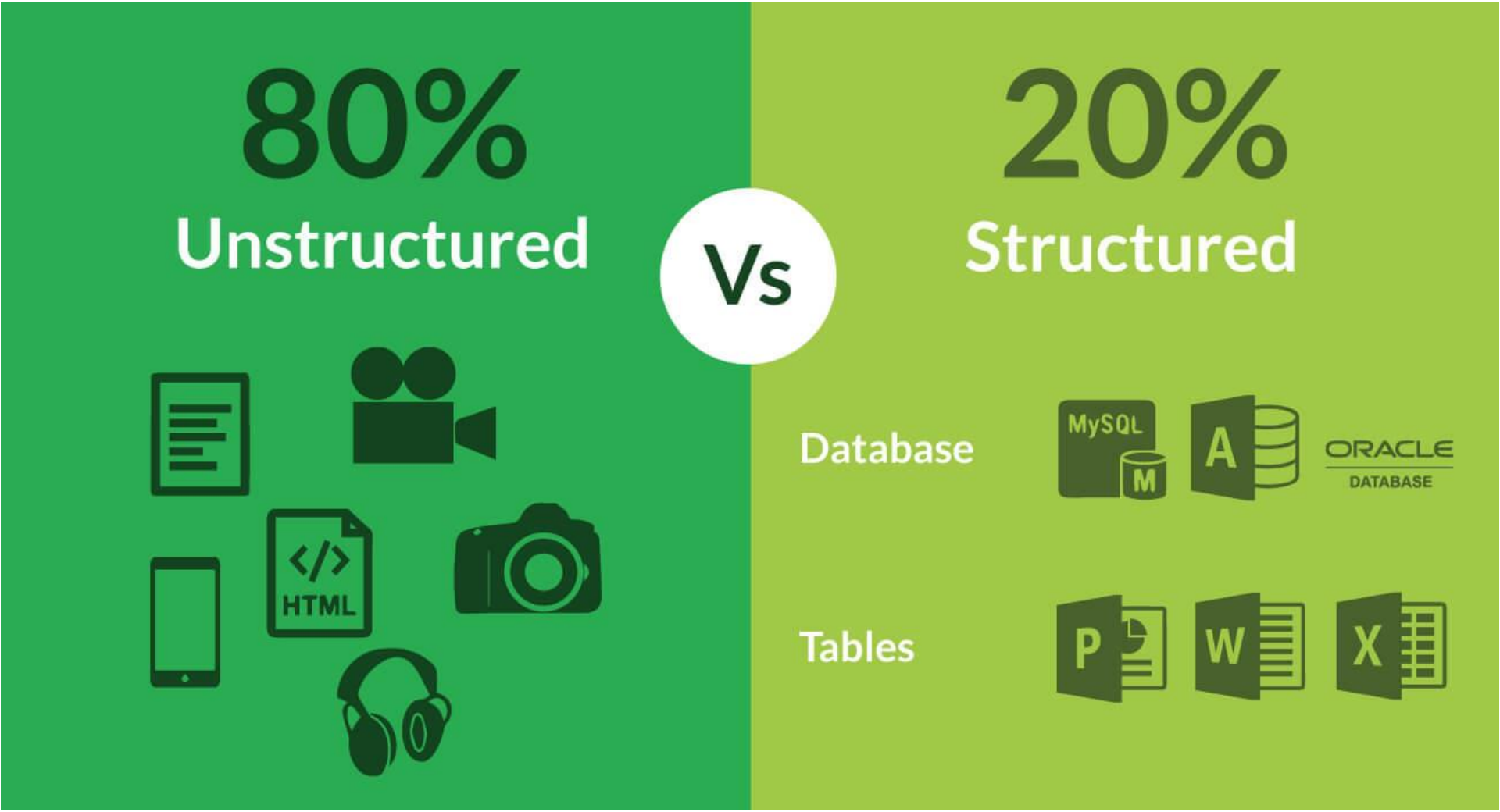
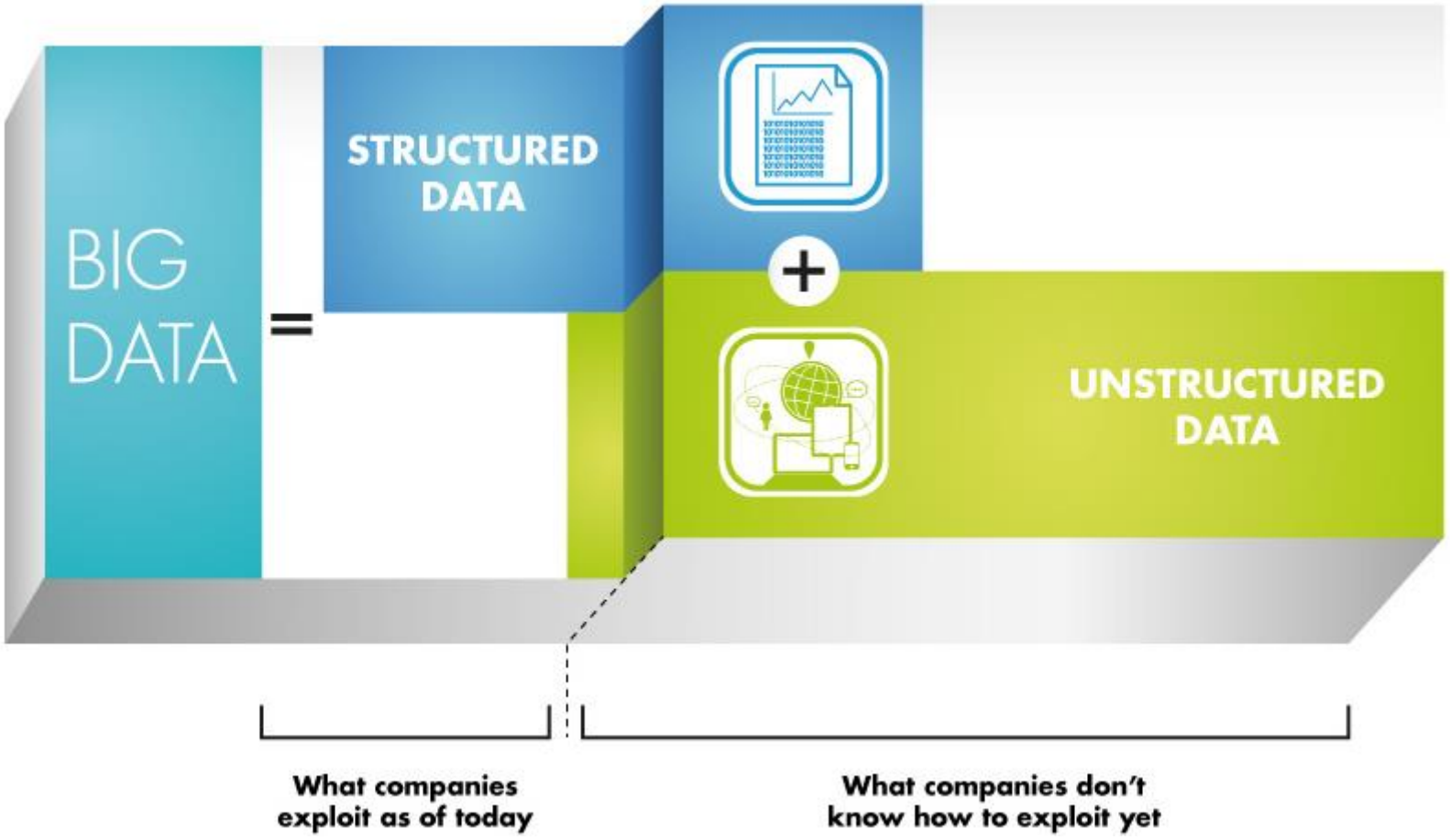
