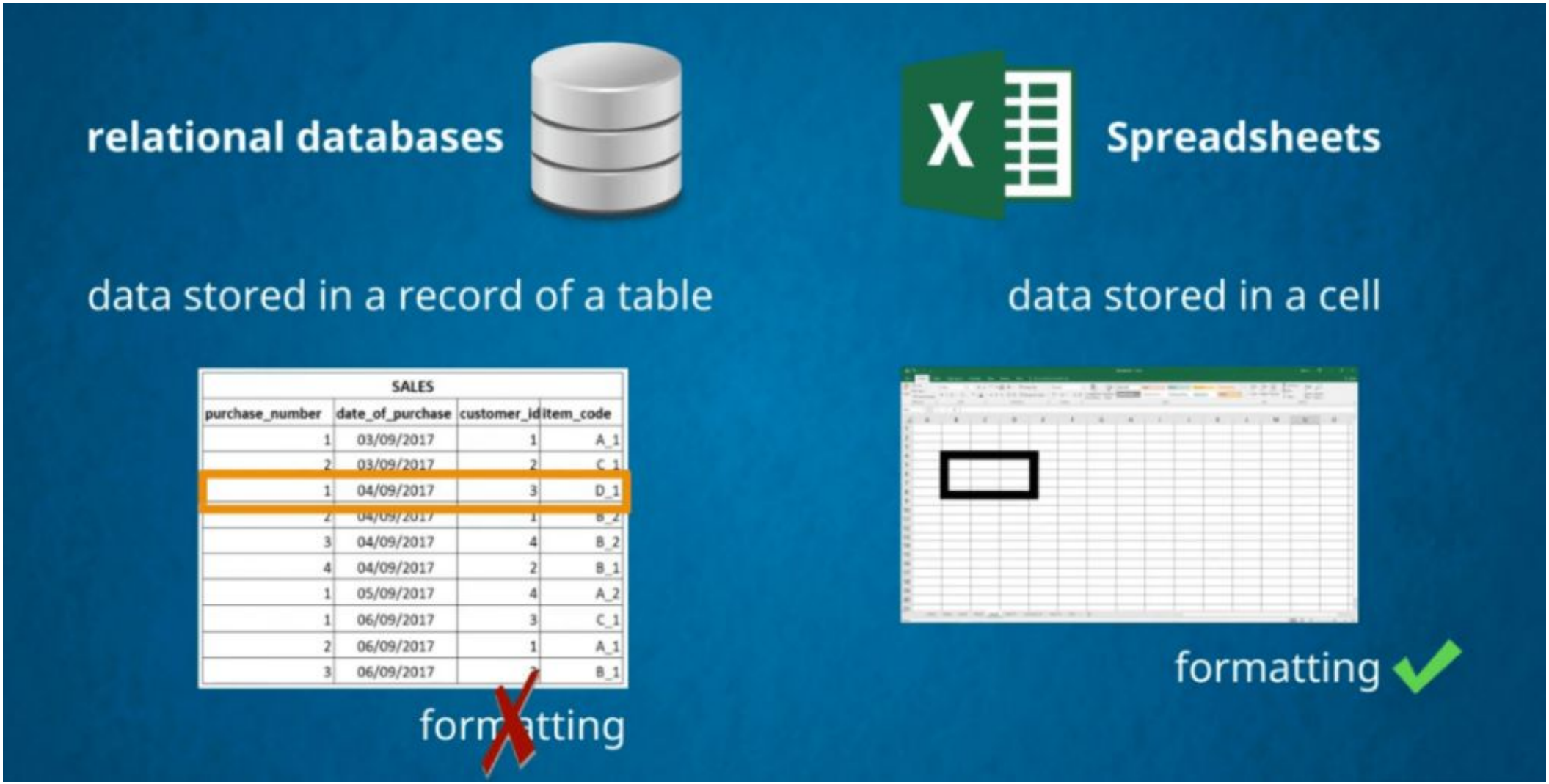
Database

sqlite3

Excel vs DataBase

- 필요 의한 발전
  - Variable(변수) < List or Dictionary(묶음) < Small Text File(파일) < Excel(계산 파일) < DB(큰 용량 파일) < Distributed system(나누어 관리 파일)
  - 유사성(Excel vs DB)



DataBase 분류

- Table 간 관계 측면
  - SQL(Structured Query Lanaguage) : 관계형 기반, Table & Record
  - NoSQL : Document-oriented, key-value 기반, Collection & Document
  - 중간 규모 앱엔 SQL vs NoSQL 성능 유사.

