MySQL 실습하기

MySQL script 파일 실행하기

- **mysql 실행**
  - window: https://withcoding.com/35
  - ubuntu: service mysql start
  - mac os: mysql.server start
- **Mysql 실행 후 (interactive mode)**
  - mysql -u root -p
  - mysql> source companyDB.sql
  - Mysql 실행 경로:
    - ㄴ codes
    - ㄴ companyDB.sql
    - ㄴ data
- **Mysql 실행 하지 않고 cmd line 에서 직접 실행**
  - mysql -u root -p dbsecure < companyDB.sql

질의 결과 output file 로 내보내기

➢ MySQL data directory 확인

- 상대 경로 사용 시에 아래 디렉토리에 저장

```
mysql> show variables like 'datadir';
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| datadir       | /usr/local/var/mysql/ |
+---------------+-----------------------+
1 row in set (0.00 sec)
```

➢ output file 로 내보내기

- select * from employees into outfile ‘ ...../employees_list.txt ’ ;
- select * from employees into outfile ‘ ...../employees_list.csv ’
- fields terminated by ‘ , ’
- enclosed by ‘ “ ‘
- lines terminated by ‘ \n ’ ;

질의 결과 output file 로 내보내기

➢ MySQL 의 환경 변수 확인 및 변경

- show variables like “secure_file_priv”;
- 파일 입출력 경로 변경
  - NULL: 어떠한 경로도 지정되어 있지 않음
  - 특정 디렉터리 경로: 해당 위치에서는 파일을 읽고 쓸 수 있음
  - 아무것도 표시되어 있지 않은 경우: 어떠한 위치에서도 파일을 읽고 쓸 수 있음
- window:
  - C:\ProgramData\MySQL\MySQL Server 5.x\my.ini 에서
  - # Secure File Priv.
  - secure-file-priv=""
- others: `mysqld --verbose --help | grep -A 1 'Default options'`(위치 찾기)
  - vi /etc/mysql/my.cnf 에서
  - [mysqld] secure-file-priv=””
  - mysql.server restart

ALTER TABLE

➢ **relation schema 를 변경**

- 이미 존재하는 테이블의 구조나 형식 등을 바꾸기 위해 사용
- 컬럼의 구조나 형식을 변경하는 데에 이용

1. 테이블 이름 변경

- ALTER TABLE [테이블명] RENAME [바꿀이름] ¦ RENAME TABLE [테이블명] TO [바꿀이름]

2. 칼럼 변경

- 컬럼 추가: ALTER TABLE [테이블명] ADD COLUMN [컬럼이름] [컬럼타입] (AFTER [컬럼이름] /FIRST)
- 컬럼 삭제: ALTER TABLE [테이블명] DROP [컬럼이름]
- 컬럼 변경: ALTER TABLE [테이블명] MODIFY [컬럼이름] [새컬럼타입] ¦ ALTER TABLE [테이블명] CHANGE [컬럼이름] [새컬럼이름] [새컬럼타입]

3. 인덱스 변경

- 인덱스 추가: ALTER TABLE [테이블명] ADD INDEX([컬럼이름])
- 인덱스 삭제: ALTER TABLE [테이블명] DROP INDEX [컬럼이름] ¦ DROP INDEX [인덱스이름] ON [테이블명]

