)	• 다른 테이블의 기본 키를 외래 키로 지정하는 경우 생성함(참조무결성제약조건)

❖ NULL(ASCII 코드 00번)은 공백(BLANK, ASCII 코드 32번)이나 숫자 0(ZERO, ASCII 48)과는 전혀 다른 값이며, 조건에 맞는 데이터가 없을 때의 공 집합과도 다르다. "NULL"은 "아직 정의되지 않은 미지의 값" 이거나 "현재 데이터를 입력하지 못하는 경우"를 의미
❖ 데이터 입력 시에 컬럼의 값이 지정되어 있지 않을 경우 기본값(DEFAULT)을 사전에 설정할 수 있다. 데이터 입력 시 명시된 값을 지정하지 않은 경우에 NULL 값이 입력되고, DEFAULT 값을 정의했다면 해당 컬럼에 NULL 값이 입력되지 않고 사전에 정의된 기본 값이 자동으로 입력

➢ 테이블 생성 및 제약 조건 생성 실습

❖TB_DEPT 테이블 생성 및 기본 키 생성

```
DROP TABLE SQLD.TB_DEPT_TEMP PURGE;
CREATE TABLE SQLD.TB_DEPT_TEMP
(
  DEPT_CD CHAR(6) NOT NULL
, DEPT_NM VARCHAR2(150) NOT NULL
, UPPER_DEPT_CD CHAR(6)
);
ALTER TABLE SQLD.TB_DEPT_TEMP ADD CONSTRAINT PK_TB_DEPT_TEMP PRIMARY KEY (DEPT_CD);
```

✓ DROP TABLE시 에러가 발생해도 바로 CREATE TABLE 실행하면됨

❖TB_EMP 테이블 생성 및 기본 키 생성

```
DROP TABLE SQLD.TB_EMP_TEMP PURGE;
CREATE TABLE SQLD.TB_EMP_TEMP
(
       EMP_NO CHAR(10) NOT NULL
     , EMP_NM VARCHAR2(150) NOT NULL
     , BIRTH_DE CHAR(8) NOT NULL
     , LUNAR_YN CHAR(1) NOT NULL
     , SEX_CD CHAR(3) NOT NULL
     , ADDR VARCHAR2(1000) NOT NULL
     , TEL_NO VARCHAR2(150) NOT NULL
     , FINAL_EDU_SE_CD CHAR(3) NOT NULL
     , SAL_TRANS_BANK_CD CHAR(3) NOT NULL
     , SAL_TRANS_ACCNT_NO VARCHAR2(20) NOT NULL
     , DIRECT_MANAGER_EMP_NO CHAR(10)
     , DEPT_CD CHAR(6) NOT NULL
);
ALTER TABLE SQLD.TB_EMP_TEMP ADD CONSTRAINT PK_TB_EMP_TEMP PRIMARY KEY (EMP_NO);
```

✓ DROP TABLE시 에러가 발생해도 바로 CREATE TABLE 실행하면됨

❖TB_EMP 테이블에 TB_DEPT 테이블의 DEPT_CD컬럼을 참조하는 FK생성

```
ALTER TABLE SQLD.TB_EMP_TEMP ADD CONSTRAINT FK_TB_EMP_TEMP_01 FOREIGN KEY (DEPT_CD) REFERENCES SQLD.TB_DEPT_TEMP (DEPT_CD);
```

➢ ALTER TABLE

❖ 컬럼을 추가/삭제하거나 제약조건을 추가/삭제하는 작업

❖ ADD COLUMN

```
ALTER TABLE SQLD.TB_EMP_TEMP ADD (MARRIED_YN CHAR(1));
```

✓ 추가되는 컬럼은 테이블의 마지막 컬럼이 되며 컬럼의 위치를 지정할 수는 없음

❖ DROP COLUMN

```
ALTER TABLE SQLD.TB_EMP_TEMP DROP COLUMN MARRIED_YN;
```

✓ 한 번 삭제된 컬럼은 복구가 불가능

❖ MODIFY COLUMN

```
ALTER TABLE SQLD.TB_EMP_TEMP ADD (MARRIED_YN CHAR(1));
ALTER TABLE SQLD.TB_EMP_TEMP MODIFY(MARRIED_YN CHAR(1) DEFAULT 'N' NOT NULL NOVALIDATE) ;
ALTER TABLE SQLD.TB_EMP_TEMP DROP COLUMN MARRIED_YN;
```

✓ 컬럼의 크기를 늘릴 수는 있지만 줄이지는 못함
✓ NULL 값만 가지고 있거나 아무 행도 없으면 줄일 수 있음
✓ DEFAULT 값을 바꾸면 변경 작업 이후 발생하는 행 삽입에만 영향을 미침

❖ RENAME COLUMN

```
CREATE TABLE SQLD.TB_EMP_TEMP_2
AS SELECT * FROM SQLD.TB_EMP_TEMP;

ALTER TABLE SQLD.TB_EMP_TEMP_2
RENAME COLUMN TEL_NO TO PHONE_NO;

ALTER TABLE SQLD.TB_EMP_TEMP_2
RENAME COLUMN PHONE_NO TO TEL_NO;
```

✓ ANSI/ISO에 명시되어 있는 기능이 아니고 Oracle 등 일부 DBMS에서만 지원하는 기능

❖ DROP CONSTRAINT

```
ALTER TABLE SQLD.TB_EMP_TEMP DROP CONSTRAINT FK_TB_EMP_TEMP_01;
```

❖ ADD CONSTRAINT

```
ALTER TABLE SQLD.TB_EMP_TEMP ADD CONSTRAINT FK_TB_EMP_TEMP_01
FOREIGN KEY (DEPT_CD) REFERENCES SQLD.TB_DEPT_TEMP (DEPT_CD);
```

✓ 참조 무결성 제약조건을 생성함으로써 TB_EMP_TEMP 테이블의 DEPT_CD 컬럼의 들어가는 값은 TB_DEPT_TEMP 테이블의 DEPT_CD 컬럼의 값으로 존재해야 한다.

➢ RENAME TABLE

❖ 테이블 명을 변경할 수 있다.

```
RENAME TB_EMP_TEMP_2 TO TB_EMP_TEMP_3;
```

➢ TRUNCATE TABLE

❖ 테이블의 데이터를 비운다. TRUNCATE 명령 수행 시 삭제한 데이터는 ROLLBACK이 불가능하다.

```
TRUNCATE TABLE SQLD.TB_EMP_TEMP_3;
```

➢ DROP TABLE

❖ 테이블을 제거한다.

```
DROP TABLE SQLD.TB_EMP_TEMP_3;
```

Chapter 03. SQL 기본

3-3. DML(DATA MANIPULATION LANGUAGE)

➢ INSERT

```
INSERT INTO SQLD.TB_CERTI T (T.CERTI_CD, T.CERTI_NM, T.ISSUE_INSTI_NM) VALUES ('100021', 'SQLD합격패스', '패스트캠퍼스');
COMMIT;
```

```
SELECT *
 FROM SQLD.TB_CERTI  A
WHERE A.CERTI_CD = '100021'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100021	SQLD 합격패스	패스트캠퍼스

➢ UPDATE

```
UPDATE SQLD.TB_CERTI A
  SET A.ISSUE_INSTI_NM = '패스트캠퍼스온라인'
WHERE A.CERTI_CD = '100021'
;

COMMIT;
```

```
SELECT *
  FROM SQLD.TB_CERTI A
 WHERE A.CERTI_CD = '100021'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100021	SQLD 합격패스	패스트캠퍼스온라인

➢ DELETE

```
DELETE
  FROM SQLD.TB_CERTI A
 WHERE A.CERTI_CD = '100021'
;

COMMIT;
```

```
SELECT *
 FROM SQLD.TB_CERTI A
WHERE A.CERTI_CD = '100021'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM

✓ 공집합

➢ SELECT

```
SELECT A.CERTI_CD
     , A.CERTI_NM
     , A.ISSUE_INSTI_NM
  FROM SQLD.TB_CERTI A;
```

✓ 출력하고 싶은 컬럼 명을 기재함

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100001	SQLD	한국데이터베이스진흥원
100002	SQLP	한국데이터베이스진흥원
100003	DASP	한국데이터베이스진흥원
100004	DAP	한국데이터베이스진흥원
100005	ADSP	한국데이터베이스진흥원
100006	ADP	한국데이터베이스진흥원
100007	정보처리기사	한국산업인력공단
100008	리눅스마스터2급	한국정보통신진흥협회
100009	리눅스마스터1급	한국정보통신진흥협회
100010	OCP	오라클
100011	OCM	오라클
100012	OCJP	오라클
100013	OCJD	오라클
100014	OCWCD	오라클
100015	워드프로세서2급	대한상공회의소
100016	워드프로세서1급	대한상공회의소
100017	컴퓨터활용능력2급	대한상공회의소
100018	컴퓨터활용능력1급	대한상공회의소
100019	정보시스템감리사	한국정보화진흥원
100020	정보관리기술사	한국산업인력공단

➢ SELECT DISTINCT

```
SELECT DISTINCT A.ISSUE_INSTI_NM
  FROM SQLD.TB_CERTI A;
```

ISSUE_INSTI_NM
한국정보통신진흥협회
오라클
대한상공회의소
한국정보화진흥원
한국산업인력공단
한국데이터베이스진흥원

✓ ISSUE_INSTI_NM 컬럼 값 기준 중복을 제거한 유일한 값만을 출력함

➢ SELECT *

```
SELECT *
  FROM SQLD.TB_CERTI A;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100001	SQLD	한국데이터베이스진흥원
100002	SQLP	한국데이터베이스진흥원
100003	DASP	한국데이터베이스진흥원
100004	DAP	한국데이터베이스진흥원
100005	ADSP	한국데이터베이스진흥원
100006	ADP	한국데이터베이스진흥원
100007	정보처리기사	한국산업인력공단
100008	리눅스마스터2급	한국정보통신진흥협회
100009	리눅스마스터1급	한국정보통신진흥협회
100010	OCP	오라클
100011	OCM	오라클
100012	OCJP	오라클
100013	OCJD	오라클
100014	OCWCD	오라클
100015	워드프로세서2급	대한상공회의소
100016	워드프로세서1급	대한상공회의소
100017	컴퓨터활용능력2급	대한상공회의소
100018	컴퓨터활용능력1급	대한상공회의소
100019	정보시스템감리사	한국정보화진흥원
100020	정보관리기술사	한국산업인력공단

✓ "*"로 조회하면 모든 컬럼이 조회된다.

➢ ALIAS 지정

```
SELECT A.CERTI_CD AS 자격증코드
     , A.CERTI_NM AS 자격증명
     , A.ISSUE_INSTI_NM AS 발급기관명
  FROM SQLD.TB_CERTI A;
```

✓ AS를 이용하여 컬럼의 이름을 지정할 수 있다.

자격증코드	자격증명	발급기관명
100001	SQLD	한국데이터베이스진흥원
100002	SQLP	한국데이터베이스진흥원
100003	DASP	한국데이터베이스진흥원
100004	DAP	한국데이터베이스진흥원
100005	ADSP	한국데이터베이스진흥원
100006	ADP	한국데이터베이스진흥원
100007	정보처리기사	한국산업인력공단
100008	리눅스마스터2급	한국정보통신진흥협회
100009	리눅스마스터1급	한국정보통신진흥협회
100010	OCP	오라클
100011	OCM	오라클
100012	OCJP	오라클
100013	OCJD	오라클
100014	OCWCD	오라클
100015	워드프로세서2급	대한상공회의소
100016	워드프로세서1급	대한상공회의소
100017	컴퓨터활용능력2급	대한상공회의소
100018	컴퓨터활용능력1급	대한상공회의소
100019	정보시스템감리사	한국정보화진흥원
100020	정보관리기술사	한국산업인력공단

➢ 합성 연산자를 이용한 문자열 연결

```
SELECT
    A.CERTI_NM || '(' || A.CERTI_CD || ')' || '-' || A.ISSUE_INSTI_NM  AS CERTI_INFO
FROM SQLD.TB_CERTI A;
```

CERTI_INFO
SQLD(100001)-한국데이터베이스진흥원
SQLP(100002)-한국데이터베이스진흥원
DASP(100003)-한국데이터베이스진흥원
DAP(100004)-한국데이터베이스진흥원
ADSP(100005)-한국데이터베이스진흥원
ADP(100006)-한국데이터베이스진흥원
정보처리기사(100007)-한국산업인력공단
리눅스마스터2급(100008)-한국정보통신진흥협회
리눅스마스터1급(100009)-한국정보통신진흥협회
OCP(100010)-오라클
OCM(100011)-오라클
OCJP(100012)-오라클
OCJD(100013)-오라클
OCWCD(100014)-오라클
워드프로세서2급(100015)-대한상공회의소
워드프로세서1급(100016)-대한상공회의소
컴퓨터활용능력2급(100017)-대한상공회의소
컴퓨터활용능력1급(100018)-대한상공회의소
정보시스템감리사(100019)-한국정보화진흥원
정보관리기술사(100020)-한국산업인력공단

✓ "||" 연산자를 이용하여 문자열을 연결할 수 있다.

