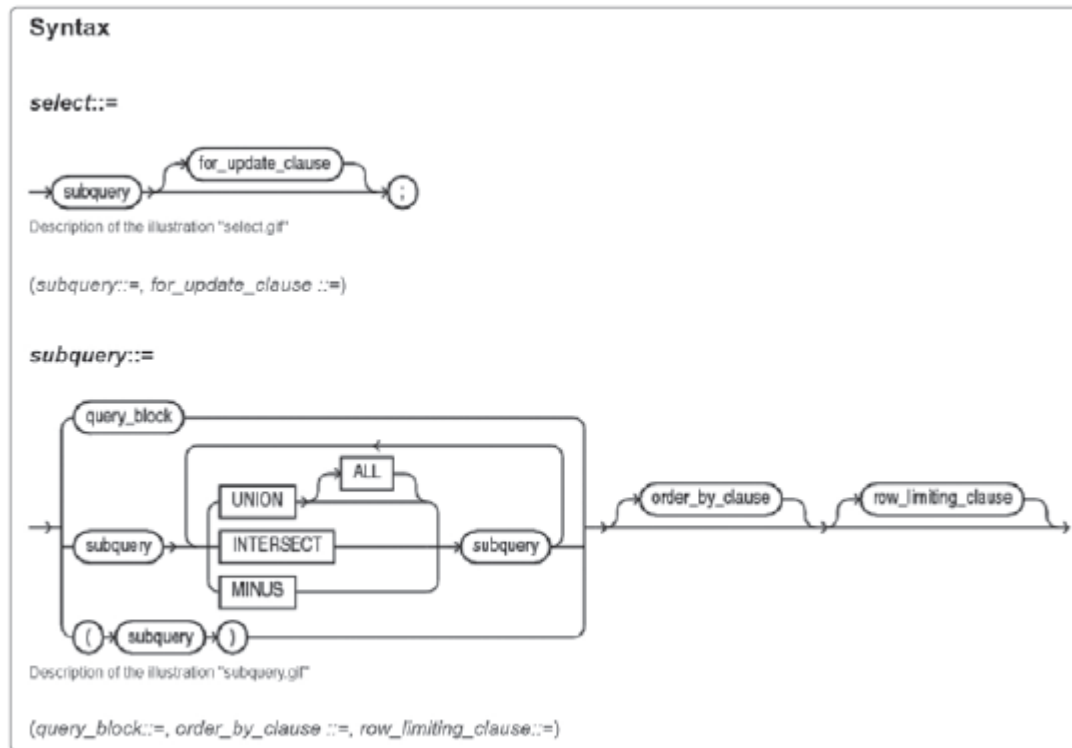


SECTION 01 SELECT문

1.1 원하는 데이터를 가져와 주는 기본적인 <SELECT ... FROM>

- SELECT문은 가장 많이 사용되는 구문임.
- SELECT의 구문 형식



[그림 6-1] Oracle에서 제공하는 도움말의 일부(출처: Oracle 웹 사이트)

[실제적으로 요약한 구조]

```
[ WITH <Sub Query>]  
SELECT select_list  
[ FROM table_source ] [ WHERE search_condition ]  
[ GROUP BY group_by_expression ]  
[ HAVING search_condition ]  
[ ORDER BY order_expression [ ASC | DESC ] ]
```

[가장 자주 쓰이는 형식]

```
SELECT 열 이름  
FROM 테이블이름  
WHERE 조건
```

SECTION 01 SELECT문

SELECT와 FROM

[간단한 SQL문을 입력, 실행]

```
SELECT * FROM employees;
```



50개의 행이 반환됨(0.329초)

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	100	Steven	King	SKING	515.123.4567	03/06/17
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	05/09/21
3	102	Lex	De Haan	LDEHAAN	515.123.4569	01/01/13
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	06/01/03
	104	Bruce	Ernst	BERNST	590.423.4568	10/05/21

[그림 6-2] [질의 결과] 창

[스크립트 실행]



107개 행이 선택되었습니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PH
199	Douglas	Grant	DGRANT	6
200	Jennifer	Whalen	JWHALEN	5
201	Michael	Hartstein	MHARTSTE	5
202	Pat	Fay	PFAY	6
203	Susan	Mavris	SMAVRIS	5
204	Hermann	Beer	HBAER	3
205	Shelley	Higgins	SHIGGINS	3
206	William	Kitz	WKITZ	5

[그림 6-3] [스크립트 출력] 창

SECTION 01 SELECT문

[테이블의 전체 이름은
'스키마이름.테이블이름(=사용자이름.테이블이름)' 형식으로 표현]

```
SELECT * FROM HR.employees;
```

[부서 테이블의 이름만 가져오기]

```
SELECT department_name FROM departments;
```

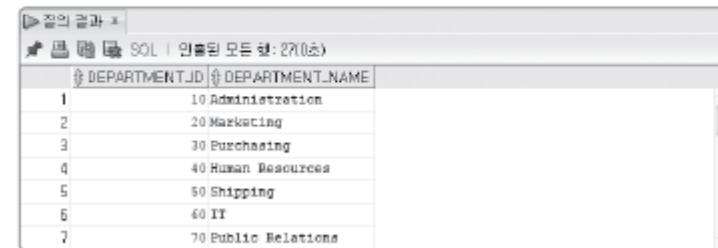


Query Results (SELECT department_name FROM departments;):

DEPARTMENT_NAME
1 Administration
2 Marketing
3 Purchasing
4 Human Resources
5 Shipping
6 IT
7 Public Relations

[그림 6-4] 쿼리 실행 결과

```
SELECT department_id, department_name FROM departments;
```



Query Results (SELECT department_id, department_name FROM departments;):

DEPARTMENT_ID	DEPARTMENT_NAME
1	10 Administration
2	20 Marketing
3	30 Purchasing
4	40 Human Resources
5	50 Shipping
6	60 IT
7	70 Public Relations


[그림 6-5] 쿼리 실행 결과

실습1

step 1

step 2

```
SELECT * FROM SYS.DBA_USERS;
```



	USERNAME	USER_ID	PASSWORD	ACCOUNT_STATUS	LOCK_DATE	EXPIRY_DATE
1	SYS	0 (null)	OPEN	(null)	18/03/16	
2	SYSTEM	5 (null)	OPEN	(null)	18/03/16	
3	ANONYMOUS	35 (null)	OPEN	(null)	14/11/25	
4	CEO	55 (null)	OPEN	(null)	18/03/28	
5	HR	43 (null)	OPEN	(null)	18/03/17	
6	ADMIN	52 (null)	OPEN	(null)	18/03/28	
7	SHOP	50 (null)	OPEN	(null)	18/03/19	
8	STAFF	56 (null)	OPEN	(null)	18/03/28	

[그림 6-7] 사용자(=스키마) 이름 조회

SECTION 01 SELECT문

step 3

HR 스키마(=사용자)에 있는 테이블의 정보를 조회한다.

```
SELECT * FROM SYS.DBA_TABLES WHERE OWNER = 'HR';
```

	OWNER	TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME	IOT_NAME	STATUS
1	HR	REGIONS	USERS	(null)	(null)	VALID
2	HR	LOCATIONS	USERS	(null)	(null)	VALID
3	HR	DEPARTMENTS	USERS	(null)	(null)	VALID
4	HR	JOBS	USERS	(null)	(null)	VALID
5	HR	EMPLOYEES	USERS	(null)	(null)	VALID
6	HR	JOB_HISTORY	USERS	(null)	(null)	VALID
7	HR	PRODUCTS	USERS	(null)	(null)	VALID
8	HR	COUNTRIES	(null)	(null)	(null)	VALID

[그림 6-8] HR 소유의 테이블 이름 조회

step 4

HR.departments 테이블의 열(=컬럼)이 무엇이 있는지 확인해 보자.

```
SELECT * FROM SYS.DBA_TAB_COLUMNS WHERE OWNER = 'HR' AND TABLE_NAME = 'DEPARTMENTS';
```