- 기술 적용 측면
  - ORM(Object-Relational Mapper), ODM(Object-Document Mapper)
  - 사용 편의성 : DB 추상화 레이어, SQL 직관적 변경 가능.
  - 도메인 변환 시 약간의 성능 저하.
  - 호환성 : 개발과 제품 플랫폼 사용 가능 여부 확인.

- DBMS 종류
  - RDB : Oracle, MySQL, MariaDB, PostgreSQL, Sqlite
  - NoSQL : MongoDB와 CouchBase, Cassandra와 HBase
  - In-memory DB : memcached, Redis(REmote DIctionary System)

Sqlite3

- install Menu > Menu > Extensions:MarketPlace >
  - Search ‘sqlite’
  - Search ‘SQLTools’
- 응용프로그램 주로 사용, 비교적 가벼운 데이터베이스, '시퀄라이트 (ˈsiːkwəl.laɪt)'
- 장 / 단점
  - 작고, 빠르며, 파일 복사로 백업이 끝난다.
  - 원격 사용 불가능하며, 동시 프로세스 접근 시 문제 발생.
- Affinity Data Type
  - NULL : The value is a NULL value.
  - INTEGER : The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
  - **REAL** : The value is a floating point value, stored as an 8-byte IEEE floating point number.
  - **TEXT** : The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
  - BLOB : The value is a blob of data, stored exactly as it was input.

Try - 기본 기능 구현.

❖ 실행결과
결과 따라 표시.

❖ 함께하기

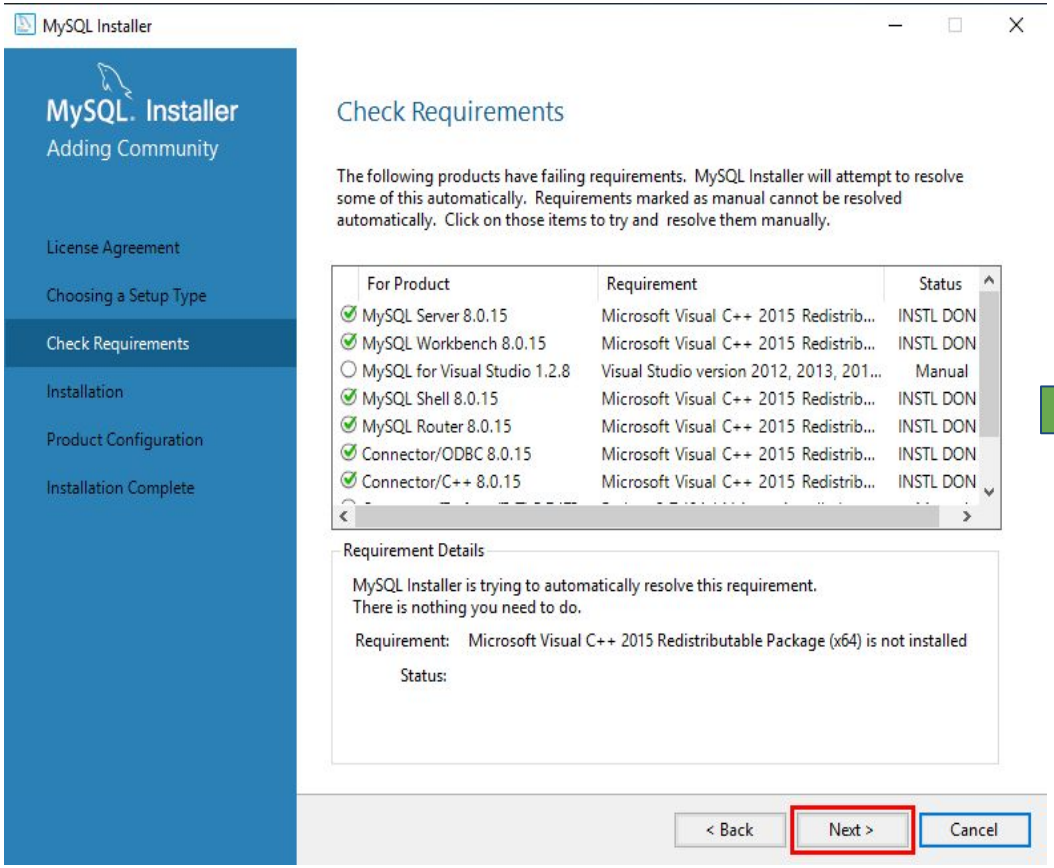
```
~$ sqlite3 testDB.db
sqlite>.help
sqlite>.header on
sqlite>SELECT * FROM ???;
sqlite>.schema sqlite_master
sqlite>.quit
~$ sqlite3 testDB.db
sqlite>.databases
```