➢ DUAL 테이블을 이용한 연산 수행

```
SELECT ( (1+1)*3 ) / 6 AS CALC_RESULT
  FROM DUAL;
```

CALC_RESULT
1

✓ 괄호 안"()"부터 연산 수행
✓ 1더하기1후 곱하기 3 = 6
✓ 6 나누기 6 = 1
✓ 결과는 1이 됨

Chapter 03. SQL 기본

3-4. TCL(TRANSACTION CONTROL LANGUAGE)

➢ 트랜잭션의 특성

① 트랜잭션은 데이터베이스의 논리적 연산단위이다.
② 하나의 트랜잭션에는 하나 이상의 SQL 문장이 포함된다. 트랜잭션은 분할할 수 없는 최소의 단위이다. 그렇기 때문에 전부 적용하거나 전부 취소한다. 즉, TRANSACTION은 ALL OR NOTHING의 개념

특성	설명
원자성(Atomicity)	• 트랜잭션에서 정의된 연산들은 모두 성공적으로 끝나거나 모두 실패해야 한다. • (All or Nothing)
일관성(Consistency)	• 트랜잭션이 실행되기 전의 데이터베이스의 내용이 잘못되어 있지 않다면 실행된 이후에서 데이터베이스의 내용에 잘못이 있으면 안된다.
고립성(Isolation)	• 트랜잭션이 실행되는 도중에 다른 트랜잭션의 영향을 받아 잘못된 결과를 만들어서는 안된다.
지속성(Durability)	• 트랜잭션이 성공적으로 수행되면 그 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장된다.

❖ 계좌이체는 한 계좌에서 현금이 인출된 후에 다른 계좌로 입금이 되는데 현금이 인출되기 전에 다른 계좌에 입금이 되는 것은 문제를 발생시킬 수 있다. 이체가 결정되기 전까지는 다른 사람이 이 계좌의 정보를 변경할 수 없다. 이것을 보통 문에 자물쇠를 채우듯이 한다고 하여 잠금(LOCKING)이라고 표현한다.

➢ COMMIT

- 입력한 자료나 수정한 자료에 대해서 또는 삭제한 자료에 대해서 전혀 문제가 없다고 판단되었을 경우 COMMIT 명령어를 통해서 트랜잭션을 완료할 수 있다.

▪ COMMIT이나 ROLLBACK 이전의 데이터 상태

- ❖ 단지 메모리 BUFFER에만 영향을 받았기 때문에 데이터의 변경 이전 상태로 복구 가능하다.
- ❖ 현재 사용자는 SELECT 문장으로 결과를 확인 가능하다.
- ❖ 다른 사용자는 현재 사용자가 수행한 명령의 결과를 볼 수 없다.
- ❖ 변경된 행은 잠금(LOCKING)이 설정되어서 다른 사용자가 변경할 수 없다.

▪ COMMIT이후의 상태

- ❖ 데이터에 대한 변경 사항이 데이터베이스에 반영된다.
- ❖ 이전 데이터는 영원히 잃어버리게 된다.
- ❖ 모든 사용자는 결과를 볼 수 있다.
- ❖ 관련된 행에 대한 잠금(LOCKING)이 풀리고, 다른 사용자들이 행을 조작할 수 있게 된다.

```
INSERT INTO SQLD.TB_CERTI T (T.CERTI_CD, T.CERTI_NM, T.ISSUE_INSTI_NM) VALUES ('100022', 'SQL지식보유자', '패스트캠퍼스');

COMMIT;
```

```
SELECT *
 FROM SQLD.TB_CERTI  A
WHERE A.CERTI_CD = '100022'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100022	SQL지식보유자	패스트캠퍼스

➢ COMMIT

```
UPDATE SQLD.TB_CERTI
   SET CERTI_NM = 'SQL경험보유자'
 WHERE CERTI_CD = '100022'
;

COMMIT;
```

```
SELECT *
  FROM SQLD.TB_CERTI  A
 WHERE A.CERTI_CD = '100022'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100022	SQL경험보유자	패스트캠퍼스

```
DELETE
  FROM SQLD.TB_CERTI
 WHERE CERTI_CD = '100022'
;

COMMIT;
```

```
SELECT *
 FROM SQLD.TB_CERTI  A
WHERE A.CERTI_CD = '100022'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM

➢ ROLLBACK

① 테이블 내 입력한 데이터나, 수정한 데이터, 삭제한 데이터에 대하여 COMMIT 이전에는 변경 사항을 취소할 수 있는데 데이터베이스에서는 롤백(ROLLBACK) 기능을 사용한다.

② 롤백(ROLLBACK)은 데이터 변경 사항이 취소되어 데이터의 이전 상태로 복구되며, 관련된 행에 대한 잠금(LOCKING)이 풀리고 다른 사용자들이 데이터 변경을 할 수 있게 된다.

```
INSERT INTO SQLD.TB_CERTI T (T.CERTI_CD, T.CERTI_NM, T.ISSUE_INSTI_NM) VALUES ('100022', 'SQL지식보유자', '패스트캠퍼스');

ROLLBACK;
```

```
SELECT *
 FROM SQLD.TB_CERTI  A
WHERE A.CERTI_CD = '100022'
;
```

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM

➢ SAVE POINT

① 저장점(SAVEPOINT)을 정의하면 롤백(ROLLBACK)할 때 트랜잭션에 포함된 전체 작업을 롤백 하는 것이 아니라 현 시점에서 SAVEPOINT까지 트랜잭션의 일부만 롤백 할 수 있다.
② 복잡한 대규모 트랜잭션에서 에러가 발생했을 때 SAVEPOINT까지의 트랜잭션만 롤백하고 실패한 부분에 대해서만 다시 실행

```
SAVEPOINT SVPT1;  여기가 세이브 포인트 지점

INSERT INTO SQLD.TB_CERTI T (T.CERTI_CD, T.CERTI_NM, T.ISSUE_INSTI_NM) VALUES ('100022', 'SQL지식보유자', '패스트캠퍼스');

UPDATE SQLD.TB_CERTI
   SET CERTI_NM = 'SQL경험보유자'
 WHERE CERTI_CD = '100022'
;

ROLLBACK TO SVPT1; 세이브 포인트 지점으로 롤백, INSERT및 UPDATE는 모두 롤백 됨

DELETE
  FROM SQLD.TB_CERTI
 WHERE CERTI_CD = '100022'
;

SELECT * FROM SQLD.TB_CERTI
 WHERE CERTI_CD = '100022'
;
```

삭제 실패(데이터가 존재하지 않음)

공집합이 나오게 됨

➢ SAVE POINT

```
SAVEPOINT SVPT1; 저장 직전에 세이브 포인트

INSERT INTO SQLD.TB_CERTI T (T.CERTI_CD, T.CERTI_NM, T.ISSUE_INSTI_NM) VALUES ('100022', 'SQL지식보유자', '패스트캠퍼스');

SAVEPOINT SVPT2; 수정 직전에 세이브 포인트

UPDATE SQLD.TB_CERTI
   SET CERTI_NM = 'SQL경험보유자'
 WHERE CERTI_CD = '100022'
;

SAVEPOINT SVPT3; 삭제하기 바로 직전에 세이브 포인트

DELETE FROM SQLD.TB_CERTI
WHERE CERTI_CD = '100022'
;

ROLLBACK TO SVPT3;
SELECT * FROM SQLD.TB_CERTI WHERE CERTI_CD = '100022' ;  000022 SQL경험보유자 패스트캠퍼스

ROLLBACK TO SVPT2;
SELECT * FROM SQLD.TB_CERTI WHERE CERTI_CD = '100022' ;  000022 SQL지식보유자 패스트캠퍼스

ROLLBACK TO SVPT1; --데이터 없음
SELECT * FROM SQLD.TB_CERTI WHERE CERTI_CD = '100022' ;  공집합(조회결과없음)
```

➢ 트랜잭션 요점

① 테이블에 데이터의 변경을 발생시키는 입력(INSERT), 수정(UPDATE), 삭제(DELETE) 수행 시 그 변경되는 데이터의 무결성을 보장하는 것이 커밋(COMMIT)과 롤백(ROLLBACK)의 목적
② 커밋(COMMIT)은 "변경된 데이터를 테이블에 영구적으로 반영해라"라는 의미를 갖는 것
③ 롤백(ROLLBACK)은 "변경전으로 복귀하라"라는 의미
④ 저장점(SAVEPOINT/SAVE TRANSACTION)은 "데이터 변경을 사전에 지정한 저장점까지만 롤백하라"는 의미
⑤ Oracle의 트랜잭션은 트랜잭션의 대상이 되는 SQL 문장을 실행하면 자동으로 시작되고, COMMIT 또는 ROLLBACK을 실행한 시점에서 종료

➢ COMMIT 및 ROLLBACK 처리와는 상관없이 트랜잭션 처리가 일어나는 상황

① CREATE, ALTER, DROP, RENAME, TRUNCATE TABLE 등 DDL 문장을 실행하면 그 전후 시점에 자동으로 커밋 된다.
② DML 문장 이후에 커밋 없이 DDL 문장이 실행되면 DDL 수행 전에 자동으로 커밋 된다.
③ 데이터베이스를 정상적으로 접속을 종료하면 자동으로 트랜잭션이 커밋 된다.
④ 애플리케이션의 이상 종료로 데이터베이스와의 접속이 단절되었을 때는 트랜잭션이 자동으로 롤백 된다.

Chapter 03. SQL 기본

3-5. WHERE 절

➢ WHERE 조건절 개요

❖ 사용자들은 자신이 원하는 자료만을 검색하기 위해서 SQL 문장에 WHERE 절을 이용하여 자료들에 대하여 제한할 수 있다. WHERE 절은 FROM 절 다음에 위치하며, 조건식은 아래 내용으로 구성된다.

① 컬럼(Column)명 (보통 조건식의 좌측에 위치)
② 비교 연산자
③ 문자, 숫자, 표현식 (보통 조건식의 우측에 위치)
④ 비교 컬럼 명 (JOIN 사용시)

➢ 연산자의 종류

① 비교 연산자 (부정 비교 연산자 포함)
② SQL 연산자 (부정 SQL 연산자 포함)
③ 논리 연산자

➢ 연산자의 우선순위

우선순위	설명
1	• () 괄호
2	• NOT 연산자
3	• 비교 연산자, SQL 비교 연산자
4	• AND
5	• OR

➢ 비교 연산자

연산자	연산자의 의미
=	~와 같다
>	~보다 크다
>=	~보다 크거나 같다
<	~보다 작다
<=	~보다 작거나 같다

➢ SQL연산자

연산자	연산자의 의미
BETWEEN A AND B	A와 B사이에 있으면 된다.
IN (LIST)	리스트에 있는 값중 하나라도 있으면 된다.
LIKE '비교문자열'	비교문자열의 형태와 일치하면 된다. (와일드카드 사용)
IS NULL	값이 NULL이면 된다.

➢ 와일드 카드 종류

와일드카드	설명
%	0개 이상의 어떤 문자를 의미한다.
_	1개인 단일 문자를 의미한다.

➢ BETWEEN, IN, LIKE의 사용

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
  FROM TB_EMP A
 WHERE A.BIRTH_DE BETWEEN '19900101' AND '19991231'
   AND A.DEPT_CD IN ('100004', '100006')  --디자인팀, 데이터팀
   AND A.ADDR LIKE '%수원시%' --"수원시"에 사는 사람
;
```

