6장. SQL

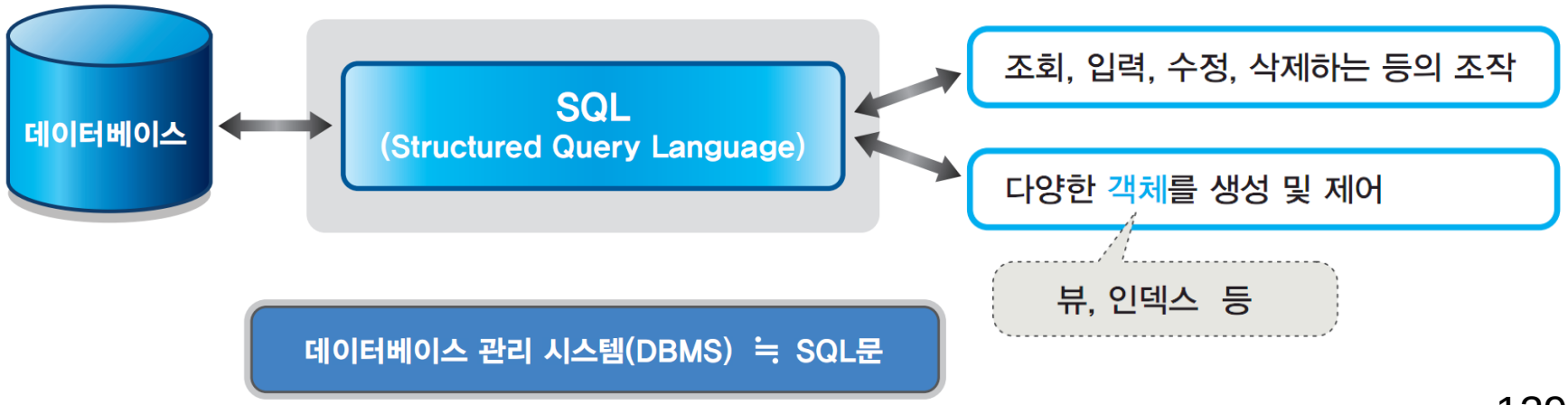
1 SQL

- **SQL**

SQL

SQL은 사용자와 데이터베이스 시스템 간에 의사소통을 하기 위한 언어.

- **데이터베이스와 SQL문**



129 Page

- **SQL**
  - 데이터베이스에 저장된 데이터를 조회, 입력, 수정, 삭제하는 등의 조작이나, 테이블을 비롯한 다양한 객체(시퀀스, 인덱스 등)를 생성 및 제어하는 역할

1-1 SQL의 역사

SQL

- 1974년에 IBM 연구소에서 데이터베이스 시스템, "시스템 R"을 질의하기 위해서 만들어진 구조화된 언어
- Structured English Query Language라고 이름을 지음
- SQL은 이후 IBM의 DB2와 SQL/DS 데이터베이스 시스템에서도 구현

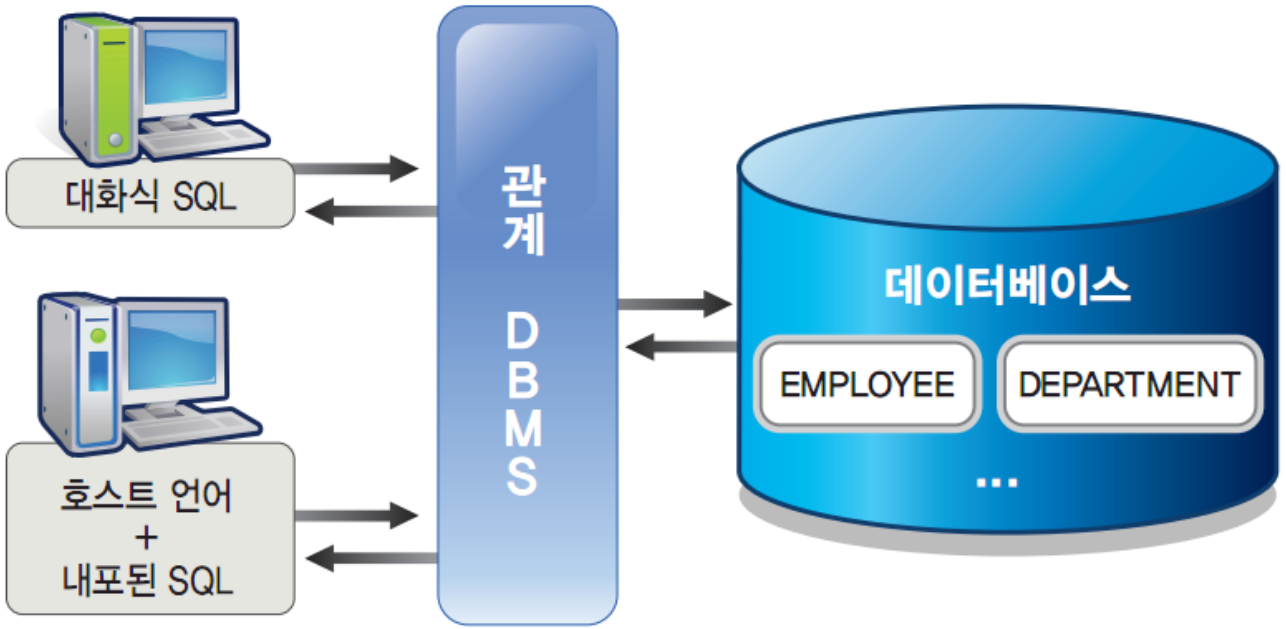
SQL 발전 역사

SEQUEL	Structured English Query Language의 약어로서 시스템 R 프로젝트에서 처음 제안됨
SQL	Structured Query Language의 약어로서 1983년 IBM의 DB2, 1991년 IBM의 SQL/DS에서 사용됨
SQL-86	1986년 미국 표준 연구소(ANSI)와 1987년 국제 표준 기구(ISO)에서 표준 언어로 채택
SQL-89	무결성 제약조건 강화
SQL2(SQL-92)	1992년 새로운 데이터베이스 조작어 기능 추가
SQL3(SQL-99)	1999년 객체 지향과 순환 멀티미디어 기능 추가
SQL4(SQL-2003)	2003년 객체 개념을 지원하는 기능 추가

1-2 SQL의 유형별 종류

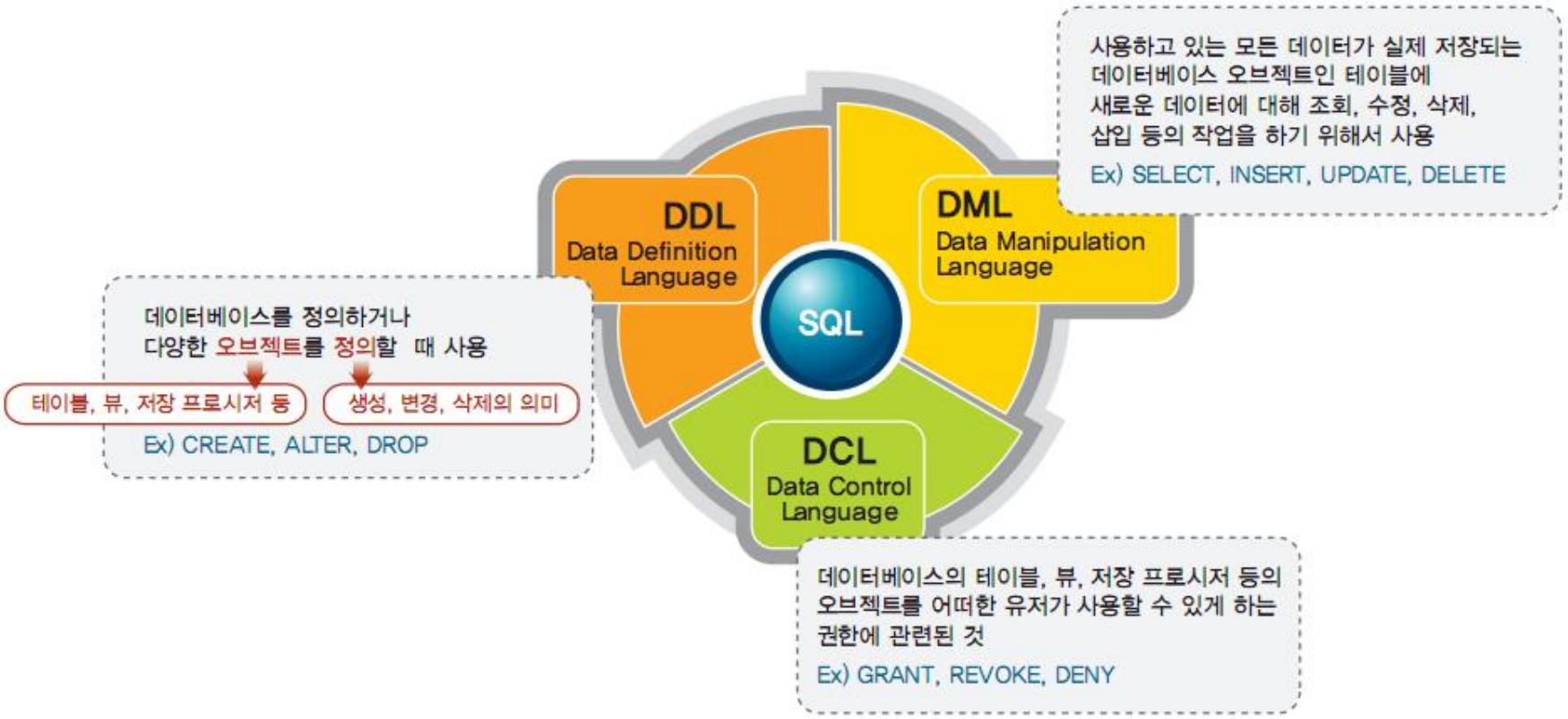
- **대화식 SQL**
  - 화면에 명령을 넣고, 결과가 바로 화면으로 나오는 방식

