

# SQL과 데이터 베이스

강사: 장순용 박사

광주인공지능사관학교 제 2기 (2021/06/16~2021/12/02) 용도로 제공되는 강의자료 입니다. 지은이의 허락 없이는 복제와 배포를 금합니다.

# 순서

## 1. SQL과 데이터 베이스.

### 1.1. 개요.

### 1.2. SQL 기초.

### 1.3. 데이터 베이스 생성과 변경.

### 1.4. SQL 함수.

### 1.5. Python에서의 접근.

## 데이터 베이스의 특징:

- 1950년대 미군이 처음 구축 (컴퓨터 도서관).
- 복수 사용자의 요구에 대응해서 데이터를 받아들이고 저장, 공급하기 위해서 일정한 구조에 따라서 편성된 데이터의 집합.
- 데이터 베이스의 구조적 특징:
  - 통합성 (integrated): 동일 데이터가 중복되지 않음.
  - 저장됨 (stored): 컴퓨터 기술을 바탕으로 저장 매체에 저장됨.
  - 공용성 (shared): 여러 사용자가 다양한 목적으로 사용할 수 있음.
  - 변화성 (changeable): 새로운 데이터의 삽입, 기존 데이터의 삭제 및 변경 가능.

## 데이터 베이스 (DB)의 종류:

- 계층형 DB: 트리 구조를 이용해서 상호 관계를 계층적으로 정의함.
- 네트워크형 DB: 그래프 구조를 이용해서 상호 관계를 계층적으로 정의함.
- 관계형 DB: 계층형과 네트워크형의 복잡한 구조를 단순화 시킨 모델. 단순한 표 (table)를 이용하여 데이터의 상호관계를 정의함.
- 객체 지향형 DB: 객체의 개념 도입. 멀티미디어와 같이 복잡한 관계를 가진 데이터를 표현하는데 효과적임.

## 데이터 베이스 (DB)의 종류:

- 계층형 DB: 트리 구조를 이용해서 상호 관계를 계층적으로 정의함.
- 네트워크형 DB: 그래프 구조를 이용해서 상호 관계를 계층적으로 정의함.
- 관계형 DB: 계층형과 네트워크형의 복잡한 구조를 단순화 시킨 모델. 단순한 표 (table)를 이용하여 데이터의 상호관계를 정의함.
- 객체 지향형 DB: 객체의 개념 도입. 멀티미디어와 같이 복잡한 관계를 가진 데이터를 표현하는데 효과적임.

## SQL (Structured Query Language):

- 1974년 IBM 연구소에서 개발한 SEQUEL에서 유래.
- 관계형 데이터 베이스 (RDB, Relational Data Base) 관리 체계.
- 3가지 용도로 사용된다:
  - 1). 데이터 정의 (DDL): 데이터 베이스를 정의, 생성, 변경, 삭제 하는 것.
  - 2). 데이터 조작 (DML): 질의 (query)를 통해서 데이터를 불러오고 처리 하는 것.
  - 3). 데이터 제어 (DCL): 데이터의 보안, 무결성, 병행 수행 제어 등을 하는 것.

# 데이터 베이스 개요

## SQL (Structured Query Language):

- 경량 (light weight) 데이터 베이스 용:

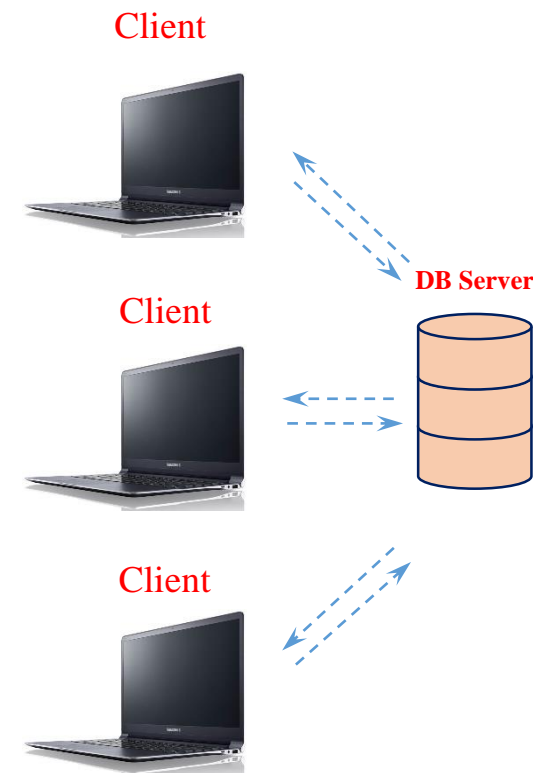
⇒ 1인 사용자 전제. 서버-클라이언트 개념 없음. 로컬 컴퓨터에 탑재.

예). SQLite (파이썬 패키지에 기본적으로 포함됨).

- 데이터 베이스 서버 (server) 구축 용:

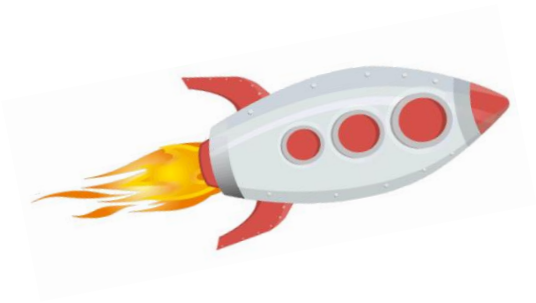
⇒ 다수의 사용자 전제. 하나의 서버에 다수의 클라이언트 접속.

예). MySQL, Microsoft SQL, Oracle SQL, PostgreSQL, MariaDB, 등.



## 데이터 베이스의 활용: CRM (Customer Relationship Management)

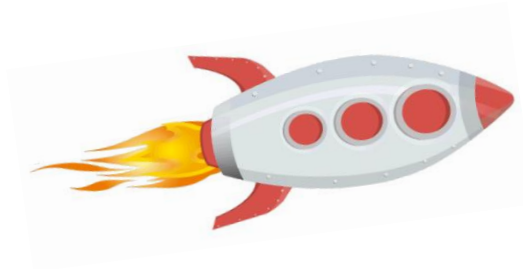
“고객관계관리” 라고도 한다. 고객과 관련된 내외부 자료를 분석, 통합하여 고객 중심 자원을 극대화 하고 이를 토대로 고객 특성에 맞게 마케팅 활동을 계획, 지원, 평가하는 과정.





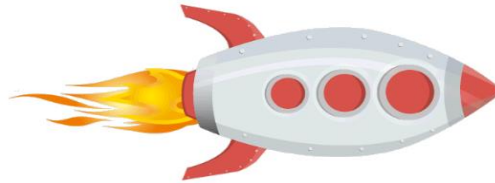
## 데이터 베이스의 활용: SCM (Supply Chain Management)

기업에서 원재료의 생산, 유통 등 모든 공급망 단계를 최적화해서 수요자가 원하는 제품을 원하는 시간과 장소에 제공하는 “공급망 관리”를 뜻한다.