4. 제약조건 변경

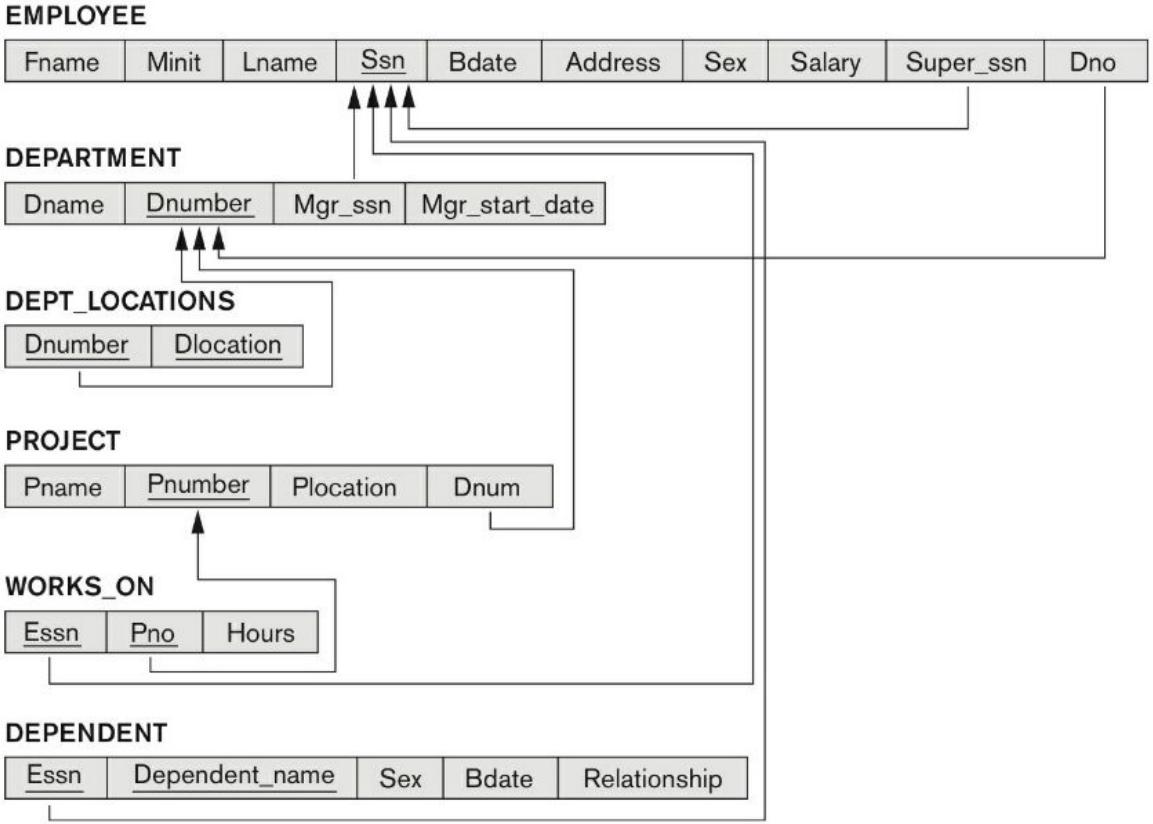
- 기본키 추가: ALTER TABLE [테이블명] ADD PRIMARY KEY([컬럼이름]) 컬럼이름
- 기본키 삭제: ALTER TABLE [테이블명] DROP PRIMARY KEY
- 외래키 추가: ALTER TABLE [테이블명] ADD FOREIGN KEY([컬럼이름]) REFERENCES [테이블이름]([컬럼이름])
- 외래키 삭제: ALTER TABLE [테이블명] DROP FOREIGN KEY [제약조건 이름]
- 제약조건 이름 확인: select * from information_schema.table_constraints where constraint_type= ' FOREIGN KEY '

데이터 추가, 수정, 삭제

1. **데이터 추가**
   - INSERT INTO [테이블명] ([col1], [col2], … , [coln]) VALUES ([val11], [val12], … , [val1n]), ([val21], [val22], … , [val2n]);
   - **<파일로부터 읽기>**
   - LOAD DATA INFILE [파일경로/파일이름] INTO TABLE [테이블이름]
   - LOAD DATA LOCAL INFILE [파일이름.csv] INTO TABLE [테이블 이름] FIELDS TERMINATED BY ‘ , ’ OPTIONALLY ENCLOSED BY “ ” LINES TERMINATED BY ‘ \n ’ IGNORE 1 LINES ([col1], [col2], … , [coln]);
   - **<NULL의 표현>**: \N
2. **데이터 수정**
   - UPDATE [테이블명] SET [컬럼이름] = [수정값] (, [컬럼이름2] = [수정값2], …) (WHERE [조건]);
3. **데이터 삭제**
   - DELETE FROM [테이블명] (WHERE [조건]);