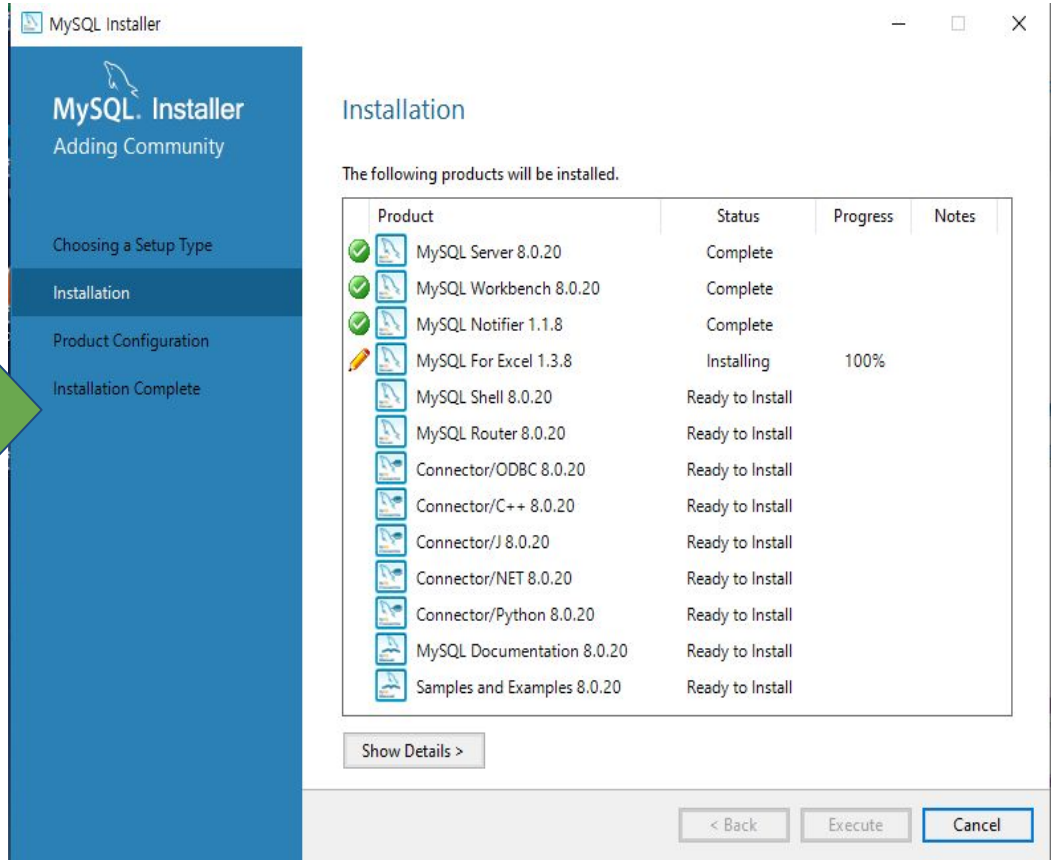
❖ 알아두기

```
~$ sqlite3 testDB.db .dump > testDB.sql
~$ sqlite3 testDB.db < testDB.sql
sqlite>ATTACH DATABASE 'testDB.db' as 'TEST';
sqlite>.databases
sqlite> DETACH DATABASE 'TEST';
sqlite>.databases
sqlite>.mode column
```

MySQL

- Download
  - https://dev.mysql.com/downloads/ > MySQL Community Server > installer





Try - 기본 기능 구현

❖ 해 보기

➢ 참조

```
CREATE TABLE lecture_INFORMATION (
  ID   FLOAT NULL  ,
  name TEXT  NULL   );
INSERT INTO lecture_INFORMATION (ID, name) VALUES (1, 'Allen');
SELECT * FROM lecture_INFORMATION;
DROP TABLE lecture_INFORMATION;
```

➢ 아래 조건 충족하고, git 적용

- Table name : member_information
- column : 아이디, 이름, 취미, 성적, 권한
- Github link 공유

Table - Create

❖ Syntax

CREATE TABLE database_name.table_name**(**
column1 datatype PRIMARY KEY(one or more columns),
.....
columnN datatype
);

```
sqlite> CREATE TABLE COMPANY(
            ID INT PRIMARY KEY     NOT NULL,
            NAME           TEXT    NOT NULL,
            AGE            INT     NOT NULL,
            ADDRESS        CHAR(50),
            SALARY         REAL
         );
sqlite> CREATE TABLE DEPARTMENT(
            ID INT PRIMARY KEY      NOT NULL,
            DEPT           CHAR(50) NOT NULL,
            EMP_ID         INT      NOT NULL
         );
sqlite>.tables
sqlite>.schema COMPANY
```

Table - Drop & Alter

❖ Drop Syntax

DROP TABLE database_name.table_name**;**

```
sqlite>DROP TABLE COMPANY;
sqlite>.tables
```