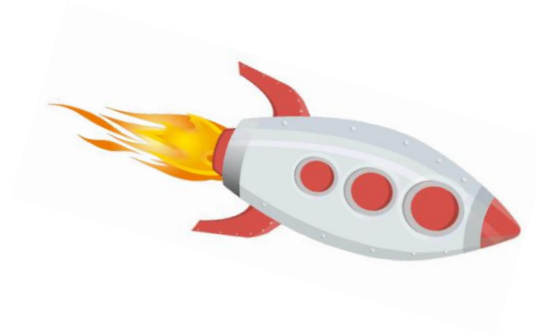
## 데이터 베이스의 활용: ERP (Enterprise Resource Planning)

인사, 재무, 생산 등 기업의 전 분야에 걸쳐 독립적으로 운영되던 관리 시스템의 경영자원을 하나의 통합 시스템으로 재구축함으로써 생산성을 극대화 하려는 경영혁신기법을 의미한다.



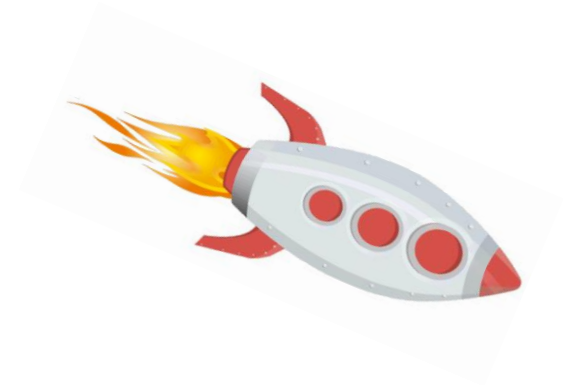
## 데이터 베이스의 활용: BI (Business Intelligence)

기업이 보유하고 있는 수많은 데이터를 정리하고 분석해 기업의 의사결정에 활용하는 일련의 프로세스를 말함. 기업의 사용자가 더 좋은 의사결정을 하도록 데이터를 수집, 저장, 분석, 접근을 지원하는 응용 시스템과 기술이다.



## 데이터 베이스의 활용: KMS (Knowledge Management System)

기업의 환경이 물품을 주로 생산하던 산업사회에서, 지적 재산의 중요성이 커지는 지식사회로 급격히 이동함에 따라, 기업 경영을 지식이라는 관점에서 새롭게 조명하는 접근방식. 의사결정의 주체인 인적 자원이 떠나면 그가 갖고 있던 지식 자원도 함께 떠나가고 기업의 지적 자원이 소실되니, 이것을 방지하는 차원에서 KMS를 활용함.



# 순서

## 1. SQL과 데이터 베이스.

### 1.1. 개요.

### 1.2. SQL 기초.

### 1.3. 데이터 베이스 생성과 변경.

### 1.4. SQL 함수.

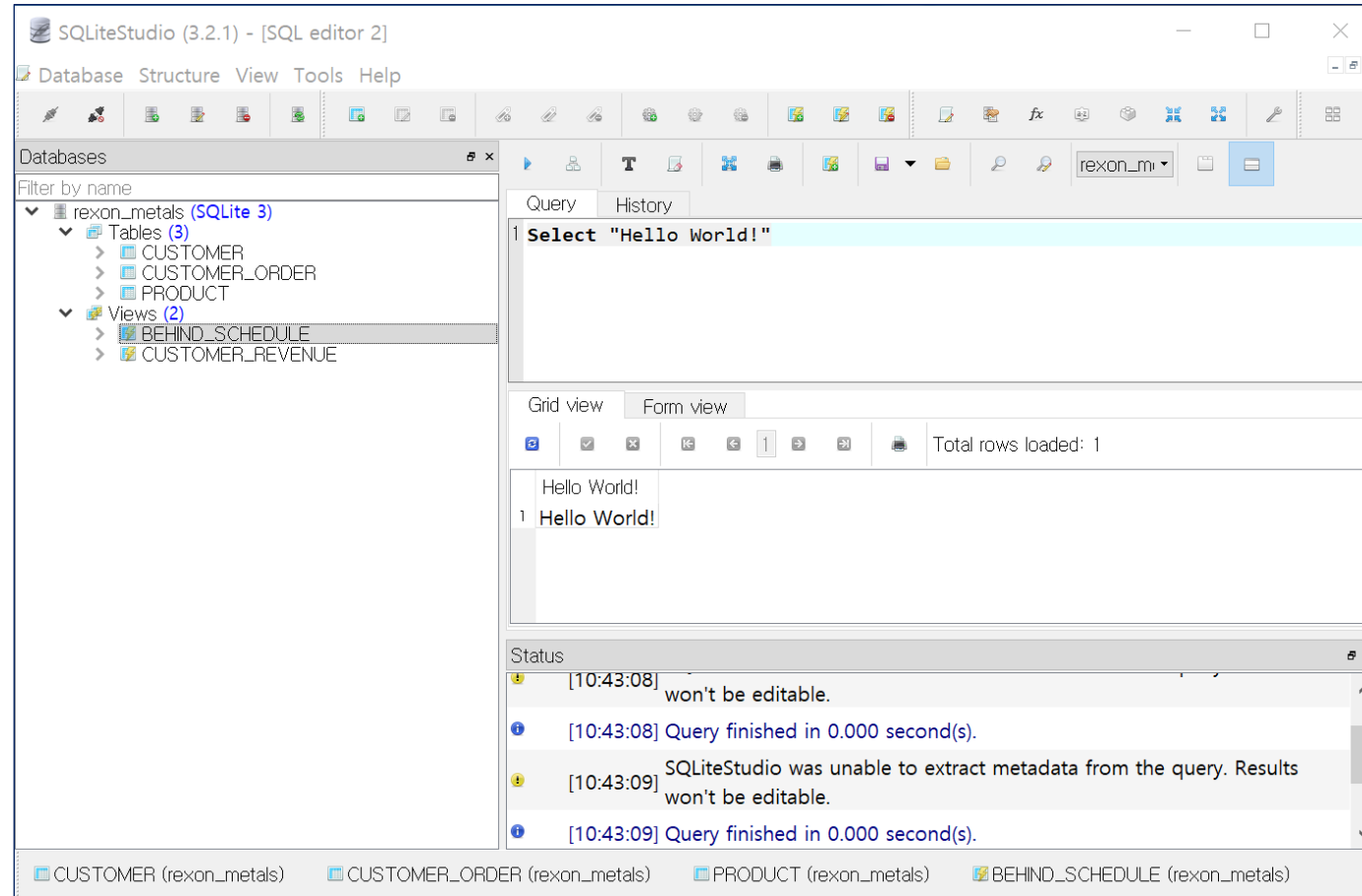
### 1.5. Python에서의 접근.

SQL (Structured Query Language): SQLiteStudio.

- SQLiteStudio는 사용하기 쉬운 SQL 개발 환경이다.
- 설치 방법:
  - 1). <https://sqlitestudio.pl> 로 가서 압축파일 내려 받음.  
**SQLiteStudio-3.3.3.zip.**
  - 2). 압축을 풀고, 폴더 전체를 원하는 위치로 옮김.
  - 3). **SQLiteStudio.exe**로 바로가기 링크를 만들어서 실행할 수 있도록 준비해 둔다.