SQL 검색 질의

COMPANY relational database schema



SQL 의 기본 검색 질의

➢ SELECT statement

- 데이터베이스에서 정보를 검색하는 기본문
- SQL 모델과 관계 모델의 중요한 차이점
  - SQL의 테이블(릴레이션)은 모든 애트리뷰트 값이 동일한 투플을 하나 이상 가지는 것을 허용함
  - SQL 테이블은 투플들의 집합이 아니고 투플들의 다중집합(multiset, **bag**)임
- 따라서, 키 제약 조건을 선언하거나 SELECT 문에 DISTINCT 옵션을 함께 사용하면 SQL이 집합의 성질을 갖도록 할 수 있음

➢ 기본 SQL 질의의 SELECT-FROM-WHERE 구조

- SELECT <애트리뷰트 리스트>
- FROM <테이블 리스트>
- WHERE <조건>
  - <애트리뷰트 리스트>: 질의에서 검색되는 값들에 대한 속성 이름들의 리스트
  - <테이블 리스트>: 질의 처리를 위해 필요한(질의의 대상이 되는) 릴레이션들의 리스트
  - <조건>: 질의를 통해 검색되는 투플들을 명시하는 조건(부울)식
- 비교 연산자
  - SQL: =, <, <=, >, >=, <> | C나 C++: =, <, <=, >, >=, !=

SQL 의 기본 검색 질의

➢ 기본 SQL 질의의 SELECT-FROM-WHERE 구조

- SELECT <애트리뷰트 리스트>
- FROM <테이블 리스트>
- WHERE <조건>
- 비교 연산자: [SQL] =, <, <=, >, >=, <> | [C나 C++] =, <, <=, >, >=, !=
- 프로젝션 애트리뷰트 (projection attribute)
  - SELECT 절에서 검색할 애트리뷰트를 명시
- 선택 조건 (selection condition)
  - WHERE 절은 검색될 투플들이 만족해야 하는 불리언 조건을 명시
- 조인 조건 (join condition)
  - 서로 다른 릴레이션의 투플들의 애트리뷰트들이 같은 두 투플을 결합하는 것을 표시

SQL 질의 예제

[질의 0] 이름이 **'john B. Smith'**인 사원(들)의 생년월일(**Bdate**)과 주소(**Adress**)를 검색하라.
질의는 **FROM** 절에 표시된 **EMPLOYEE** 릴레이션만 검색한다

[질의 1] **'Research'** 부서에서 근무하는 모든 사원의 이름(**Fname, Lname**)과 주소(**Address**)를 검색하라.

[질의 2] **'Stafford'**에 위치한 모든 프로젝트 들에 대해서 프로젝트 번호(**Pnumber**), 담당부서 번호(**Dnum**), 부서 관리자의 성(**Lname**), 주소(**Address**), 생년월일(**Bdate**)을 검색하라

SQL 질의 예제

[질의 0] 이름이 **'john B. Smith'**인 사원(들)의 생년월일(**Bdate**)과 주소(**Adress**)를 검색하라.

질의는 **FROM** 절에 표시된 **EMPLOYEE** 릴레이션만 검색한다

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname= 'John' AND Minit='B' AND Lname='Smith'
```

[질의 1] **'Research'** 부서에서 근무하는 모든 사원의 이름(**Fname, Lname**)과 주소(**Address**)를 검색하라.

```
SELECT Fname, Lname, Address
FROM EMPLOYEE, DEPARTMENT
WHERE Dname='Research' AND Dnumber = Dno;
```

조인 조건 (Dnumber = Dno)

SQL 질의 예제

[질의 2] 'Stafford'에 위치한 모든 프로젝트 들에 대해서 프로젝트 번호(Pnumber), 담당부서 번호(Dnum), 부서 관리자의 성(Lname), 주소(Address), 생년월일(Bdate)을 검색하라

SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber **AND** Mgr_ssn=Ssn **AND** Plocation='Stafford'

두 개의 조인 조건 (Dnum = Dnumber, Mgr_ssn=Ssn)

SQL 질의 예제: 결과

Bdate	Address
1965-01-09	731Fondren, Houston, TX

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

모호한 애트리뷰트

➢ 모호한 애트리뷰트 이름

- SQL 에서는 동일한 이름을 갖는 애트리뷰트가 여러 테이블에 사용될 수 있음
  - 이 경우에, 다중 테이블 질의를 통해 동일한 이름을 갖는 두 개 이상의 애트리뷰트를 참조하려면 모호함을 피하기 위해 릴레이션 이름과 함께 애트리뷰트 이름을 사용함 (**[릴레이션이름].[애트리뷰트이름]**)
  - 예: 만약 EMPLOYEE 의 Dno → Dnumber, Lname → Name 으로 가정, DEPARTMENT의 Dname →Name 으로 가정

[질의 1A]