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR
1000000010	박혜령	2	19920202	100004	경기도 수원시 팔달구 매탄동 987
1000000015	이정직	1	19980715	100006	경기도 수원시 팔달구 인계동 124

❖ BETWEEN A AND B 로 1990년대 생 직원을 조회
❖ IN 으로 특정 팀들을 조회
❖ LIKE로 특정 지역을 조회

➢ LIKE 연산자의 사용

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
  FROM TB_EMP A
 WHERE A.EMP_NM LIKE '박__'
  ;
```

❖ 성이 "박"씨인 직원을 출력함

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR
1000000009	박태범	1	19880629	100003	경기도 용인시 수지구 죽전3동 998
1000000010	박혜령	2	19920202	100004	경기도 수원시 팔달구 매탄동 987
1000000018	박바른	2	19820629	100006	경기도 용인시 수지구 죽전1동 484
1000000019	박정혜	2	19920202	100007	경기도 수원시 팔달구 매탄동 987
1000000024	박선영	1	19870615	100009	경기도 수원시 팔달구 매탄2동 445
1000000025	박호진	2	19720128	100009	경기도 성남시 분당구 정자동 123
1000000027	박이수	2	19980529	100009	서울특별시 강남구 역삼동 221
1000000036	박여진	2	19980529	100012	경기도 용인시 수지구 죽전4동 998

➢ NULL인 행 조회

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
     , NVL(A.DIRECT_MANAGER_EMP_NO, '상사없음') DIRECT_MANAGER_EMP_NO
  FROM TB_EMP A
 WHERE A.DIRECT_MANAGER_EMP_NO IS NULL -- NULL은 '='로 비교할 수 없음
  ;
```

❖ 직속관리자사원번호가 NULL인 직원을 조회함

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR	DIRECT_MANAGER_EMP_NO
9999999999	김회장	1	19651105	999999	경기도 용인시 수지구 죽전1동 215	상사없음

➢ 논리 연산자

연산자	연산자의 의미
AND	앞 조건과 뒤 조건이 모두 참이어야 한다.
OR	앞 조건과 뒤 조건 중 하나라도 참이어야 한다.
NOT	조건이 거짓이면 된다.

➢ AND, OR 연산자 실습

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
  FROM TB_EMP A
 WHERE A.BIRTH_DE BETWEEN '19900101' AND '19991231'
   AND A.DEPT_CD IN ('100004', '100006')  --디자인팀, 데이터팀
   AND (   A.ADDR LIKE '%수원시%' --"수원시"에 사는 사람
        OR A.ADDR LIKE '%성남시%' --"성남시"에 사는 사람
       )
;
```

❖ 수원시 혹은 성남시에 사는 사람을 포함

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR
1000000010	박혜령	2	19920202	100004	경기도 수원시 팔달구 매탄동 987
1000000011	최수자	2	19951122	100004	경기도 성남시 분당구 야탑동 487
1000000015	이정직	1	19980715	100006	경기도 수원시 팔달구 인계동 124
1000000016	이진실	2	19920128	100006	경기도 성남시 분당구 정자동 998

➢ AND, OR, NOT 연산자 실습

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
  FROM TB_EMP A
 WHERE A.BIRTH_DE BETWEEN '19900101' AND '19991231'
   AND A.DEPT_CD IN ('100004', '100006')  --디자인팀, 데이터팀
   AND NOT (   A.ADDR LIKE '%수원시%' --"수원시"에 사는 사람
            OR A.ADDR LIKE '%성남시%' --"성남시"에 사는 사람
           )
;
```

❖ 수원시 혹은 성남시에 사는 사람을 제외

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR
1000000012	김호형	1	19910227	100004	경기도 고양시 일산서구 일산동 987

➢ 부정비교연산자

연산자	연산자의 의미
!=	같지않다.
<>	같지않다.
^=	같지않다.
NOT 컬럼명 =	~와 같지 않다.
NOT 컬럼명 >	~보다 크지 않다.

➢ 비교 연산자 실습

```
SELECT A.SAL_HIS_NO
     , A.PAY_DE
     , A.PAY_AMT
     , A.EMP_NO
  FROM SQLD.TB_SAL_HIS A
 WHERE A.PAY_DE >= '20200501'
   AND A.PAY_DE <= '20200531'
   AND A.PAY_AMT >= 5500000
  ;
```

❖ 급여지급일자가 2020년 5월인 지급 내역 중 지급 액수가 550만원 이상인 내역을 출력

SAL_HIS_NO	PAY_DE	PAY_AMT	EMP_NO
20200701000065	20200525	5950000	1000000024
20200701000073	20200525	5570000	1000000032
20200701000075	20200525	5890000	1000000034

➢ 부정 비교 연산자 실습

```
SELECT A.SAL_HIS_NO
     , A.PAY_DE
     , A.PAY_AMT
     , A.EMP_NO
  FROM SQLD.TB_SAL_HIS A
 WHERE A.PAY_DE >= '20200501'
   AND A.PAY_DE <= '20200531'
   AND NOT A.PAY_AMT >= 5500000
;
```

❖ 급여지급일자가 2020년 5월인 지급 내역 중 지급 액수가 550만원 미만인 내역을 출력

SAL_HIS_NO	PAY_DE	PAY_AMT	EMP_NO
20200701000042	20200525	3700000	1000000001
20200701000043	20200525	3760000	1000000002
20200701000044	20200525	4440000	1000000003
20200701000045	20200525	4910000	1000000004
20200701000046	20200525	4320000	1000000005
중간생략 …			
20200701000069	20200525	4590000	1000000028
20200701000070	20200525	4800000	1000000029
20200701000071	20200525	5020000	1000000030
20200701000072	20200525	5150000	1000000031
20200701000074	20200525	3660000	1000000033
20200701000076	20200525	4350000	1000000035
20200701000077	20200525	2650000	1000000036
20200701000078	20200525	5140000	1000000037
20200701000079	20200525	2650000	1000000038
20200701000080	20200525	3930000	1000000039
20200701000081	20200525	4210000	1000000040
20200701000082	20200525	5470000	9999999999

➢ 부정비교 연산자 실습 - 2

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
  FROM TB_EMP A
 WHERE A.BIRTH_DE BETWEEN '19900101' AND '19991231'
   AND A.DEPT_CD IN ('100004', '100006')  --디자인팀, 데이터팀
   AND A.SEX_CD <> '1'
;
```

❖ 성별이 남자가 아닌 직원을 출력함

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR
1000000010	박혜령	2	19920202	100004	경기도 수원시 팔달구 매탄동 987
1000000011	최수자	2	19951122	100004	경기도 성남시 분당구 야탑동 487
1000000016	이진실	2	19920128	100006	경기도 성남시 분당구 정자동 998

➢ 부정SQL연산자

연산자	연산자의 의미
NOT BETWEEN A AND B	A와 B의 값 사이에 있지 않다.
NOT IN (LIST)	LIST에 있는 값과 모두 일치하지 않는다.
IS NOT NULL	NULL값이 아니다.

➢ 부정SQL연산자 - 실습

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.SEX_CD
     , A.BIRTH_DE
     , A.DEPT_CD
     , A.ADDR
     , A.DIRECT_MANAGER_EMP_NO
  FROM TB_EMP A
 WHERE A.BIRTH_DE NOT BETWEEN '19900101' AND '19991231' --90년대 생이 아니고
   AND A.DEPT_CD NOT IN ('100004', '100006')  --디자인팀과 데이터 팀이 아니고
   AND A.SEX_CD <> '1'   --성별이 남자가 아니고
   AND DIRECT_MANAGER_EMP_NO IS NOT NULL --직속관리자사원번호가 존재하는 직원
```

EMP_NO	EMP_NM	SEX_CD	BIRTH_DE	DEPT_CD	ADDR	DIRECT_MANAGER_EMP_NO
1000000006	김혜진	2	19840715	100003	경기도 수원시 팔달구 권선동 887	1000000001
1000000008	김려원	2	19890624	100003	경기도 고양시 일산서구 마두동 112	1000000006
1000000022	김순수	2	19871201	100007	경기도 용인시 수지구 죽전5동 228	1000000019
1000000025	박호진	2	19720128	100009	경기도 성남시 분당구 정자동 123	1000000024
1000000031	김사랑	2	19811201	100010	서울특별시 서초구 방배동 357	1000000028
1000000034	김익정	2	19720128	100012	경기도 성남시 분당구 정자동 996	1000000033
1000000040	김여진	2	19871205	100013	경기도 용인시 수지구 죽전1동 877	1000000037

➢ 문자유형비교방법 - 양쪽이 모두 CHAR 타입인 경우

상황	방법
양쪽이 모두 CHAR 타입인 경우	• 길이가 서로 다르면 작은 쪽에 공백을 추가하여 길이를 같게 함 • 서로 다른 문자가 나올 때 까지 비교 • 달라진 첫번째 값에 따라 크기를 결정 • 공백의 수만 다르다면 같은 값으로 결정

```
DROP TABLE SQLD.CHAR_COMPARE;

CREATE TABLE SQLD.CHAR_COMPARE
(
  SN CHAR(10)
, CHAR_COMPARE_4 CHAR(4)
, CHAR_COMPARE_6 CHAR(6)
)
;

INSERT INTO SQLD.CHAR_COMPARE VALUES ('1000000001', 'SQLD', 'SQLD');
INSERT INTO SQLD.CHAR_COMPARE VALUES ('1000000002', 'SQLD', 'SQLA');
COMMIT;
```

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS CHAR_COMPARE_4
     , REPLACE(CHAR_COMPARE_6, ' ', '_') AS CHAR_COMPARE_6
  FROM SQLD.CHAR_COMPARE
;
```

CHAR_COMPARE_4	CHAR_COMPARE_6
SQ LD	SQ LA__
SQ LD	SQ LD__

❖ 공백이 들어간 것을 알 수 있음

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS CHAR_COMPARE_4
     , REPLACE(CHAR_COMPARE_6, ' ', '_') AS CHAR_COMPARE_6
  FROM SQLD.CHAR_COMPARE
 WHERE SN = '1000000001'
   AND CHAR_COMPARE_4 = CHAR_COMPARE_6
;
```

CHAR_COMPARE_4	CHAR_COMPARE_6
SQ LD	SQ LD__

❖ 공백만 다르다면 같다고 판단함

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS CHAR_COMPARE_4
     , REPLACE(CHAR_COMPARE_6, ' ', '_') AS CHAR_COMPARE_6
  FROM SQLD.CHAR_COMPARE
 WHERE SN = '1000000002'
   AND CHAR_COMPARE_4 > CHAR_COMPARE_6 --알파벳D가 더 큼
;
```

CHAR_COMPARE_4	CHAR_COMPARE_6
SQ LD	SQ LA__

❖ SQLD가 더 큼, SQLA 뒤에 있는 공백은 크기 비교에 영향을 주지 못함

➢ 문자유형비교방법 - 비교연산자중 한쪽이 VARCHAR인 경우

상황	방법
비교연산자중 한쪽이 VARCHAR인 경우	• 서로 다른 문자가 나올 때까지 비교한다. • 길이가 다르다면 짧은 것이 끝날 때 까지만 비교한 후 길이가 긴 것이 크다고 판단한다. • 길이가 같고 다른 것이 없다면 같다고 판단한다. • VARCHAR는 공백도 문자로 판단한다.