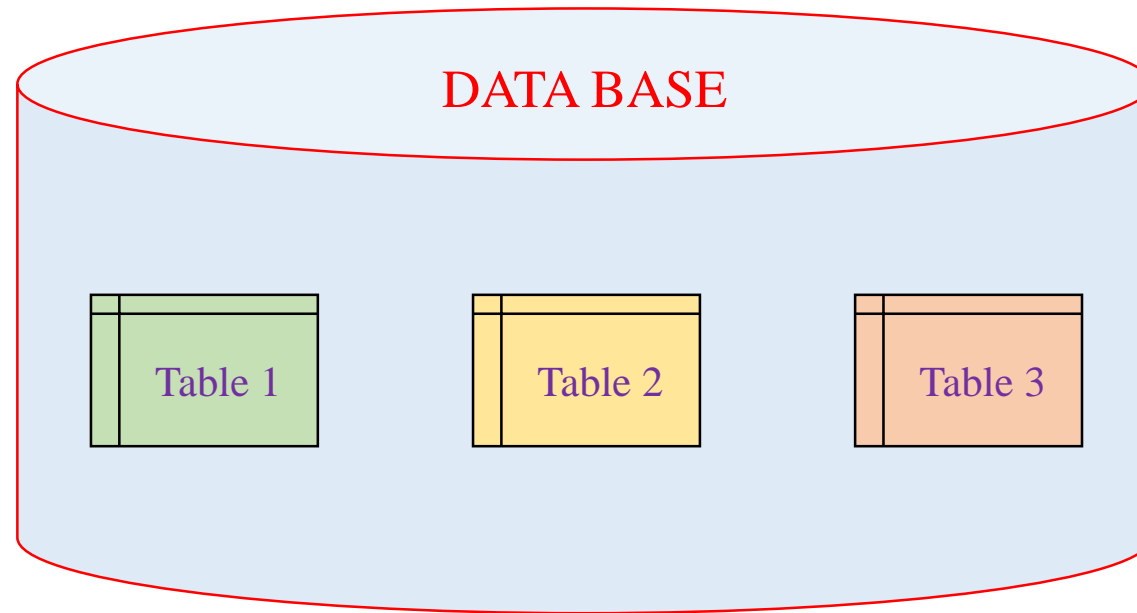
# 데이터 베이스 개요

SQL (Structured Query Language): SQLiteStudio.



# 데이터 베이스 개요

SQL (Structured Query Language): 데이터 베이스와 테이블.





# 데이터 베이스 개요

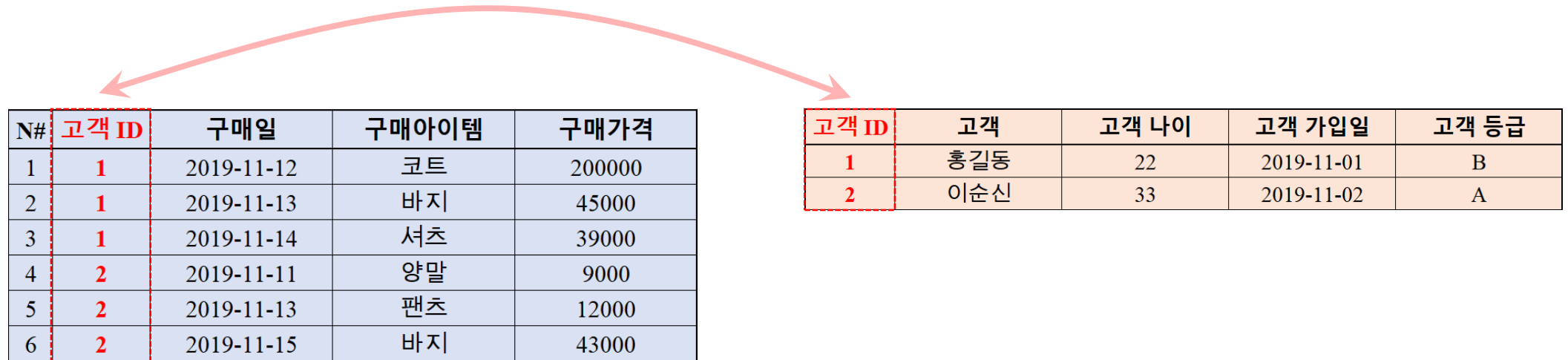
SQL (Structured Query Language): 여러 테이블로 분산하는 이유.

N#	고객 ID	고객	고객 나이	고객 가입일	고객 등급	구매일	구매아이템	구매가격
1	1	홍길동	22	2019-11-01	B	2019-11-12	코트	200000
2	1	홍길동	22	2019-11-01	B	2019-11-13	바지	45000
3	1	홍길동	22	2019-11-01	B	2019-11-14	셔츠	39000
4	2	이순신	33	2019-11-02	A	2019-11-11	양말	9000
5	2	이순신	33	2019-11-02	A	2019-11-13	팬츠	12000
6	2	이순신	33	2019-11-02	A	2019-11-15	바지	43000

한 개의 테이블로 통합해 놓으면 중복이 발생할 수 있다.

# 데이터 베이스 개요

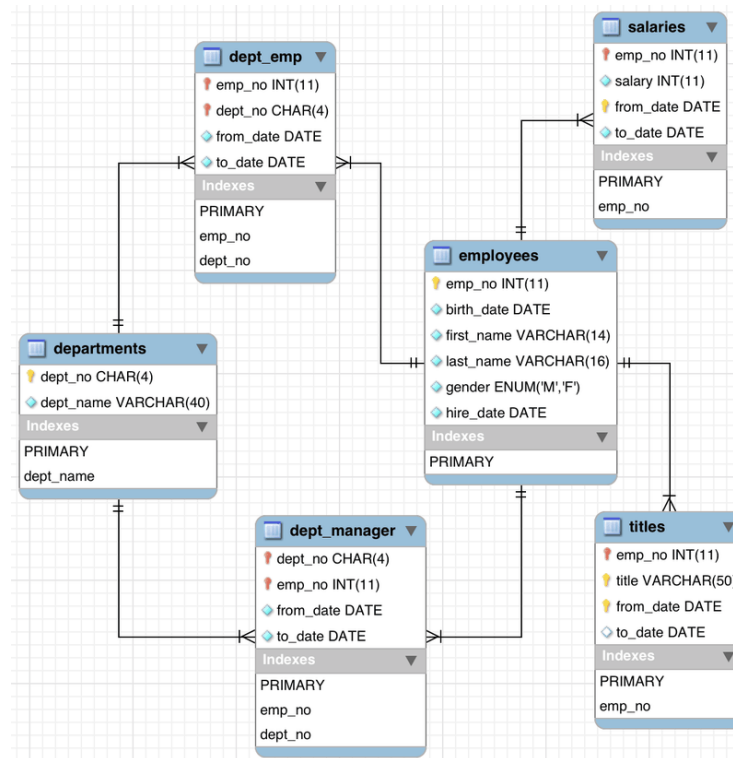
SQL (Structured Query Language): 여러 테이블로 분산하는 이유.



두 개의 테이블로 분산해 놓으면 효율적이다.  
일종의 KEY 변수로 연결해 놓을 수 있다.

# 데이터 베이스 개요

SQL (Structured Query Language): 여러 테이블로 분산하는 이유.



관계형 데이터 베이스 관리 시스템 (RDBMS)

Relational Data Base Management System

# SQL 기초 문법

SQL 기초 문법: 행 가져오기.

- 1행 주석은 '--'로 시작. Multi-line 주석은 '/\*'로 시작하여 '\*/'로 끝냄.
- SELECT 명령어 사용해 보기.