```
SELECT Fname, EMPLOYEE.Name, Address
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Name='Research' AND DEPARTMENT.Dnumber = EMPLOYEE.Dnumber;
```

[질의 1']

```
SELECT EMPLOYEE.Fname, EMPLOYEE.Lname, EMPLOYEE.Address
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Dname='Research' AND DEPARTMENT.Dnumber = EMPLOYEE.Dno;
```

SQL 재명명

➢ SQL 재명명 (AS)

- 릴레이션의 별명
  - 같은 릴레이션을 두 번 참조하는 질의에서도 애트리뷰트 이름의 모호함이 나타남
  - 한 릴레이션(EMPLOYEE)에 대해서 **별명** 또는 **투플 변수**라고 부르는 또 다른 릴레이션의 이름들(E 와 S)을 선언하여 사용 가능

[질의 8] 각 사원에 대해 사원의 이름(Fname)과 성(Lname), 직속 상사의 이름(Fname)과 성(Lname)을 검색하라

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn = S.Ssn;
```

- 위의 질의 예제에서는 조인 조건 (E.Super_ssn = S.ssn)을 만족하는 투플들을 찾아냄으로써 한 릴레이션을 **자기 자신과 조인**
- 애트리뷰트 재명명
  - SQL 별명을 사용하여 질의 내의 릴레이션의 애트리뷰트 이름들을 재명명 가능
  - EMPLOYEE AS E(Fn, Mi, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)
- SQL 질의에서 **WHERE** 내에 있는 릴레이션들에 대해 투플 변수를 명시하기 위해 별명 또는 재명명 방식 사용 가능 → **질의 이해에 도움**

WHERE 절의 생략과 '*'의 사용

➢ WHERE 절의 생략

- 투플 선택에 대한 조건이 없다는 것을 의미
- FROM 절에서 명시한 릴레이션의 모든 투플이 질의 결과로 검색됨
- 릴레이션들의 크로스 프로덕트(cross product)
  - FROM 절에 두 개 이상의 릴레이션이 명시되고, WHERE 절이 없다면 이 릴레이션들의 모든 가능한 투플의 조합이 선택됨

[질의 9와 10] EMPLOYEE의 모든 Ssn을 선택하고 (Q9), EMPLOYEE 의 Ssn 과 DEPARTMENT의 Dname의 모든 조합을 선택하라 (Q10)

(Q9)
```
SELECT Ssn
FROM EMPLOYEE
```

(Q10)
```
SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT
```

➢ '*'의 사용

- 선택된 투플들의 모든 애트리뷰트 값을 검색하려면 SELECT 절에 모든 애트리뷰트의 이름을 명시적으로 열거하는 대신에 별표(*)를 명시
- 별표(*) 앞에 릴레이션 이름 또는 별명이 올 수 있음
  - SELECT * / FROM EMPLOYEE / WHERE Dno=5;

SQL 집합으로의 테이블

➢ 중복된 투플 제거

- SQL은 질의 결과에서 자동적으로 중복된 튜플을 제거하지 않음
  - 중복을 없애는 연산은 비용이 큼 (정렬 + 제거)
  - 사용자가 질의 결과에 중복된 투플들이 나타나는 것을 원하는 경우가 있음
  - 집단 함수(평균, 합 등)가 투플들에 적용될 때 대부분의 경우에 중복 삭제가 필요하지 않음
- DISTINCT 사용
  - 키를 가진 SQL 테이블은 키 값이 각 투플마다 구별되야 하므로 집합으로 표현되도록 제한
  - SQL 질의 결과에서 중복된 투플을 삭제하려면 SELECT 절에 키워드 DISTINCT 사용
- SELECT ALL 사용
  - 그냥 SELECT 와 동일
  - 중복 투플을 제거하지 않음

[질의 11] 모든 사원의 급여(Salary)를 검색하고(Q11), 구별되는 급여를 모두 검색하라(Q11A).