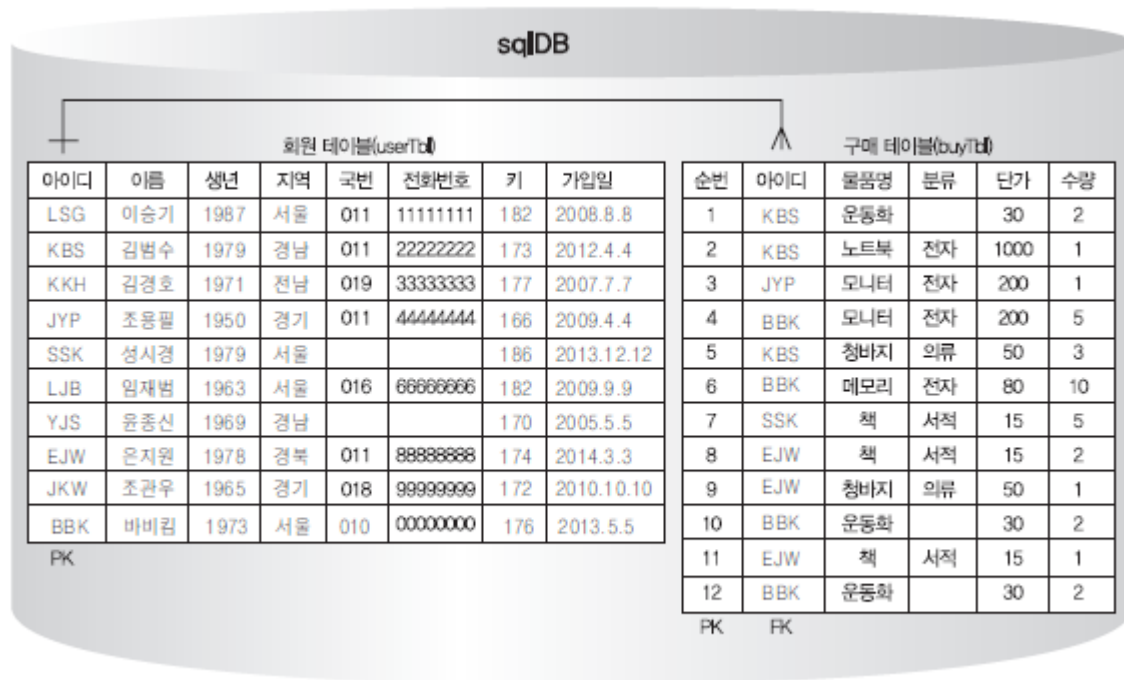
	OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_TYPE_MOD	DATA_TYPE
1	HR	DEPARTMENTS	DEPARTMENT_ID	NUMBER	(null)	(null)
2	HR	DEPARTMENTS	DEPARTMENT_NAME	VARCHAR2	(null)	(null)
3	HR	DEPARTMENTS	MANAGER_ID	NUMBER	(null)	(null)
4	HR	DEPARTMENTS	LOCATION_ID	NUMBER	(null)	(null)

[그림 6-9] 열 이름 조회

step 5

최종적으로 데이터를 조회한다.

```
SELECT department_name FROM HR.departments;
```



[그림 6-11] 샘플 데이터베이스(테이블의 내용은 필자가 임의로 작성한 것임)

SECTION 01 SELECT문

실습2

앞으로 책의 전 과정에서 사용할 스키마(=사용자)와 테이블을 생성하자.

아직 배우지 않은 SQL문이 많이 나올 것이므로 잘 이해가 안 가더라도 우선은 똑같이 진행하자. 앞으로 하나씩 계속 배워 나갈 것이다.

step 0

SQL Developer를 종료하고 다시 실행하자. 그리고 [로컬-SYSTEM]으로 연결해서 워크시트를 연다. 이번엔 입력할 쿼리는 앞으로 다른 장에서도 거의 비슷하게 많이 사용될 것이다. 그러므로, 이번 실습에서 입력한 쿼리를 저장해 놓는 것이 나중에 편리할 것이다.

step 1

우선 sqlDB 스키마(=사용자)를 만들자.

```
CREATE USER sqlDB IDENTIFIED BY 1234 -- 사용자 이름: sqlDB, 비밀번호 : 1234
      DEFAULT TABLESPACE USERS
      TEMPORARY TABLESPACE TEMP;
```

결과 메시지:

User SQLDB이(가) 생성되었습니다.

```
GRANT connect, resource, dba TO sqlDB ;
```

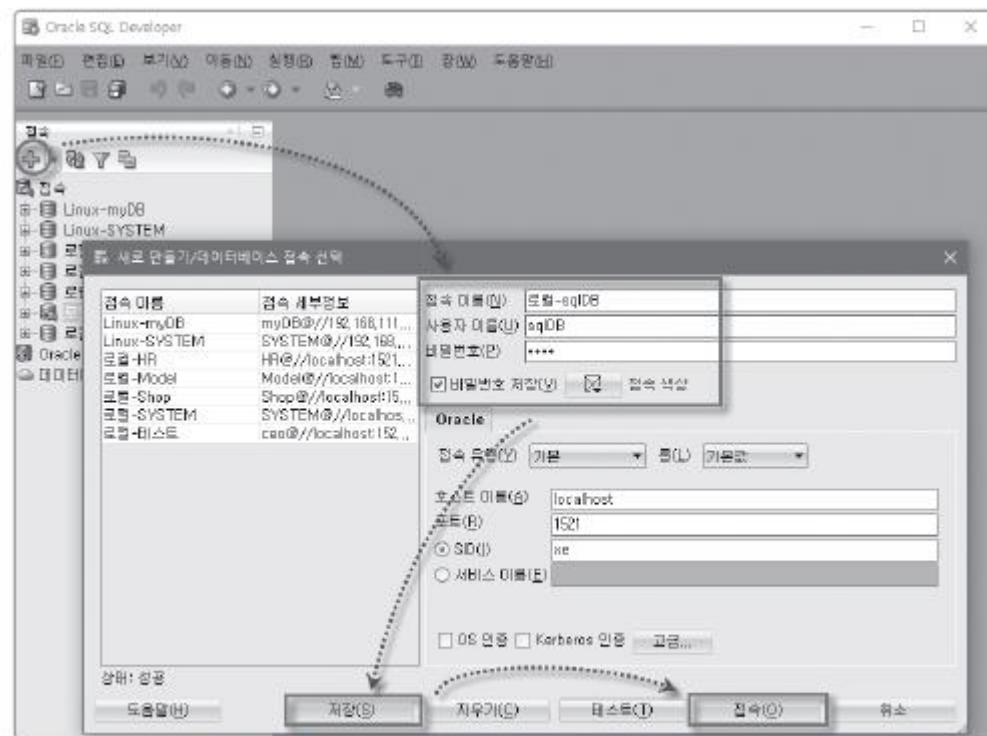
결과 메시지:

Grant을(를) 성공했습니다.

>> 이것이 오라클이다

step 2

새로운 사용자 sqlDB의 접속을 만들자.



[그림 6-12] 새로운 접속의 생성

SECTION 01 SELECT문

step3

테이블을 만들자.

```
CREATE TABLE userTBL -- 회원 테이블
( userID      CHAR(8) NOT NULL PRIMARY KEY, -- 사용자 아이디(PK)
  userName    NVARCHAR2(10) NOT NULL, -- 이름
  birthYear   NUMBER(4) NOT NULL, -- 출생년도
  addr        NCHAR(2) NOT NULL, -- 지역(경기, 서울, 경남 식으로 2글자만 입력)
  mobile1     CHAR(3), -- 휴대폰의 국번(010, 011, 016, 017, 018, 019 등)
  mobile2     CHAR(8), -- 휴대폰의 나머지 전화번호(하이픈 제외)
  height      NUMBER(3), -- 키
  mDate       DATE -- 회원 가입일
);

CREATE TABLE buyTBL -- 회원 구매 테이블
( idNum       NUMBER(8) NOT NULL PRIMARY KEY, -- 순번(PK)
  userID      CHAR(8) NOT NULL, -- 아이디(FK)
  prodName    NCHAR(6) NOT NULL, -- 물품명
  groupName   NCHAR(4), -- 분류
  price       NUMBER(8) NOT NULL, -- 단가
  amount      NUMBER(3) NOT NULL, -- 수량
  FOREIGN KEY (userID) REFERENCES userTBL(userID)
);
```

결과 메시지:

Table USERTBL이(가) 생성되었습니다.

Table BUYTBL이(가) 생성되었습니다.