- **내포된 SQL**
  - Java, COBOL, C, C++ 등과 같은 고급 프로그래밍 언어 사이에 SQL문을 끼어 넣는 방식



1-2 SQL의 유형별 종류

- DDL, DML, DCL



131 Page

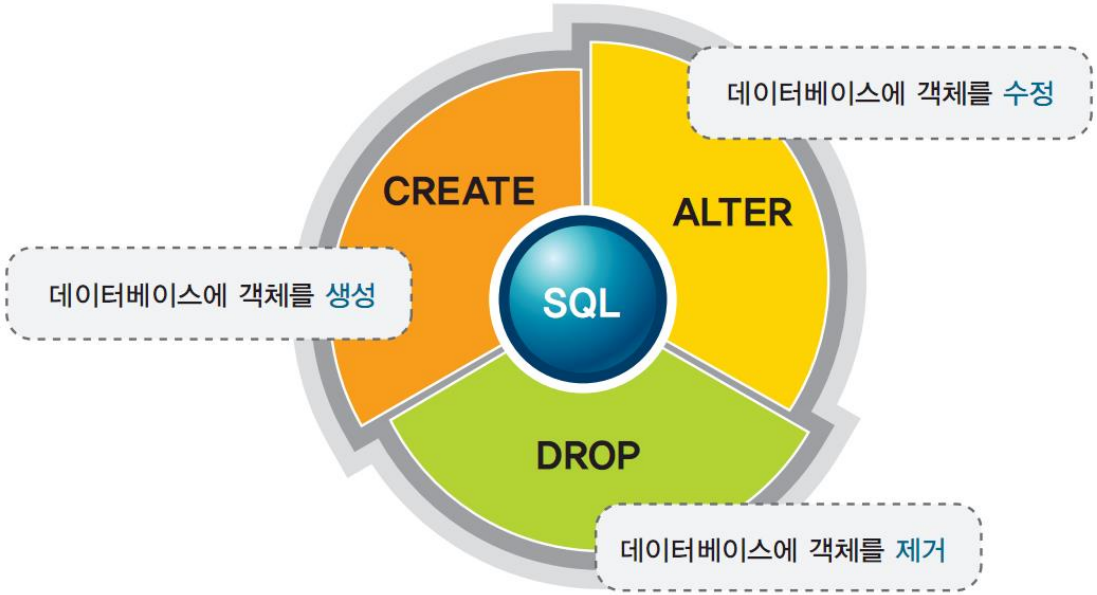
1-2 SQL의 유형별 종류

- SQL 명령문의 유형

유형	명령문
DML:Data ManipulatiON Language (데이터 조작어)-데이터 변경 시 사용	SELECT (데이터 검색 시 사용) INSERT (데이터 입력) UPDATE (데이터 수정) DELETE (데이터 삭제)
DDL:Data DefinitiON Language (데이터 정의어)-객체 생성 및 변경 시 사용	CREATE (데이터베이스 생성) ALTER (데이터베이스 변경) DROP (데이터베이스 삭제) RENAME (데이터베이스 객체이름 변경) TRUNCATE (데이터베이스 저장 공간 삭제)
TCL:TransactiON CONtrol Language (트랜잭션 처리어)	COMMIT (트랜잭션의 정상적인 종료처리) ROLLBACK (트랜잭션 취소) SAVEPOINT (트랜잭션 내에 임시 저장점 설정)
DCL:Data CONtrol Language (데이터 제어어)	GRANT (데이터베이스에 대한 일련의 권한 부여) REVOKE (데이터베이스에 대한 일련의 권한 취소)

2 데이터 정의어(DDL)

- 데이터 정의어(DDL)



133 Page

- DDL 명령문의 종류

SQL문	내용
CREATE	데이터베이스 및 객체 생성
DROP	데이터베이스 및 객체 삭제
ALTER	기존에 존재하는 데이터베이스 객체를 변경

3 데이터 조작어(DML)

3-1 데이터 검색(SELECT)

SQL문의 기본 형식

- Sql은 대소문자를 구별하지 않는다.
  - 보통은 소문자
- 여러 줄로 작성이 가능하다.
- 구문은 주로 나누어 작성한다.
- 세미콜론(;)으로 문장을 마친다.

3 데이터 조작어(DML)

3-1 데이터 검색(SELECT)

- **SELECT 문의 기본 형식**

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

```
SELECT 열이름
FROM 테이블이름
WHERE 조건
```

3-1 데이터 검색(SELECT)

- **과정 테이블의 내용을 살펴보기 위한 쿼리문**

예

```
SELECT  *     FROM     COURSE
  ①     ②       ③          ④
```

- SELECT 는 데이터베이스 내에 저장되어 있는 테이블을 조회하기 위한 명령어
- SELECT 다음에는 보고자 하는 대상의 칼럼 명을 기술한다. SELECT 다음에 *를 기술하면 지정된 테이블(④COURSE)의 모든 칼럼을 조회
- FROM 다음에는 보고자 하는 대상의 테이블 명을 기술
- ④에 COURSE를 기술하였기에 COURSE 테이블에 등록된 과정의 정보보기

Tip

*는 에스테리스크라고 불리는 표시로서 테이블 내의 "모든(all)" 칼럼의 데이터를 선정하고자 할 때 사용한다. 해당 테이블의 모든 칼럼 명을 지정할 필요가 없기 때문에 편리하게 사용되는 특수 문자이다.

3-1 데이터 검색(SELECT)

- **칼럼 이름을 명시해서 특정 칼럼만 보기**

예

```
SELECT * FROM STUDENT
```

- 칼럼 명은 STU_ID
- 칼럼 명은 STU_NAME

- **STUDENT테이블의 STU_ID, STU_NAME 칼럼 내용만 출력**

예

```
SELECT STU_ID, STU_NAME        FROM STUDENT
```

3 데이터 조작어(DML)

3-1 데이터 검색(SELECT)

- 특정 칼럼만 보기(예제)

sample.sql - AD...-PC₩admin (52))*

```
select STU_ID, STU_NAME
from STUDENT
```

결과 | 메시지

	STU_ID	STU_NAME
1	101	문종헌
2	102	오한솔
3	103	제용석
4	104	정국철
5	105	박홍진
6	106	김현우
7	107	박시준
8	108	김준형
9	109	문혜진
10	110	박기석
11	111	윤효선
12	112	안창범
13	113	공지훈
14	114	이봉림
15	115	안창범
16	116	장희성

in-PC₩admin (52) | EduManager | 00:00:00 | 16개의 행

sample.sql - AD...-PC₩admin (52))*

```
select *
from STUDENT
```

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	101	문종헌	24	moon@nate.com	10	M
2	102	오한솔	22	five@nate.com	20	M
3	103	제용석	22	again@nate.com	20	M
4	104	정국철	22	cook@nate.com	20	M
5	105	박홍진	24	red@nate.com	10	M
6	106	김현우	21	kim@nate.com	20	M
7	107	박시준	22	season@nate.com	20	M
8	108	김준형	27	brother@nate.com	10	M
9	109	문혜진	22	sun@nate.com	20	F
10	110	박기석	34	flag@nate.com	10	M
11	111	윤효선	24	good@nate.com	30	F
12	112	안창범	34	window@nate.com	30	M
13	113	공지훈	28	empty@nate.com	10	M
14	114	이봉림	29	bbong@nate.com	10	M
15	115	안창범	24	chang@nate.com	30	M
16	116	장희성	34	shine@nate.com	10	M