(Q11)
```
SELECT All Salary
FROM EMPLOYEE
```

(Q11A)
```
SELECT DISTINCT Salary
FROM EMPLOYEE
```

SQL 에서의 관계 대수 연산

➢ 집합 관계 대수 연산

- 합집합(UNION), 차집합(EXCEPT), 교집합(INTERSECT) 연산들을 포함
- 집합 관계 대수 연산들도 중복 투플 제거 옵션을 추가할 수 있음
- 집합 연산들은 타입 호환성을 갖는 릴레이션들에만 적용 가능함
  - 동일한 애트리뷰트를 가지며 이 애트리뷰트들이 양쪽 릴레이션에서 같은 순서로 나타남

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  (SELECT    DISTINCT Pnumber
       FROM      PROJECT, DEPARTMENT, EMPLOYEE
       WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                 AND Lname='Smith' )
       UNION
      ( SELECT   DISTINCT Pnumber
       FROM      PROJECT, WORKS_ON, EMPLOYEE
       WHERE     Pnumber=Pno AND Essn=Ssn
                 AND Lname='Smith' );
```

부분 문자열 패턴 비교

➢ LIKE 비교 연산자

- 문자열의 패턴 비교에 사용
- 부분 문자열(substring)의 표현
  - '%' : 0 보다 큰 임의의 개수의 문자
  - '_' : 임의의 한 개의 문자를 의미
- 탈출문자(escape character)
  - 문자열 내의 상수로 밑줄이나 %가 필요한 경우 그 문자열 뒤에 ESCAPE를 써서 표시한 **탈출문자**(예: \)를 앞에 붙임
  - **작은따옴표**('')는 문자열의 시작과 끝을 나타내므로 이를 문자열 내에 포함시키려면 **두 개의 연속된** 작은따옴표로 표현

[질의 12] 주소가 'Houston, Texas'인 모든 사원을 검색하라

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Address LIKE '%Houston,Tx%';
```

[질의 13] 1970년대에 태어난 모든 사원을 검색하라

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Bdate LIKE '_ _ 7 _ _ _ _ _';
```

산술 연산자 및 기타 연산자

- **질의 내에서 산술식 허용**
  - 표준 산술 연산자: 더하기(+), 빼기(-), 곱하기(*), 나누기(/)
    - 수치 값 혹은 숫치 애트리뷰트에 적용할 수 있음
- **기타 연산자**
  - 집합 연산자 (concatenate operator)
    - 문자열 데이터 타입에 대해서 집합연산자 ‘ ‘ 를 사용하여 문자열을 합칠 수 있음
  - 기간 증가 연산자
    - 날짜, 시간, 타임스탬프 또는 기간 데이터 타입에 대해 날짜, 시간 또는 타임스탬프를 주어진 기간만큼 증가(+)시키거나 감소(-)시킬 수 있음
  - 비교 연산자(BETWEEN)

질의 결과의 정렬

➢ ORDER BY 절 사용

- 하나 이상의 애트리뷰트를 기준으로 질의 결과에 들어 있는 투플들을 정렬하는 것이 가능
- 정렬 순서
  - 디폴트 정렬은 오름차순 (ASC)
  - 내림 차순으로 보고자 한다면 키워드 (DESC)를 명시
  - `ORDER BY D.Dname DESC E.lname ASC E.Fname ASC`

[질의 15] 사원 및 각 사원이 근무하는 프로젝트들의 리스트를 검색하는데, 부서 이름 순서대로, 그리고 각 부서 내에서는 사원의 성과 이름의 알파벳 순서대로 구하라.

```
SELECT D.Dname, E.Lname, E.Fname, P.Pname
FROM DEPARTMENT AS D, EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
WHERE D.Dname=E.Dno AND E.Ssn=W.Essn, AND W.Pno=P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;
```

기본 SQL 검색 질의에 대한 논의와 요약

➢ SQL 에서의 검색 질의

- SQL 에서 간단한 검색 질의 하나는 4개의 절까지 가질 수 있으며 첫 두 항목(**SELECT, FROM** 절)은 필수적으로 가져야 함