```
DROP TABLE SQLD.VARCHAR_COMPARE;

CREATE TABLE SQLD.VARCHAR_COMPARE
(
  SN CHAR(10)
, CHAR_COMPARE_4 CHAR(4)
, VARCHAR_COMPARE_6 VARCHAR2(6)
)
;
INSERT INTO SQLD.VARCHAR_COMPARE VALUES ('1000000001', 'SQLD', 'SQLD  ');
INSERT INTO SQLD.VARCHAR_COMPARE VALUES ('1000000002', 'SQLD', 'SQLA  ');

COMMIT;
```

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS VARCHAR_COMPARE_4
     , REPLACE(VARCHAR_COMPARE_6, ' ', '_') AS VARCHAR_COMPARE_6
  FROM SQLD.VARCHAR_COMPARE
;
```

VARCHAR_COMPARE_4	VARCHAR_COMPARE_6
SQ LD	SQ LD __
SQ LD	SQ LA __

❖ VARCHAR2 컬럼의 값은 뒤에 공백이 있음

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS VARCHAR_COMPARE_4
     , REPLACE(VARCHAR_COMPARE_6, ' ', '_') AS VARCHAR_COMPARE_6
  FROM SQLD.VARCHAR_COMPARE
 WHERE SN = '1000000001'
   AND CHAR_COMPARE_4 = VARCHAR_COMPARE_6
;
```

VARCHAR_COMPARE_4	VARCHAR_COMPARE_6

❖ 결과집합없음 길이가 달라서 같지 않다고 판단

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS VARCHAR_COMPARE_4
     , REPLACE(VARCHAR_COMPARE_6, ' ', '_') AS VARCHAR_COMPARE_6
  FROM SQLD.VARCHAR_COMPARE
 WHERE SN = '1000000001'
   AND CHAR_COMPARE_4 <> VARCHAR_COMPARE_6
;
```

VARCHAR_COMPARE_4	VARCHAR_COMPARE_6
SQ LD	SQ LD __

❖ SQLD까지 문자는 동일하나 공백이 틀려서 같지 않다고 판단함

➢ 문자유형비교방법 - 비교연산자중 한쪽이 VARCHAR인 경우 - TRIM함수의 사용

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS VARCHAR_COMPARE_4
     , REPLACE(VARCHAR_COMPARE_6, ' ', '_') AS VARCHAR_COMPARE_6
  FROM SQLD.VARCHAR_COMPARE
 WHERE SN = '1000000001'
   AND CHAR_COMPARE_4 = TRIM(VARCHAR_COMPARE_6)
```

VARCHAR_COMPARE_4	VARCHAR_COMPARE_6
SQLD	SQLD__

❖ TRIM함수를 이용하여 VARCHAR2컬럼 뒤에 있는 공백을 제거한 후 비교하면 같은 값으로 비교된다.

➢ 문자유형비교방법 - 상수 값과 비교

상황	방법
상수값과 비교	• 상수 쪽을 변수 타입과 동일하게 바꾸고 비교한다. • 변수 쪽이 CHAR이면 CHAR타입인 경우를 적용한다. • 변수 쪽이 VARCHAR이면 VARCHAR타입인 경우를 적용한다.

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS VARCHAR_COMPARE_4
     , REPLACE(VARCHAR_COMPARE_6, ' ', '_') AS VARCHAR_COMPARE_6
  FROM SQLD.VARCHAR_COMPARE
 WHERE SN = '1000000001'
   AND CHAR_COMPARE_4 = 'SQLD'
;
```

VARCHAR_COMPARE_4	VARCHAR_COMPARE_6
SQLD	SQLD__

❖ 상수 값이 CHAR로 비교됨

```
SELECT
       REPLACE(CHAR_COMPARE_4, ' ', '_') AS VARCHAR_COMPARE_4
     , REPLACE(VARCHAR_COMPARE_6, ' ', '_') AS VARCHAR_COMPARE_6
  FROM SQLD.VARCHAR_COMPARE
 WHERE SN = '1000000001'
   AND VARCHAR_COMPARE_6 = 'SQLD'
;
```

VARCHAR_COMPARE_4	VARCHAR_COMPARE_6

❖ 상수 값이 VARCHAR2로 비교됨
❖ 상수 값은 뒤에 공백이 없으므로 같지 않음

➢ 집합의 일부분만을 출력

```
SELECT A.SAL_HIS_NO
     , A.PAY_DE
     , A.PAY_AMT
     , A.EMP_NO
  FROM SQLD.TB_SAL_HIS A
 WHERE A.PAY_DE >= '20200501'
   AND A.PAY_DE <= '20200531'
   AND ROWNUM <= 10
 ORDER BY A.PAY_AMT DESC
 ;
```

SAL_HIS_NO	PAY_DE	PAY_AMT	EMP_NO
20200701000048	20200525	5000000	1000000007
20200701000045	20200525	4910000	1000000004
20200701000044	20200525	4440000	1000000003
20200701000046	20200525	4320000	1000000005
20200701000050	20200525	4100000	1000000009
20200701000043	20200525	3760000	1000000002
20200701000042	20200525	3700000	1000000001
20200701000049	20200525	3600000	1000000008
20200701000051	20200525	2910000	1000000010
20200701000047	20200525	2510000	1000000006

❖ PAY_DE조건을 만족하는 집합에서 10건만을 먼저 출력하는데
❖ 그 10건의 출력 결과를 PAY_AMT로 내림차순 정렬함

```
SELECT A.SAL_HIS_NO
     , A.PAY_DE
     , A.PAY_AMT
     , A.EMP_NO
  FROM
     (
      SELECT
             A.SAL_HIS_NO
           , A.PAY_DE
           , A.PAY_AMT
           , A.EMP_NO
        FROM SQLD.TB_SAL_HIS A
       WHERE A.PAY_DE >= '20200501'
         AND A.PAY_DE <= '20200531'
       ORDER BY A.PAY_AMT DESC
     ) A
WHERE ROWNUM <= 10
;
```

SAL_HIS_NO	PAY_DE	PAY_AMT	EMP_NO
20200701000065	20200525	5950000	1000000024
20200701000075	20200525	5890000	1000000034
20200701000073	20200525	5570000	1000000032
20200701000082	20200525	5470000	9999999999
20200701000072	20200525	5150000	1000000031
20200701000078	20200525	5140000	1000000037
20200701000056	20200525	5100000	1000000015
20200701000067	20200525	5080000	1000000026
20200701000071	20200525	5020000	1000000030
20200701000048	20200525	5000000	1000000007

❖ PAY_DE 조건을 만족하는 집합에서 PAY_AMT기준으로 내림차순 정렬한 집합 중 상위 10건을 출력함

Chapter 03. SQL 기본

3-6. 함수(FUNCTION)

➢ 단일 행 함수의 주요 특징

① SELECT, WHERE, ORDER BY 절에 사용 가능하다.
② 각 행(Row)들에 대해 개별적으로 작용하여 데이터 값들을 조작하고, 각각의 행에 대한 조작 결과를 리턴 한다.
③ 여러 인자(Argument)를 입력해도 단 하나의 결과만 리턴 한다.
④ 함수의 인자(Arguments)로 상수, 변수, 표현식이 사용 가능하고, 하나의 인수를 가지는 경우도 있지만 여러 개의 인수를 가질 수도 있다.
⑤ 특별한 경우가 아니면 함수의 인자(Arguments)로 함수를 사용하는 함수의 중첩이 가능하다.

➢ 단일 행 함수의 종류

종류	설명
문자 형 함수	문자를 입력하면 문자나 숫자값을 반환한다. (LOWER, UPPER, SUBSTR, LENGTH, LTRIM, RTRIM, TRIM, ASCII)
숫자 형 함수	숫자를 입력하면 숫자값을 반환한다. (ABS, MOD, ROUND, TRUNC, SIGN, CHR, CEIL, FLOOR, EXP, LOG, LN, POWER, SIN, COS, TAN)
날짜 형 함수	DATE 타입의 값을 연산한다. (SYSDATE, EXTRACT, TO_NUMBER)
변환 형 함수	문자, 숫자, 날짜형의 값의 데이터 타입을 변환한다. (TO_NUMBER, TO_CHAR, TO_DATE, CONVERT)
NULL관련 함수	NULL을 처리하기 위한 함수 (NVL, NULLIF, COALESCE)

➢ 단일 행 문자형 함수 사용 예시

```
SELECT
      LOWER('SQL Developer') AS "LOWER('SQL Developer')" --소문자로 변환
    , UPPER('SQL Developer') AS "UPPER('SQL Developer')" --대문자로 변환
    , ASCII('A') AS "ASCII('A')" --아스키코드값 출력
    , CHR('65') AS "CHR('65')" --아스키코드값의 문자 출력
    , CONCAT('SQL', 'Developer') AS "CONCAT('SQL', 'Developer')" --문자열 결합
    , SUBSTR('SQL Developer', 1, 3) AS "SUBSTR('SQL Developer', 1, 3)" --문자열 잘라내기
    , LENGTH('SQL') AS "LENGTH('SQL')" --문자열의 길이 출력
    , LTRIM(' SQL') AS "LTRIM(' SQL')" --왼쪽 공백 제거
    , RTRIM('SQL ') AS "RTRIM('SQL ')" --오른쪽 공백 제거
 FROM DUAL
 ;
```

Name	Value
LOWER('SQL Developer')	sql developer
UPPER('SQL Developer')	SQL DEVELOPER
ASCII('A')	65
CHR('65')	A
CONCAT('SQL', 'Developer')	SQLDeveloper
SUBSTR('SQL Developer', 1, 3)	SQL
LENGTH('SQL')	3
LTRIM(' SQL')	SQL
RTRIM('SQL ')	SQL

➢ 단일행 숫자형 함수 사용 예시

```
SELECT
      ABS(-15) AS "ABS(-15)"  --절대값을 반환
    , SIGN(10) AS "SIGN(10)"  --양수일경우 1, 음수일경우 -1, 0일 경우 0 반환
    , MOD(8,3) AS "MOD(8,3)" --나머지를 반환
    , CEIL(38.678) AS "CEIL(38.678)" --무조건 올림
    , FLOOR(38.678) AS "FLOOR(38.678)" --무조건 버림
    , ROUND(38.678, 2) AS "ROUND(38.678, 2)" --소수점 2번째 자리에서 반올림
    , TRUNC(38.678) AS "TRUNC(38.678)" --0의 자리에서 무조건 자름
    , TRUNC(38.678, 1) AS "TRUNC(38.678, 1)" --1의 자리에서 무조건 자름
    , TRUNC(38.678, 2) AS "TRUNC(38.678, 2)" --2의 자리에서 무조건 자름
    , TRUNC(38.678, 3) AS "TRUNC(38.678, 3)" --3의 자리에서 무조건 자름
 FROM DUAL
 ;
```

Name	Value
ABS(-15)	15
SIGN(10)	1
MOD(8,3)	2
CEIL(38.678)	39
FLOOR(38.678)	38
ROUND(38.678, 2)	38.68
TRUNC(38.678)	38
TRUNC(38.678, 1)	38.6
TRUNC(38.678, 2)	38.67
TRUNC(38.678, 3)	38.678