쿼리가 실행되었습니다. | ADMIN-PC(10.0 RTM) | admin-PC₩admin (52) | EduManager | 00:00:00 | 16개의 행

3-1 데이터 검색(SELECT)

- **중복된 데이터를 한 번씩만 출력하게 하는 DISTINCT**
  - 중복된 것을 골라서 세기 어려울 때 사용하는 구문
  - 테이블의 크기가 클수록 효율적
  - 중복된 것은 1개씩만 보여주면서 출력

- **DISTINCT로 중복된 데이터를 한 번씩만 출력하기**

3-1 데이터 검색(SELECT)

- **DISTINCT로 중복된 데이터를 한 번씩만 출력하기(예제)**

예

```
SELECT COU_ID
FROM STUDENT
```

명령 프롬프트 - sqlplus sc...

```
SQL> select cou_id from student;

    COU_ID
----------
        10
        20
        20
        20
        10
        20
        20
        10
        20
        10
        30
        30
        10
        10
        30
        10

16 개의 행이 선택되었습니다.

SQL>
```

예

```
SELECT distinct COU_ID
FROM STUDENT
```

명령 프롬프트 - sqlplus scott/11...

```
SQL> select distinct cou_id from student;

    COU_ID
----------
        30
        20
        10

SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
```

3-1 데이터 검색(SELECT)

- 특정 조건의 데이터만 조회

```
SELECT * [column1, column2, .. ,columnn]
FROM  table_name
WHERE 조건절;
```

- 나이(AGE)가 30 이상인 학생 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE AGE >= 30
```

조건절

WHERE age 〉= 30;

① 칼럼 ② 연산자 ③ 비교 대상 값

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	110	박기석	34	flag@nate.com	10	M
2	112	안창범	34	window@nate.com	30	M
3	116	장희성	34	shine@nate.com	10	M

171 Page

3-1 데이터 검색(SELECT)

비교 연산자

- (같다), >(크다), >=(크거나 같다), <(작다), <=(작거나 같다), <>(같지 않다)

연산자	의미	예제
=	같다	SELECT STU_ID, STU_NAME, AGE FROM STUDENT WHERE AGE = 34;
>	보다 크다	SELECT STU_ID, STU_NAME, AGE FROM STUDENT WHERE AGE > 30;
<	보다 작다	SELECT STU_ID, STU_NAME, AGE FROM STUDENT WHERE AGE < 30;
>=	보다 크거나 같다	SELECT STU_ID, STU_NAME, AGE FROM STUDENT WHERE AGE >= 30;
<=	보다 작거나 같다	SELECT STU_ID, STU_NAME, AGE FROM STUDENT WHERE AGE <= 30;
<>, !=, ^=	다르다	SELECT STU_ID, STU_NAME, AGE FROM STUDENT WHERE AGE <> 34;

3-1 데이터 검색(SELECT)

- 10번 과정 소속 학생 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE COU_ID=10
```

- 특정 조건의 데이터만 조회

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	101	문종헌	24	moon@nate.com	10	M
2	105	박홍진	24	red@nate.com	10	M
3	108	김준형	27	brother@nate.c...	10	M
4	110	박기석	34	flag@nate.com	10	M
5	113	공지훈	28	empty@nate.co...	10	M
6	114	이봉림	29	bbong@nate.c...	10	M
7	116	장희성	34	shine@nate.com	10	M

3-1 데이터 검색(SELECT)

● 이름이 김준형인 학생을 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE STU_NAME='김준형'
```

- 문자열은 싱글쿼트(''), 숫자는 그대로 조건식에 표현

● 특정 조건의 데이터만 조회

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	108	김준형	27	brother@nate.com	10	M

173 Page

3-1 데이터 검색(SELECT)

논리 연산자

- AND, OR, NOT

연산자	의미
AND	두 가지 조건을 모두 만족해야만 검색할 수 있다. SELECT * FROM STUDENT WHERE COU_ID=10 AND SEX= 'F';
OR	두 가지 조건 중에서 한 가지만 만족하더라도 검색할 수 있다. SELECT * FROM STUDENT WHERE COU_ID=10 OR SEX= 'F';
NOT	조건에 만족하지 못하는 것만 검색한다. SELECT * FROM STUDENT WHERE NOT COU_ID=10;

BETWEEN AND 연산자

예

```
SELECT * FROM STUDENT
WHERE AGE >= 21 AND AGE <= 29
```

- BETWEEN AND 연산자형식

```
column_name BETWEEN A AND B
```

3-1 데이터 검색(SELECT)

- 연령이 21에서 29 사이인 학생 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE AGE BETWEEN 21 AND 29
```

- BETWEEN AND 연산자 결과

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	101	문종헌	24	moon@nate.com	10	M
2	102	오한솔	22	five@nate.com	20	M
3	103	제용석	22	again@nate.com	20	M
4	104	정국철	22	cook@nate.com	20	M

175 Page

3-1 데이터 검색(SELECT)

- **IN 연산자**

예

```
SELECT * FROM STUDENT
WHERE AGE=21 OR  AGE=27 OR  AGE=28
```

- IN 연산자형식

```
column_name IN (A, B, C)
```

- **연령이 21이거나 27이거나 28인 학생 출력하기(예제)**

```
1:  SELECT * FROM STUDENT
2:  WHERE AGE IN(21, 27, 28)
```

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	106	김현우	21	kim@nate.com	20	M
2	108	김준형	27	brother@nate.com	10	M
3	113	공지훈	28	empty@nate.com	10	M

176 Page

3-1 데이터 검색(SELECT)

LIKE 연산자와 와일드카드형식

```
column_name LIKE pattern
```

와일드카드	의미
%	문자가 없거나, 하나 이상의 문자가 어떤 값이 와도 상관없다.
_	하나의 문자가 어떤 값이 와도 상관없다.

- %나, _ 검색할 때는 ₩%, ₩_
- ' 검색할 때는 '' 또는 ₩'

3-1 데이터 검색(SELECT)

● 와일드카드(%) 사용하기

- 성이 '김'인 학생을 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE STU_NAME LIKE '김%'
```

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	106	김현우	21	kim@nate.com	20	M
2	108	김준형	27	brother@nate.com	10	M

● 와일드카드(_) 사용하기

- 이름의 두 번째 글자가 '봉'이고 그 뒤는 무엇이든 관계없는 사원 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE STU_NAME LIKE '_봉%'
```

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	114	이봉림	29	bbong@nate.com	10	M

3-1 데이터 검색(SELECT)

● NULL을 위한 IS NULL과 IS NOT NULL 연산자

- 이름과 과정번호와 이메일 출력하기(예제)

```
1:  SELECT STU_NAME, COU_ID, STU_EMAIL
2:  FROM STUDENT
```

결과 | 메시지

	STU_NAME	COU_ID	STU_EMAIL
1	문종헌	10	moon@nate.com
2	오한솔	20	five@nate.com
3	제용석	20	again@nate.com
4	정국철	20	cook@nate.com
5	박홍진	10	red@nate.com
6	김현우	20	NULL
7	박시준	20	season@nate.c...
8	김준형	10	brother@nate.c...
9	문혜진	20	NULL
10	박기석	10	flag@nate.com
11	윤효선	30	good@nate.com
12	안창범	30	window@nate....
13	공지훈	10	NULL
14	이봉림	10	bbong@nate.c...
15	안창범	30	chang@nate.c...
16	장희성	10	shine@nate.com

178 Page

3-1 데이터 검색(SELECT)

● NULL을 위한 IS NULL과 IS NOT NULL 연산자

- 이메일이 NULL인 학생을 찾는데 실패하는 예제

```
1:  SELECT * FROM STUDENT
2:  WHERE STU_EMAIL=NULL
```

결과 | 메시지

STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX

179 Page

- 특정 칼럼 값이 NULL 값인지를 비교할 경우 연산자(=)를 사용하지 않고 IS 연산자를 사용

대상칼럼 IS (연산자) NULL(비교값)