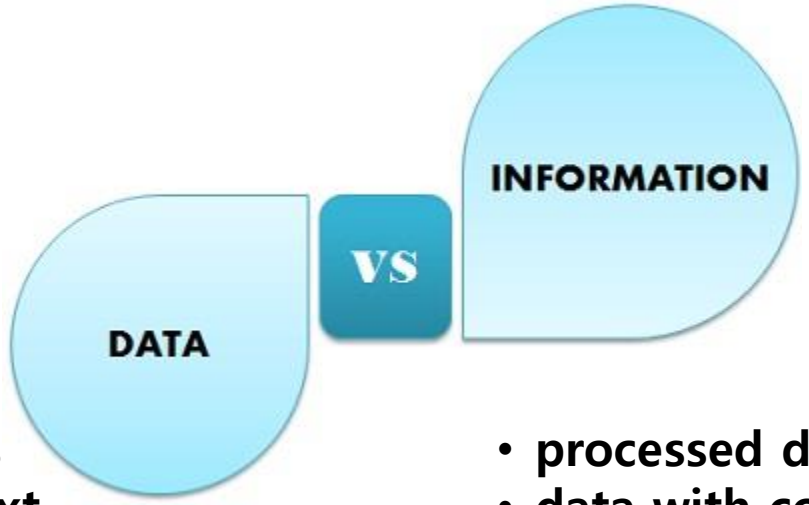
1. DBMS