❖ Alter Syntax

ALTER TABLE database_name.table_name **RENAME TO** new_table_name;

ALTER TABLE database_name.table_name **ADD COLUMN** column_def...;

```
sqlite> ALTER TABLE COMPANY RENAME TO OLD_COMPANY;
sqlite> ALTER TABLE OLD_COMPANY ADD COLUMN SEX INTEGER;
sqlite> ALTER TABLE OLD_COMPANY ADD COLUMN PHONE VARCHAR(10);
```

❖ Truncate Command

➢ 블록 단위 저장 단편화 해소 : 질의 성능 영향

```
sqlite> VACUUM;
```

❖ Explain Query Command

```
sqlite> EXPLAIN SELECT * FROM COMPANY;
```

Query - Insert

- Syntax

  INSERT INTO TABLE_NAME [(column1, column2, column3,...columnN)]
  VALUES (value1, value2, value3,...valueN);

```
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        VALUES (1, 'Paul', 32, 'California', 20000.00 );
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        VALUES (2, 'Allen', 25, 'Texas', 15000.00 );
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        VALUES (5, 'David', 27, 'Texas', 85000.00 );
sqlite> INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        VALUES (6, 'Kim', 22, 'South-Hall', 45000.00 );
sqlite> INSERT INTO COMPANY VALUES (6, 'James', 24, 'Houston', 10000.00 );
sqlite> INSERT INTO COMPANY VALUES (7, 'James', 24, 'Houston', 10000.00 );
```

Query - Select

- Syntax

  SELECT column1, column2, columnN **FROM** table_name;

  SELECT * FROM table_name;

```
sqlite> .header on
sqlite> .mode column
sqlite> SELECT * FROM COMPANY;
sqlite> SELECT ID, NAME, SALARY FROM COMPANY;
sqlite> SELECT NAME FROM COMPANY;
```

- Populate One Table Using Another Table

  INSERT INTO first_table_name [(column1, column2, ... columnN)]

  SELECT column1, column2, ...columnN

  FROM second_table_name

  [**WHERE** condition];

```
sqlite> INSERT INTO OLD_COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
        SELECT ID,NAME,AGE,ADDRESS,SALARY
            FROM COMPANY;
```

Query - Update & Delete

- Update Syntax

  UPDATE table_name
  SET column1 = value1, column2 = value2...., columnN = valueN
  WHERE [condition];

```
sqlite> UPDATE COMPANY SET ADDRESS = 'Texas' WHERE ID = 6;
sqlite> UPDATE COMPANY SET ADDRESS = 'Texas', SALARY = 20000.00;
```

- Delete Syntax

  DELETE FROM table_name
  WHERE [condition];

```
sqlite> DELETE FROM COMPANY WHERE ID = 7;
sqlite> DELETE FROM COMPANY;
```

WHERE Clause

- Syntax

  SELECT column1, column2, columnN
  FROM table_name
  WHERE [condition]
  → **WHERE** [condition1] **AND** [condition2]... **AND** [conditionN];
  → **WHERE** [condition1] **OR** [condition2]... **OR** [conditionN]

```
sqlite> SELECT * FROM COMPANY WHERE AGE IS NOT NULL;
sqlite> SELECT * FROM COMPANY WHERE NAME LIKE 'Ki%';
sqlite> SELECT * FROM COMPANY WHERE NAME LIKE 'P__l';
sqlite> SELECT * FROM COMPANY WHERE NAME LIKE '%a%';
sqlite> SELECT * FROM COMPANY WHERE NAME LIKE '_a%';
sqlite> SELECT * FROM COMPANY WHERE AGE IN ( 25, 27 );
sqlite> SELECT * FROM COMPANY WHERE AGE NOT IN ( 25, 27 );
sqlite> SELECT * FROM COMPANY WHERE AGE BETWEEN 25 AND 27;
sqlite> SELECT * FROM COMPANY
         WHERE AGE >= 25 AND SALARY >= 65000;
sqlite> SELECT * FROM COMPANY WHERE AGE >= 25 OR SALARY >= 65000;
sqlite> SELECT * FROM COMPANY
         WHERE AGE > (SELECT AGE FROM COMPANY
                              WHERE SALARY > 65000);
```

Try - WHERE Clause

❖ 같이 하기

```
import pandas as pd
from sklearn import datasets
boston = datasets.load_boston()
pd_boston = pd.DataFrame(boston['data'], columns=boston['feature_names'])
import sqlite3
conn = sqlite3.connect('boston.db')
pd_boston.to_sql('boston_table', conn, if_exists='replace')
```