3-1 데이터 검색(SELECT)

● NULL을 위한 IS NULL과 IS NOT NULL 연산자

- 이메일이 NULL인 학생 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE STU_EMAIL IS NULL
```

- 널 값 검색-1

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	106	김현우	21	NULL	20	M
2	109	문혜진	22	NULL	20	F
3	113	공지훈	28	NULL	10	M

180 Page

3-1 데이터 검색(SELECT)

● NULL을 위한 IS NULL과 IS NOT NULL 연산자

- 이메일이 NULL이 아닌 학생 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  WHERE STU_EMAIL IS NOT NULL
```

- 널 값 검색-2

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	101	문종헌	24	moon@nate.com	10	M
2	102	오한솔	22	five@nate.com	20	M
3	103	제용석	22	again@nate.com	20	M
4	104	정국철	22	cook@nate.com	20	M
5	105	박홍진	24	red@nate.com	10	M
6	107	박시준	22	season@nate.c...	20	M
7	108	김준형	27	brother@nate.c...	10	M

180 Page

3-1 데이터 검색(SELECT)

정렬을 위한 ORDER BY 절

- 정렬이란 크기 순서대로 나열하는 것을 의미
- 작은 것이 위에 출력되고 아래로 갈수록 큰 값이 출력하는 오름차순(ascending) 정렬 방식
- 큰 값이 위에 출력되고 아래로 갈수록 작은 값이 출력하는 내림차순(descending) 정렬 방식

ORDER BY형식

```
SELECT * [column1, column2, .. ,columnn]
FROM  table_name
WHERE 조건절
ORDER BY column_name  sorting
```

ORDER BY절의 정렬 방식

	ASC(오름차순)	DESC(내그차순)
숫자	작은 값부터 정렬	큰 값부터 정렬
문자	사전 순서로 정렬	사전 반대 순서로 정렬
날짜	빠른 날짜 순서로 정렬	늦은 날짜 순서로 정렬
NULL	가장 마지막에 나온다.	가장 먼저 나온다.

3-1 데이터 검색(SELECT)

● 오름차순 정렬을 위한 ASC

- 나이를 기준으로 오름차순으로 정렬하여 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  ORDER BY AGE ASC
```

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	106	김현우	21	NULL	20	M
2	107	박시준	22	season@nate.com	20	M
3	109	문혜진	22	NULL	20	F
4	102	오한솔	22	five@nate.com	20	M
5	103	제용석	22	again@nate.com	20	M
6	104	정국철	22	cook@nate.com	20	M
7	105	박홍진	24	red@nate.com	10	M
8	111	윤효선	24	good@nate.com	30	F
9	115	안창범	24	chang@nate.com	30	M
10	101	문종헌	24	moon@nate.com	10	M
11	108	김준형	27	brother@nate.com	10	M
12	113	공지훈	28	NULL	10	M
13	114	이봉림	29	bbong@nate.com	10	M
14	112	안창범	34	window@nate.c...	30	M
15	116	장희성	34	shine@nate.com	10	M
16	110	박기석	34	flag@nate.com	10	M

182 Page

3-1 데이터 검색(SELECT)

● 내림차순 정렬을 위한 DESC

- 나이가 많은 사람부터 적은 사람 순으로 순차적으로 출력하기(예제)

```
1:  SELECT * FROM STUDENT
2:  ORDER BY AGE DESC
```

결과 | 메시지

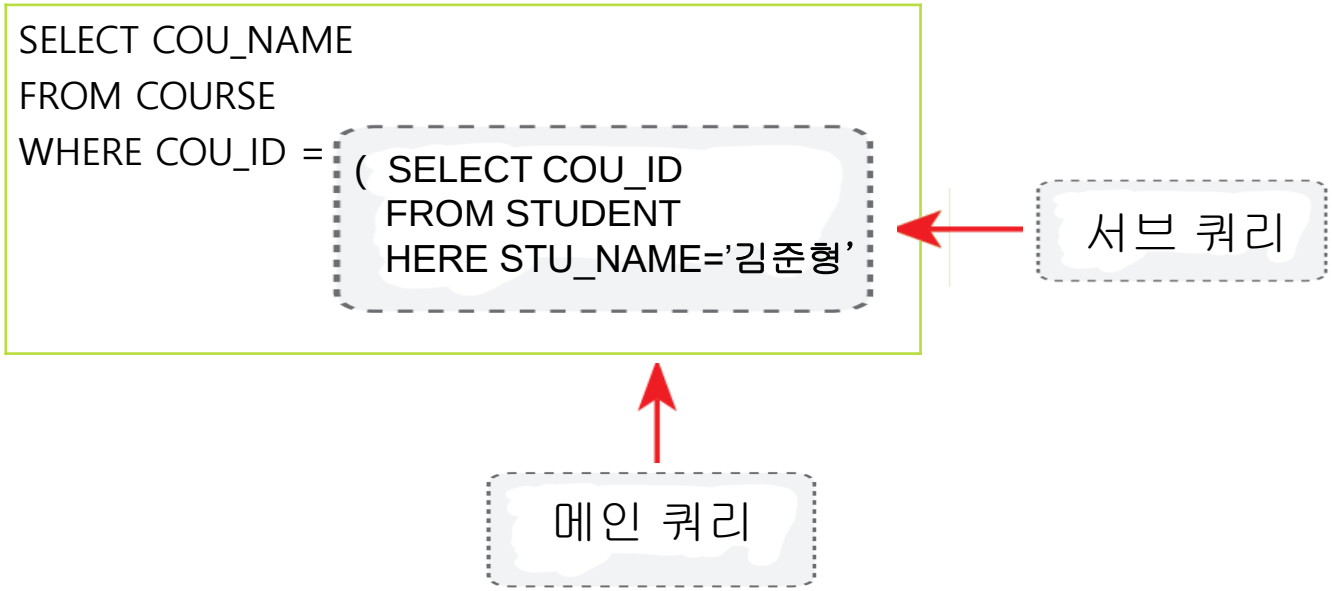
	STU_ID	STU_NAME	AGE	STU_EMAIL	COU_ID	SEX
1	110	박기석	34	flag@nate.com	10	M
2	112	안창범	34	window@nate.com	30	M
3	116	장희성	34	shine@nate.com	10	M
4	114	이봉림	29	bbong@nate.com	10	M
5	113	공지훈	28	NULL	10	M
6	108	김준형	27	brother@nate.com	10	M
7	105	박홍진	24	red@nate.com	10	M
8	101	문종헌	24	moon@nate.com	10	M
9	111	윤효선	24	good@nate.com	30	F
10	115	안창범	24	chang@nate.com	30	M
11	107	박시준	22	season@nate.com	20	M
12	102	오한솔	22	five@nate.com	20	M
13	103	제용석	22	again@nate.com	20	M
14	104	정국철	22	cook@nate.com	20	M
15	109	문혜진	22	NULL	20	F
16	106	김현우	21	NULL	20	M

183 Page

3-1 데이터 검색(SELECT)

● 서브 쿼리

- 메인 쿼리와 서브 쿼리 관계



- 서브 쿼리형식

```
SELECT SELECT _LIST                          → 메인 쿼리
FROM TABLE_LIST
WHERE COLUMN =  (SELECT EXPRESSION  → 서브 쿼리
                FROM .....
                WHERE.....)
```

3-1 데이터 검색(SELECT)

● 서브 쿼리에는 몇 가지 규칙

1. 서브 쿼리 안에 서브 쿼리가 들어갈 수 있다. 이것을 네스팅(nesting)이라고 하며 메모리가 허용하는 한 무제한으로 중첩할 수 있다.
2. 메인 쿼리에서 서브 쿼리의 결과 값을 조건으로 사용할 때 SOME, ANY 또는 ALL 연산자를 사용하지 않는 일반적인 경우에는 서브 쿼리에서는 하나의 레코드 값만 리턴해야 한다. 그러므로 대부분의 경우 서브 쿼리에는 GROUP BY, HAVING문을 사용할 수 없다.
3. 서브 쿼리에서 ORDER BY문은 TOP 연산자와 함께 있을 때만 사용 가능하다.
4. 서브 쿼리에서 SELECT 하지 않은 칼럼은 주 쿼리에서 사용할 수 없다.

- 단일 행(Single Row) 서브 쿼리는 수행 결과가 오직 하나의 로우(행, row)만을 반환하는 서브 쿼리를 가진 것

3-1 데이터 검색(SELECT)

● 다중 행 서브 쿼리

- 서브 쿼리에서 반환되는 결과가 하나 이상의 행일 때 사용하는 서브 쿼리

종류	의미
IN	메인 쿼리의 비교 조건('=' 연산자로 비교할 경우)이 서브 쿼리의 결과 중에서 하나라도 일치하면 참이다.
ANY, SOME	메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 하나 이상이 일치하면 참이다.("any ="은 in과 동일)
ALL	메인 쿼리의 비교 조건이 서브 쿼리의 검색 결과와 모든 값이 일치하면 참이다.
EXISTS	메인 쿼리의 비교 조건이 서브 쿼리의 결과 중에서 만족하는 값이 하나라도 존재하면 참이다.

3-1 데이터 검색(SELECT)

● IN 연산자

- 메인 쿼리의 비교 조건에서 서브 쿼리의 출력 결과와 하나라도 일치하면 메인 쿼리의 WHERE 절이 참이 되도록 하는 연산자

- 25세 이상인 학생들로 구성된 과정의 학생 정보 출력(예제)