1.1. 데이터베이스 개요


80%
Unstructured
Vs
20%
Structured
HTML
Database
MySQL
ORACLE
DATABASE
Tables


BIG DATA
=
STRUCTURED DATA
+
UNSTRUCTURED DATA
What companies exploit as of today
What companies don't know how to exploit yet



- **raw facts**
- **no context**
- **just numbers and text**

business transaction, sales, payments, etc.

- **processed data**
- **data with context**
- **value-added data**

customer balance, sales trends, etc.

➡ ***make decisions***

Database

- Collection of data organized in a manner that allows access, retrieval, and use of that data

Data

- Collection of unprocessed items
  - Text
  - Numbers
  - Images
  - Audio
  - Video

Information

- Processed data
  - Documents
  - Audio
  - Images
  - Video

structured

■ What is Database?

○ 데이터베이스

- a collection of interrelated data
- a set of integrated, stored, shared and operational data

■ 데이터베이스 조건

○ 통합(Integrated)

- 동일한 데이터가 중복되지 않도록 구성
- 최소한의 중복 또는 통제된 중복 만 허용

○ 저장(Stored)

- 컴퓨터로 접근 가능한 물리적 저장 매체 저장

○ 공유(Shared)

- 공동으로 소유하고 유지하며 이용하는 데이터

○ 운영(Operational)

- 존재 목적이나 기능 수행에 꼭 필요한 데이터 집합

■ 데이터베이스 특징

○ 실시간 접근성(Real-time Accessibility)
- ➢ 데이터들 간의 밀접한 관계로 연결
- ➢ 중복 데이터를 배제하도록 지양
- ➢ 사용자의 어떤 요구에도 즉각 응답

○ 계속적인 변화(Continuous evolution)
- ➢ 현실 세계의 상태를 정확히 반영
- ➢ 동적으로 삽입/삭제/수정하여 현재의 데이터 유지

○ 동시 공유 가능(Concurrent sharing)
- ➢ 여러 사용자들이 동시에 이용
- ➢ 같은 시간에 같은 데이터에 접근하여 이용

○ 내용에 의한 참조 가능(Content reference)
- ➢ 저장된 주소나 위치에 의해서 참조하지 않고
- ➢ 사용자가 요구하는 데이터의 내용/값에 따라 참조

■ What is DBMS?

○ DBMS

- ➢ a collection of programs
- ➢ manage the database structure
- ➢ controls access to the data stored in the database