❖ 해 보기

➢ iris, wine, breast cancer, titanic 위와 같이 하기

GLOB Clause

- Syntax

SELECT FROM table_name **WHERE column GLOB** 'XXXX*'
or
SELECT FROM table_name WHERE column GLOB '*XXXX*'
or
SELECT FROM table_name WHERE column GLOB 'XXXX?'
or
SELECT FROM table_name WHERE column GLOB '?XXXX'
or
SELECT FROM table_name WHERE column GLOB '?XXXX?'
or
SELECT FROM table_name WHERE column GLOB '????'

```
sqlite> SELECT * FROM COMPANY WHERE AGE  GLOB '2*';
sqlite> SELECT * FROM COMPANY WHERE ADDRESS  GLOB '*-*';
```

Limit Clause

- Syntax

  SELECT column1, column2, columnN
  FROM table_name
  LIMIT [no of rows] **OFFSET** [row num]

```
sqlite> SELECT * FROM COMPANY LIMIT 6;
sqlite> SELECT * FROM COMPANY LIMIT 3 OFFSET 2;
```

Order by Clause

❖ Syntax

SELECT column-list
FROM table_name
[**WHERE** condition]
[**ORDER BY** column1, column2, .. columnN] [**ASC** | **DESC**];

```
sqlite> SELECT * FROM COMPANY ORDER BY SALARY ASC;
sqlite> SELECT * FROM COMPANY ORDER BY NAME, SALARY ASC;
sqlite> SELECT * FROM COMPANY ORDER BY NAME DESC;
```

❖ 해 보기

➢ iris, wine, breast cancer, titanic 위와 같이 하기

GROUP BY Clause

- Syntax

```
SELECT column-list
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2....columnN
ORDER BY column1, column2....columnN
```

```
sqlite> SELECT NAME, SUM(SALARY) AS TOTAL
        FROM COMPANY
        GROUP BY NAME;
sqlite> INSERT INTO COMPANY VALUES (8, 'Paul', 24, 'Houston', 20000.00 );
sqlite> INSERT INTO COMPANY VALUES (9, 'James', 44, 'Norway', 5000.00 );
sqlite> INSERT INTO COMPANY VALUES (10, 'James', 45, 'Texas', 5000.00 );
sqlite> SELECT NAME, SUM(SALARY)
        FROM COMPANY
        GROUP BY NAME
        ORDER BY NAME;
sqlite>  SELECT NAME, SUM(SALARY)
        FROM COMPANY
        GROUP BY NAME
        ORDER BY NAME DESC;
```

Having Clause

❖ Syntax

SELECT column1, column2 **FROM** table1, table2
WHERE [conditions]
GROUP BY column1, column2
HAVING [conditions]
ORDER BY column1, column2

```
sqlite > SELECT * FROM COMPANY
         GROUP BY name
         HAVING count(name) < 2;          → Try HAVING count(name) > 2;
sqlite > SELECT *
         FROM COMPANY
         GROUP BY ADDRESS
         HAVING count(ADDRESS) >= 2;
```

❖ Distinct Keyword

SELECT **DISTINCT** column1, column2,.....columnN
FROM table_name
WHERE [condition]

```
sqlite> SELECT SALARY FROM COMPANY;
sqlite> SELECT DISTINCT SALARY FROM COMPANY;
```

Try - Fill Values

❖ 해 보기

➢ import DB from good influence excel

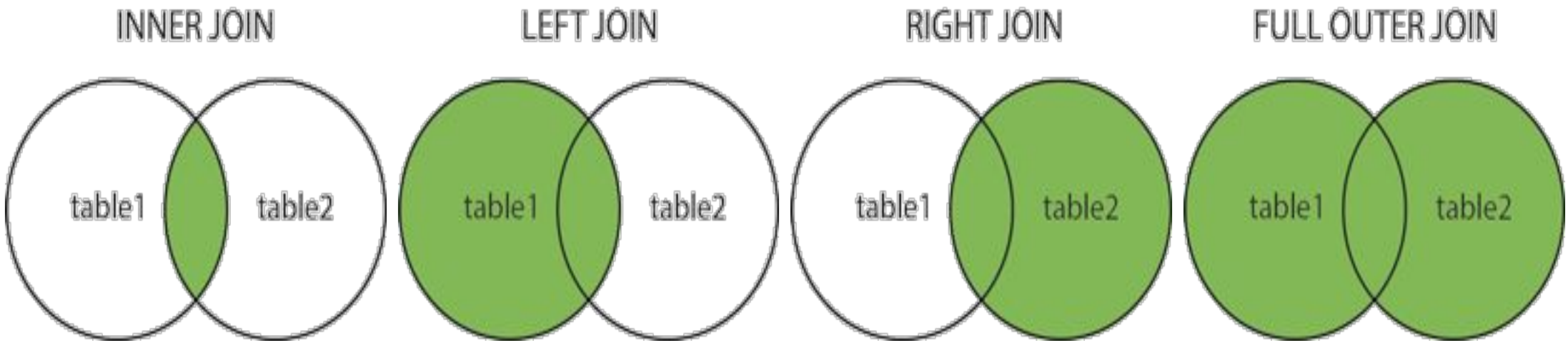
➢ 작성한ERD 맞게 분리해 Table에 넣기

```
sqlite> INSERT INTO OLD_COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
                    SELECT ID,NAME,AGE,ADDRESS,SALARY
                        FROM COMPANY;
sqlite> ALTER TABLE OLD_COMPANY ADD COLUMN GENDER INTEGER;
sqlite> ALTER TABLE OLD_COMPANY DROP COLUMN GENDER;
```