```
1:  SELECT STU_NAME, age, COU_ID
2:  FROM STUDENT
3:  WHERE COU_ID IN (SELECT DISTINCT COU_ID
4:                   FROM STUDENT
5:                   WHERE age>=25)
```

- IN 연산자 사용 예

결과 | 메시지

	STU_NAME	age	COU_ID
1	문종헌	24	10
2	박홍진	24	10
3	김준형	27	10
4	박기석	34	10
5	윤효선	24	30
6	안창범	34	30
7	공지훈	28	10
8	이봉림	29	10
9	안창범	24	30
10	장희성	34	10

192 Page

4.4 SELECT 문(계속)

예 : IN

(3426 IN
2106
3426
3011
)은 참이다.

(1365 IN
2106
3426
3011
)은 거짓이다.

(1365 NOT IN
2106
3426
3011
)은 참이다.

4.4 SELECT 문(계속)

예 : ANY

(3000000 < ANY | 2500000
3000000
4000000) 은 참이다.

(4000000 < ANY | 2500000
3000000
4000000) 은 거짓이다.

4.4 SELECT 문(계속)

예 : ALL

(3000000 < ALL | 2500000
3000000
4000000 |)은 거짓이다.

(1500000 < ALL | 2500000
3000000
4000000 |)은 참이다.

(3000000 = ALL | 2500000
3000000
4000000 |)은 거짓이다.

(1500000 <> ALL | 2500000
3000000
4000000 |)은 참이다.

4.4 SELECT 문(계속)

예 : IN을 사용한 질의

질의: 영업부나 개발부에 근무하는 사원들의 이름을 검색하라.

```
SELECT    EMPNAME
FROM      EMPLOYEE
WHERE     DNO  IN  (1, 3)
          (SELECT   DEPTNO
           FROM     DEPARTMENT
           WHERE    DEPTNAME = '영업' OR DEPTNAME = '개발') ;
```

4.4 SELECT 문(계속)

이 질의를 중첩 질의를 사용하지 않은 다음과 같은 조인 질의로 나타낼 수 있다. 실제로, 중첩 질의를 사용하여 표현된 대부분의 질의를 중첩 질의가 없는 조인 질의로 표현할 수 있다.

```
SELECT    EMPNAME
FROM      EMPLOYEE E, DEPARTMENT D
WHERE     E.DNO = D.DEPTNO
          AND (D.DEPTNAME = '영업' OR D.DEPTNAME = '개발');
```

EMPNAME
박영권
이수민
최종철
김상원

3-1 데이터 검색(SELECT)

- **별칭사용(alias)**

컬럼에 사용

```
SELECT stu_name as name from student;
SELECT stu_name "Name" from student;
```

테이블에 사용

```
SELECT stu_name "Name" from student s;
```

3-1 데이터 검색(SELECT)

별칭사용(alias)

- AS는 생략 가능
- ""사용하면 예약어도 사용가능
- 숫자로 시작하는 객체명은 사용불가능.(mysql, mssql은 사용가능)
- 숫자만으로 구성된 객체명은 사용불가능.
- ORDER BY절에 사용가능
- WHERE절에 사용불가능

3-1 데이터 검색(SELECT)

● 집계함수

- 수(COUNT), 합계(SUM), 평균(AVG), 최대값(MAX), 최소값(MIN) 등을 구하기 위한 함수구성

예

SELECT AVG(AGE) FROM STUDENT ➡1개의 행으로 결과가 구해짐

- 집계 함수

129 Page

3-1 데이터 검색(SELECT)

- 집계함수의 종류

구분	설명
SUM	그룹의 누적 합계를 반환한다.
AVG	그룹의 평균을 반환한다.
COUNT	그룹의 총 개수를 반환한다.
COUNT_BIG	그룹의 총 개수를 반환한다. 단, 결과 값이 bigint형이다.
MAX	그룹의 최대값을 반환한다.
MIN	그룹의 최소값을 반환한다.
STDDEV	그룹의 표준편차를 반환한다

- NULL은 해당 정보가 무시된다.

3-1 데이터 검색(SELECT)

● 합계 구하는 SUM 함수

- 학생들의 나이 총합 출력하기(예제)

```
1:  SELECT SUM(AGE) AS  나이총합
2:  FROM STUDENT
```

- 합계 구하는 SUM 함수

결과 | 메시지

	나이 총합
1	413

184 Page

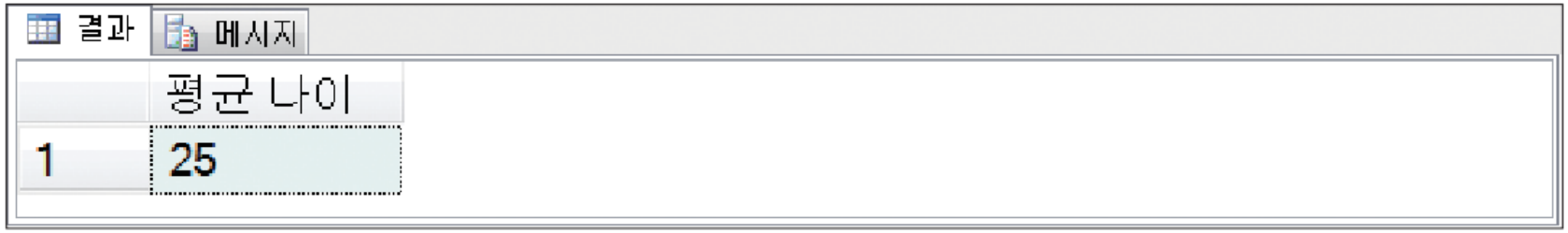
3-1 데이터 검색(SELECT)

● 평균 구하는 AVG 함수

- 학생들의 나이 총합 출력하기(예제)

```
1:  SELECT AVG (AGE) AS 평균나이
2:  FROM STUDENT
```

- 평균 구하는 AVG 함수



● Null은 무시

- SELECT COUNT(*) FROM STUDENT; // 모든 학생 집계
- SELECT COUNT(stu_email) FROM STUDENT; // 이메일이 있는 학생만 집계

3-1 데이터 검색(SELECT)

GROUP BY

- GROUP BY 절 뒤에 해당 칼럼을 기술

```
SELECT 칼럼 명, 집계함수
FROM 테이블 명
WHERE 조건 (연산자)
GROUP BY 칼럼 명;
ORDER BY 칼럼 명;
```

GROUP BY

- 소속 과정별 평균 연령을 과정 번호와 함께 출력하기(예제)

```
1:  SELECT COU_ID, AVG (AGE) AS [평균 나이] FROM STUDENT
2:  GROUP BY COU_ID
```

- 그룹별 평균 급여 구하기

결과 | 메시지

	COU_ID	평균 나이
1	10	28
2	20	21
3	30	27

185 Page

3-1 데이터 검색(SELECT)

- **서브 쿼리와 함께 조합**
  - 가장 많은 급여와 가장 적은 급여를 받는 사원의 이름을 출력하라.

```
SELECT first_name, MAX(salary), MIN(salary)
FROM employees
```

=> 오류. 이름이 그룹에 사용하는 것이 아니라서

```
SELECT first_name, MAX(salary), MIN(salary)
FROM employees
GROUP BY first_name;
```

  - 그냥 모두 다 나왔다.