```
SELECT      <애트리뷰트 리스트>
FROM        <테이블 리스트>
[WHERE      <조건>]
[ORDER BY   <애트리뷰트 리스트>]
```

  - **SELECT** 절: 검색하려고 하는 애트리뷰트들의 리스트 나열
  - **FROM** 절: 질의에 필요한 모든 릴레이션(테이블)을 명시
  - **WHERE** 절: 필요하다면 조인 조건을 포함하여 **FROM** 절에 명시된 릴레이션들로부터 투플들을 선택하기 위한 조건을 명시
  - ORDER_BY절: 질의 결과를 출력하는 순서를 명시
- 추가로 사용되는 두 개의 절: **GROUP_BY, HAVING**
  - 집단 함수에 추가적인 기능을 부여

SQL 언어 고급기능

질의 논리 연산자

➢ 세 값을 갖는 논리

- SQL 에서 NULL 값의 세 가지 의미
  - 알려지지 않은 값 | 이용할 수 없거나 보류해 둔 값 | 적용할 수 없는 애트리뷰트
  - 어떤 경우든지 가능하기 때문에 SQL은 NULL 값의 의미를 구분하지 않음
- 따라서, NULL 값과 비교 연산을 수행하게 될 경우에 TRUE/FALSE 가 아닌 다른 구분자가 필요
  - UNKNOWN 을 추가해 총 세가지 논리 값을 사용함
  - TRUE | FALSE | UNKNOWN
- 이와 관련된 비교 조건의 결과 값은 뒤에 [표 7.1] 과 같음

➢ SQL 질의에서 NULL 값의 검사

- NULL과 비교하기 위해서는 =, <, > 를 사용하지 않고 IS 나 IS NOT 을 비교 연산자로 사용함

[질의 18] 상사가 없는 모든 사원의 이름을 검색하라.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Super_ssn IS NULL;
```

세 값을 갖는 논리에서 논리 연산자들

Table 7.1 Logical Connectives in Three-Valued Logic

(a)	**AND**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	FALSE	UNKNOWN
	FALSE	FALSE	FALSE	FALSE
	UNKNOWN	UNKNOWN	FALSE	UNKNOWN
(b)	**OR**	TRUE	FALSE	UNKNOWN
	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	UNKNOWN
	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
(c)	**NOT**			
	TRUE	FALSE		
	FALSE	TRUE		
	UNKNOWN	UNKNOWN		

중첩 질의의 사용

➢ 중첩질의

- 다른 질의의 WHERE 절 내에서 완전한 select-from-where 형태를 가짐
- 내부 질의의 결과인 값들의 집합 V 와 외부 질의의 WHERE 절에서의 값 v 를 비교하는 연산자들은 다음과 같음
  - v IN V: v 값이 V 집합 중에 존재하는지를 비교
  - v ANY(SOME) V: v 값이 V 내의 어떤 값들과 같으면 참을 반환 (IN 과 비슷)
  - 하지만 ALL, ANY 와 같은 연산자들은 >, >=, <, <=, <> 등과 같은 연산자들과 결합 가능
  - 예) v > ALL V : 모든 V들의 값들보다 큰 경우에 참을 반환

[질의 16] 부양 가족의 성(Fname) 과 성별 (Sex) 이 같은 사원들의 이름 (Fname, Lname)을 검색하라.

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E
WHERE E.Ssn IN  ( SELECT D.Essn FROM DEPENDENT AS D
                  WHERE E.Fname = D.Dependent_name AND E.Sex = D.Sex);
```

중첩 질의의 사용

➢ 상관중첩질의

- 중첩 질의의 WHERE 절에 있는 조건에서 외부 질의에 선언된 릴레이션의 일부 애트리뷰트를 참조하는 경우 두 질의가 서로 **상관(correlated)**된다고 말함

[질의 16A] 부양 가족의 성(Fname) 과 성별 (Sex) 이 같은 사원들의 이름 (Fname, Lname)을 검색하라.

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE AS E, DEPENDENT AS D
WHERE E.Ssn = D.Essn AND E.Sex=D.Sex AND E.Fname = D.Dependent_name;
```

➢ EXIST 와 UNIQUE

- 질의(Q)를 입력으로 하여 TRUE 또는 FALSE 를 반환하는 함수
- **EXIST(Q):** 질의 결과가 한 투플도 갖지 않을 경우 FALSE / 질의 결과가 한 투플이라도 있을 경우 TRUE
- **UNIQUE(Q)**: 질의 결과에 중복된 투플이 없으면 TRUE/ 그렇지 않으면 (다중 집합) FALSE