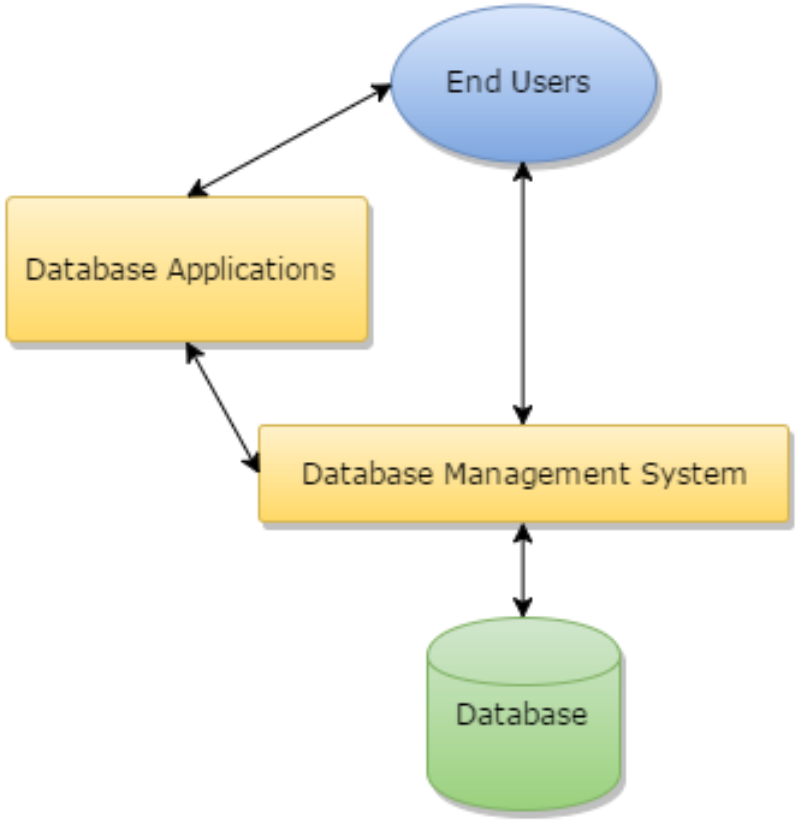
■ DBMS의 역할

○ create databases

○ insert, update and delete data

○ sort and query data

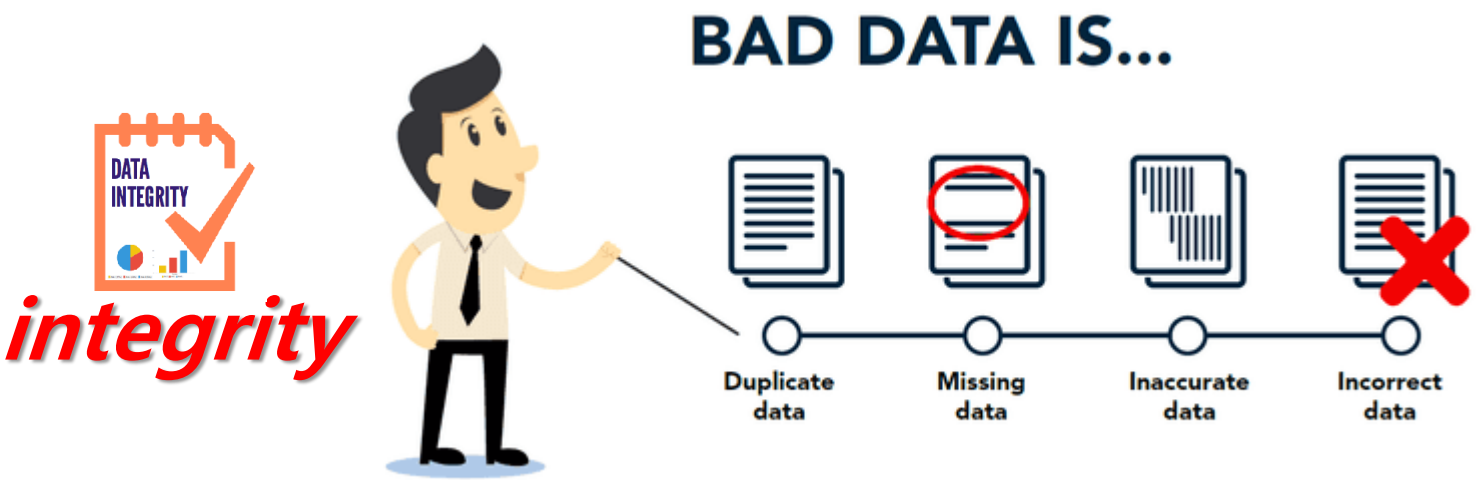
○ create form and report



Components of a Database Management System

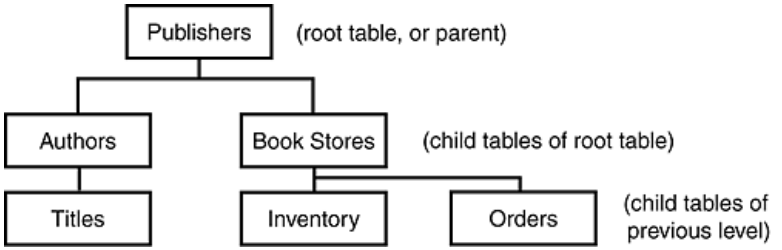
■ DBMS 역할과 장점

- ○ Improved data sharing
- ○ Improved data security
- ○ Minimized data inconsistency
- ○ Improved data access
- ○ Improved decision making
- ○ Increased end-user productivity
- ○ Reduce application development time