Joins - Ready

❖ Ready

```
sqlite> CREATE TABLE DEPARTMENT(
                ID INT PRIMARY KEY      NOT NULL,
                DEPT           CHAR(50) NOT NULL,
                EMP_ID         INT      NOT NULL       );
sqlite> INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID) VALUES (1, 'IT Billing', 1 );
sqlite> INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID) VALUES (2, 'Engineering', 2 );
sqlite> INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID) VALUES (3, 'Finance', 7 );
sqlite> INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID) VALUES (4, 'IT Billing', 1 );
sqlite> INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID) VALUES (5, 'Finance', 7 );
```



Joins - INNER JOIN

- Cross Syntax

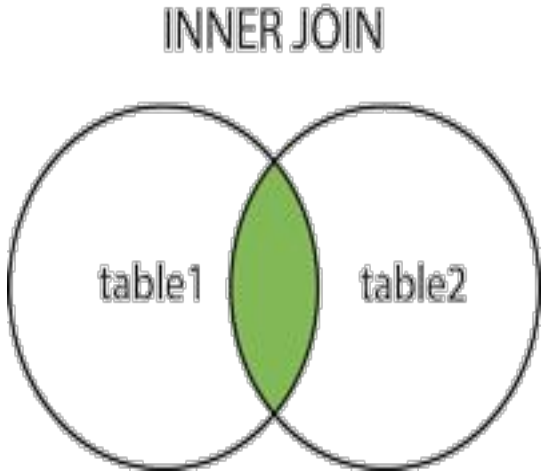
  SELECT ... FROM table1 **CROSS JOIN** table2 ...

```
sqlite> SELECT EMP_ID, NAME, DEPT
        FROM COMPANY CROSS JOIN DEPARTMENT;
```

- Inner Syntax

  SELECT ... FROM table1 **[INNER] JOIN** table2 **ON** conditional_expression ...

```
sqlite> SELECT EMP_ID, NAME, DEPT
        FROM COMPANY
            INNER JOIN DEPARTMENT
            ON COMPANY.ID = DEPARTMENT.EMP_ID
            AND COMPANY.NAME = 'Paul';
sqlite> SELECT EMP_ID, COMPANY.NAME, DEPT, OLD_COMPANY.NAME
        FROM COMPANY
            INNER JOIN DEPARTMENT
            ON COMPANY.ID = DEPARTMENT.EMP_ID
                INNER JOIN OLD_COMPANY
                ON COMPANY.ID = OLD_COMPANY.ID;
```