[회원 테이블 입력]

```
INSERT INTO userTBL VALUES('LSG', '이승기', 1987, '서울', '011', '11111111', 182, '2008-8-8');
INSERT INTO userTBL VALUES('KBS', '김범수', 1979, '경남', '011', '22222222', 173, '2012-4-4');
INSERT INTO userTBL VALUES('KKH', '김경호', 1971, '전남', '019', '33333333', 177, '2007-7-7');
INSERT INTO userTBL VALUES('JYP', '조용필', 1950, '경기', '011', '44444444', 166, '2009-4-4');
INSERT INTO userTBL VALUES('SSK', '성시경', 1979, '서울', NULL, NULL, 186, '2013-12-12');
INSERT INTO userTBL VALUES('LJB', '임재범', 1963, '서울', '016', '66666666', 182, '2009-9-9');
INSERT INTO userTBL VALUES('YJS', '윤종신', 1969, '경남', NULL, NULL, 170, '2005-5-5');
INSERT INTO userTBL VALUES('EJW', '은지원', 1972, '경북', '011', '88888888', 174, '2014-3-3');
INSERT INTO userTBL VALUES('JKW', '조관우', 1965, '경기', '018', '99999999', 172, '2010-10-10');
INSERT INTO userTBL VALUES('BBK', '바비킴', 1973, '서울', '010', '00000000', 176, '2013-5-5');
```

결과 메시지:

1 행 이(가) 삽입되었습니다. (10회 반복됨)

SECTION 01 SELECT문

[구매 테이블 입력]

```
CREATE SEQUENCE idSEQ; -- 순차번호 입력을 위해서 시퀀스 생성
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'KBS', '운동화', NULL, 30, 2);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'KBS', '노트북', '전자', 1000, 1);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'JYP', '모니터', '전자', 200, 1);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'BBK', '모니터', '전자', 200, 5);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'KBS', '청바지', '의류', 50, 3);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'BBK', '메모리', '전자', 80, 10);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'SSK', '책', '서적', 15, 5);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'EJW', '책', '서적', 15, 2);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'EJW', '청바지', '의류', 50, 1);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'BBK', '운동화', NULL, 30, 2);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'EJW', '책', '서적', 15, 1);
INSERT INTO buyTBL VALUES(idSEQ.NEXTVAL, 'BBK', '운동화', NULL, 30, 2);
```

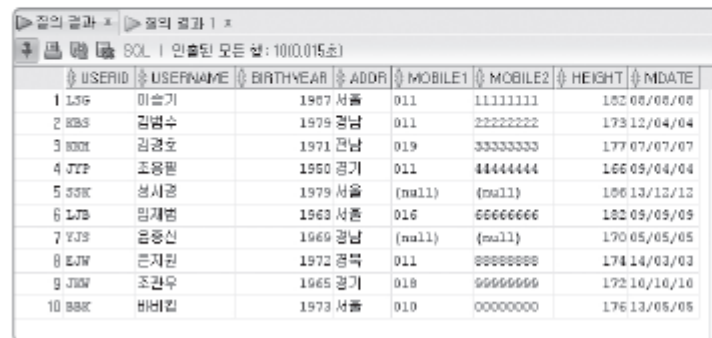
결과 메시지:

1 행 이(가) 삽입되었습니다. (12회 반복됨)

step 5

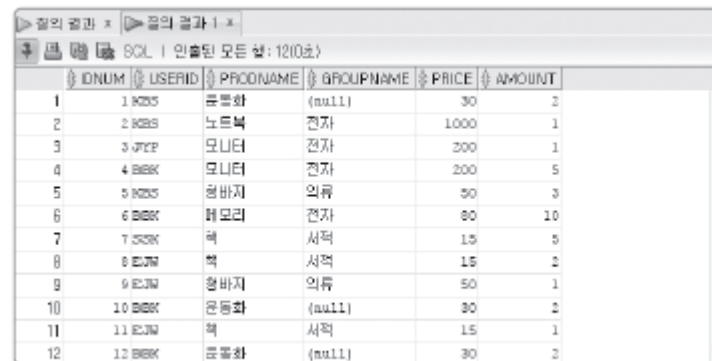
입력한 데이터를 커밋하고, 데이터를 확인하자.

```
COMMIT;
SELECT * FROM userTBL;
SELECT * FROM buyTBL;
```



USERID	USERNAME	BIRTHYEAR	ADDR	MOBILE1	MOBILE2	HEIGHT	MDATE
1	LSG	1987	서울	011	11111111	182	08/08/08
2	KBS	1979	경남	011	22222222	173	12/04/04
3	KOI	1971	전남	019	33333333	177	07/07/07
4	JYP	1980	경기	011	44444444	166	05/04/04
5	SSK	1979	서울	(null)	(null)	186	13/12/13
6	LJS	1963	서울	016	55555555	182	05/05/05
7	VJS	1969	경남	(null)	(null)	170	05/05/05
8	EJW	1972	경북	011	88888888	174	14/03/03
9	JIN	1965	경기	018	99999999	172	16/10/16
10	BBK	1973	서울	010	00000000	176	13/05/05

[그림 6-13] 회원 테이블(userTBL)



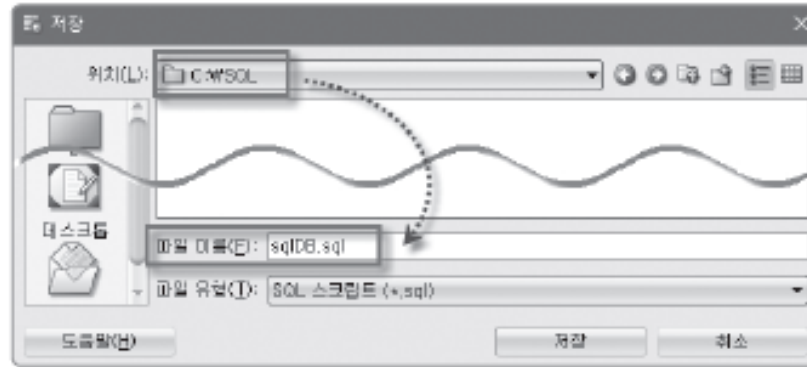
IDNUM	USERID	PRCONAME	GROUPNAME	PRICE	AMOUNT
1	1 KOS	운동화	(null)	30	2
2	2 KBS	노트북	전자	1000	1
3	3 JYP	모니터	전자	200	1
4	4 BBK	모니터	전자	200	5
5	5 KOS	청바지	의류	50	3
6	6 BBK	메모리	전자	80	10
7	7 SSK	책	서적	15	5
8	8 EJW	책	서적	15	2
9	9 EJW	청바지	의류	50	1
10	10 BBK	운동화	(null)	30	2
11	11 EJW	책	서적	15	1
12	12 BBK	운동화	(null)	30	2

[그림 6-14] 구매 테이블(buyTBL)

SECTION 01 SELECT문

step 6

앞으로는 이 책의 많은 부분에서 이 sqlDB 스키마를 사용하게 될 것이다. 혹, 실수로 이 스키마가 변경되어도 다시 입력하는 번거로움이 없도록 SQL문을 저장해 놓자.



[그림 6-15] C:\SQL\sqlDB.sql로 스크립트를 저장

SECTION 01 SELECT문

1.2 특정한 조건의 데이터만 조회하는 <SELECT... FROM ... WHERE>

- 기본적인 WHERE절

```
SELECT 필드이름 FROM 테이블이름 WHERE 조건식;
```

[WHERE절은 조회하는 결과_ 원하는 데이터만 보고 싶을 때]

```
SELECT * FROM userTBL;
```

[WHERE 조건 없이 조회]

```
SELECT * FROM userTBL WHERE userName = '김경호';
```

	USERID	USERNAME	BIRTHYEAR	ADDR	MOBILE1	MOBILE2	HEIGHT	MDATE
1	KKH	김경호	1971	전남	019	33333333	177	07/07/07

[그림 6-16] 쿼리 실행 결과

SECTION 01 SELECT문

- 관계 연산자의 사용

- 조건 연산자(=, <, >, <=, >=, < >, != 등)와 관계 연산자(NOT, AND, OR 등)를 잘 조합하면 다양한 조건의 쿼리를 생성할 수 있음..

```
SELECT userID, userName FROM userTBL WHERE birthYear >= 1970 AND height >= 182;
```

[아이디와 이름을 조회]

```
SELECT userID, userName FROM userTBL WHERE birthYear >= 1970 OR height >= 182;
```

[추가 조건 아이디와 이름을 조회]

SECTION 01 SELECT문

- BETWEEN... AND와 IN() 그리고 LIKE

```
SELECT userName, height FROM userTBL WHERE height BETWEEN 180 AND 183;
```

[연속적인 값 일 경우 BETWEEN... AND 사용 조회]

```
SELECT userName, addr FROM userTBL WHERE addr='경남' OR addr='전남' OR addr='경북';
```

[비연속적인 값 일 경우 사용 조회]

```
SELECT userName, addr FROM userTBL WHERE addr IN ('경남', '전남', '경북');
```

[IN() 사용 조회]

```
SELECT userName, height FROM userTBL WHERE userName LIKE '김%';
```

[LIKE 사용 조회]