EXIST 사용 예

[질의 6] 부양가족이 없는 종업원들의 이름을 검색하라.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT * FROM DEPENDENT WHERE Ssn = Essn);
```

[질의 7] 부양가족을 적어도 한 명 이상 가진 관리자의 이름을 검색하라.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE EXISTS (SELECT * FROM DEPENDENT WHERE Ssn = Essn)
      AND
      EXISTS (SELECT * FROM DEPARTMENT WHERE Ssn=Mgr_ssn);
```

중첩 질의의 사용

➢ 명시적 집학과 애트리뷰트 재명명

- 명시적 집합은 괄호로 묶어서 바로 나타낼 수 있음
  - 예) Pno IN (1,2,3)
- 질의 결과에 나타나는 임의의 애트리뷰트를 정량자 AS 다음에 원하는 새 이름을 사용하여 재명명 가능
  - 변경시, 질의 결과 컬럼 이름이 바뀌어 보임

[질의 17] 프로젝트 번호 1, 2, 3에서 일하는 모든 사원의 사회보장번호를 검색하라.

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE Pno IN (1,2,3);
```

[질의 8A] 각 사원에 대해 사원의 이름(Fname)과 직속 상사의 이름(Fname)을 검색하라.

```
SELECT E.Fname AS Employee_name, S.Fname AS Supervisor_name
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn = S.Ssn;
```

SQL 집단 함수

➢ 집단 함수 (Aggregate Function)

- 집단 함수: 여러 투플의 정보를 요약하여 하나의 투플로 요약하는 데에 사용함
- 그룹화 (grouping): 요약하기 전에 투플들을 부분 그룹으로 나누기 위해 사용
- 집단 함수의 종류: COUNT, SUM, MAX, MIN, AVG
  - COUNT: 질의의 결과로 검색된 투플이나 값들의 개수를 구함
  - SUM, MAX, MIN AVG: 숫자들의 집합이나 다중집합 (중복 허용) 에 적용되어 합, 최댓값, 최솟값, 평균값을 구함

[질의 19] 모든 사원의 급여의 합, 최고 급여, 최저 급여, 평균 급여를 구하라.

```
SELECT SUM(Salary) AS Total_Sal, MAX(Salary) AS Highest_Sal, MIN(Salary) AS Lowest_Sal,
       AVG(Salary) AS Average_Sal
FROM EMPLOYEE;
```

[질의 20] 'Research' 부서에 근무하는 모든 사원의 급여의 합, 최고 급여, 최소 급여, 평균 급여를 구하라.

```
SELECT SUM(E.Salary), MAX(E.Salary), MIN(E.Salary), AVG(E.Salary)
FROM EMPLOYEE AS E, DEPARTMENT AS D
WHERE E.Dno=D.Dnumber AND D.Dname='Research';
```

SQL 집단 함수

➢ 집단 함수 (Aggregate Function)

- 집단 함수: 여러 투플의 정보를 요약하여 하나의 투플로 요약하는 데에 사용함
- 그룹화 (grouping): 요약하기 전에 투플들을 부분 그룹으로 나누기 위해 사용
- 집단 함수의 종류: COUNT, SUM, MAX, MIN, AVG
  - COUNT: 질의의 결과로 검색된 투플이나 값들의 개수를 구함
  - SUM, MAX, MIN AVG: 숫자들의 집합이나 다중집합 (중복 허용) 에 적용되어 합, 최댓값, 최솟값, 평균값을 구함

[질의 21과 22] 회사의 총 사원 수 (Q21)와 'Research '부서에서 근무하는 총 사원수 (Q22)를 검색하라

```
SELECT COUNT(*) FROM EMPLOYEE;
SELECT COUNT (*) FROM EMPLOYEE, DEPARTMENT WHERE Dno=Dnumber AND Dname = ' Research ' ;
```

[질의 23] 데이터베이스에서 서로 다른 급여들의 개수를 구하라.