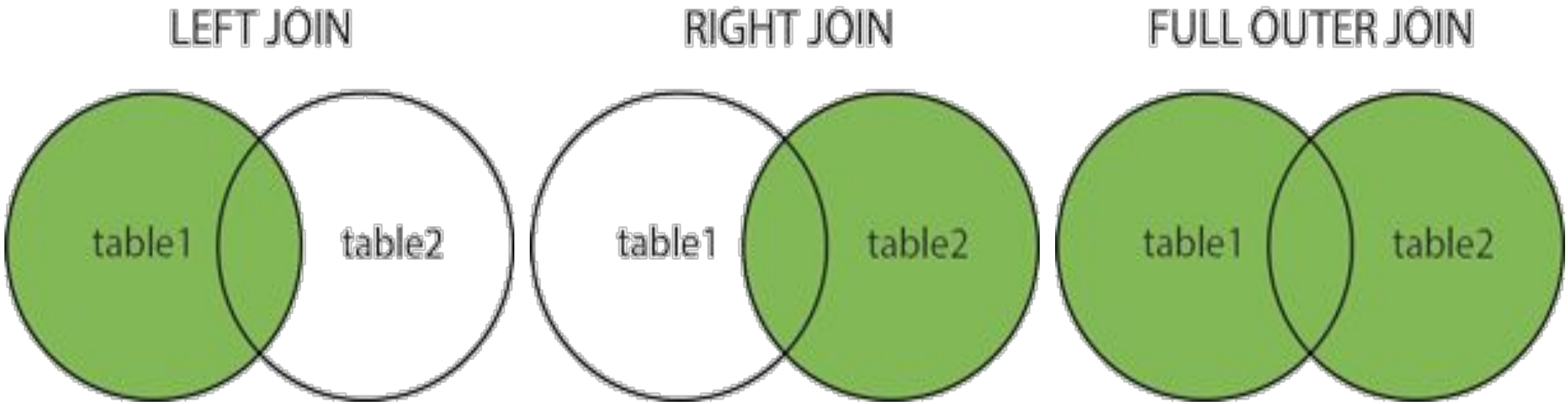


Joins - OUTER JOIN

- Outer Syntax : **SQLite only supports the LEFT OUTER JOIN**

```
SELECT ...
FROM table1 LEFT OUTER JOIN table2 ON conditional_expression ...
SELECT ... FROM table1 LEFT OUTER JOIN table2 USING ( column1 ,... ) ...
```

```
sqlite> SELECT EMP_ID, NAME, DEPT
        FROM COMPANY LEFT OUTER JOIN DEPARTMENT
        ON COMPANY.ID = DEPARTMENT.EMP_ID;
```



Union & Union All Clause

❖ Syntax

SELECT column1 [, column2] FROM table1 [, table2] [WHERE condition]

UNION → Try **UNION ALL**

SELECT column1 [, column2] FROM table1 [, table2] [WHERE condition]

```
sqlite> SELECT EMP_ID, NAME, DEPT FROM COMPANY
                                    INNER JOIN DEPARTMENT
                                     ON COMPANY.ID = DEPARTMENT.EMP_ID
        UNION
        SELECT EMP_ID, NAME, DEPT FROM COMPANY
                                    LEFT OUTER JOIN DEPARTMENT
                                    ON COMPANY.ID = DEPARTMENT.EMP_ID;
sqlite>  SELECT EMP_ID, NAME, DEPT FROM COMPANY
                                    INNER JOIN DEPARTMENT
                                    ON COMPANY.ID = DEPARTMENT.EMP_ID
        UNION ALL
        SELECT EMP_ID, NAME, DEPT FROM COMPANY
                                    LEFT OUTER JOIN DEPARTMENT
                                    ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

Fill Values

```
sqlite> UPDATE COMPANY
           SET ADDRESS = NULL, SALARY = NULL
           WHERE ID IN(6,7);
sqlite> SELECT  ID, NAME, AGE, ADDRESS, SALARY
        FROM COMPANY
        WHERE SALARY IS NOT NULL;
sqlite> SELECT  ID, NAME, AGE, ADDRESS, SALARY
        FROM COMPANY
        WHERE SALARY IS NULL;
```

Alias

- Syntax

```
SELECT column_name AS alias_name, column2....
FROM table_name AS alias_name
WHERE [condition];
```

```
sqlite> SELECT C.ID, C.NAME, C.AGE, D.DEPT
        FROM COMPANY AS C, DEPARTMENT AS D
        WHERE  C.ID = D.EMP_ID;
sqlite> SELECT
             C.ID AS COMPANY_ID,
             C.NAME AS COMPANY_NAME,
             C.AGE, D.DEPT
        FROM COMPANY AS C, DEPARTMENT AS D
        WHERE  C.ID = D.EMP_ID;
```

Indexes

- CREATE INDEX Syntax

  CREATE INDEX index_name ON table_name;

- Column Indexes Syntax

  CREATE INDEX index_name
  ON table_name (column_name1, column_name2);

```
sqlite> CREATE INDEX salary_index ON COMPANY (salary);
sqlite> .indices COMPANY
sqlite> SELECT * FROM sqlite_master WHERE type = 'index';
```

- Unique Indexes Syntax

  CREATE UNIQUE INDEX index_name
  on table_name (column_name);

- DROP INDEX

  DROP INDEX index_name;

```
sqlite> DROP INDEX salary_index;
```

Views

- Syntax

```
CREATE [TEMP | TEMPORARY] VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

```
sqlite> CREATE VIEW COMPANY_VIEW AS
            SELECT ID, NAME, AGE
            FROM  COMPANY;
```

```
sqlite> DROP VIEW COMPANY_VIEW;
```

Transaction

- Syntax

  BEGIN; or BEGIN TRANSACTION;