SECTION 01 SELECT문

- ANY/ALL/SOME 그리고 서브쿼리(SubQuery, 하위쿼리)
 - 서브쿼리란 간단히 얘기하면 쿼리문 안에 또 쿼리문이 들어 있는 것을 얘기함.

```
SELECT userName, height FROM userTBL WHERE height > 177;
```

[WHERE 조건에 김경호의 키를 직접입력]

```
SELECT userName, height FROM userTBL  
WHERE height > (SELECT height FROM userTBL WHERE userName = '김경호');
```

[쿼리를 통해서 사용, 동일값 추출]

```
SELECT userName, height FROM userTBL  
WHERE height >= (SELECT height FROM userTBL WHERE addr = '경남');
```



[그림 6-17] 쿼리 실행 결과

[쿼리문 만들기]

[ANY 구문]

```
SELECT rownum, userName, height FROM userTBL  
WHERE height >= ANY (SELECT height FROM userTBL WHERE addr = '경남');
```

	USERNAME	HEIGHT
1	성시경	186
2	임재범	182
3	이승기	182
4	김경호	177
5	변비림	176
6	문지원	174
7	김범수	173
8	조관우	172
9	윤종신	170

[그림 6-18] 쿼리 실행 결과

	USERNAME	HEIGHT
1	김범수	173
2	문지원	174
3	변비림	176
4	김경호	177
5	이승기	182
6	임재범	182
7	성시경	186

[그림 6-19] 쿼리 실행 결과

```
SELECT userName, height FROM userTBL  
WHERE height = ANY (SELECT height FROM userTBL WHERE addr = '경남');
```

	USERNAME	HEIGHT
1	김범수	173
2	윤종신	170

[그림 6-20] 쿼리 실행 결과

[‘= ANY’ 를 사용]

SECTION 01 SELECT문

- 원하는 순서대로 정렬하여 출력 : ORDER BY
 - ORDER BY절은 결과물에 대해 영향을 미치지 않지만, 결과가 출력되는 순서를 조절하는 구문임.
 - ORDER BY에 나온 열이 SELECT 다음에 꼭 있을 필요는 없음. 즉, SELECT userID FROM userTBL ORDER BY height 문을 사용해도 됨.

```
SELECT userName, mDate FROM userTBL ORDER BY mDate;
```

	USERNAME	MDATE
1	김종민	05/05/05
2	김영호	07/07/07
3	이승기	08/08/08
4	조용필	09/04/04
5	최재범	09/09/09
6	조권우	10/10/10
7	김범수	12/04/04
8	나비킴	13/05/05
9	성시경	13/12/12
10	윤지원	14/03/03

[그림 6-21] 쿼리 실행 결과

[ORDER BY 출력]

SECTION 01 SELECT문

- 중복된 것은 하나만 남기는 DISTINCT
 - 중복된 것은 1개씩만 보여주면서 출력됨

```
SELECT addr FROM userTBL;
```

ADDR
1 서울
2 경남
3 전남
4 경기
5 서울
6 서울
7 경남
8 경북
9 경기
10 서울

[거주지역이 몇 군데인지 출력]

[그림 6-22] 쿼리 실행 결과

```
SELECT DISTINCT addr FROM userTBL;
```

ADDR
1 경남
2 전남
3 경북
4 경기
5 서울

[DISTINCT 사용 출력]

[그림 6-24] 쿼리 실행 결과

```
SELECT addr FROM userTBL ORDER BY addr;
```

ADDR
1 경기
2 경기
3 경남
4 경남
5 경북
6 서울
7 서울
8 서울
9 서울
10 전남

[ORDER BY 사용 출력]

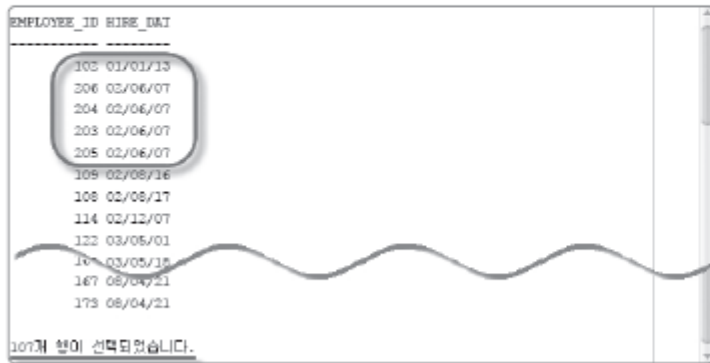
[그림 6-23] 쿼리 실행 결과

SECTION 01 SELECT문

- ROWNUM 열과 SAMPLE문
 - ROWNUM을 활용하면 출력되는 개수를 조절할 수 있음.
 - 임의의 데이터를 추출하고 싶다면 SAMPLE(퍼센트)문을 사용.

[상위의 N개만 출력하려면 서브 쿼리와
'WHERE ROWNUM <= 개수' 구문을 함께 사용]

```
SELECT employee_id, hire_date FROM employees  
ORDER BY hire_date ASC;
```

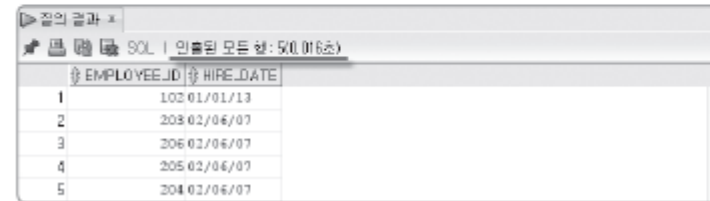


EMPLOYEE_ID	HIRE_DATE
103	01/01/13
206	02/06/07
204	02/06/07
203	02/06/07
205	02/06/07
109	02/08/16
108	02/08/17
114	02/12/07
122	03/05/01
140	03/05/18
167	08/04/21
173	08/04/21

[그림 6-25] 쿼리 실행 결과

[ORDER BY 사용 출력]

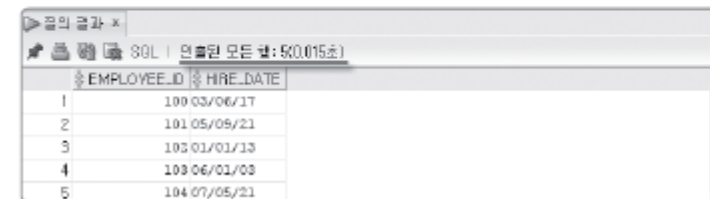
```
SELECT * FROM  
(SELECT employee_id, hire_date FROM employees ORDER BY hire_date ASC)  
WHERE ROWNUM <= 5;
```



EMPLOYEE_ID	HIRE_DATE
1	103 01/01/13
2	203 02/06/07
3	206 02/06/07
4	205 02/06/07
5	204 02/06/07

[그림 6-26] 쿼리 실행 결과

```
SELECT employee_id, hire_date FROM employees  
WHERE ROWNUM <= 5;
```



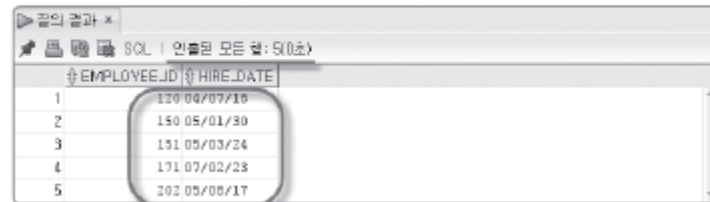
EMPLOYEE_ID	HIRE_DATE
1	100 03/06/17
2	101 05/09/21
3	103 01/01/13
4	103 06/01/08
5	104 07/05/21

[그림 6-27] 쿼리 실행 결과

SECTION 01 SELECT문

- 임의의 데이터를 추출하고 싶다면 SAMPLE(퍼센트)문을 사용하면 됨.

```
SELECT employee_id, hire_date FROM EMPLOYEES SAMPLE(5);
```



The screenshot shows a SQL query result window titled '결과 보기' (View Results). The query is 'SELECT employee_id, hire_date FROM EMPLOYEES SAMPLE(5);'. The result is displayed as a table with 5 rows. The first two rows are highlighted with a rounded rectangle.

	EMPLOYEE_ID	HIRE_DATE
1	120	09/07/19
2	150	05/01/30
3	191	05/03/24
4	171	07/02/28
5	202	05/05/17

[그림 6-28] 추출 결과

SECTION 01 SELECT문

테이블을 복사하는 CREATE TABLE ... AS SELECT

- CREATE TABLE ... AS SELECT 구문은 테이블을 복사해서 사용할 경우에 주로 사용됨.