예).

```
SELECT 'Hello, World';           -- 문자열 출력.
SELECT 1 + 2;                    -- 계산결과 출력.
SELECT * FROM Country;          -- "Country"는 table 이름.
SELECT * FROM Country ORDER BY Name; -- 정렬 적용.
SELECT Name, LifeExpectancy AS 'Life Expectancy' FROM Country; -- 이름 바꿈.
SELECT COUNT(*) FROM Country;    -- 행 수 카운팅.
SELECT * FROM Country ORDER BY Name LIMIT 5; -- 출력 수 한정.
SELECT * FROM Country ORDER BY Name LIMIT 5 OFFSET 5; -- 첫 5개 건너 뛴.
```

# SQL 기초 문법

## SQL 기초 문법: 특정 컬럼 가져오기.

- SELECT를 사용해서 특정 컬럼을 불러올 수 있다.

예).

```
SELECT * FROM Country ORDER BY Code;
```

-- 모든 컬럼.

```
SELECT Name, Code, Region, Population FROM Country ORDER BY Code;
```

- 컬럼을 불러와서 이름을 변경하여 출력할 수 있다.

예).

```
SELECT Name AS Country, Code FROM Country ORDER BY Code;
```

- 간단한 수식을 적용할 수도 있다.

예).

```
SELECT Name AS Country, Population / 1000 AS 'Pop (1000s)' FROM Country;
```

# SQL 기초 문법

SQL 기초 문법: 정렬된 결과 보여주기.

- SELECT + ORDER BY를 사용해서 정렬된 결과를 보여줄 수 있다.

예).

SELECT Name FROM Country; -- 정렬되지 않음.

SELECT Name FROM Country ORDER BY Name; -- 오름차순 정렬 (Default).

SELECT Name FROM Country ORDER BY Name **DESC**; -- 내림차순 정렬.

SELECT Name FROM Country ORDER BY Name **ASC**; -- 오름차순 정렬.

- 두 가지 이상의 기준 (컬럼)으로 정렬할 수 있다. 먼저 나오는 컬럼을 우선시.

예).

SELECT Name, Continent FROM Country ORDER BY Continent, Name;

SELECT Name, Continent FROM Country ORDER BY **Continent DESC, Name**;

# SQL 기초 문법

SQL 기초 문법: 필터링 결과 보여주기.

- SELECT + WHERE를 사용해서 필터링 결과를 보여줄 수 있다.
- AND, OR, NOT 등으로 조건문을 연결할 수 있다.
- BETWEEN n1 AND n2로 n1~n2 레인지에 속하는 값만 필터링 할 수 있다.

예).

```
SELECT Name, Continent, Population FROM Country WHERE Population < 100000  
ORDER BY Population DESC;
```

```
SELECT Name, Continent, Population FROM Country WHERE Population < 100000  
AND Continent = 'Oceania' ORDER BY Population DESC;
```

```
SELECT Name, Population FROM Country WHERE Population BETWEEN 1000000  
AND 10000000 ORDER BY Population DESC;
```

# SQL 기초 문법

SQL 기초 문법: 문자열 필터링 결과 보여주기.

- SELECT + LIKE을 사용해서 필터링 결과를 보여줄 수 있다.
- '%' 는 불특정 문자열 부분을 의미하고 '\_'는 불특정 문자를 의미한다.

예).

```
SELECT Name, Continent FROM Country WHERE Name LIKE '%island' -- 끝.
```

```
SELECT Name, Continent FROM Country WHERE Name LIKE 'island%' -- 시작.
```

```
SELECT Name, Continent FROM Country WHERE Name LIKE '%island%' -- 중간.
```

```
SELECT Name, Continent FROM Country WHERE Name LIKE '_a%' -- 두번째 문자.
```

- SELECT + IN을 사용해서 멤버십 필터링 결과를 보여줄 수 있다.

예).

```
SELECT Name, Continent FROM Country WHERE Continent IN ( 'Europe', 'Asia' );
```



# SQL 기초 문법

SQL 기초 문법: 중복 제거해서 출력.

- DISTINCT 키워드를 사용해서 중복을 제거하고 출력할 수 있다.

예).

```
SELECT CountryCode, Name FROM City;
```

```
SELECT DISTINCT CountryCode FROM City;           -- 중복없이 국가코드 출력.
```

```
SELECT COUNT(CountryCode) FROM City;           -- 국가코드 수 카운팅.
```

```
SELECT COUNT(DISTINCT CountryCode) FROM City; -- 중복 없는 국가코드 수.
```

## SQL 기초 문법

SQL 기초 문법: 출력된 결과 이어서 사용.

- SELECT로 출력된 결과를 "가상"의 테이블로 사용하여 새로운 SELECT 문을 만들 수 있다.

예).

```
SELECT Name FROM (SELECT Code, Name, Continent FROM Country);
```

```
SELECT COUNT(*) FROM (SELECT DISTINCT CountryCode FROM City);
```

## 코딩 예제 #01

---

→ SQLiteStudio 에서 기초 SQL Query를 실행해 본다 - Part I. ←

→ 사용: **script\_01.txt** , **world.db** ←

# SQL 기초 문법

SQL 기초 문법: 테이블 생성 및 행 추가.

- CREATE TABLE로 데이터 베이스에 테이블을 추가할 수 있다.

예).

```
CREATE TABLE test ( a INT, b TEXT, c TEXT );      -- "a", "b", "c" 컬럼의 테이블.  
SHOW TABLES;                                     -- 테이블 목록 (비 SQLite).  
SELECT * FROM sqlite_master WHERE TYPE = "table"; -- 테이블 목록 (SQLite).
```

# SQL 기초 문법

SQL 기초 문법: 테이블 생성 및 행 추가.

- INSERT INTO ~ VALUES로 행을 추가할 수 있다.

예).

```
INSERT INTO test VALUES ( 1, 'This', 'Right here!' );
```

 -- 1행 추가.

```
INSERT INTO test ( b, c ) VALUES ( 'That', 'Over there!' );
```

 -- "a"는 NULL 상태!

- 다른 테이블에서 SELECT로 가져온 결과로 행 추가 가능하다.

예). "item" 테이블에서 가져온 결과로 "test"테이블에 행 추가.

```
INSERT INTO test ( a, b, c ) SELECT id, name, description FROM item;
```

# SQL 기초 문법

## SQL 기초 문법: 변경.

- UPDATE + SET로 특정 행의 특정 컬럼 값을 변경할 수 있다.

예).

**UPDATE** test **SET** c = 'Extra funny.' WHERE a = 2;      -- "c" 컬럼의 값 변경.

**UPDATE** test **SET** c = **NULL** WHERE a = 2;      -- NULL 값으로 변경.

## SQL 기초 문법: 삭제.

- DELETE FROM으로 특정 행을 삭제할 수 있다.

예).

**DELETE FROM** test WHERE a = 2;

-- 조건에 맞는 행 삭제.

**DELETE FROM** test WHERE a = 1;

-- 조건에 맞는 행 삭제.

**DELETE FROM** test;

-- **주의!** 모든 행 삭제됨! 테이블은 남음.

- DROP TABLE로 특정 테이블을 삭제할 수 있다.

예).

**DROP TABLE** test;

-- 데이터 베이스에서 "test" 테이블 삭제!

# SQL 기초 문법

## SQL 기초 문법: NULL에 대해서.

- NULL은 값이 아니라 결측치를 의미하며 0 또는 '' (공백)과는 다르다.  
예).

CREATE TABLE test ( a INT, b TEXT, c TEXT );	
INSERT INTO test ( b, c ) VALUES ( 'This', 'That' );	-- "a"에는 NULL을 넣어 줌.
SELECT * FROM test;	-- NULL 포함 다 보여줌.
SELECT * FROM test WHERE a = NULL;	-- <b>작동 안함.</b>
SELECT * FROM test WHERE a <b>IS NULL</b> ;	-- 작동 함.
SELECT * FROM test WHERE a <b>IS NOT NULL</b> ;	-- 작동 함.
SELECT count(*) FROM test WHERE a IS NULL;	-- "a" 컬럼의 NULL 수 세어 줌.