```
SELECT first_name
FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees)
OR salary = (SELECT MIN(salary) FROM employees);
```

3-1 데이터 검색(SELECT)

● HAVING 조건

- 과정별 평균 연령이 25세 이상인 과정의 번호와 평균 연령 구하기(예제)

```
1:  SELECT COU_ID, AVG (AGE) AS [평균 나이]
2:  FROM STUDENT
3:  WHERE AVG(AGE) >= 25
4:  GROUP BY COU_ID
```

=> 오류

```
1:  SELECT COU_ID, AVG (AGE) AS [평균 나이]
2:  FROM STUDENT
3:  GROUP BY COU_ID
4:  HAVING AVG(AGE) >= 25
```

결과 | 메시지

	COU_ID	평균 나이
1	10	28
2	20	21
3	30	27

결과 | 메시지

	COU_ID	평균 나이
1	10	28
2	30	27

186 Page

3-1 데이터 검색(SELECT)

조인 (JOIN)

- 조인의 개념?
  - 두 개 이상의 테이블을 서로 묶어서 하나의 결과 집합으로 만들어 내는 작업

- 데이터베이스의 테이블
  - 여러 개의 테이블로 분리하여 저장
    - 중복과 공간 낭비를 피하고 데이터의 무결성 위함
  - 분리된 테이블들은 서로 관계(Relation)를 가짐
    - 1대 다 관계에서 일어나는 데이터 처리 필요성

3-1 데이터 검색(SELECT)

- 조인

```
SELECT STU_NAME, COU_ID
FROM STUDENT
order by COU_ID
```

```
SELECT COU_ID, COU_NAME
FROM COURSE
```

결과 메시지

	STU_NAME	COU_ID
1	문종헌	10
2	박홍진	10
3	김준형	10
4	박기석	10
5	공지훈	10
6	이봉림	10
7	장희성	10
8	문혜진	20
9	김현우	20
10	박시준	20
11	오한솔	20
12	제용석	20
13	정국철	20
14	안창범	30
15	윤효선	30
16	안창범	30

결과 메시지

	COU_ID	COU_NAME
1	10	모바일
2	20	자바
3	30	윈도우
4	40	웹표준

3 데이터 조작어(DML)

3-1 데이터 검색(SELECT)

- **INNER JOIN**
- 조인 대상이 되는 테이블의 공통적으로 존재하는 칼럼 값이 같은 값을 가지는 조건(=)에 대해서 조인 결과를 얻는 것

```
SELECT <열 목록>
FROM <첫 번째 테이블>
      INNER JOIN <두 번째 테이블>
      ON <조인될 조건>
[WHERE 검색조건]
```

예

```
SELECT *
FROM STUDENT
      INNER JOIN COURSE
      ON STUDENT.COU_ID = COURSE.COU_ID
```

STUDENT 테이블의 COU_ID 칼럼

COURSE 테이블의 COU_ID 칼럼

3-1 데이터 검색(SELECT)

- 학생과 과정 테이블 관계

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	SEX	COU_ID	COU_ID	COU_NAME	TEA_NAME
1	101	문종헌	24	moon@nate.com	M	10	10	모바일	성윤정
2	102	오한솔	22	five@nate.com	M	20	20	자바	김혜경
3	103	제용석	22	again@nate.com	M	20	20	자바	김혜경
4	104	정국철	22	cook@nate.com	M	20	20	자바	김혜경
5	105	박홍진	24	red@nate.com	M	10	10	모바일	성윤정
6	106	김현우	21	NULL	M	20	20	자바	김혜경
7	107	박시준	22	season@nate.c...	M	20	20	자바	김혜경
8	108	김준형	27	brother@nate.c...	M	10	10	모바일	성윤정
9	109	문혜진	22	NULL	F	20	20	자바	김혜경
10	110	박기석	34	flag@nate.com	M	10	10	모바일	성윤정
11	111	윤효선	24	good@nate.com	F	30	30	윈도우	황연주
12	112	안창범	34	window@nate....	M	30	30	윈도우	황연주
13	113	공지훈	28	NULL	M	10	10	모바일	성윤정
14	114	이봉림	29	bbong@nate.c...	M	10	10	모바일	성윤정
15	115	안창범	24	chang@nate.c...	M	30	30	윈도우	황연주
16	116	장희성	34	shine@nate.com	M	10	10	모바일	성윤정

3-1 데이터 검색(SELECT)

INNER Join

- 옛날방식

예

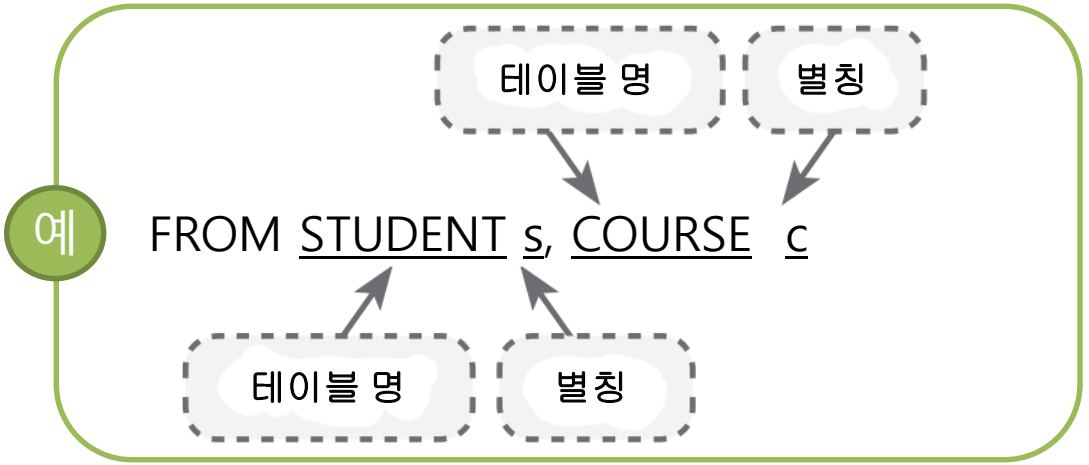
```
SELECT STU_NAME, COU_NAME                          .....①
FROM   STUDENT, COURSE                             .....②
WHERE STUDENT.COU_ID = COURSE.COU_ID                .....③
```

- 필요한 항목만 출력 – 오류.

```
SELECT STU_ID, STU_NAME, COU_ID, COU_NAME
FROM STUDENT
        INNER JOIN COURSE
        ON STUDENT.COU_ID = COURSE.COU_ID
```

3-1 데이터 검색(SELECT)

- 테이블에 별칭 부여하기



3-1 데이터 검색(SELECT)

● 테이블에 별칭 부여하기

FROM 절에서 테이블 명 다음에 공백을 두고 별칭을 부여함.

```
SELECT s.STU_NAME, c.COU_NAME,
c.COU_ID
FROM   STUDENT s
  inner join COURSE c
  ON s.COU_ID = c.COU_ID
WHERE  s.STU_NAME='김준형'
```

FROM 절에서 지정한 별칭을 **SELECT** 절과 **WHERE** 절에서 사용하고 있음.

결과 | 메시지

	STU_NAME	COU_NAME	COU_ID
1	김준형	모바일	10

189 Page

3-1 데이터 검색(SELECT)

- **Join**
  - 모든 과목에 대한 수강생 목록 검색

```
SELECT c.COU_NAME, c.COU_ID, s.STU_NAME
FROM  STUDENT s
  inner join COURSE c
  ON s.COU_ID = c.COU_ID
```

- **수강생이 없는 과목은 검색되지 않는다. 해결책은?**
- **한쪽에 없으면 출력이 되지 않는 것이 INNER JOIN이다.**

3-1 데이터 검색(SELECT)

outer Join(외부조인)

- 내부 조인을 사용하면 수강하지 않는 과목은 출력되지 않는다.
- 조인의 조건에 만족되지 않는 행까지도 포함시키는 것

```
SELECT <열 목록>
FROM <첫 번째 테이블(LEFT 테이블)>
 <LEFT ¦ RIGHT ¦ FULL> OUTER JOIN <두 번째 테이블(RIGHT 테이블)>
      ON <조인될 조건>
[WHERE 검색조건] ;
```

- LEFT OUTER JOIN 은 '왼쪽 테이블의 것은 모두 출력되어야 한다' 고 해석하면 이해 쉬움