형식:

CREATE TABLE 새로운테이블 AS (SELECT 복사할 열 FROM 기존 테이블)

```
CREATE TABLE buyTBL2 AS (SELECT * FROM buyTBL);  
SELECT * FROM buyTBL2;
```

[buyTBL을 buyTBL2로 복사하는 구문]

```
CREATE TABLE buyTBL3 AS (SELECT userID, prodName FROM buyTBL);  
SELECT * FROM buyTBL3;
```

USERID	PRODNAME
1 QBS	운동화
2 QBS	노트북
3 JYP	모니터
4 QBS	책
5 JYP	책
12 QBS	운동화

[그림 6-29] 쿼리 실행 결과

[지정한 일부 열만 복사]



[그림 6-30] BUYTBL 및 BUYTBL2의 제약 조건 확인

[P, K나 FK 등의 제약 조건은 복사되지 않음]

SECTION 01 SELECT문

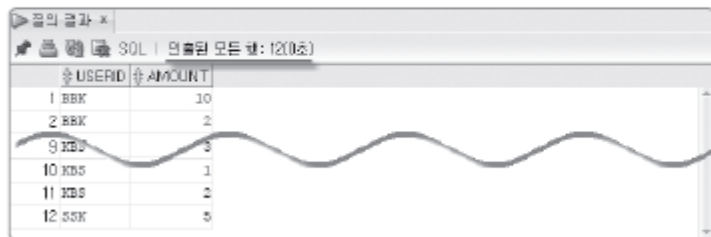
1.3 GROUP BY 및 HAVING 그리고 집계 함수

◦ GROUP BY절

형식 :

```
SELECT select_expr  
  [FROM table_references]  
  [WHERE where_condition]  
  [GROUP BY {col_userName | expr | position}]  
  [HAVING where_condition]  
  [ORDER BY {col_userName | expr | position}]
```

```
SELECT userID, amount FROM buyTBL ORDER BY userID;
```

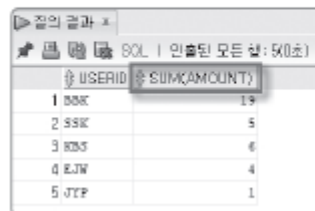


USERID	AMOUNT
1 BKK	10
2 BKK	2
9 KDS	3
10 KDS	1
11 KDS	2
12 SSK	5

[그림 6-31] 쿼리 실행 결과

[sqlDB의 구매 테이블에서 사용자가 구매한 물품 개수 확인]

```
SELECT userID, SUM(amount) FROM buyTBL GROUP BY userID;
```

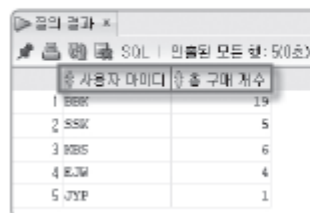


USERID	SUM(AMOUNT)
1 BKK	19
2 SSK	5
3 KDS	6
4 EJV	4
5 JYP	1

[그림 6-32] GROUP BY 사용 결과

[각 사용자별로 구매한 개수를 합쳐서 출력]

```
SELECT userID AS "사용자 아이디", SUM(amount) AS "총 구매 개수"  
FROM buyTBL GROUP BY userID;
```



사용자 아이디	총 구매 개수
1 BKK	19
2 SSK	5
3 KDS	6
4 EJV	4
5 JYP	1

[그림 6-33] 별칭의 활용

[별칭을 사용해서 결과보기]

SECTION 01 SELECT문

```
SELECT userID AS "사용자 아이디", SUM(price*amount) AS "총 구매액"  
FROM buyTBL GROUP BY userID;
```

사용자 아이디	총 구매액
1 BBSK	1920
2 JSJK	75
3 BBSB	1210
4 JSJK	95
5 JSJK	200

[구매액의 총합을 출력]

[그림 6-34] 총 구매액

○ 집계 함수

- SUM() 외에 GROUP BY와 함께 자주 사용되는 집계 함수(또는 집합 함수)

함수명	설명
AVG()	평균을 구한다.
MIN()	최소값을 구한다.
MAX()	최대값을 구한다.
COUNT()	행의 개수를 센다.
COUNT(DISTINCT)	행의 개수를 센다(중복은 1개만 인정).
STDEV()	표준편차를 구한다.
VARIANCE()	분산을 구한다.

[표 6-1] 자주 사용되는 Oracle 집계 함수

SECTION 01 SELECT문

```
SELECT AVG(amount) AS "평균 구매 개수" FROM buyTBL;
```

[illegible]

[전체 구매자가 구매한 물품의 개수의 평균]

[그림 6-35] 쿼리 실행 결과

```
SELECT CAST(AVG(amount) AS NUMBER(5,3)) AS "평균 구매 개수" FROM buyTBL;
```

평균 구매 저수 2,912

[소수점을 조절 CAST() 함수 사용]

[그림 6-36] 쿼리 실행 결과

```
SELECT userID, CAST(AVG(amount) AS NUMBER(5,3)) AS "평균 구매 개수" FROM buyTBL
GROUP BY userID;
```

USERID	평균 구매 개수
1 BBS	4.75
2 SSK	5
3 KRS	2
4 EJW	1.333
5 JYP	1

[평균적으로 몇 개 구매했는지 평균 구하기]

[그림 6-37] 쿼리 실행 결과

```
SELECT userName, MAX(height), MIN(height) FROM userTBL;
```

ORA-00937: not a single-group group function
00937, 00000 - "not a single-group group function"
--Cause:
--Action:
9999, 8월 16일 오후 발생

[그림 6-38] 오류 메시지

[GROUP BY 없이는 별도의 이름 열을 집계 함수와 같이 사용할 수 없다는 오류]

```
SELECT userName, MAX(height), MIN(height) FROM userTBL GROUP BY userName;
```

	USERNAME	MAX(HEIGHT)	MIN(HEIGHT)
1	조만우	172	172
2	마슬기	182	182
3	바비킴	176	176
4	조름철	166	166
5	강민수	173	173
6	임재범	182	182
7	은지원	174	174
8	김광호	177	177
9	성시경	186	186
10	윤종신	170	170

[GROUP BY 수정]

[그림 6-39] 쿼리 실행 결과

SECTION 01 SELECT문

```
SELECT userName, height
FROM userTBL
WHERE height = (SELECT MAX(height)FROM userTBL)
OR height = (SELECT MIN(height)FROM userTBL);
```

	USERNAME	HEIGHT
1	조용필	166
2	강시경	166

[서브쿼리와 조합]

[그림 6-40] 쿼리 실행 결과

```
SELECT COUNT(*) FROM userTBL;
```

[사용자의 수를 카운트]

```
SELECT COUNT(mobile1) AS "휴대폰이 있는 사용자" FROM userTBL;
```

	휴대폰이 있는 사용자
1	8

[NULL 값인 것은 제외하고 카운트]

[그림 6-41] 쿼리 실행 결과

SECTION 01 SELECT문

◦ Having절

- HAVING은 WHERE와 비슷한 개념으로 조건을 제한하는 것이지만, 집계 함수에 대해서 조건을 제한하는 것임.

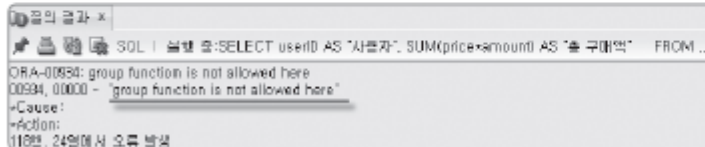
```
SELECT userID AS "사용자", SUM(price*amount) AS "총 구매액"  
FROM buyTBL  
GROUP BY userID;
```

순 사용자	총 구매액
1 BDK	1900
2 SKK	75
3 KBS	1210
4 KPW	90
5 JYP	200

[사용자별 총 구매액]

[그림 6-42] 쿼리 실행 결과

```
SELECT userID AS "사용자", SUM(price*amount) AS "총 구매액"  
FROM buyTBL  
WHERE SUM(price*amount) > 1000  
GROUP BY userID;
```



[그림 6-43] 오류 메시지

[집계 함수는 WHERE절에 나타날 수 없음]

```
SELECT userID AS "사용자", SUM(price*amount) AS "총 구매액"  
FROM buyTBL  
GROUP BY userID  
HAVING SUM(price*amount) > 1000;
```

순 사용자	총 구매액
1 BDK	1900
2 KBS	1210

[HAVING절 사용, HAVING절은 꼭 GROUP BY절 다음에 나와야 함]

[그림 6-44] 쿼리 실행 결과

```
SELECT userID AS "사용자", SUM(price*amount) AS "총 구매액"  
FROM buyTBL  
GROUP BY userID  
HAVING SUM(price*amount) > 1000  
ORDER BY SUM(price*amount);
```