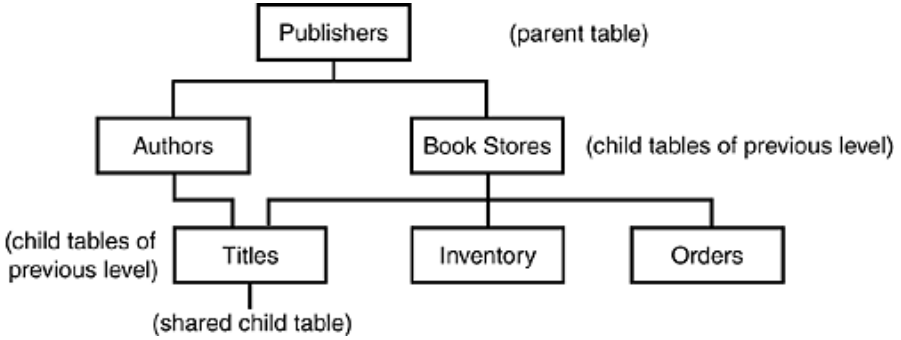


■ DBMS 종류

○ 계층형 모델(Hierarchical Database Model)
- ➢ 트리 형태로 표현 (1:N)
- ➢ 개체를 노드, 개체 집합들 사이의 관계를 링크로 표현

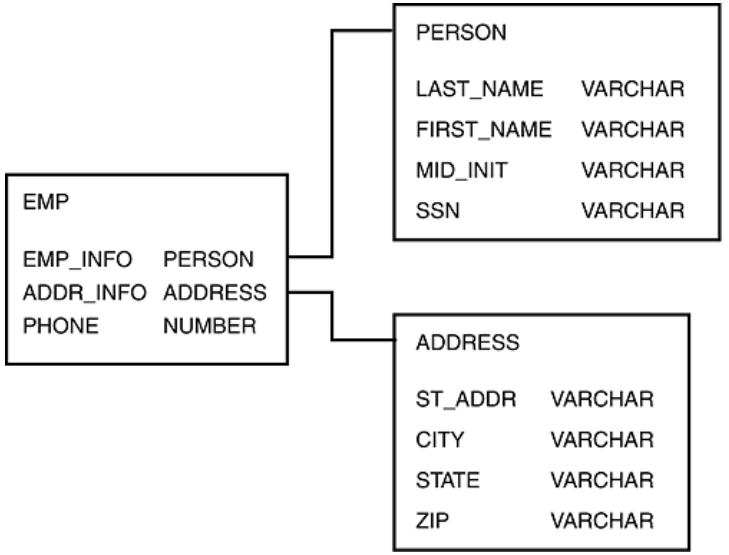


○ 네트워크형 모델(Network Database Model)
- ➢ 그래프 형태로 표현 (N:M)
- ➢ 각 개체가 여러 관계를 가질 수 있음



○ 관계 데이터 모델(Relational Database Model)
- ➢ 개체를 테이블, 개체 집합들 사이의 관계를 공통 속성으로 연결

○ 객체 관계 모델(Object Relational Database Model)
- ➢ 속성-관계
- ➢ 개체-관계



1.2. RDBMS

■ Why use an RDBMS?

○ 데이터 안정성(Data Safety)
- ➢ data is immune to program crashes

○ 동시 접근성(Concurrent Access)
- ➢ atomic updates via transactions

○ 장애 허용성(Fault Tolerance)
- ➢ replicated DBS for instant failover on machine/disk crashes