## 코딩 예제 #02

→ SQLiteStudio 에서 기초 SQL Query를 실행해 본다 - Part II. ←

→ 사용: **script\_02.txt** , **scratch.db** ←

# 순서

## 1. SQL과 데이터 베이스.

1.1. 개요.

1.2. SQL 기초.

1.3. 데이터 베이스 생성과 변경.

1.4. SQL 함수.

1.5. Python에서의 접근.


## 데이터 베이스 생성과 삭제:

- 비 SQLite (MySQL 등)과 SQLite의 방법에 차이가 있다.

예). 비 SQLite (MySQL 등).

<b>CREATE DATABASE</b> myDB;	-- 새롭게 데이터 베이스 생성.
<b>USE</b> myDB;	-- 연결하여 사용.
<b>SHOW TABLES;</b>	-- 테이블 목록 출력.
<b>DROP DATABASE</b> myDB;	-- 데이터 베이스 삭제.

예). SQLite의 경우에는 SQLite Studio를 통해서 생성과 연결을 한다.

- ⇒ 신규 생성: Database → Add a Database → 녹색  버튼 클릭 후 생성.
- ⇒ 패널에 불러 들이기: Database → Add a Database → 폴더 버튼 클릭 후 선택.
- ⇒ 연결 (선택): 왼쪽 패널에서 마우스 오른쪽 클릭 후 "Connect to the database".
- ⇒ 연결 끊기: 왼쪽 패널에서 마우스 오른쪽 클릭 후 "Disconnect from the database".
- ⇒ 패널에서 삭제: 왼쪽 패널에서 마우스 오른쪽 클릭 후 "Remove the database".

### 테이블 생성과 삭제:

- 테이블은 CREATE로 생성 DROP으로 삭제.

예).

```
CREATE TABLE myTB ( a INT, b TEXT );           -- 새롭게 테이블 생성 (및 추가).  
SHOW TABLES;                                -- 테이블 목록 (비 SQLite).  
SELECT * FROM sqlite_master WHERE TYPE = "table"; -- 테이블 목록 (SQLite).  
INSERT INTO myTB VALUES ( 1, 'foo' );  
SELECT * FROM myTB;  
DROP TABLE myTB;                             -- 테이블 삭제.
```

## 테이블 정의 및 생성:

- 테이블 생성시 컬럼의 자료형을 명시할 수 있다. CHAR, VARCHAR = TEXT형.
- 타 SQL버전과는 다르게 SQLite는 자료형 준수를 강요하지는 않는다.

예).

```
CREATE TABLE test (                                -- 컬럼의 자료형 명시!
    id INTEGER,
    name VARCHAR(255),                               -- 길이 변동가능. 최대 255문자.
    address VARCHAR(255),
    city VARCHAR(255),
    state CHAR(2),                                    -- 길이 고정적.
    zip CHAR(10)
);
```

### 테이블 정의 및 생성:

- 테이블 생성시 컬럼 별 제약을 둘 수도 있다.

예).

```
CREATE TABLE test (  
    a INTEGER NOT NULL,          -- NULL 허용하지 않음.  
    b VARCHAR(255)  
);
```

```
CREATE TABLE test (  
    a INTEGER NOT NULL DEFAULT 0,  -- NULL 대신 default값 0.  
    b VARCHAR(255) UNIQUE NOT NULL -- NULL아닌 고유한 값.  
);
```

## 테이블 정의 및 생성:

- 테이블 생성시 PRIMARY KEY 지정 (고유한 값을 가진 INDEX 컬럼).
  - AUTOINCREMENT 는 1씩 자동적으로 증가하는 정수 값을 넣어준다 (SQLite).
- ⇒ 타 SQL (MySQL 등)에서는 AUTO\_INCREMENT.

예).

```
CREATE TABLE test (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    a VARCHAR(255),  
    b VARCHAR(255)  
);  
  
INSERT INTO test ( a, b ) VALUES ( 'one', 'two' );           -- id 자동 삽입!
```

### 테이블 생성 후 구조 변경:

- ALTER TABLE ~ ADD COLUMN으로 컬럼 추가.

예).

```
CREATE TABLE test (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    a VARCHAR(255),  
    b VARCHAR(255)  
);  
  
ALTER TABLE test ADD COLUMN c VARCHAR(100);
```



### 테이블 생성 후 구조 변경:

- SQLite에서는 테이블의 컬럼 전체를 삭제할 수 없다.
- 대신 다음과 같이 새로운 테이블을 만든다.

예).

**CREATE TABLE** test2 **AS** SELECT id, a, b FROM test; -- "c" 컬럼 없는 새로운 테이블.

DROP TABLE test; -- 기존 테이블 삭제.

**ALTER TABLE** test2 **RENAME TO** test; -- 새 테이블 이름 변경.

SELECT \* FROM test; -- 내용을 확인해 본다.

### SQLite의 Storage Class:

- SQLite는 타 SQL과의 호환성 유지를 위해서 다양한 “자료형”을 지원한다.
- 하지만 SQLite의 실질적인 “기초 자료형”은 다음 5가지 storage class이다.
  - 1). NULL: 결측치.
  - 2). INTEGER: 8 바이트 까지로 표현할 수 있는 정수.
  - 3). REAL: 8 바이트로 표현할 수 있는 실수.
  - 4). TEXT: 길이에 제약이 없는 문자열.
  - 5). BLOB: 입력된 자료를 원형 그대로 저장.
- 부울형 storage class는 없다.

### SQLite의 자료형: 수치형.

- 정수형: INT, INTEGER, TINYINT, SMALLINT, MEDIUMINT, BIGINT, UNSIGNED BIG INT, INT2, INT8.  
⇒ INT 또는 INTEGER는 동일한 의미이다.  
⇒ 금액 (money)를 나타내기 위해서는 정수 (INT, INTEGER)를 사용한다.
- 실수형: REAL, DOUBLE, FLOAT, NUMERIC, DECIMAL.  
⇒ DECIMAL(p,s)는 실수형을 의미하며 p = 전체 자릿수, s = 소수점 이하의 자릿수.  
⇒ NUMERIC(p,s)과 DECIMAL(p,s)은 동일한 실수형을 의미한다.  
⇒ DECIMAL은 DECIMAL(10,0)과 같다.

SQLite의 자료형: 문자열 관련 자료형.

- 문자열 자료 나타내기: TEXT, CHAR, VARCHAR.
  - ⇒ TEXT에는 길이 제약이 없다.
  - ⇒ CHAR(n)은 n개의 문자 길이로 고정된 문자열을 의미한다.
  - ⇒ VARCHAR(n)은 최대 n개의 문자 까지 유동적인 길이의 문자열을 의미한다.

SQLite의 자료형: 날짜와 시간 관련 자료형.

- 날짜와 시간 나타내기: DATE, DATETIME.
  - ⇒ 날짜: 'YYYY-MM-DD' 형태의 문자열.
  - ⇒ 시간: 'HH:MM:SS' 형태의 문자열.
  - ⇒ 날짜와 시간: 'YYYY-MM-DD HH:MM:SS' 형태의 문자열.