➢ 날짜형 데이터 변환 함수 사용 예시

```
SELECT SYSDATE AS "SYSDATE" --현재 년월일시분초 출력
     , EXTRACT(YEAR FROM SYSDATE) AS "EXTRACT(YEAR FROM SYSDATE)" --년 출력
     , EXTRACT(MONTH FROM SYSDATE) AS "EXTRACT(MONTH FROM SYSDATE)" --월 출력
     , EXTRACT(DAY FROM SYSDATE) AS "EXTRACT(DAY FROM SYSDATE)" --일 출력
     , TO_CHAR(SYSDATE, 'YYYY') AS "TO_CHAR(SYSDATE, 'YYYY')" --년 출력(문자열)
     , TO_CHAR(SYSDATE, 'MM') AS "TO_CHAR(SYSDATE, 'MM')" --월 출력(문자열)
     , TO_CHAR(SYSDATE, 'DD') AS "TO_CHAR(SYSDATE, 'DD')" --일 출력(문자열)
     , TO_CHAR(SYSDATE, 'HH24') AS "TO_CHAR(SYSDATE, 'HH24')" --시 출력(문자열)
     , TO_CHAR(SYSDATE, 'MI') AS "TO_CHAR(SYSDATE, 'MI')" --분 출력(문자열)
     , TO_CHAR(SYSDATE, 'SS') AS "TO_CHAR(SYSDATE, 'SS')" --초 출력(문자열)
     , TO_NUMBER(TO_CHAR(SYSDATE, 'YYYY')) AS "TO_NUMBER(TO_CHAR(SYSDATE, 'YYYY'))" --년 출력
     , TO_NUMBER(TO_CHAR(SYSDATE, 'MM')) AS "TO_NUMBER(TO_CHAR(SYSDATE, 'MM'))" --월  출력
     , TO_NUMBER(TO_CHAR(SYSDATE, 'DD')) AS "TO_NUMBER(TO_CHAR(SYSDATE, 'DD'))" --일 출력
     , TO_NUMBER(TO_CHAR(SYSDATE, 'HH24')) AS "TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'))" --시 출력
     , TO_NUMBER(TO_CHAR(SYSDATE, 'MI')) AS "TO_NUMBER(TO_CHAR(SYSDATE, 'MI'))" --분 출력
     , TO_NUMBER(TO_CHAR(SYSDATE, 'SS')) AS "TO_NUMBER(TO_CHAR(SYSDATE, 'SS'))" --초 출력
  FROM DUAL
;
```

Name	Value
SYSDATE	2021-11-21 13:40:19.000
EXTRACT(YEAR FROM SYSDATE)	2021
EXTRACT(MONTH FROM SYSDATE)	11
EXTRACT(DAY FROM SYSDATE)	21
TO_CHAR(SYSDATE, 'YYYY')	2021
TO_CHAR(SYSDATE, 'MM')	11
TO_CHAR(SYSDATE, 'DD')	21
TO_CHAR(SYSDATE, 'HH24')	13
TO_CHAR(SYSDATE, 'MI')	40
TO_CHAR(SYSDATE, 'SS')	19
TO_NUMBER(TO_CHAR(SYSDATE, 'YYYY'))	2021
TO_NUMBER(TO_CHAR(SYSDATE, 'MM'))	11
TO_NUMBER(TO_CHAR(SYSDATE, 'DD'))	21
TO_NUMBER(TO_CHAR(SYSDATE, 'HH24'))	13
TO_NUMBER(TO_CHAR(SYSDATE, 'MI'))	40
TO_NUMBER(TO_CHAR(SYSDATE, 'SS'))	19

➢ 날짜형 데이터 연산

```
SELECT
       SYSDATE AS "SYSDATE" --현재 년월일 시분초
     , SYSDATE - 1 AS "SYSDATE - 1" --1일을 뺀 년월일 시분초
     , SYSDATE - (1/24) AS "SYSDATE - (1/24)" --1시간을뺀 년월일 시분초
     , SYSDATE - (1/24/60) AS "SYSDATE - (1/24/60)" --1분을뺀 년월일 시분초
     , SYSDATE - (1/24/60/60) AS "SYSDATE - (1/24/60/60)" --1초을뺀 년월일 시분초
     , SYSDATE - (1/24/60/60) * 10 AS "SYSDATE - (1/24/60/60) * 10" --10초을뺀 년월일 시분초
     , SYSDATE - (1/24/60/60) * 30 AS "SYSDATE - (1/24/60/60) * 30" --30초을뺀 년월일 시분초
  FROM DUAL
;
```

Name	Value
SYSDATE	2020-06-21 18:28:54
SYSDATE - 1	2020-06-20 18:28:54
SYSDATE - (1/24)	2020-06-21 17:28:54
SYSDATE - (1/24/60)	2020-06-21 18:27:54
SYSDATE - (1/24/60/60)	2020-06-21 18:28:53
SYSDATE - (1/24/60/60) * 10	2020-06-21 18:28:44
SYSDATE - (1/24/60/60) * 30	2020-06-21 18:28:24

➢ 데이터 변환의 종류

종류	설명
명시적 형 변환	데이터 변환 형 함수로 데이터 유형을 변환하도록 명시해주는 경우
암시적 형 변환	DBMS가 자동으로 데이터 유형을 변환하는 경우

➢ 데이터 변환 함수 사용 예시

```
SELECT
       TO_CHAR(SYSDATE, 'YYYY/MM/DD') "TO_CHAR(SYSDATE, 'YYYY/MM/DD')"
     , TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS') "TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS')"
     , TO_CHAR(10.25, '$999,999,999.99') AS "TO_CHAR(10.25, '$999,999,999.99')"
     , TO_CHAR(12500, 'L999,999,999') AS "TO_CHAR(12500, 'L999,999,999')"
     , TO_NUMBER('100') + TO_NUMBER('100') AS "TO_NUMBER('100') + TO_NUMBER('100')"
  FROM DUAL
;
```

```
Name                                          |Value
|
----------------------------------------------+-------------
---------+
TO_CHAR(SYSDATE, 'YYYY/MM/DD')                |2021/11/21
|
TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS')|2021/11/21
01:54:56    |
TO_CHAR(10.25, '$999,999,999.99')             |
$10.25       |
TO_CHAR(12500, 'L999,999,999')                |
￦12,500|
TO_NUMBER('100') + TO_NUMBER('100')           |200
|
```

➢ 단일 행 CASE표현의 종류

종류	설명
CASE WHEN 조건 THEN 값 혹은 SQL문 ELSE 값 혹은 SQL문 END	조건이 맞으면 THEN절 수행하고 그렇지 않으면 ELSE절을 수행한다.
DECODE(표현식 혹은 기준컬럼, 비교값1, 결과값1, 비교값2, 결과값2, 디폴트값)	표현식 혹은 기준컬럼과 비교하여 비교값1과 같으면 결과값1을 출력하고, 표현식 혹은 기준컬럼이 비교값2랑 같으면 결과값2를 가져오고, 그렇지않으면 디폴트 값을 결과로 출력한다.

```
SELECT CASE WHEN SAL_CD = '100001' THEN '기본급'
            WHEN SAL_CD = '100002' THEN '상여급'
            WHEN SAL_CD = '100003' THEN '특별상여급'
            WHEN SAL_CD = '100004' THEN '야근수당'
            WHEN SAL_CD = '100005' THEN '주말수당'
            WHEN SAL_CD = '100006' THEN '점심식대'
            WHEN SAL_CD = '100007' THEN '복지포인트'
            ELSE '유효하지않음'
            END SAL_NM
  FROM TB_SAL
;
```

SAL_NM
기본급
상여급
특별상여급
야근수당
주말수당
점심식대
복지포인트

```
SELECT DECODE(SAL_CD, '100001', '기본급', '100002', '상여급', '기타')  AS SAL_NM
  FROM TB_SAL
;
```

SAL_NM
기본급
상여급
기타
기타
기타
기타
기타

➢ NULL 연산 결과

연산	결과
NULL+2	NULL
NULL-2	NULL
NULL*2	NULL
NULL/2	NULL

➢ 널 관련 함수 사용 예시

```
SELECT NVL(DIRECT_MANAGER_EMP_NO, '최상위자') AS 관리자 --관리자사원번호가 NULL이면 "최상위자"로 출력
  FROM TB_EMP
 WHERE DIRECT_MANAGER_EMP_NO IS NULL;
```

관리자
최상위자

```
SELECT NVL(UPPER_DEPT_CD , '최상위부서') AS 상위부서
  FROM TB_DEPT
 WHERE UPPER_DEPT_CD  IS NULL;
```

상위부서
최상위부서

```
SELECT NVL(MAX(EMP_NM), '존재안함') AS EMP_NM
FROM TB_EMP
WHERE EMP_NM = '김회장';
```

EMP_NM
김회장

```
SELECT NVL(MAX(EMP_NM), '존재안함') AS EMP_NM
FROM TB_EMP
WHERE EMP_NM = '박찬호';
```

EMP_NM
존재안함

❖ MAX함수를 쓰면 공집합인 경우에도 한개의 ROW가 나옴

➢ 널 관련 함수 사용 예시 - 계속

```
SELECT NULLIF('박찬호', '박지성') --같지않으면 박찬호 리턴
  FROM DUAL
;
```

박찬호

```
SELECT NVL(NULLIF('박찬호', '박찬호'), '같으면널')
  FROM DUAL
;
```

같으면널

```
SELECT COALESCE(NULL, NULL, 0) AS SAL
FROM DUAL;
```

0

❖ 널이 아닌 첫번째 수식/값을 리턴 함

```
SELECT COALESCE(5000, NULL, 0) AS SAL
  FROM DUAL;
```

5000

```
SELECT COALESCE(NULL, 6000, 0) AS SAL
  FROM DUAL;
```

6000

Chapter 03. SQL 기본

3-7. GROUP BY, HAVING 절

➢ 집계 함수

① 여러 행들의 그룹이 모여서 그룹당 단 하나의 결과를 돌려주는 함수이다.
② GROUP BY 절은 행들을 소 그룹화 한다.
③ SELECT 절, HAVING 절, ORDER BY 절에 사용할 수 있다.

➢ ALL과 DISTINCT

항목	설명
ALL	DEFAULT옵션 임 생략 가능
DISTINCT	유일한 값을 출력함

➢ 집계 함수의 종류

항목	설명
COUNT(*)	NULL값을 포함한 행의 수를 출력
COUNT(표현식)	표현식의 값이 NULL아닌 행의 수를 출력
SUM(표현식)	표현식이 NULL값인 것을 제외한 합계를 출력
AVG(표현식)	표현식이 NULL값인 것을 제외한 평균을 출력
MAX(표현식)	표현식이 NULL값인 것을 제외한 최대값을 출력
MIN(표현식)	표현식이 NULL값인 것을 제외한 최소값을 출력
STDDEV(표현식)	표현식이 NULL값인 것을 제외한 표준편차를 출력
VARIAN(표현식)	표현식이 NULL값인 것을 제외한 분산을 출력

➢ 집계 함수 사용 예시

```
SELECT MAX(BIRTH_DE)
     , MIN(BIRTH_DE)
     , COUNT(*)
  FROM TB_EMP;
```

MAX(BIRTH_DE)	MIN(BIRTH_DE)	COUNT(*)
19980715	19651105	41

❖ 사원들 중 가장 생일이 늦은 사원과 빠른 사원 그리고 총 사원수를 리턴 함

➢ GROUP BY 절

① GROUP BY 절을 통해 소그룹 별 기준을 정한 후, SELECT 절에 집계 함수를 사용한다.
② 집계 함수의 통계 정보는 NULL 값을 가진 행을 제외하고 수행한다.
③ GROUP BY 절에서는 SELECT 절과는 달리 ALIAS 명을 사용할 수 없다.
④ 집계 함수는 WHERE 절에는 올 수 없다. (집계 함수를 사용할 수 있는 GROUP BY 절보다 WHERE 절이 먼저 수행된다)
⑤ WHERE 절은 전체 데이터를 GROUP으로 나누기 전에 행들을 미리 제거시킨다.
⑥ HAVING 절은 GROUP BY 절의 기준 항목이나 소그룹의 집계 함수를 이용한 조건을 표시할 수 있다.
⑦ GROUP BY 절에 의한 소 그룹별로 만들어진 집계 데이터 중, HAVING 절에서 제한 조건을 두어 조건을 만족하는 내용만 출력한다.
⑧ HAVING 절은 일반적으로 GROUP BY 절 뒤에 위치한다.

➢ GROUP BY 절 사용 예시

```
SELECT A.DEPT_CD
     , (SELECT L.DEPT_NM FROM TB_DEPT L WHERE L.DEPT_CD = A.DEPT_CD) AS DEPT_NM
     , MAX(A.BIRTH_DE) AS "가장 늦은 생년월일"
     , MIN(A.BIRTH_DE) AS "가장 빠른 생년월일"
     , COUNT(*) AS "직원수"
  FROM TB_EMP A
GROUP BY A.DEPT_CD
ORDER BY A.DEPT_CD ;
```

❖ 사원들 중 부서 코드 별 가장 늦은 생일, 가장 빠른 생일, 직원 수를 출력한다.