```
SELECT COUNT(DISTINCT Salary)
FROM EMPLOYEE;
```

SQL 그룹핑

➢ GROUP BY

- 릴레이션 내의 투플들을 어떤 애트리뷰트의 값을 기준으로 서로 겹치지 않는 여러 부분 집단으로 나누고자 할 경우
  - 이 때 기준이 되는 애트리뷰트를 **그룹화 애트리뷰트 (grouping attribute)** 라고 함
  - 그룹화 애티르뷰트를 GROUP BY 절에 명시

[질의 24] 각 부서에 대해서 부서 번호, 부서에 속한 사원들의 수, 각 부서에 속한 사원들의 평균 급여를 구하라.

```
SELECT Dno, COUNT (*), AGV(Salary) FROM EMPLOYEE GROUP BY Dno;
```

[질의 25] 각 프로젝트에 대해서 프로젝트 번호, 프로젝트 이름, 그 프로젝트에서 근무하는 사원들의 수를 검색하라.

```
SELECT Pnumber, Pname, COUNT(*) FROM PROJECT, WORKS_ON WHERE Pnumber=Pno
GROUP BY Pnumber, Pname;
```

SQL 그룹핑

➢ HAVING

- 그룹화 애트리뷰트에 같은 값을 갖는 투플들의 그룹에 대한 요약된 정보의 조건을 나타내며 이런 조건을 만족하는 그룹들만 질의 결과로 검색됨
  - 집단 함수를 적용할 그룹들을 선택

[질의 26] 두 명 이상의 사원이 근무하는 각 프로젝트에 대해서 프로젝트 번호, 프로젝트 이름, 프로젝트에서 근무하는 사원의 수를 검색하라.

```
SELECT Pnumber, Pname, COUNT(*) FROM PROJECT, WORKS_ON WHERE Pnumber = Pno
GROUP BY Pnumber, Pname HAVING COUNT(*) > 2;
```

[질의 27] 각 프로젝트에 대해서 프로젝트 번호, 프로젝트 이름, 5번 부서에 속하면서 프로젝트에서 근무하는 사원의 수를 검색하라.

```
SELECT Pnumber, Pname, COUNT(*) FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber=Pno AND Ssn=Essn AND Dno=5 GROUP BY Pnumber, Pname;
```

SQL 그룹핑

➢ HAVING

- 그룹화 애트리뷰트에 같은 값을 갖는 투플들의 그룹에 대한 요약된 정보의 조건을 나타내며 이런 조건을 만족하는 그룹들만 질의 결과로 검색됨
  - 집단 함수를 적용할 그룹들을 선택

[질의 28] 6명 이상의 사원이 근무하는 각 부서에 대해서 부서 번호와 40,000 달러가 넘는 급여를 받는 사원의 수를 검색하라.

```
SELECT Dno, COUNT (*) FROM EMPLOYEE
WHERE Salary>40000 AND Dno IN (SELECT Dno FROM EMPLOYEE GROUP BY Dno HAVING COUNT (*) > 5)
GROUP BY Dno;
```

숙제 1

- **제출일: ~10/14 15:00**

<참고사항>
- Late penalty: 하루에 20% 씩 감점 (5일 이후 점수 없음)
- Copy 시 점수 없음 (함께 과제 시에 반드시 명시)

<제출방법>
- 교육 시스템에 과제 제출 란에 제출 시간 전(10/14 수업시간 전)까지 업로드 후 프린트 해서 10/14 수업시간에 제출
- 늦게 제출하는 학생의 경우에도 동일하게 교육 시스템에 파일 업로드 후 프린트 해서 제출 (제출 시간은 업로드 시간 기준)

숙제 1

- **제출일: ~10/14 15:00**

<숙제 협업>

- 숙제를 서로 함께 도와가면서 했다면, 누구에게 어떤 부분에 대해서 물어보았는지 명시하시오
- 숙제를 함께 도와가며 할 수는 있지만, 협업을 명시 하였더라도 그대로 베끼는 행위는 카피로 0점처리 할 것임
- 서로 도와서 숙제를 하게되는 경우에도 해당 내용을 분명히 이해하고 자기만의 언어로 다시 작성하여야 함

SQL JOIN

➢ 특정 애트리뷰트를 기준으로 두 개의 테이블을 연결하여 합침

- INNER JOIN | LEFT (RIGHT) OUTER JOIN | FULL OUTER JOIN