## SQLite의 자료형: 부울형 (BOOLEAN).

- SQLite에서 부울 (BOOLEAN) 자료 나타내기.
  - ⇒ 실질적으로는 INTEGER이며 0 (False)과 1 (True)로 매핑된다.
  - ⇒ 다음과 같이 0,1 이외의 값은 reject 되도록 정의해 둔다.

예).

```
CREATE TABLE myTB (  
    aColumn BOOLEAN NOT NULL CHECK (aColumn IN (0,1))  
);
```

```
INSERT INTO myTB VALUES (0);           -- OK.  
INSERT INTO myTB VALUES (1);           -- OK.  
INSERT INTO myTB VALUES (2);           -- NOT OK!
```

### 테이블 사이의 관계 설정:

- FOREIGN KEY를 사용하여 테이블 사이의 관계 설정.
- 일종의 제약 (constraint)으로서 그 역할을 한다.

예). 먼저 "Franchisee" 테이블을 만들어 둔다.

```
CREATE TABLE Franchisee (  
    FranchiseeID    INTEGER PRIMARY KEY,  
    FranchiseeName TEXT NOT NULL  
);  
  
INSERT INTO Franchisee VALUES (111,"John");    -- 데이터를 입력해 둔다.  
INSERT INTO Franchisee VALUES (112,"Paul");    -- 데이터를 입력해 둔다.  
INSERT INTO Franchisee VALUES (113,"David");    -- 데이터를 입력해 둔다.
```

### 테이블 사이의 관계 설정:

- FOREIGN KEY를 사용하여 테이블 사이의 관계 설정.
- 일종의 제약 (constraint)으로서 그 역할을 한다.

예). 다음은 "Store" 테이블을 만들어 둔다.

```
CREATE TABLE Store (  
    StoreID    INTEGER PRIMARY KEY,  
    StoreName  TEXT NOT NULL,  
    FranchiseeID INTEGER NOT NULL,  
    FOREIGN KEY(FranchiseeID) REFERENCES Franchisee(FranchiseeID)  
); -- Store 테이블의 FranchiseeID와 Franchisee 테이블의 FranchiseeID 연결.
```



### 테이블 사이의 관계 설정:

- FOREIGN KEY를 사용하여 테이블 사이의 관계 설정.
- 일종의 제약 (constraint)으로서 그 역할을 한다.

예). 이젠 "Store" 테이블에 값을 넣어서 테스트 해본다.

```
INSERT INTO Store VALUES (1,"711 Gangnam", 111);      -- OK.
INSERT INTO Store VALUES (2,"711 Munjeong", 111);      -- OK.
INSERT INTO Store VALUES (3,"711 Suyu", 113);          -- OK.
INSERT INTO Store VALUES (4,"711 Youido", 114);         -- NOT OK!
DROP TABLE Franchisee;                                  -- 먼저 삭제할 수 없다!
DROP TABLE Store;                                       -- 먼저 삭제 가능.
```

### 테이블 JOIN:

- 왼쪽 테이블과 오른쪽 테이블의 공통 값으로 INNER JOIN.

예).

```
SELECT a.name, b.item_id, b.quantity, b.price FROM customer AS a  
           INNER JOIN sale AS b ON a.id = b.customer_id;
```

- 왼쪽 테이블을 우선시하는 LEFT JOIN.

예).

```
SELECT a.name, b.item_id, b.quantity, b.price FROM customer AS a  
           LEFT JOIN sale AS b ON a.id = b.customer_id;
```

- SQLite는 RIGHT JOIN과 FULL OUTER JOIN를 지원하지 않는다.

## 데이터 테이블의 결합

TABLE A

이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

TABLE B

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

## 데이터 테이블의 결합

TABLE A

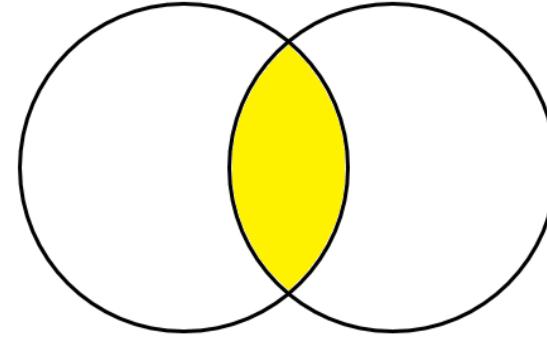
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

TABLE B

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

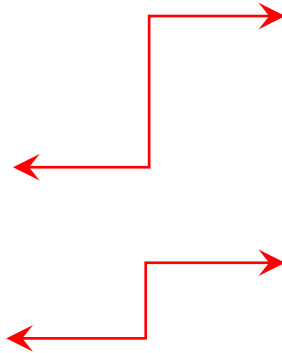
## 데이터 테이블의 결합 : Inner Join

조건: A.이름 = B.직원이름



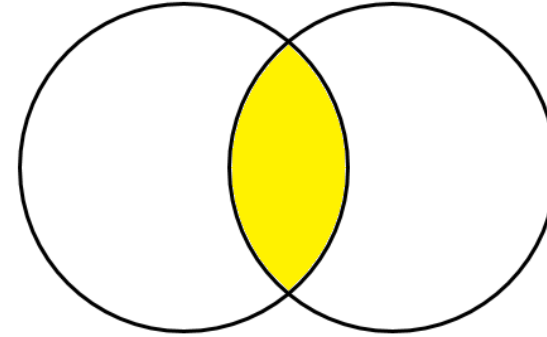
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000



## 데이터 테이블의 결합 : Inner Join

조건: A.이름 = B.직원이름

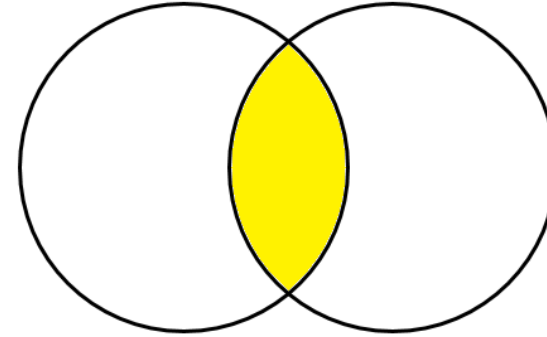


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

## 데이터 테이블의 결합 : Inner Join

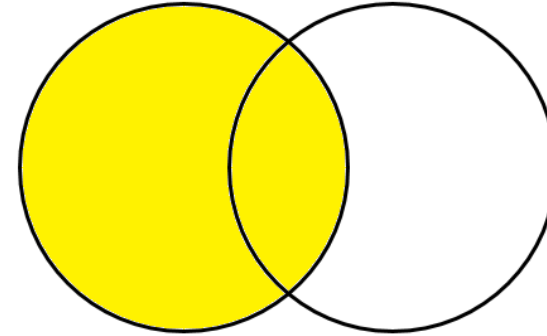
조건: A.이름 = B.직원이름



이름	성별	나이	직원이름	직책	시급
홍길동	남	28	홍길동	직원	9000
사임당	여	30	사임당	과장	20000

## 데이터 테이블의 결합 : Left Join

조건: A.이름 = B.직원이름



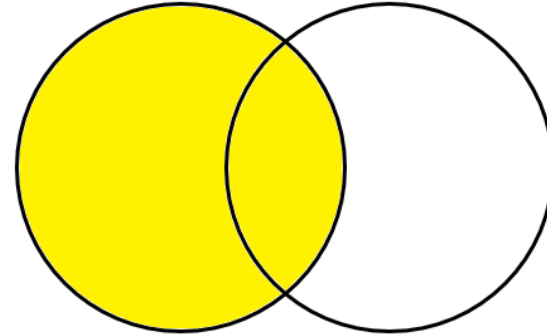
이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000