DEPT_CD	DEPT_NM	가장 늦은 생년월일	가장 빠른 생년월일	직원수
100001	운영본부	19840612	19840612	1
100002	지원팀	19930129	19870623	4
100003	기획팀	19941228	19840715	4
100004	디자인팀	19951122	19871201	4
100005	플랫폼사업본부	19780213	19780213	1
100006	데이터팀	19980715	19820629	4
100007	개발팀	19951122	19871201	4
100008	솔루션사업본부	19780213	19780213	1
100009	운영팀	19980529	19690524	4
100010	개발팀	19970627	19811201	4
100011	신사업본부	19771202	19771202	1
100012	인공지능팀	19980529	19690524	4
100013	빅데이터팀	19940709	19871205	4
999999	회장실	19651105	19651105	1

```
SELECT A.DEPT_CD
     , (SELECT L.DEPT_NM
          FROM TB_DEPT L
         WHERE L.DEPT_CD = A.DEPT_CD) AS DEPT_NM
     , MAX(A.BIRTH_DE) AS "가장 늦은 생년월일"
     , MIN(A.BIRTH_DE) AS "가장 빠른 생년월일"
     , COUNT(*) AS "직원수"
 FROM TB_EMP A
ORDER BY A.DEPT_CD ;
```

```
SQL Error [937] [42000]: ORA-00937: 단일 그룹의 그룹 함수가 아닙니다
```

❖ DEPT_CD컬럼을 SELECT절에 기재하면서 집계 함수를 쓰면 그룹핑 대상이 없기 때문에 SQL 에러가 발생한다.

➢ HAVING 절

① WHERE절에서는 집계 함수를 쓸 수 없다.
② 집계된 결과 집합을 기준으로 특정 조건을 주고 싶은 경우 HAVING절을 이용하면 된다.
③ HAVING 절은 WHERE 절과 비슷하지만 그룹을 나타내는 결과 집합의 행에 조건이 적용된다는 점에서 차이가 있다.

➢ HAVING 사용 예시

```
SELECT A.DEPT_CD
     , (SELECT L.DEPT_NM FROM TB_DEPT L WHERE L.DEPT_CD = A.DEPT_CD) AS DEPT_NM
     , MAX(A.BIRTH_DE) AS "가장 늦은 생년월일"
     , MIN(A.BIRTH_DE) AS "가장 빠른 생년월일"
     , COUNT(*) AS "직원수"
 FROM TB_EMP A
 GROUP BY A.DEPT_CD
HAVING(COUNT(*)) > 1
ORDER BY A.DEPT_CD ;
```

DEPT_CD	DEPT_NM	가장 늦은 생년월일	가장 빠른 생년월일	직원수
100002	지원팀	19930129	19870623	4
100003	기획팀	19941228	19840715	4
100004	디자인팀	19951122	19871201	4
100006	데이터팀	19980715	19820629	4
100007	개발팀	19951122	19871201	4
100009	운영팀	19980529	19690524	4
100010	개발팀	19970627	19811201	4
100012	인공지능팀	19980529	19690524	4
100013	빅데이터팀	19940709	19871205	4

❖ 사원들 중 부서 코드 별 가장 늦은 생일, 가장 빠른 생일, 직원 수를 출력한다.
❖ 그룹핑 한 결과에서 COUNT(*)가 1보다 큰 집합만 출력한다.
❖ 즉 부서의 직원수가 1명을 초과하는 부서의 집계 데이터를 출력한 것이다.

➢ HAVING 사용 예시 - 2

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
GROUP BY A.EMP_NO
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4700000
ORDER BY A.EMP_NO
;
```

❖ 2019년도의 급여 내역 중에서 직원 별로 평균 급여액이 470만원 이상인 직원들의
❖ 직원번호, 직원명, 최대급여액수, 최소급여액수, 평균 급여 액수를 출력하는 SQL문

EMP_NO	EMP_NM	MAX_PAY_AMT	MIN_PAY_AMT	AVG_PAY_AMT
1000000014	이관심	5610000	2890000	4890833.33
1000000036	박여진	5990000	2790000	4765833.33

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , SUM(A.PAY_AMT) AS SUM_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
HAVING SUM(A.PAY_AMT) > 55000000
GROUP BY A.EMP_NO
ORDER BY SUM_PAY_AMT DESC
;
```

EMP_NO	EMP_NM	SUM_PAY_AMT
1000000014	이관심	58690000
1000000036	박여진	57190000
1000000025	박호진	56220000
1000000034	김익정	55390000
1000000019	박정혜	55270000

❖ 2019년도의 급여 내역 중 급여의 합계가 (연봉) 5500만원을 초과하는 직원의 직원번호, 직원명, 합계금여액수를 출력하는 SQL문

➢ CASE WHEN문 사용

```
SELECT
      SUM(CASE WHEN BIRTH_DE LIKE '195%' THEN 1 ELSE 0 END) AS "1950년대생"
    , SUM(CASE WHEN BIRTH_DE LIKE '196%' THEN 1 ELSE 0 END) AS "1960년대생"
    , SUM(CASE WHEN BIRTH_DE LIKE '197%' THEN 1 ELSE 0 END) AS "1970년대생"
    , SUM(CASE WHEN BIRTH_DE LIKE '198%' THEN 1 ELSE 0 END) AS "1980년대생"
    , SUM(CASE WHEN BIRTH_DE LIKE '199%' THEN 1 ELSE 0 END) AS "1990년대생"
    , COUNT(*) CNT
 FROM TB_EMP;
```

❖ 직원들의 정보 중 몇 십 년대생인지에 대한 카운트를 구하는 SQL문
❖ BIRTH_DE의 값이 매칭되면 1을 더해서 카운트를 하고 그렇지 않으면 0을 더해서 카운트를 안함

1950년대생	1960년대생	1970년대생	1980년대생	1990년대생	CNT
0	3	5	14	19	41

```
SELECT
      DEPT_CD
    , (SELECT L.DEPT_NM FROM TB_DEPT L WHERE L.DEPT_CD = A.DEPT_CD) AS DEPT_NM
    , SUM(CASE WHEN A.BIRTH_DE LIKE '195%' THEN 1 ELSE 0 END) AS "1950년대생"
    , SUM(CASE WHEN A.BIRTH_DE LIKE '196%' THEN 1 ELSE 0 END) AS "1960년대생"
    , SUM(CASE WHEN A.BIRTH_DE LIKE '197%' THEN 1 ELSE 0 END) AS "1970년대생"
    , SUM(CASE WHEN A.BIRTH_DE LIKE '198%' THEN 1 ELSE 0 END) AS "1980년대생"
    , SUM(CASE WHEN A.BIRTH_DE LIKE '199%' THEN 1 ELSE 0 END) AS "1990년대생"
    , COUNT(*) CNT
 FROM TB_EMP A
GROUP BY A.DEPT_CD
ORDER BY A.DEPT_CD;
```

❖ 부서별로 직원들이 몇 십년대생인지에 대한 카운트를 함
❖ BIRTH_DE의 값이 매칭되면 1을 더해서 카운트를 하고 그렇지 않으면 0을 더해서 카운트를 안함

DEPT_CD	DEPT_NM	1950년대생	1960년대생	1970년대생	1980년대생	1990년대생	CNT
100001	운영본부	0	0	0	1	0	1
100002	지원팀	0	0	0	2	2	4
100003	기획팀	0	0	0	3	1	4
100004	디자인팀	0	0	0	1	3	4
100005	플랫폼사업본부	0	0	1	0	0	1
100006	데이터팀	0	0	0	2	2	4
100007	개발팀	0	0	0	1	3	4
100008	솔루션사업본부	0	0	1	0	0	1
100009	운영팀	0	1	1	1	1	4
100010	개발팀	0	0	0	1	3	4
100011	신사업본부	0	0	1	0	0	1
100012	인공지능팀	0	1	1	1	1	4
100013	빅데이터팀	0	0	0	1	3	4
999999	회장실	0	1	0	0	0	1

➢ 집계 함수와 널

```
SELECT SUM(NUM), AVG(NUM), MAX(NUM), MIN(NUM)
FROM
(
      SELECT NULL AS NUM FROM DUAL
      UNION ALL
      SELECT 10 AS NUM FROM DUAL
      UNION ALL
      SELECT 20 AS NUM FROM DUAL
      UNION ALL
      SELECT 30 AS NUM FROM DUAL
      UNION ALL
      SELECT 40 AS NUM FROM DUAL
)
```

SUM(NUM)	AVG(NUM)	MAX(NUM)	MIN(NUM)
100	25	40	10

- ❖ 널은 집계 함수의 계산에 포함되지 않음
- ❖ AVG(NUM)의 값이 25임 즉 집계 함수는 평균을 구할 때 관여하지 않았음

Chapter 03. SQL 기본

3-8 ORDER BY 절

➢ ORDER BY 정렬

① ORDER BY 절은 SQL 문장으로 조회된 데이터들을 다양한 목적에 맞게 특정 컬럼을 기준으로 정렬하여 출력하는데 사용
② ORDER BY 절에 컬럼(Column)명 대신에 SELECT 절에서 사용한 ALIAS 명이나 컬럼 순서를 나타내는 정수도 사용 가능
③ 별도로 정렬 방식을 지정하지 않으면 기본적으로 오름차순이 적용되며, SQL 문장의 제일 마지막에 위치
④ 숫자 형 데이터 타입은 오름차순으로 정렬했을 경우에 가장 작은 값부터 출력, 날짜 형 데이터 타입은 오름차순으로 정렬했을 경우 날짜 값이 가장 빠른 값이 먼저 출력
⑤ NULL 값을 가장 큰 값으로 간주하여 오름차순으로 정렬했을 경우에는 가장 마지막에, 내림차순으로 정렬했을 경우에는 가장 먼저 위치

➢ ORDER BY 실습

```
SELECT
       A.CERTI_CD
     , A.CERTI_NM
     , A.ISSUE_INSTI_NM
  FROM TB_CERTI A
 ORDER BY A.CERTI_NM;
```

❖ 자격증명을 기준으로 정렬
❖ 오름차순으로 정렬됨(디폴트)

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100006	ADP	한국데이터베이스진흥원
100005	ADSP	한국데이터베이스진흥원
100004	DAP	한국데이터베이스진흥원
100003	DASP	한국데이터베이스진흥원
100013	OCJD	오라클
100012	OCJP	오라클
100011	OCM	오라클
100010	OCP	오라클
100014	OCWCD	오라클
100001	SQLD	한국데이터베이스진흥원
100002	SQLP	한국데이터베이스진흥원
100009	리눅스마스터1급	한국정보통신진흥협회
100008	리눅스마스터2급	한국정보통신진흥협회
100016	워드프로세서1급	대한상공회의소
100015	워드프로세서2급	대한상공회의소
100020	정보관리기술사	한국산업인력공단
100019	정보시스템감리사	한국정보화진흥원
100007	정보처리기사	한국산업인력공단
100018	컴퓨터활용능력1급	대한상공회의소
100017	컴퓨터활용능력2급	대한상공회의소

➢ ORDER BY 실습 – 역순 정렬

```
SELECT
       A.CERTI_CD
     , A.CERTI_NM
     , A.ISSUE_INSTI_NM
  FROM TB_CERTI A
ORDER BY A.CERTI_NM DESC;
```

❖ 자격증명을 기준으로 정렬
❖ 내림차순으로 정렬함

CERTI_CD	CERTI_NM	ISSUE_INSTI_NM
100017	컴퓨터활용능력2급	대한상공회의소
100018	컴퓨터활용능력1급	대한상공회의소
100007	정보처리기사	한국산업인력공단
100019	정보시스템감리사	한국정보화진흥원
100020	정보관리기술사	한국산업인력공단
100015	워드프로세서2급	대한상공회의소
100016	워드프로세서1급	대한상공회의소
100008	리눅스마스터2급	한국정보통신진흥협회
100009	리눅스마스터1급	한국정보통신진흥협회
100002	SQLP	한국데이터베이스진흥원
100001	SQLD	한국데이터베이스진흥원
100014	OCWCD	오라클
100010	OCP	오라클
100011	OCM	오라클
100012	OCJP	오라클
100013	OCJD	오라클
100003	DASP	한국데이터베이스진흥원
100004	DAP	한국데이터베이스진흥원
100005	ADSP	한국데이터베이스진흥원
100006	ADP	한국데이터베이스진흥원

➢ ORDER BY 실습 – 널 포함 정렬

```
SELECT
       A.EMP_NO
     , A.EMP_NM
     , A.BIRTH_DE
     , A.ADDR
     , A.TEL_NO
     , A.DIRECT_MANAGER_EMP_NO
  FROM TB_EMP A
WHERE BIRTH_DE LIKE '196%'
ORDER BY A.DIRECT_MANAGER_EMP_NO DESC;
```

❖ 오라클은 널 값이 가장 크다고 인식한다
❖ 즉 DIRECT_MANAGER_EMP_NO 컬럼이 널인 행이 가장 위에 위치하였다.