[ORDER BY를 사용]

SECTION 01 SELECT문

◦ ROLLUP(), GROUPING_ID(), CUBE() 함수

- 총합 또는 중간 합계가 필요하다면 GROUP BY절과 함께 ROLLUP() 또는 CUBE()를 사용하면 됨.

```
SELECT idNum, groupName, SUM(price * amount) AS "비용"
FROM buyTbl
GROUP BY ROLLUP (groupName, idNum);
```

IDNUM	GROUPNAME	비용
1	1 (null)	60
2	10 (null)	60
3	12 (null)	60
4	(null) (null)	180
5	7 서적	75
6	8 서적	30
7	11 서적	15
8	(null) 서적	120
9	9 의류	150
10	5 의류	50
11	(null) 의류	200
12	2 전자	1000
13	3 전자	200
14	4 전자	1000
15	6 전자	800
16	(null) 전자	3000
17	(null) (null)	3500

[분류별로 합계 및 총합 구하기]

```
SELECT groupName, SUM(price * amount) AS "비용"
FROM buyTbl
GROUP BY ROLLUP (groupName);
```

GROUPNAME	비용
1 서적	120
2 의류	200
3 전자	3000
4 (null)	180
5 (null)	3500

[소합계 및 총합계만 필요]

[그림 6-46] 쿼리 실행 결과

```
SELECT groupName, SUM(price * amount) AS "비용"
, GROUPING_ID(groupName) AS "추가행 여부"
FROM buyTbl
GROUP BY ROLLUP(groupName) ;
```

GROUPNAME	비용	추가행 여부
1 서적	120	0
2 의류	200	0
3 전자	3000	0
4 (null)	180	0
5 (null)	3500	1

[GROUPING_ID() 함수 사용]

[그림 6-47] 쿼리 실행 결과

SECTION 01 SELECT문

물품	색상	수량
컴퓨터	검정	11
컴퓨터	파랑	22
모니터	검정	33
모니터	파랑	44

[표 6-2] 다차원 정보용 샘플 테이블

```
CREATE TABLE cubeTbl(prodName NCHAR(3), color NCHAR(2), amount INT);
INSERT INTO cubeTbl VALUES('컴퓨터', '검정', 11);
INSERT INTO cubeTbl VALUES('컴퓨터', '파랑', 22);
INSERT INTO cubeTbl VALUES('모니터', '검정', 33);
INSERT INTO cubeTbl VALUES('모니터', '파랑', 44);
SELECT prodName, color, SUM(amount) AS "수량합계"
  FROM cubeTbl
 GROUP BY CUBE (color, prodName)
 ORDER BY prodName, color;
```

형식 :

```
[ WITH <Sub Query>]
SELECT select_list [ INTO new_table ]
[ FROM table_source ]
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

[WITH절을 살펴보기]

PRODNAME	COLOR	수량합계
1 모니터	검정	33
2 모니터	파랑	44
3 모니터	{null}	77
4 컴퓨터	검정	11
5 컴퓨터	파랑	22
6 컴퓨터	{null}	33
7 {null}	검정	44
8 {null}	파랑	66
9 {null}	{null}	110

[물품별 소합계 및 색상별 소합계 확인
CUBE() 사용]

[그림 6-48] CUBE()의 결과

SECTION 01 SELECT문

1.4 WITH절과 CTE

- WITH절은 CTE를 표현하기 위한 구문임.
 - CTE는 기존의 뷰, 파생 테이블, 임시 테이블 등으로 사용되던 것을 대신할 수 있으며, 더 간결한 식으로 보여지는 장점.
- 비재귀적 CTE
 - 재귀적이지 않은 CTE임. 단순한 형태이며, 복잡한 쿼리 문장을 단순화시키는 데에 적합하게 사용될 수 있음.

```
WITH CTE_테이블이름(열 이름)
AS
(
    <쿼리문>
)
SELECT 열 이름 FROM CTE_테이블이름 ;
```

[SELECT 열 이름 FROM CTE_테이블이름]

SECTION 01 SELECT문

1단계 → '각 지역별로 가장 큰 키'를 뽑는 쿼리는 다음과 같다.

```
SELECT addr, MAX(height) FROM userTbl GROUP BY addr
```

2단계 → 위의 쿼리를 WITH 구문으로 묶는다.

```
WITH cte_테이블이름(addr, maxHeight)
AS
( SELECT addr, MAX(height) FROM userTbl GROUP BY addr)
```

3단계 → '키의 평균'을 구하는 쿼리를 작성한다.

```
SELECT AVG(키) FROM cte_테이블이름
```

4단계 → 2단계와 3단계의 쿼리를 합친다.

```
WITH cte_userTbl(addr, maxHeight)
AS
( SELECT addr, MAX(height) FROM userTbl GROUP BY addr)
SELECT AVG(maxHeight) AS "각 지역별 최고키 평균" FROM cte_userTbl;
```

각 지역별 최고키 평균	
1	176.4

[그림 6-52] 쿼리 실행 결과

형식 :

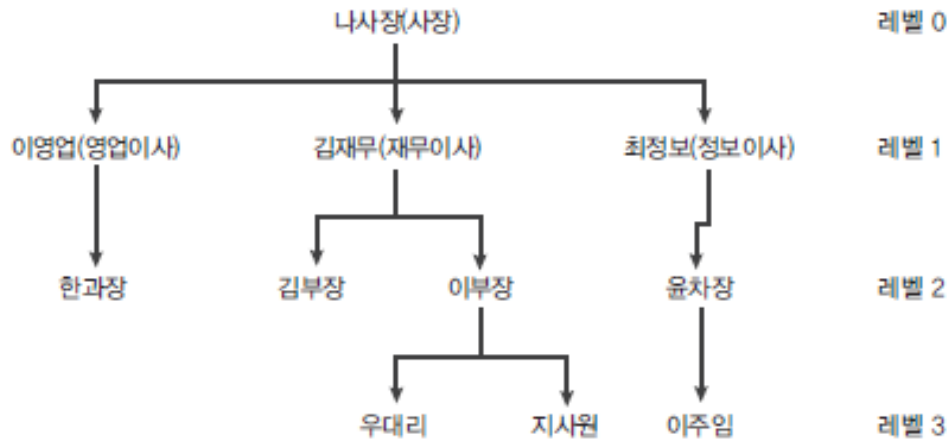
```
WITH
AAA (컬럼들)
AS ( AAA의 쿼리문 ),
BBB (컬럼들)
AS ( BBB의 쿼리문 ),
CCC (컬럼들)
AS ( CCC의 쿼리문 )
SELECT * FROM [AAA 또는 BBB 또는 CCC]
```

[중복 CTE가 허용됨]

SECTION 01 SELECT문

◦ 재귀적 CTE

- 재귀적이라는 의미는 자기자신을 반복적으로 호출한다는 의미를 내포함.



[그림 6-53] 간단한 조직도 예

직원 이름(EMP) - 기본 키	상관 이름(MANAGER)	부서(DEPARTMENT)
나사장	없음(NULL)	없음(NULL)
김재무	나사장	재무부
김부장	김재무	재무부
이부장	김재무	재무부
우대리	이부장	재무부
지사원	이부장	재무부
이영업	나사장	영업부
한과장	이영업	영업부
최정보	나사장	정보부
윤차장	최정보	정보부
이주임	윤차장	정보부

[표 6-3] 조직도 테이블

SECTION 01 SELECT문

[재귀적 CTE의 기본 형식]

```
형식:
WITH CTE_테이블이름(열 이름)
AS
(
    <쿼리문1 : SELECT * FROM 테이블A >
    UNION ALL
    <쿼리문2 : SELECT * FROM 테이블A JOIN CTE_테이블이름>
)
SELECT * FROM CTE_테이블이름;
```

작동 원리를 살펴보면

- ① <쿼리문1>을 실행한다. 이것이 전체 흐름의 최초 호출에 해당한다. 그리고, 레벨의 시작은 0으로 초기화된다.
- ② <쿼리문2>를 실행한다. 레벨을 레벨+1로 증가시킨다. 그런데, SELECT의 결과가 빈 것이 아니라면, 'CTE_테이블이름'을 다시 재귀적으로 호출한다.
- ③ 계속 ②번을 반복한다. 단, SELECT의 결과가 아무것도 없다면 재귀적인 호출이 중단된다.
- ④ 외부의 SELECT문을 실행해서 앞 단계에서의 누적된 결과(UNION ALL)를 가져온다.

실습3

하나의 테이블에서 회사의 조직도가 출력되도록, 재귀적 CTE를 구현하자.

step 0

우선 위의 테이블을 정의하고 데이터를 입력하자.