## 데이터 테이블의 결합 : Left Join

조건: A.이름 = B.직원이름

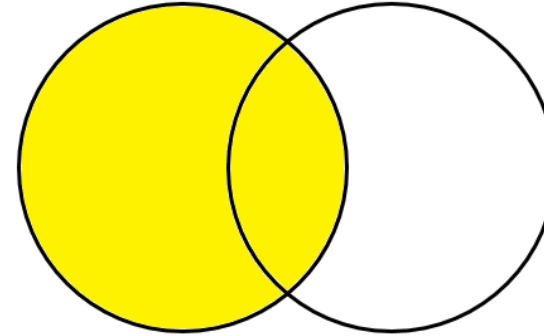


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

## 데이터 테이블의 결합 : Left Join

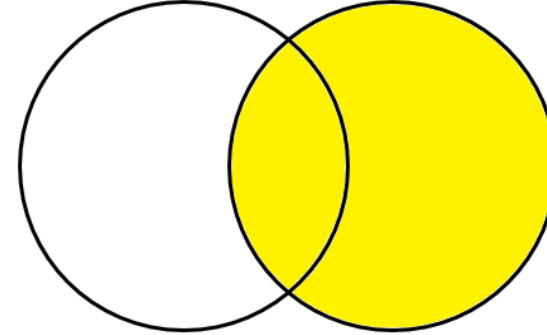
조건: A.이름 = B.직원이름



이름	성별	나이	직원이름	직책	시급
김철수	남	23			
이민수	남	31			
홍길동	남	28	홍길동	직원	9000
임꺽정	남	36			
사임당	여	30	사임당	과장	20000

## 데이터 테이블의 결합 : Right Join

조건: A.이름 = B.직원이름

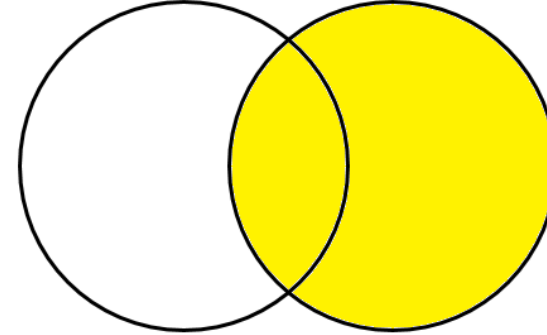


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

## 데이터 테이블의 결합 : Right Join

조건: A.이름 = B.직원이름

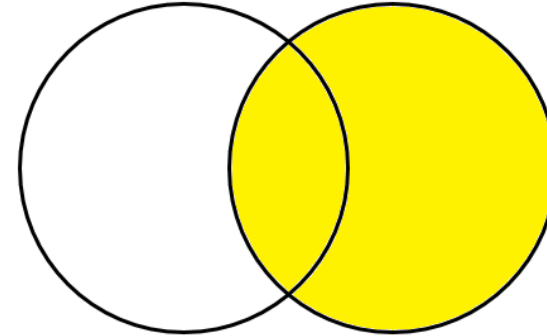


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

## 데이터 테이블의 결합 : **Right Join**

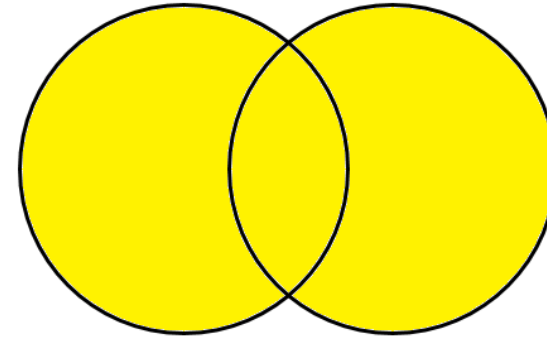
조건: A.이름 = B.직원이름



이름	성별	나이	직원이름	직책	시급
홍길동	남	28	홍길동	직원	9000
			이세종	직원	9500
			대장금	과장	19000
사임당	여	30	사임당	과장	20000

## 데이터 테이블의 결합 : Full Outer Join

조건: A.이름 = B.직원이름

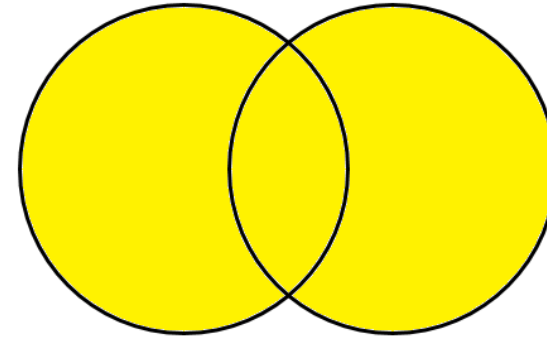


이름	성별	나이
김철수	남	23
이민수	남	31
홍길동	남	28
임꺽정	남	36
사임당	여	30

직원이름	직책	시급
홍길동	직원	9000
이세종	직원	9500
대장금	과장	19000
사임당	과장	20000

## 데이터 테이블의 결합 : Full Outer Join

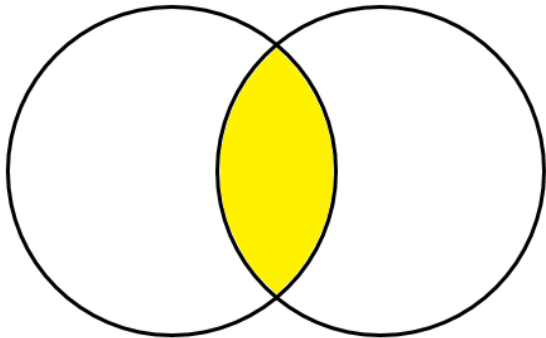
조건: A.이름 = B.직원이름



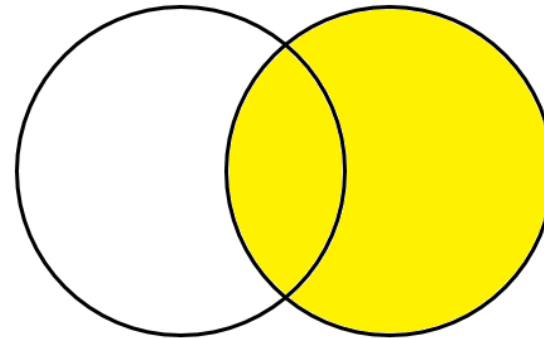
이름	성별	나이	직원이름	직책	시급
김철수	남	23			
이민수	남	31			
홍길동	남	28	홍길동	직원	9000
임꺽정	남	36			
사임당	여	30	사임당	과장	20000
			이세종	직원	9500
			대장금	과장	19000

## 데이터 테이블의 결합 : 정리

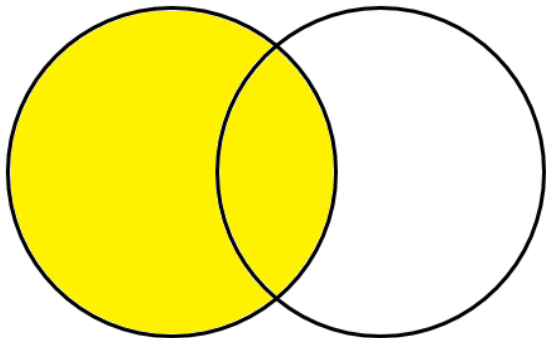
**Inner Join**



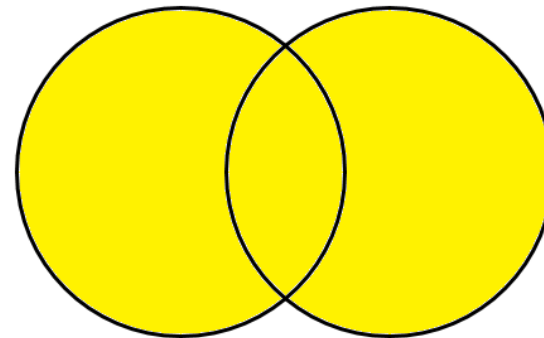
**Right Join**



**Left Join**



**Full Outer Join**





## VIEW 생성과 삭제:

- 출력 방법을 VIEW로 저장해 둘 수 있다.