EMP_NO	EMP_NM	BIRTH_DE	ADDR	TEL_NO	DIRECT_MANAGER_EMP_NO
9999999999	김회장	19651105	경기도 용인시 수지구 죽전1동 215	010-3212-7777	(null)
1000000035	최창수	19690524	경기도 고양시 일산서구 백석동 278	010-1147-2158	1000000033
1000000026	김길정	19690524	경기도 고양시 일산서구 백석동 666	010-1147-2158	1000000024

➢ SELECT문의 실행 순서

순번	절	설명
1	FROM절	조회 테이블을 참조한다.
2	WHERE절	조회 대상 행을 조회한다.
3	GROUP BY절	대상 행을 그룹화 한다.
4	HAVING절	그룹화한 값에서 조건에 맞는 것을 출력한다.
5	SELECT절	SELECT절에 기재한 컬럼이나 식을 계산한다.
6	ORDER BY절	출력되는 결과 집합을 정렬한다.

➢ ORDER BY 실습 - SELECT절에 존재하지 않는 컬럼으로 정렬

```
SELECT
       A.CERTI_CD
     , A.ISSUE_INSTI_NM
  FROM TB_CERTI A
ORDER BY A.CERTI_NM DESC;
```

❖ SELECT절에 기재하지 않은 컬럼을 기준으로 ORDER BY해도 정상적으로 실행됨
❖ SELECT절에 CERTI_NM컬럼은 존재하지 않음
❖ ORDER BY 절에 CERTI_NM컬럼으로 ORDER BY함 정상 수행됨

CERTI_CD	ISSUE_INSTI_NM
100017	대한상공회의소
100018	대한상공회의소
100007	한국산업인력공단
100019	한국정보화진흥원
100020	한국산업인력공단
100015	대한상공회의소
100016	대한상공회의소
100008	한국정보통신진흥협회
100009	한국정보통신진흥협회
100002	한국데이터베이스진흥원
100001	한국데이터베이스진흥원
100014	오라클
100010	오라클
100011	오라클
100012	오라클
100013	오라클
100003	한국데이터베이스진흥원
100004	한국데이터베이스진흥원
100005	한국데이터베이스진흥원
100006	한국데이터베이스진흥원

➢ ORDER BY 실습 - HAVING절의 결과를 정렬

```
SELECT A.DEPT_CD
     , (SELECT L.DEPT_NM FROM TB_DEPT L WHERE L.DEPT_CD = A.DEPT_CD) AS DEPT_NM
     , MAX(A.BIRTH_DE) AS "가장 늦은 생년월일"
     , MIN(A.BIRTH_DE) AS "가장 빠른 생년월일"
     , COUNT(*) AS "직원수"
  FROM TB_EMP A
 GROUP BY A.DEPT_CD
HAVING(COUNT(*)) > 1
ORDER BY A.DEPT_CD ;
```

DEPT_CD	DEPT_NM	가장 늦은 생년월일	가장 빠른 생년월일	직원수
100002	지원팀	19930129	19870623	4
100003	기획팀	19941228	19840715	4
100004	디자인팀	19951122	19871201	4
100006	데이터팀	19980715	19820629	4
100007	개발팀	19951122	19871201	4
100009	운영팀	19980529	19690524	4
100010	개발팀	19970627	19811201	4
100012	인공지능팀	19980529	19690524	4
100013	빅데이터팀	19940709	19871205	4

```
SELECT A.DEPT_CD
     , (SELECT L.DEPT_NM FROM TB_DEPT L WHERE L.DEPT_CD = A.DEPT_CD) AS DEPT_NM
     , MAX(A.BIRTH_DE) AS "가장 늦은 생년월일"
     , MIN(A.BIRTH_DE) AS "가장 빠른 생년월일"
     , COUNT(*) AS "직원수"
 FROM TB_EMP A
GROUP BY A.DEPT_CD
HAVING(COUNT(*)) > 1
ORDER BY A.EMP_NM ;
```

SQL Error [979] [42000]: ORA-00979: GROUP BY 표현식이 아닙니다.

❖ GROUP BY 절과 같이 쓰이는 ORDER BY는 일반 컬럼을 사용 할 수 없음

➢ ORDER BY와 부분 범위 처리

```
SELECT A.SAL_HIS_NO
     , A.PAY_DE
     , A.PAY_AMT
     , A.EMP_NO
  FROM SQLD.TB_SAL_HIS A
 WHERE A.PAY_DE >= '20200501'
   AND A.PAY_DE <= '20200531'
   AND ROWNUM <= 10
 ORDER BY A.PAY_AMT DESC
 ;
```

SAL_HIS_NO	PAY_DE	PAY_AMT	EMP_NO
20200701000048	20200525	5000000	1000000007
20200701000045	20200525	4910000	1000000004
20200701000044	20200525	4440000	1000000003
20200701000046	20200525	4320000	1000000005
20200701000050	20200525	4100000	1000000009
20200701000043	20200525	3760000	1000000002
20200701000042	20200525	3700000	1000000001
20200701000049	20200525	3600000	1000000008
20200701000051	20200525	2910000	1000000010
20200701000047	20200525	2510000	1000000006

❖ 급여가 높은 10건의 데이터가 나오는 것이 아니라 랜덤으로 10건이 나오는데 나온 결과 집합 10건은 정렬됨

```
SELECT A.SAL_HIS_NO
     , A.PAY_DE
     , A.PAY_AMT
     , A.EMP_NO
  FROM
     (
      SELECT
             A.SAL_HIS_NO
           , A.PAY_DE
           , A.PAY_AMT
           , A.EMP_NO
        FROM SQLD.TB_SAL_HIS A
       WHERE A.PAY_DE >= '20200501'
         AND A.PAY_DE <= '20200531'
       ORDER BY A.PAY_AMT DESC
     ) A
WHERE ROWNUM <= 10
;
```

SAL_HIS_NO	PAY_DE	PAY_AMT	EMP_NO
20200701000065	20200525	5950000	1000000024
20200701000075	20200525	5890000	1000000034
20200701000073	20200525	5570000	1000000032
20200701000082	20200525	5470000	9999999999
20200701000072	20200525	5150000	1000000031
20200701000078	20200525	5140000	1000000037
20200701000056	20200525	5100000	1000000015
20200701000067	20200525	5080000	1000000026
20200701000071	20200525	5020000	1000000030
20200701000048	20200525	5000000	1000000007

❖ 급여가 높은 순으로 10건이 리턴 됨

Chapter 03. SQL 기본

3-9. 조인

➢ 조인이란?

① 두 개 이상의 테이블들을 연결 또는 결합하여 데이터를 출력하는 것을 조인이라고 하며 일반적으로 사용 되는 SQL문의 상당수가 조인으로 이루어져 있다.
② 일반적인 경우 PRIMARY KEY와 FOREIGN KEY의 값 연관에 의해 조인이 이루어지며 PK, FK관계와는 별도로 일반 컬럼 끼리 조인이 이루어지는 경우도 있다.

➢ 2개의 테이블 조인

```
SELECT A.EMP_NO
     , A.EMP_NM
     , A.DEPT_CD
     , B.DEPT_NM
  FROM TB_EMP A
     , TB_DEPT B
WHERE A.DEPT_CD = B.DEPT_CD
  AND B.DEPT_NM = '지원팀'
;
```

❖ 부서명이 지원팀인 직원의 사원번호, 사원명, 부서코드, 부서명을 출력

EMP_NO	EMP_NM	DEPT_CD	DEPT_NM
1000000005	김현희	100002	지원팀
1000000002	이현승	100002	지원팀
1000000003	이정수	100002	지원팀
1000000004	이승준	100002	지원팀

➢ 3개의 테이블 조인

```
SELECT
       A.EMP_NO
     , A.EMP_NM
     , A.DEPT_CD
     , B.DEPT_NM
     , C.CERTI_CD
  FROM TB_EMP A
     , TB_DEPT B
     , TB_EMP_CERTI C
WHERE A.DEPT_CD = B.DEPT_CD
  AND B.DEPT_NM = '지원팀'
  AND A.EMP_NO = C.EMP_NO
;
```

❖ 부서명이 지원팀인 직원의 사원번호, 사원명, 부서코드, 부서명, 보유자격증 코드를 출력

EMP_NO	EMP_NM	DEPT_CD	DEPT_NM	CERTI_CD
1000000002	이현승	100002	지원팀	100010
1000000002	이현승	100002	지원팀	100009
1000000002	이현승	100002	지원팀	100005
1000000003	이정수	100002	지원팀	100010
1000000003	이정수	100002	지원팀	100010
1000000003	이정수	100002	지원팀	100007
1000000004	이승준	100002	지원팀	100014
1000000004	이승준	100002	지원팀	100005
1000000004	이승준	100002	지원팀	100017
1000000005	김현희	100002	지원팀	100004
1000000005	김현희	100002	지원팀	100017
1000000005	김현희	100002	지원팀	100002

➢ 4개의 테이블 조인

```
SELECT
       A.EMP_NO
     , A.EMP_NM
     , A.DEPT_CD
     , B.DEPT_NM
     , C.CERTI_CD
     , D.CERTI_NM
  FROM TB_EMP A
     , TB_DEPT B
     , TB_EMP_CERTI C
     , TB_CERTI D
WHERE A.DEPT_CD = B.DEPT_CD
  AND B.DEPT_NM = '지원팀'
  AND A.EMP_NO = C.EMP_NO
  AND C.CERTI_CD = D.CERTI_CD
;
```

❖ 부서명이 지원팀인 직원의 사원번호, 사원명, 부서코드, 부서명, 보유자격증 코드 및 자격증명을 출력

EMP_NO	EMP_NM	DEPT_CD	DEPT_NM	CERTI_CD	CERTI_NM
1000000005	김현희	100002	지원팀	100002	SQLP
1000000005	김현희	100002	지원팀	100004	DAP
1000000002	이현승	100002	지원팀	100005	ADSP
1000000004	이승준	100002	지원팀	100005	ADSP
1000000003	이정수	100002	지원팀	100007	정보처리기사
1000000002	이현승	100002	지원팀	100009	리눅스마스터1급
1000000002	이현승	100002	지원팀	100010	OCP
1000000003	이정수	100002	지원팀	100010	OCP
1000000003	이정수	100002	지원팀	100010	OCP
1000000004	이승준	100002	지원팀	100014	OCWCD
1000000004	이승준	100002	지원팀	100017	컴퓨터활용능력2급
1000000005	김현희	100002	지원팀	100017	컴퓨터활용능력2급

Chapter 03. SQL 기본

3-10. 연습문제

문제 1. 다음 중 DML의 종류가 아닌 것은?