```
CREATE TABLE empTbl (emp NCHAR(3), manager NCHAR(3), department NCHAR(3));
```

```
INSERT INTO empTbl VALUES('나사장', '없음', '없음');
INSERT INTO empTbl VALUES('김재무', '나사장', '재무부');
INSERT INTO empTbl VALUES('김부장', '김재무', '재무부');
INSERT INTO empTbl VALUES('이부장', '김재무', '재무부');
INSERT INTO empTbl VALUES('우대리', '이부장', '재무부');
INSERT INTO empTbl VALUES('지사원', '이부장', '재무부');
INSERT INTO empTbl VALUES('이영업', '나사장', '영업부');
INSERT INTO empTbl VALUES('한과장', '이영업', '영업부');
INSERT INTO empTbl VALUES('최정보', '나사장', '정보부');
INSERT INTO empTbl VALUES('윤차장', '최정보', '정보부');
INSERT INTO empTbl VALUES('이주임', '윤차장', '정보부');
```


SECTION 01 SELECT문

step 1

재귀적 CTE의 구문 형식에 맞춰서 쿼리문을 만들자. 아직 조인^{join}을 배우지 않아서 <쿼리문2> 부분이 좀 어려울 것이지만, 형식대로 empTbl과 empCTE를 조인하는 방식이다. 어려우면, 그냥 결과를 위주로 보자.

```
WITH empCTE(empName, mgrName, dept, empLevel)
AS
(
  ( SELECT emp, manager, department , 0
    FROM empTbl
    WHERE manager = '없음' ) -- 상관이 없는 사람이 바로 사장
  UNION ALL
  (SELECT empTbl.emp, empTbl.manager, empTbl.department, empCTE.empLevel+1
   FROM empTbl INNER JOIN empCTE
    ON empTbl.manager = empCTE.empName)
)
SELECT * FROM empCTE ORDER BY dept, empLevel;
```

EMPNAME	MGRNAME	DEPT	EMPLEVEL
1 나사장	없음	없음	0
2 이영일	나사장	영업부	1
3 한과장	이영일	영업부	2
4 김재무	나사장	재무부	1
5 김부장	김재무	재무부	2
6 이부장	김재무	재무부	2
7 우대리	이부장	재무부	3
8 지사원	이부장	재무부	3
9 최경보	나사장	정보부	1
10 윤차장	최경보	정보부	2
11 이주임	윤차장	정보부	3

[그림 6-54] 재귀적 CTE 결과 1

step 2

그래도 [그림 6-53]처럼 안보이므로, 쿼리문을 약간 수정해서 [그림 6-53]과 더욱 비슷하게 만들자.

```
WITH empCTE(empName, mgrName, dept, empLevel)
AS
(
  ( SELECT emp, manager, department , 0
    FROM empTbl
    WHERE manager = '없음' ) -- 상관이 없는 사람이 바로 사장
  UNION ALL
  (SELECT empTbl.emp, empTbl.manager, empTbl.department, empCTE.empLevel+1
   FROM empTbl INNER JOIN empCTE
    ON empTbl.manager = empCTE.empName)
)
SELECT CONCAT(RPAD('L', empLevel*2 + 1, 'L'), empName) AS "직원이름", dept AS "직원부서"
FROM empCTE ORDER BY dept, empLevel;
```

직원이름	직원부서
1 나사장	없음
2 L이영일	영업부
3 L L한과장	영업부
4 L 김재무	재무부
5 L L김부장	재무부
6 L L이부장	재무부
7 L L L우대리	재무부
8 L L L지사원	재무부
9 L최경보	정보부
10 L L윤차장	정보부
11 L L L이주임	정보부

[그림 6-55] 재귀적 CTE 결과 2

SECTION 01 SELECT문

step3

이번에는 직원급을 제외한, 부장/차장/과장급까지만 출력해 보자. 레벨이 2이므로, 간단히 'WHERE empLevel < 2'만 <쿼리문2> 부분에 추가해 주면 된다.

```
WITH empCTE(empName, mgrName, dept, empLevel)
AS
(
  ( SELECT emp, manager, department , 0
    FROM empTbl
    WHERE manager = '없음' ) -- 상관이 없는 사람이 바로 사장

  UNION ALL
  (SELECT empTbl.emp, empTbl.manager, empTbl.department, empCTE.empLevel+1
   FROM empTbl INNER JOIN empCTE
    ON empTbl.manager = empCTE.empName
   WHERE empLevel < 2)
)
SELECT CONCAT(RPAD('L', empLevel*2 + 1, 'L'), empName) AS "직원이름", dept AS "직원부서"
FROM empCTE ORDER BY dept, empLevel;
```

	직원이름	직원부서
1	나사꾼	없음
2	L대형업	영업부
3	L L한과장	영업부
4	L 김개무	재무부
5	L L김부장	재무부
6	L L이부장	재무부
7	L 최장보	정보부
8	L L김차장	정보부

[그림 6-56] 재귀적 CTE 결과 3

형식 :

```
[ WITH <common_table_expression>]
SELECT select_list [ INTO new_table ]
[ FROM table_source ]
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

SECTION 01 SELECT문

1.5 SQL의 분류

- SQL문은 크게 DML, DDL, DCL로 분류함.
- DML
 - DML은 데이터 조작 언어은 데이터를 조작(선택, 삽입, 수정, 삭제)하는 데 사용되는 언어임.
 - SQL문 중에 SELECT, INSERT, UPDATE, DELETE가 DML문에 해당됨
 - 트랜잭션이 발생하는 SQL도 DML임.
- DDL
 - DDL은 데이터 정의 언어은 데이터베이스, 테이블, 뷰, 인덱스 등의 데이터베이스 개체를 생성/삭제/변경하는 역할을 함.
 - DDL은 CREATE, DROP, ALTER 등임. 트랜잭션을 발생시키지 않음.
- DCL
 - DCL은 데이터 제어 언어은 사용자에게 어떤 권한을 부여하거나 빼앗을 때 주로 사용하는 구문임.
 - DCL은 GRANT/REVOKE/ DENY 등임.

SECTION 02 데이터의 변경을 위한 SQL문

2.1 데이터의 삽입 : INSERT

- INSERT는 테이블에 데이터를 삽입하는 명령어임.
- INSERT문 기본

[기본형식]

```
INSERT INTO 테이블[(열1, 열2, ...)] VALUES (값1, 값2 ...)
```

주의점 1: VALUE 다음에 나오는 값들의 순서 및 개수가 테이블이 정의된 열 순서 및 개수와 동일해야 함.

```
CREATE TABLE testTBL1 (id NUMBER(4), userName NCHAR(3), age NUMBER(2));  
INSERT INTO testTBL1 VALUES (1, '홍길동', 25);
```

주의점 2: 테이블 이름 뒤에 입력할 열의 목록을 나열해줘야 함.

```
INSERT INTO testTBL1(id, userName) VALUES (2, '설현');
```

주의점 3: 열의 순서를 바꿔서 입력하고 싶을 때는 열 이름을 입력할 순서에 맞춰 나열해줘야 함.

```
INSERT INTO testTBL1(userName, age, id) VALUES ('지민', 26, 3);
```

SECTION 02 데이터의 변경을 위한 SQL문

◦ 자동으로 증가하는 시퀀스

```
CREATE TABLE testTBL2
(id NUMBER(4),
userName NCHAR(3),
age NUMBER(2),
nation NCHAR(4) DEFAULT '대한민국');
```

[testTBL2를 생성]

```
CREATE SEQUENCE idSEQ
START WITH 1 -- 시작값
INCREMENT BY 1 ; -- 증가값
```

[시퀀스를 생성]

```
INSERT INTO testTBL2 VALUES (idSEQ.NEXTVAL, '유나' ,25 , DEFAULT);
INSERT INTO testTBL2 VALUES (idSEQ.NEXTVAL, '혜정' ,24 , '영국');
SELECT * FROM testTBL2;
```

[데이터를 입력]

ID	USERNAME	AGE	NATION
1	유나	25	대한민국
2	혜정	24	영국

[그림 6-57] 쿼리 실행 결과

```
INSERT INTO testTBL2 VALUES (11, '조위' , 18, '대만'); -- 강제로 11 입력
ALTER SEQUENCE idSEQ
INCREMENT BY 10; -- 증가값을 다시 설정(현재 자동으로 2까지 입력되었으므로, 10을 더하면
-- 다음 값은 12가 됨)
INSERT INTO testTBL2 VALUES (idSEQ.NEXTVAL, '미나' , 21, '일본'); -- 12가 들어감
ALTER SEQUENCE idSEQ
INCREMENT BY 1; -- 다시 증가값을 1로 변경 (이후는 13부터 들어감)
SELECT * FROM testTBL2;
```