예).

```
CREATE VIEW myView1 AS SELECT Name AS Country, Population / 1000 AS "Pop  
(1000s)" FROM Country;
```

```
CREATE VIEW myView2 AS SELECT Name, Continent, Population FROM Country  
WHERE Population < 100000 ORDER BY Population DESC;
```

```
CREATE VIEW myView3 AS SELECT Name, Continent FROM Country WHERE Name  
LIKE '_a%';
```

### VIEW 생성과 삭제:

- 정의된 VIEW를 테이블과도 같이 불러서 사용할 수 있다.

예).

```
SELECT * FROM myView1;
```

```
SELECT * FROM myView2;
```

```
SELECT Name as CountryName FROM myView3;
```

- 다음과 같이 VIEW를 삭제할 수 있다.

예).

```
DROP VIEW myView1;
```

```
DROP VIEW myView2;
```

```
DROP VIEW myView3;
```

## 코딩 예제 #03

→ SQLiteStudio 에서 데이터 베이스의 생성과 접속을 실행해 본다. ←

→ 사용: **script\_03.txt** , **scratch.db** ←

# 순서

## 1. SQL과 데이터 베이스.

1.1. 개요.

1.2. SQL 기초.

1.3. 데이터 베이스 생성과 변경.

1.4. SQL 함수.

1.5. Python에서의 접근.

## 문자열 함수와 연산자:

⇒  $x \parallel y$  : 문자열  $x$ 와  $y$  연결.

⇒ `LENGTH(x)`: 문자열  $x$ 의 길이.

⇒ `SUBSTR(x, n, m)`:  $x$ = 문자열,  $n$  = 시작 위치,  $m$  = 반환 문자 개수.

⇒ `TRIM(x)`: 문자열  $x$ 에서 스페이스 제거.

⇒ `TRIM(x, y)`: 문자열  $x$ 에서 문자  $y$  제거.

⇒ `LTRIM()`, `RTRIM()`: `TRIM()`과 유사. 단, Left 또는 Right에 국한되게 문자 제거.

⇒ `UPPER(x)`, `LOWER(x)`: 문자열  $x$ 의 대문자화 또는 소문자화.

⇒ `REPLACE(x, y, z)`: 문자열  $x$ 에서 패턴  $y$ 를 패턴  $z$ 로 대체.

## SQL 함수

집계 (aggregation)하지 않는 계산용 함수와 연산자:

⇒ 사칙연산: +, -, \*, /.

⇒  $x \% y$ : 나머지.

⇒ ABS(x): 절대값.

⇒ ROUND(x, n): x 소수점 이하 n 자리까지.

# SQL 함수

## 날짜와 시간 관련 함수:

⇒ DATE('now'): 현재 날짜.

⇒ TIME('now'): 현재 시각.

⇒ DATETIME('now'): 날짜와 시각.

⇒ STRFTIME(<FORMAT>, 'now')

FORMAT	의미
'%d'	일
'%m'	월 (1~12)
'%Y'	연도
'%w'	요일 (0~6, 0=일요일)

## 집계 (aggregation) 함수:

- 집계 함수와 순위 함수를 통 털어서 윈도우 함수 (Window Function)라 부른다.

⇒ AVG(X): 그룹별 X 컬럼의 평균을 구해준다.

⇒ SUM(X): 그룹별 X 컬럼의 합을 구해준다.

⇒ COUNT(X): 그룹별 X 컬럼의 카운트를 세어준다.

⇒ MIN(X), MAX(X): 그룹별 X 컬럼의 최소값, 최대값 계산.



# SQL 함수

집계 (aggregation) 함수:

예).

SELECT **COUNT(\*)** FROM Country; -- 행수 카운트.

SELECT **COUNT(Population)** FROM Country; -- NULL이 아닌 Population 값의 개수.

SELECT COUNT(\*) FROM Country **GROUP BY** Continent; -- 대륙별 국가의 수.

SELECT COUNT(**DISTINCT** Continent) FROM Country ; -- 고유한 대륙의 개수.

SELECT **AVG(Population)** FROM Country WHERE Region = 'Western Europe'; -- 서유럽 지역의 국가의 인구 평균.

SELECT **MIN(Population), MAX(Population)** FROM Country WHERE Region = 'Western Europe'; -- 서유럽 지역의 국가 인구 최소값과 최대값.

## 코딩 예제 #04

→ SQLiteStudio 에서 SQL 함수를 실행해 본다. ←

→ 사용: **script\_04.txt** , **world.db** ←

# 순서

## 1. SQL과 데이터 베이스.

1.1. 개요.

1.2. SQL 기초.

1.3. 데이터 베이스 생성과 변경.

1.4. SQL 함수.

1.5. Python에서의 접근.

## Python 에서의 접근:

- Python에는 SQLite가 이미 설치되어 있다 (Anaconda로 설치된 경우).
- 커서 (cursor)객체를 만들고 SQL 커맨드를 쉽게 처리할 수 있다.

예).

```
import sqlite3                # 라이브러리를 가져온다.
conn = sqlite3.connect('customer.db') # 신규 생성 또는 접속.
cur = conn.cursor()           # 커서 객체.
cur.execute("""CREATE TABLE customers (
    first_name text, last_name text,
    email text)""")            # Multi-line.
conn.commit()                 # 확인.
conn.close()                  # 데이터 베이스 연결 종료.
```

## Python 에서의 접근:

- "?"를 사용해서 편리하게 외부 입력을 받을 수 있다.

예).

```
import sqlite3                # 라이브러리를 가져온다.
conn = sqlite3.connect('scratch.db')  # 접속.
cur = conn.cursor()           # 커서 객체.
sql = "INSERT INTO test VALUES (?, ?, ?);" # ?를 placeholder로 사용.
x = [2, "This", "That"]        # 삽입될 행 (list 또는 tuple).
cur.execute(sql, x)            # 실행.
conn.commit()                 # 확인.
conn.close()                  # 데이터 베이스 연결 종료.
```

## 코딩 예제 #0101

---

→ Jupyter 노트북을 열어서 SQLite query를 실행해 본다. ←

→ 사용: [ex\\_0101.ipynb](#) ←

## 코딩 예제 #0102

---

→ Pandas 데이터 프레임과 SQL 테이블. ←

→ 사용: [ex\\_0102.ipynb](#) ←

끝

문의:

sychang1@gmail.com