① INSERT
② UPDATE
③ GRANT
④ SELECT

문제 2. TB_EMP 테이블에 CHAR 1바이트 크기의 본사근무여부(HEAD_WORK_YN) 컬럼을 추가하고자 한다. 올바른 DDL문은 무엇인가? (오라클 기준)

① ALTER TABLE SQLD.TB_EMP ADD HEAD_WORK_YN DATATYPE CHAR(1);
② ALTER TABLE SQLD.TB_EMP ADD HEAD_WORK_YN CHAR(1);
③ ALTER TABLE SQLD.TB_EMP ADD (HEAD_WORK_YN CHAR(1));
④ ALTER TABLE SQLD.TB_EMP ADD COLUMN(HEAD_WORK_YN CHAR(1));

문제 3. 다음 중 트랜잭션의 4대 특성이 아닌 것은 무엇인가?

① 원자성(Atomicity)
② 일관성(Consistency)
③ 지속성(Durability)
④ 동시성(Concurrency)

문제 1. 다음 중 DML의 종류가 아닌 것은?

① INSERT
② UPDATE
③ GRANT
④ SELECT

문제 2. TB_EMP 테이블에 CHAR 1바이트 크기의 본사근무여부(HEAD_WORK_YN) 컬럼을 추가하고자 한다. 올바른 DDL문은 무엇인가? (오라클 기준)

① ALTER TABLE SQLD.TB_EMP ADD HEAD_WORK_YN DATATYPE CHAR(1);
② ALTER TABLE SQLD.TB_EMP ADD HEAD_WORK_YN CHAR(1);
③ ALTER TABLE SQLD.TB_EMP ADD (HEAD_WORK_YN CHAR(1));
④ ALTER TABLE SQLD.TB_EMP ADD COLUMN(HEAD_WORK_YN CHAR(1));

문제 3. 다음 중 트랜잭션의 4대 특성이 아닌 것은 무엇인가?

① 원자성(Atomicity)
② 일관성(Consistency)
③ 지속성(Durability)
④ 동시성(Concurrency)

문제 4. 2019년의 평균급여액수가 450만원 이상인 직원을 출력하는 SQL문이다. 올바른 SQL문을 모두 고르시오.

①

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4500000
ORDER BY A.EMP_NO;
```

②

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
GROUP BY A.EMP_NO
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4500000
ORDER BY A.EMP_NO;
```

③

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
  AND ROUND(AVG(A.PAY_AMT), 2) >= 4500000
GROUP BY A.EMP_NO
ORDER BY A.EMP_NO;
```

④

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4500000
GROUP BY A.EMP_NO
ORDER BY A.EMP_NO;
```

문제 4. 2019년의 평균급여액수가 450만원 이상인 직원을 출력하는 SQL문이다. 올바른 SQL문을 모두 고르시오.

①

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4500000
ORDER BY A.EMP_NO;
```

②

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
GROUP BY A.EMP_NO
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4500000
ORDER BY A.EMP_NO;
```

③

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
  AND ROUND(AVG(A.PAY_AMT), 2) >= 4500000
GROUP BY A.EMP_NO
ORDER BY A.EMP_NO;
```

④

```
SELECT
       A.EMP_NO
     , (SELECT L.EMP_NM FROM TB_EMP L WHERE L.EMP_NO = A.EMP_NO) AS EMP_NM
     , MAX(A.PAY_AMT) AS MAX_PAY_AMT
     , MIN(A.PAY_AMT) AS MIN_PAY_AMT
     , ROUND(AVG(A.PAY_AMT), 2) AS AVG_PAY_AMT
FROM TB_SAL_HIS A
WHERE A.PAY_DE BETWEEN '20190101' AND '20191231'
HAVING(ROUND(AVG(A.PAY_AMT), 2)) >= 4500000
GROUP BY A.EMP_NO
ORDER BY A.EMP_NO;
```

문제 5. 다음 중 각 SQL문의 실행 결과를 올바르게 설명한 것을 2개 고르시오. (DUAL테이블에는 단, 한개의 레코드가 있다고 가정한다.)

①

```
SELECT 1, 2 FROM DUAL WHERE 1=2;
-> 실행 시 에러가 발생한다.
```

②

```
SELECT NVL(1, 0) FROM DUAL WHERE 1=2;
-> 실행 시 공집합으로 나온다. (실행 결과가 없다)
```

③

```
SELECT NVL(MAX(1), 1) FROM DUAL WHERE 1=2;
-> 실행 시 공집합으로 나온다. (실행 결과가 없다)
```

④

```
SELECT MAX(NVL(1, 1)) FROM DUAL WHERE 1=2;
-> 실행 시 NULL이 출력된다.
```

문제 5. 다음 중 각 SQL문의 실행 결과를 올바르게 설명한 것을 2개 고르시오. (DUAL테이블에는 단, 한개의 레코드가 있다고 가정한다.)

①

```
SELECT 1, 2 FROM DUAL WHERE 1=2;
-> 실행 시 에러가 발생한다.
```

②

```
SELECT NVL(1, 0) FROM DUAL WHERE 1=2;
-> 실행 시 공집합으로 나온다. (실행 결과가 없다)
```

③

```
SELECT NVL(MAX(1), 1) FROM DUAL WHERE 1=2;
-> 실행 시 공집합으로 나온다. (실행 결과가 없다)
```

④

```
SELECT MAX(NVL(1, 1)) FROM DUAL WHERE 1=2;
-> 실행 시 NULL이 출력된다.
```

문제 6. 아래의 SQL문은 널 연산을 처리하는 SQL문이다. 아래의 SQL문 실행 시 결과는 무엇인가?

```
SELECT COUNT(*), COUNT(NUM), SUM(NUM), AVG(NUM), MAX(NUM), MIN(NUM)
FROM
(
      SELECT NULL AS NUM FROM DUAL
      UNION ALL
      SELECT 10 AS NUM FROM DUAL
      UNION ALL
      SELECT 20 AS NUM FROM DUAL
      UNION ALL
      SELECT 30 AS NUM FROM DUAL
      UNION ALL
      SELECT 40 AS NUM FROM DUAL
);
```

①

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	5	NULL	25	NULL	10

②

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	5	NULL	25	NULL	10

③

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	4	100	25	40	10

④

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	4	100	20	40	10

문제 6. 아래의 SQL문은 널 연산을 처리하는 SQL문이다. 아래의 SQL문 실행 시 결과는 무엇인가?

```
SELECT COUNT(*), COUNT(NUM), SUM(NUM), AVG(NUM), MAX(NUM), MIN(NUM)
FROM
(
      SELECT NULL AS NUM FROM DUAL
      UNION ALL
      SELECT 10 AS NUM FROM DUAL
      UNION ALL
      SELECT 20 AS NUM FROM DUAL
      UNION ALL
      SELECT 30 AS NUM FROM DUAL
      UNION ALL
      SELECT 40 AS NUM FROM DUAL
);
```

①

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	5	NULL	25	NULL	10

②

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	5	NULL	25	NULL	10

③

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	4	100	25	40	10

④

COUNT(*)	COUNT(NUM)	SUM(NUM)	AVG(NUM)	MAX(NUM)	SUM_NUM
5	4	100	20	40	10

문제 7. 다음 중 NULL에 대한 설명으로 가장 부적절한 것은?

① 모르는 값을 의미한다.
② 값의 부재를 의미한다.
③ 공백 혹은 숫자 0을 의미하기도 한다.
④ NULL과 숫자의 연산은 NULL을 반환한다.

문제 8. 다음 중 테이블명으로 사용 가능한 것은 무엇인가?

① TB-EMP
② 1_TB_EMP
③ TB-EMP_100
④ TB_EMP_100$

문제 9. COMMIT과 ROLLBACK의 장점으로 적합하지 않은 것은 무엇인가?

① 데이터의 무결성을 보장한다.
② 영구적인 변경을 하기 전에 데이터의 변경사항을 확인 할 수 있다.
③ 영구적인 변경을 미연에 방지한다.
④ 논리적으로 연관된 작업을 그룹핑 하여 처리할 수 있다.

문제 7. 다음 중 NULL에 대한 설명으로 가장 부적절한 것은?

① 모르는 값을 의미한다.
② 값의 부재를 의미한다.
③ 공백 혹은 숫자 0을 의미하기도 한다.
④ NULL과 숫자의 연산은 NULL을 반환한다.

문제 8. 다음 중 테이블명으로 사용 가능한 것은 무엇인가?

① TB-EMP
② 1_TB_EMP
③ TB-EMP_100
④ TB_EMP_100$

문제 9. COMMIT과 ROLLBACK의 장점으로 적합하지 않은 것은 무엇인가?

① 데이터의 무결성을 보장한다.
② 영구적인 변경을 하기 전에 데이터의 변경사항을 확인 할 수 있다.
③ 영구적인 변경을 미연에 방지한다.
④ 논리적으로 연관된 작업을 그룹핑 하여 처리할 수 있다.

문제 10. 아래의 SQL문은 ORDER BY절을 이용해 결과 집합을 정렬한 SQL문이다. 결과 집합의 정렬 순서가 동일한 ORDER BY절의 내용은 무엇인가?

```
SELECT
       A.CERTI_CD
     , A.CERTI_NM
     , A.ISSUE_INSTI_NM
  FROM TB_CERTI A
ORDER BY A.ISSUE_INSTI_NM DESC, A.CERTI_CD ASC;
```

① ORDER BY A.ISSUE_INSTI_NM, A.CERTI_CD ASC
② ORDER BY 3 DESC, 1 ASC
③ ORDER BY 1 DESC, 2 ASC
④ ORDER BY 1 ASC, 3 DESC

문제 11. 아래의 SQL문은 문법적으로 실패한 SQL문이다. 실패의 원인이 된 줄의 번호는 몇 번인가?

```
① SELECT EMP_NM, EMP_NO, DEPT_CD, DEPT_NM
② FROM TB_EMP A, TB_DEPT B
③ WHERE A.DEPT_CD = B.DEPT_CD
④ ORDER BY BIRTH_DE ;
```

문제 10. 아래의 SQL문은 ORDER BY절을 이용해 결과 집합을 정렬한 SQL문이다. 결과 집합의 정렬 순서가 동일한 ORDER BY절의 내용은 무엇인가?

```
SELECT
       A.CERTI_CD
     , A.CERTI_NM
     , A.ISSUE_INSTI_NM
  FROM TB_CERTI A
ORDER BY A.ISSUE_INSTI_NM DESC, A.CERTI_CD ASC;
```

① ORDER BY A.ISSUE_INSTI_NM, A.CERTI_CD ASC
② ORDER BY 3 DESC, 1 ASC
③ ORDER BY 1 DESC, 2 ASC
④ ORDER BY 1 ASC, 3 DESC

문제 11. 아래의 SQL문은 문법적으로 실패한 SQL문이다. 실패의 원인이 된 줄의 번호는 몇 번인가?

```
① SELECT EMP_NM, EMP_NO, DEPT_CD, DEPT_NM
② FROM TB_EMP A, TB_DEPT B
③ WHERE A.DEPT_CD = B.DEPT_CD
④ ORDER BY BIRTH_DE ;
```

문제 12. 4개의 테이블을 조인하여 필요한 집합 및 컬럼들을 조회하고자 한다. 최소 몇개의 조인 조건이 필요한가?

① 4
② 3
③ 2
④ 1

문제 13. 다음 중 관리자사원번호의 값이 널인 레코드를 찾아내는 문장으로 가장 적절한 것은?

① `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO IS NULL;`
② `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO NOT NULL;`
③ `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO <> NULL;`
④ `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO != NULL;`

문제 12. 4개의 테이블을 조인하여 필요한 집합 및 컬럼들을 조회하고자 한다. 최소 몇개의 조인 조건이 필요한가?

① 4
② 3
③ 2
④ 1

문제 13. 다음 중 관리자사원번호의 값이 널인 레코드를 찾아내는 문장으로 가장 적절한 것은?

① `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO IS NULL;`
② `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO NOT NULL;`
③ `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO <> NULL;`
④ `SELECT * FROM TB_EMP WHERE DIRECT_MANAGER_EMP_NO != NULL;`