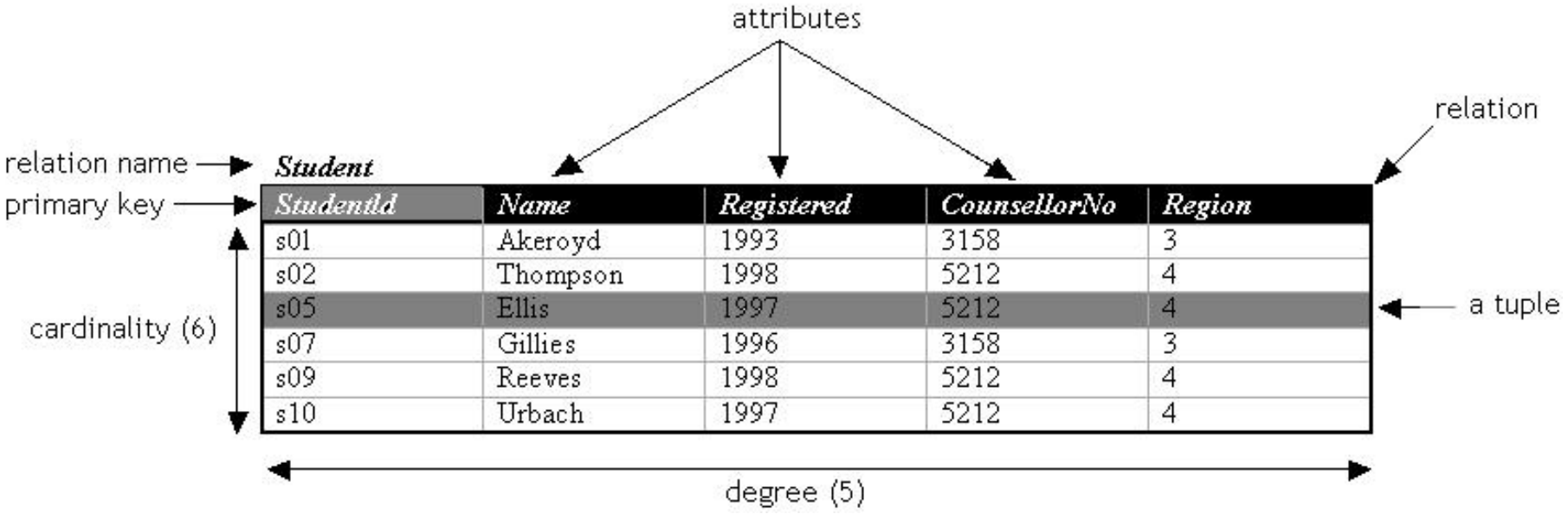
○ 데이터 무결성(Data Integrity)
- ➢ aids to keep data meaningful

○ 데이터 확장성(Scalability)
- ➢ can handle small/large quantities of data in a uniform manner

○ 데이터 보고서(Reporting)
- ➢ easy to write SQL programs to generate arbitrary reports

■ RDBMS Concepts

- ○ Relation / Table
- ○ Tuple / Row or Record
- ○ Attribute / Column or Field
- ○ Cardinality / Number of Rows
- ○ Degree / Number of Columns
- ○ Domain / Pool or legal values
- ○ Primary Key / Unique identifier



Student

stu_no	stu_name	stu_ename	tel
20001001	김유신	Kim Yoo-Shin	061)685-7818
20001015	박도준	Park Do-Jun	061)744-6126
20001021	이상길	Lee Sang-Gil	031)691-5423
20041002	김유미	Kim Yoo-Mi	061)763-1439
20041007	정인정	Jeung Yin-Jeung	061)723-1078
20041033	연개소문	Yean Gae-So-Moon	061)642-9304
20061011	박정인	Park Jung-In	02)652-2439
20061014	고혜진	Ko Hea-Jin	061)781-5135
20061048	김영호	Kim Young-Ho	062)548-8881
20071001	장수인	Jang Soo-In	061)791-1236
20071010	홍길동	Hong Gil-Dong	061)642-4034
20071022	이순신	Lee Sun-Shin	061)745-7667
20071300	유하나	Yoo Ha-Na	061)651-5992
20071307	김문영	Kim Moon-Young	061)745-5485

■ 개체 관계 모델(Entity-Relationship Model)

○ 개체(Entity)
- ➢ 실 세계에 존재하는 분리된 실체 하나를 표현, 일반적으로 명사 하나에 해당

○ 관계(Relationship)
- ➢ 개체들 사이에 존재하는 연관이나 연결, 일반적으로 동사에 해당
- ➢ 최소 대응수(minimum cardinality)와 최대 대응수(maximum cardinality)로 구성