```
SELECT c.COU_NAME, c.COU_ID, s.STU_NAME
FROM COURSE c
  LEFT OUTER JOIN STUDENT s
  ON c.COU_ID = s.COU_ID
```

3-1 데이터 검색(SELECT)

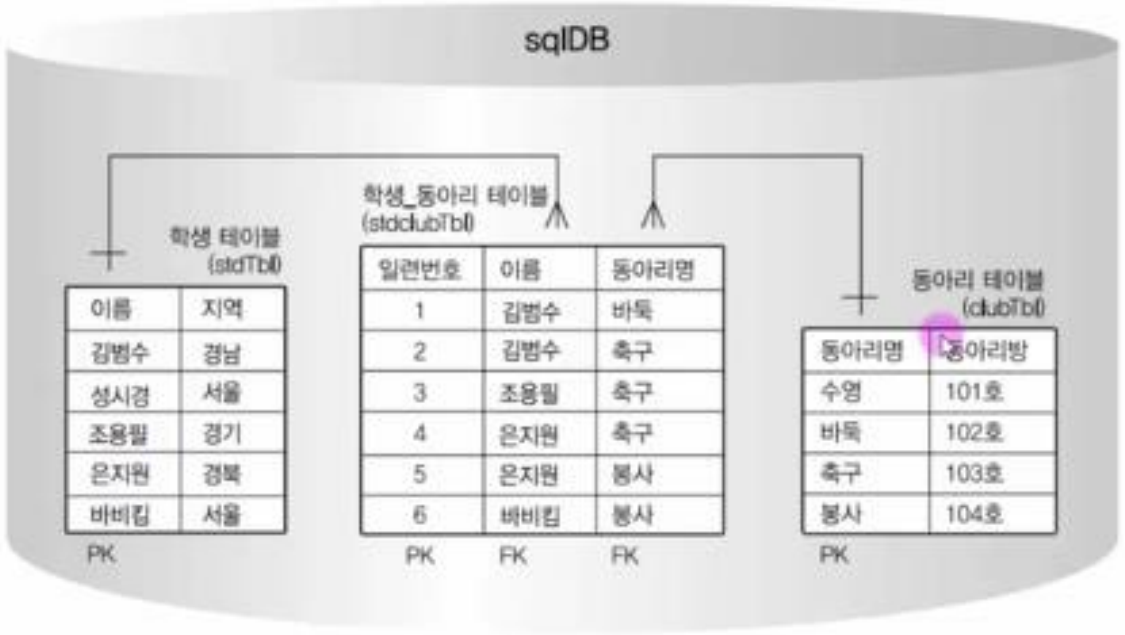
- **outer Join(외부조인)**

  - INNER JOIN과 반대로 수강생이 없는 과목 출력

```
SELECT c.COU_NAME, c.COU_ID, s.STU_NAME
FROM COURSE c
  LEFT OUTER JOIN STUDENT s
  ON c.COU_ID = s.COU_ID
WHERE s.STU_NAME is null
```

3-1 데이터 검색(SELECT)

- 3개 테이블의 join



[그림 7-31] 세 개의 테이블 샘플

[그림 7-31]의 구조는 테이블의 복잡성을 없애려고 학생의 이름 및 동아리명을 Primary Key로 설정했다.

@ sqldb_club.sql

3-1 데이터 검색(SELECT)

3개 테이블의 join

- 학생을 기준으로 학생이름/지역/가입한 동아리/동아리방을 출력

```
SELECT S.stdName, S.addr, C.clubname, C.roomNo
FROM stdTbl S
INNER JOIN stdclubTbl SC
   ON S.stdName = SC.stdName
INNER JOIN clubTbl C
   ON SC.clubName = C.clubName
ORDER BY S.stdName;
```

- 동아리를 기준으로 출력

```
SELECT C.clubname, C.roomNo, S.stdName, S.addr
FROM stdTbl S
INNER JOIN stdclubTbl SC
   ON S.stdName = SC.stdName
INNER JOIN clubTbl C
   ON SC.clubName = C.clubName
ORDER BY C.clubname;
```

3-1 데이터 검색(SELECT)

3개 테이블의 join

- 학생 기준으로 동아리를 가입하지 않은 학생도 출력

```
SELECT S.stdName, S.addr, C.clubname, C.roomNo
FROM stdTbl S
LEFT OUTER JOIN stdclubTbl SC
   ON S.stdName = SC.stdName
LEFT OUTER JOIN clubTbl C
   ON SC.clubName = C.clubName
```

- (연습) 동아리 기준으로 아무도 가입하지 않은 동아리도 출력

3-1 데이터 검색(SELECT)

3개 테이블의 join

- 동아리에 가입하지 않은 학생도 출력되고 학생이 아무도 없는 동아리도 출력(full OUTER JOIN)

```
SELECT S.stdName, S.addr, C.clubname, C.roomNo
FROM stdTbl S
FULL OUTER JOIN stdclubTbl SC
   ON S.stdName = SC.stdName
FULL OUTER JOIN clubTbl C
   ON SC.clubName = C.clubName
```

3-1 데이터 검색(SELECT)

CROSS JOIN(상호 조인)

- 한쪽 테이블의 모든 행들과 다른 쪽 테이블의 모든 행 조인
  - CROSS JOIN의 결과 개수는 두 테이블 개수를 곱한 개수
  - 카티션곱(Cartesian Product) 이라고도 부름
  - 상호조인 도식화 예시

```
SELECT *
   FROM buyTbl
     CROSS JOIN userTbl ;
```

회원 테이블(us...)

아이디	이름	생년	지역	국번	전화번호	키	가입일
LSG	이승기	1987	서울	011	1111111	182	2008.8.8
KBS	김범수	1979	경남	011	2222222	173	2012.4.4
KKH	김경호	1971	전남	019	3333333	177	2007.7.7
JYP	조용필	1950	경기	011	4444444	166	2009.4.4
SSK	성시경	1979	서울			186	2013.12.12
LJB	임재범	1963	서울	016	6666666	182	2009.9.9
YJS	윤종신	1969	경남			170	2005.5.5
EJW	은지원	1978	경북	011	8888888	174	2014.3.3
JKW	조관우	1965	경기	018	9999999	172	2010.10.10
BBK	바비킴	1973	서울	010	0000000	176	2013.5.5

PK

순번	아이디	물품명	분류	단가	수량
1	KBS	운동화		30	2
2	KBS	노트북	전자	1000	1
3	JYP	모니터	전자	200	1
4	BBK	모니터	전자	200	5
5	KBS	청바지	의류	50	3
6	BBK	메모리	전자	80	10
7	SSK	책	서적	15	5
8	EJW	책	서적	15	2
9	EJW	청바지	의류	50	1
10	BBK	운동화		30	2
11	EJW	책	서적	15	1
12	BBK	운동화		30	2

PK FK

3-1 데이터 검색(SELECT)

SELF JOIN(자체 조인)

- 자기 자신과 자기 자신이 조인한다는 의미

```
SELECT A.first_name AS “부하직원”, B.first_name AS “직속상관”, B.phone_number AS “직속상관연락처”
FROM employees A
   INNER JOIN employees B
      ON A.manager_id = B.employee_id
WHERE A. first_name = ‘Lisa’;
```

3-1 데이터 검색(SELECT)

● UNION / UNION ALL

- 두 쿼리의 결과를 행으로 합치는 것

SELECT stdName, addr FROM stdTbl

stdName	addr
김범수	경남
성시경	서울
조용필	경기
은지원	경북
바비킴	서울

SELECT clubName, roomNo FROM clubTbl

clubName	roomNo
수영	101호
바둑	102호
축구	103호
봉사	104호

UNION ALL

stdName	addr
김범수	경남
성시경	서울
조용필	경기
은지원	경북
바비킴	서울
수영	101호
바둑	102호
축구	103호
봉사	104호

```
SELECT 문장1
   UNION [ALL]
SELECT 문장2
```

3-1 데이터 검색(SELECT)

- **UNION / UNION ALL**

```
SELECT stdName, addr FROM stdTbl
uniON all
SELECT clubName, roomNo FROM clubTbl;
```

- union 중복된 열 하나만 출력 / union all 중복 상관 없이 다 출력
- SELECT 문장 1, 과 2의 결과 열의 개수가 같아야 한다. 호환되는 데이터 형식도 같아야 한다.
- 열의 이름은 문장1의 것을 따른다.

3-1 데이터 검색(SELECT)

- Concatenate operator

```
SELECT last_name || ' is a ' || job_id AS "Employee Details"
FROM employees;
```

- Escape Key (₩)