  COMMIT; or END TRANSACTION; or ROLLBACK;

```
sqlite> BEGIN;
sqlite> DELETE FROM COMPANY WHERE AGE = 25;
sqlite> ROLLBACK;
```

```
sqlite> BEGIN;
sqlite> DELETE FROM COMPANY WHERE AGE = 25;
sqlite> COMMIT;
```

- AUTOINCREMENT

```
sqlite> CREATE TABLE COMPANY(
   ID INTEGER PRIMARY KEY AUTOINCREMENT,
   NAME           TEXT      NOT NULL,
   AGE            INT       NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL
);
```

Subqueries(1)

- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators such as **=, <, >, >=, <=, IN, BETWEEN, etc**
- Subqueries with SELECT Statement

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR  (SELECT column_name [, column_name ]
                             FROM table1 [, table2 ] [WHERE])
```

```
sqlite> SELECT * FROM COMPANY
        WHERE ID IN (SELECT ID FROM COMPANY
                     WHERE SALARY > 45000) ;
sqlite> SELECT AGE FROM COMPANY
        WHERE EXISTS (SELECT AGE FROM COMPANY
                      WHERE SALARY > 65000);
```

- Subqueries with INSERT Statement

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
   SELECT [ *|column1 [, column2 ]
   FROM table1 [, table2 ] [ WHERE VALUE OPERATOR ]
```

```
sqlite> INSERT INTO COMPANY_BKP
        SELECT * FROM COMPANY
        WHERE ID IN (SELECT ID FROM COMPANY) ;
```

Subqueries(2)

- Subqueries with UPDATE Statement

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
      FROM TABLE_NAME)
   [ WHERE) ]
sqlite> UPDATE COMPANY
        SET SALARY = SALARY * 0.50
        WHERE AGE IN (SELECT AGE FROM COMPANY_BKP
                              WHERE AGE >= 27 );
```

- Subqueries with DELETE Statement

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
      FROM TABLE_NAME)
   [ WHERE) ]
sqlite> DELETE FROM COMPANY
        WHERE AGE IN (SELECT AGE FROM COMPANY_BKP
                              WHERE AGE > 27 );
```

Date & Time

```
sqlite> SELECT date('now');
2013-05-07
sqlite> SELECT date('now','start of month','+1 month','-1 day');
2013-05-31
sqlite> SELECT datetime(1092941466, 'unixepoch');
2004-08-19 18:51:06
sqlite> SELECT datetime(1092941466, 'unixepoch', 'localtime');
2004-08-19 13:51:06
sqlite> SELECT strftime('%s','now');
1393348134
sqlite> SELECT strftime('%s','now')
      - strftime('%s','2004-01-01 02:34:56');
295001572
sqlite> SELECT date('now','start of year','+9 months','weekday 2');
2013-10-01
sqlite> SELECT time('12:00', 'localtime');
05:00:00
sqlite> SELECT time('12:00', 'utc');
19:00:00
```

Sub	Description
%d	Day of month, 01-31
%f	seconds, SS.SSS
%H	Hour, 00-23
%m	Month, 00-12
%M	Minute, 00-59
%s	Seconds since 1970-01-01
%S	Seconds, 00-59
%w	Day of week, 0-6 (0 is Sunday)
%W	Week of year, 01-53
%Y	Year, YYYY
%%	% symbol

Useful Functions

```
sqlite> SELECT count(*) FROM COMPANY;
sqlite> SELECT max(salary) FROM COMPANY;
sqlite> SELECT min(salary) FROM COMPANY;
sqlite> SELECT avg(salary) FROM COMPANY;
sqlite> SELECT sum(salary) FROM COMPANY;
sqlite> SELECT random() AS Random;
sqlite> SELECT abs(5), abs(-15), abs(NULL), abs(0), abs("ABC");
sqlite> SELECT upper(name) FROM COMPANY;
sqlite> SELECT lower(name) FROM COMPANY;
sqlite> SELECT name, length(name) FROM COMPANY;
sqlite> SELECT sqlite_version() AS 'SQLite Version';
```

Try - ERD DB 적용 구현과 이해

❖ 해보기

- ➢ 회원 정보(Member) : ID(Email), password, name, signup Time
- ➢ 회원 취미(Hobby) : name - 입력(0,1,2) → Relationship Member
- ➢ 회원 로그 정보(LogHistory) : login Time, logout Time - 입력(하루 0회 이상 가능)
- ➢ 공지사항(Notice) : title, content, insert Time → Relationship Member
- ➢ 파일 정보(Files) : name, directory, size, insert Time, → Relationship Notice
- ➢ Apply database with your table schema.

```
~$ sqlite3 testDB.db < testDB.sql
 sqlite>.databases
 sqlite>.tables
```

- ➢ insert 3 record each Table.
- ➢ delete 1 record any relationship


수고하셨습니다.