ID	USERNAME	AGE	NATION
1	유나	25	대한민국
2	혜정	24	영국
3	11 조위	18	대만
4	12 미나	21	일본

[id 열의 값이 1부터 차례로 들어가는 것을 확인]

[그림 6-58] SEQUENCE 사용 결과


```
SELECT idSEQ.CURRVAL FROM DUAL;
```

[시퀀스의 현재 값을 확인]

SECTION 02 데이터의 변경을 위한 SQL문

- 자동으로 증가하는 시퀀스

```
CREATE TABLE testTBL3 (id NUMBER(3));
CREATE SEQUENCE cycleSEQ
  START WITH 100
  INCREMENT BY 100
  MINVALUE 100 -- 최소값
  MAXVALUE 300 -- 최대값
  CYCLE        -- 반복 설정
  NOCACHE ;    -- 캐시 사용 안 함
INSERT INTO testTBL3 VALUES (cycleSEQ.NEXTVAL);
INSERT INTO testTBL3 VALUES (cycleSEQ.NEXTVAL);
INSERT INTO testTBL3 VALUES (cycleSEQ.NEXTVAL);
INSERT INTO testTBL3 VALUES (cycleSEQ.NEXTVAL);
SELECT * FROM testTBL3;
```



1	100
2	200
3	300
4	100

[특정 범위의 값이 계속 반복되어서 입력]

[그림 6-59] SEQUENCE 활용

SECTION 02 데이터의 변경을 위한 SQL문

- 대량의 샘플 데이터 생성

형식 :

```
INSERT INTO 테이블이름 (열 이름1, 열 이름2, ...)  
SELECT문 ;
```

[기본형식]

```
CREATE TABLE testTBL4 (empID NUMBER(6), FirstName VARCHAR2(20),  
    LastName VARCHAR2(25), Phone VARCHAR2(20));  
INSERT INTO testTBL4  
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, PHONE_NUMBER  
FROM HR.employees ;
```

결과 메시지:

107개 행 이(가) 삽입되었습니다.

[HR.employees의 데이터를 가져와서 입력]

```
CREATE TABLE testTBL5 AS  
(SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME, PHONE_NUMBER  
FROM HR.employees) ;
```

[테이블의 정의까지 생략]

SECTION 02 데이터의 변경을 위한 SQL문

2.2 데이터의 수정 : UPDATE

```
UPDATE 테이블이름  
SET 열1=값1, 열2=값2 ...  
WHERE 조건 ;
```

[기존에 입력되어 있는 값을 변경하는 UPDATE문 형식]

```
UPDATE testTBL4  
SET Phone = '없음'  
WHERE FirstName = 'David';
```

['David'의 Phone을 '없음'으로 변경하는 예]

```
UPDATE buyTBL SET price = price * 1.5 ;
```

[예시 : 구매 테이블에서 현재의 단가가 모두 1.5배 인상]

SECTION 02 데이터의 변경을 위한 SQL문

2.3 데이터의 삭제 : DELETE FROM

- DELETE도 UPDATE와 거의 비슷한 개념이다. DELETE는 행 단위로 삭제함.

형식 :
DELETE FROM 테이블이름 WHERE 조건 ;

[DELETE 문 형식]

```
DELETE FROM testTBL4 WHERE FirstName = 'Peter';
```

[testTBL4에서 'Peter' 사용자 삭제 예시]

```
ROLLBACK; -- 앞에서 지운 'Peter'를 되돌림  
DELETE FROM testTBL4 WHERE FirstName = 'Peter' AND ROWNUM <= 2;
```

[3건의 'Peter' 중에서 2건만 삭제하려면 ROWNUM을 활용]

SECTION 02 데이터의 변경을 위한 SQL문

2.4 조건부 데이터 변경 : MERGE

- MERGE문은 하나의 문장에서 경우에 따라서 INSERT, UPDATE, delete를 수행할 수 있는 구문임.

```
형식 :  
MERGE [ hint ]  
  INTO [ schema. ] { table | view } [ t_alias ]  
  USING { [ schema. ] { table | view }  
        | subquery  
        } [ t_alias ]  
  ON ( condition )  
  [WHEN MATCHED THEN  
    UPDATE SET column = { expr | DEFAULT }  
    [, column = { expr | DEFAULT } ]...  
    [ where_clause ]  
    [ DELETE where_clause ]]  
  [WHEN NOT MATCHED THEN  
    INSERT [ (column [, column ]...) ]  
    VALUES ({ expr [, expr ]... | DEFAULT } )  
    [ where_clause ]]  
  [LOG ERRORS  
    [ INTO [schema.] table ]  
    [ (simple_expression) ]  
    [ REJECT LIMIT { integer | UNLIMITED } ] ] ;
```

[MERGE문의 원형]

SECTION 02 데이터의 변경을 위한 SQL문

실습5

위 시나리오 대로 MERGE 구문의 활용을 연습하자.

step 1

우선 멤버 테이블(memberTBL)을 정의하고, 데이터를 입력하자. 지금은 연습 중이므로 기존 userTBL에서 아이디, 이름, 주소만 가져와서 간단히 만들겠다. 앞에서 배운 CREATE TABLE... AS문을 활용하면 된다.

```
CREATE TABLE memberTBL AS
(SELECT userID, userName, addr FROM userTbl );
SELECT * FROM memberTBL;
```

USERID	USERNAME	ADDR
1 LSG	이승거	서울
2 KDS	김병수	경남
3 KSK	김경호	전남
4 JYP	조용필	경기
5 SSK	성시경	서울
6 LJB	임재범	서울
7 YJS	윤종신	경남
8 EJM	윤지현	경북
9 JSM	조잔무	경기
10 BHK	배태환	서울

[그림 6-63] 멤버 테이블

step 2

변경 테이블(changeTBL)을 정의하고 데이터를 입력하자. 1명의 신규가입, 2명의 주소변경, 2명의 회원탈퇴가 있는 것으로 가정하자. 이런 방식으로 계속 변경된 데이터를 누적해 가면 된다.

```
CREATE TABLE changeTBL
( userID CHAR(8) ,
  userName NVARCHAR2(10),
  addr NCHAR(2),
  changeType NCHAR(4) -- 변경 사유
);
INSERT INTO changeTBL VALUES('TKV', '태권브이', '한국', '신규가입');
INSERT INTO changeTBL VALUES('LSG', null, '제주', '주소변경');
INSERT INTO changeTBL VALUES('LJB', null, '영국', '주소변경');
INSERT INTO changeTBL VALUES('BBK', null, '탈퇴', '회원탈퇴');
INSERT INTO changeTBL VALUES('SSK', null, '탈퇴', '회원탈퇴');
```

SECTION 02 데이터의 변경을 위한 SQL문

step 3

1주일이 지났다고 가정하고, 이제는 변경사유(chageType) 열에 의해서 기존 멤버 테이블의 데이터를 변경한다. 이 예에서 5개 행이 영향을 받을 것이다.

```
MERGE INTO memberTBL M  -- 변경될 테이블 (target 테이블)
-- 변경할 기준이 되는 테이블 (source 테이블)
USING (SELECT changeType, userID, userName, addr FROM changeTBL) C
ON (M.userID = C.userID)  -- userID를 기준으로 두 테이블을 비교한다.
-- target 테이블에 source 테이블의 행이 있으면 주소를 변경한다.
WHEN MATCHED THEN
    UPDATE SET M.addr = C.addr
    -- target 테이블에 source 테이블의 행이 있고, 사유가 '회원탈퇴'라면 해당 행을 삭제한다.
    DELETE WHERE C.changeType = '회원탈퇴'
-- target 테이블에 source 테이블의 행이 없으면 새로운 행을 추가한다.
WHEN NOT MATCHED THEN
    INSERT (userID, userName, addr) VALUES(C.userID, C.userName, C.addr);
```

step 4

멤버 테이블을 조회해 보자. 계획대로 1개 행이 추가되고, 2개 행은 주소가 변경되고, 2개 행은 삭제되었다.

```
SELECT * FROM memberTBL;
```

USERID	USERNAME	ADDR
1 LSG	이승기	제주
2 KDS	김철수	경남
3 KKH	김경호	전남
4 JYP	조용필	경기
5 LJB	임재범	영국
6 YJS	유종신	경남
7 EJM	문지현	경북
8 JKW	조관우	경기
9 TBY	태권브이	한국

[그림 6-64] 변경된 멤버 테이블