```
SELECT employee_id, last_name, job_id
FROM employees
WHERE job_id LIKE '%SA₩_%' ESCAPE '₩';
```

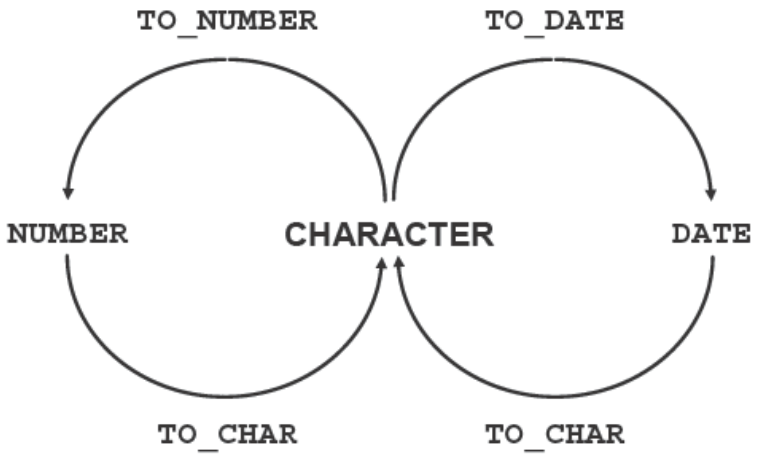
- Arithmetic Expressions
  - +, -, *, /

```
SELECT last_name, salary, salary+300
FROM employees;
SELECT last_name, salary, 12*(salary+100)
FROM employees;
```

3-1 데이터 검색(SELECT)

- Conversion Functions

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2



-

3-1 데이터 검색(SELECT)

- Conversion Functions
  - TO_CHAR(*date, 'format_model'*)
    - 날짜 표현 변경
    - single quotation (') 를 사용해야 한다.
    - 대소문자 구분
    - In case-sensitive

```
SELECT employee_id,
TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE last_name='Higgins';
```

3-1 데이터 검색(SELECT)

- Conversion Functions
  - TO_CHAR(***date***, *'format_model'*)
    - 날짜 표현 변경
    - single quotation (') 를 사용
    - 대소문자 구분
    - In case-sensitive

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for month
MONTH	Full name of the month
MON	Three-letter abbreviation of the month
DY	Three-letter abbreviation of the day of the week
DAY	Full name of the day of the week
DD	Numeric day of the month

```
SELECT employee_id,
TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE last_name='Higgins';
```

3-1 데이터 검색(SELECT)

Conversion Functions

- TO_CHAR(*number, 'format_model'*)
  - 숫자 표현 변경

Element	Result
9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a comma as thousands indicator

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM employees
WHERE last_name='Ernst';
```

3-1 데이터 검색(SELECT)

- 우선순위

Operator	Meaning
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

- 가로(parentheses) 사용가능

3-2 데이터 삽입(INSERT)

- **INSERT문의 기본 형식**

```
INSERT INTO table_name (column_name, ...)
VALUES(column_value, ...) ;
```

```
1:  INSERT INTO COURSE
2:  (COU_ID, COU_NAME, TEA_NAME)


3:  VALUES(50, '보안', '주상면')
4:
5:  SELECT * FROM COURSE
```

- 데이터 삽입

결과 | 메시지

	COU_ID	COU_NAME	TEA_NAME
1	10	모바일	성윤정
2	20	자바	김혜경
3	30	윈도우	황연주
4	40	웹표준	전혜영
5	50	보안	주상면

194 Page

3-2 데이터 삽입(INSERT)

INSERT

- 테이블 이름 다음에 나오는 열 생략 가능
  - 생략할 경우에 VALUE 다음에 나오는 값들의 순서 및 개수가 테이블이 정의된 열 순서 및 개수와 동일해야 함

```
1:  INSERT INTO COURSE
2:  VALUES(50, '보안', '주상면')
```

  - 두 값만 넣고 싶다면 열을 넣어줘야 한다.

```
1:   INSERT INTO COURSE(COU_ID, COU_NAME)
2:   VALUES(50, '보안')
```

  - 열 순서를 바꾸고 싶다면 열을 넣어줘야 한다.

```
1:   INSERT INTO COURSE(COU_NAME, COU_ID)
2:   VALUES('보안', 50)
```

3-2 데이터 삽입(INSERT)

INSERT

- 대량의 샘플 데이터 생성
  - INSERT INTO... SELECT 구문 사용

```
형식:
INSERT INTO 테이블이름 (열이름1, 열이름2, …)
   SELECT문  ;
```

  - 다른 테이블의 데이터를 가져와 대량으로 입력하는 효과
  - SELECT 문의 열의 개수 = INSERT 할 테이블의 열의 개수

```
CREATE TABLE test(id int, name varchar(50),lname varchar(50));


INSERT INTO TEST1
SELECT employee_id, first_name, last_name
FROM employees
```

3-2 데이터 삽입(INSERT)

- **INSERT**
  - 테이블을 복사하는 CREATE TABLE … SELECT
    - 테이블 정의까지 생략하고 싶을 때 사용
    - 테이블을 복사해서 사용할 경우 주로 사용
    - CREATE TABLE 새로운테이블 (SELECT 복사할열 FROM 기존테이블)
    - 지정된 일부 열만 테이블로 복사하는 것도 가능

```
CREATE TABLE buyTbl2(SELECT * FROM buyTbl);
```

- PK나 FK 같은 제약 조건은 복사되지 않음

3-3 데이터 갱신(UPDATE)

● UPDATE문의 기본 형식

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

- WHERE절 생략 가능하나 테이블의 전체 행의 내용 변경

  UPDATE buyTbl set price = price * 1.5;

● UPDATE문의 형식

- 과정 번호가 10번인 학생의 과정 번호를 그0번으로 수정(예제)

```
1:    UPDATE STUDENT
2:    SET COU_ID=30
3:    WHERE COU_ID=10
4:
5:    SELECT * FROM STUDENT
```

→ 과정 번호(COU_ID)가 10인 자료를 과정 번호 30으로 수정한다.

3-3 데이터 갱신(UPDATE)

UPDATE문의 형식

- 데이터 수정 결과

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	SEX	COU_ID
1	101	문종헌	24	moon@nate.com	M	30
2	102	오한솔	22	five@nate.com	M	20
3	103	제용석	22	again@nate.com	M	20
4	104	정국철	22	cook@nate.com	M	20
5	105	박홍진	24	red@nate.com	M	30
6	106	김현우	21	NULL	M	20
7	107	박시준	22	season@nate.c...	M	20
8	108	김준형	27	brother@nate.c...	M	30
9	109	문혜진	22	NULL	F	20
10	110	박기석	34	flag@nate.com	M	30
11	111	윤효선	24	good@nate.com	F	30
12	112	안창범	34	window@nate....	M	30
13	113	공지훈	28	NULL	M	30
14	114	이봉림	29	bbong@nate.c...	M	30
15	115	안창범	24	chang@nate.c...	M	30
16	116	장희성	34	shine@nate.com	M	30

195 Page

3-4 데이터 삭제(DELETE)

DELETE문의 기본 형식

- 30번 과정 소속 학생을 삭제(예제)

```
1:  DELETE STUDENT
2:  WHERE COU_ID=30
3:
4:  SELECT * FROM STUDENT
```

- 데이터 삭제

결과 | 메시지

	STU_ID	STU_NAME	AGE	STU_EMAIL	SEX	COU_ID
1	102	오한솔	22	five@nate.com	M	20
2	103	제용석	22	again@nate.com	M	20
3	104	정국철	22	cook@nate.com	M	20
4	106	김현우	21	NULL	M	20
5	107	박시준	22	season@nate.com	M	20
6	109	문혜진	22	NULL	F	20

196 Page

3-4 데이터 삭제(DELETE)

- **DELETE문의**
  - WHERE절 생략 가능하나 테이블의 전체 행의 내용 변경
    - Truncate table 사용 (물리적인 삭제)
    - Drop table (물리적인 삭제)
  - 테이블을 삭제하는 경우의 속도 비교
    - DML문인 DELETE는 트랜잭션 로그 기록 작업 때문에 삭제 느림
    - DDL문인 DROP table과 TRUNCATE table문은 트랜잭션 없어 빠름

Reference

데이터베이스 배움터(오라클 기반) / 생능출판사/ 홍의경

SqI 첫걸음 / 아사이 아츠시 / 한빛미디어