○ 속성(Attribute)
- ➢ 개체의 성질, 분류, 식별, 수량, 상태 등을 나타내는 세부 항목
- ➢ 관계 또한 속성을 보유할 수 있음

○ 기본키(Primary Key)
- ➢ 모든 개체를 고유하게 식별할 수 있는 속성

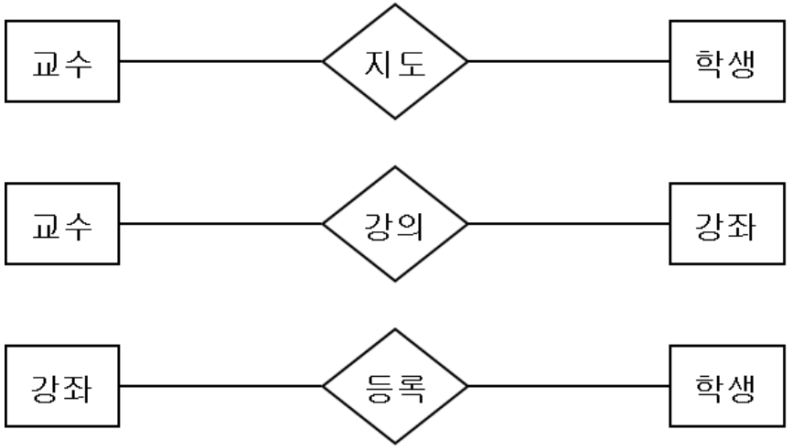
■ 예제: 교수-학생 ER모델

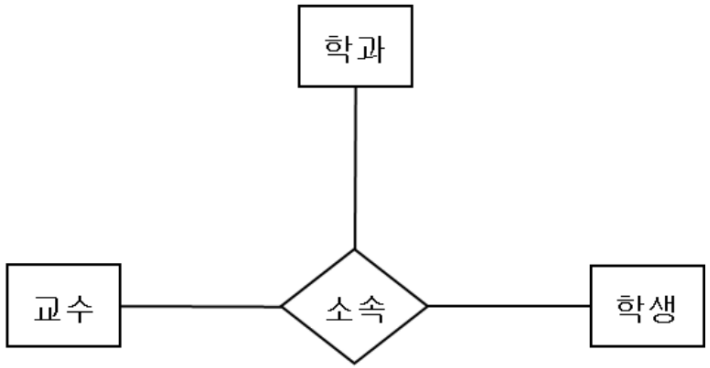
○ 이항 관계

- 교수는 학생을 지도한다.
- 교수는 강좌를 강의한다.
- 강좌는 학생들이 등록한다.

○ 삼항 관계

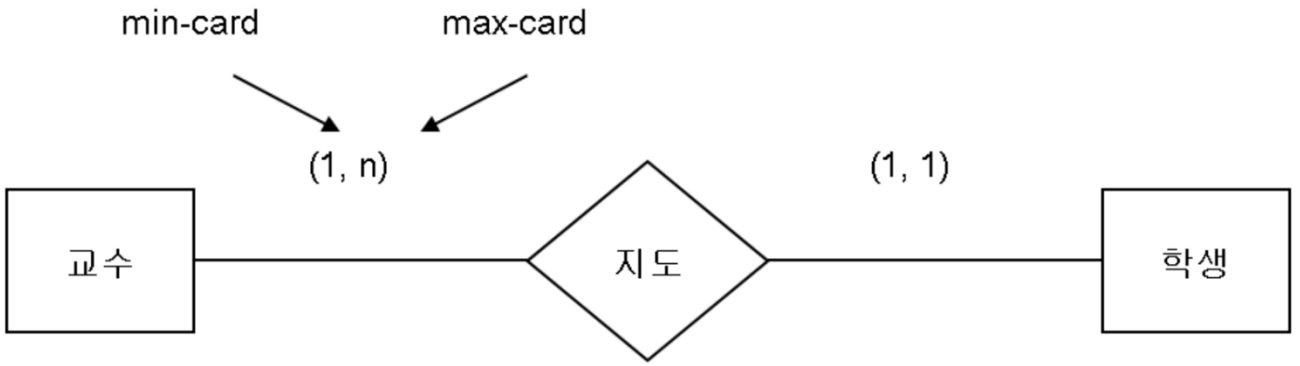
- 교수와 학생은 특정 학과에 소속된다.





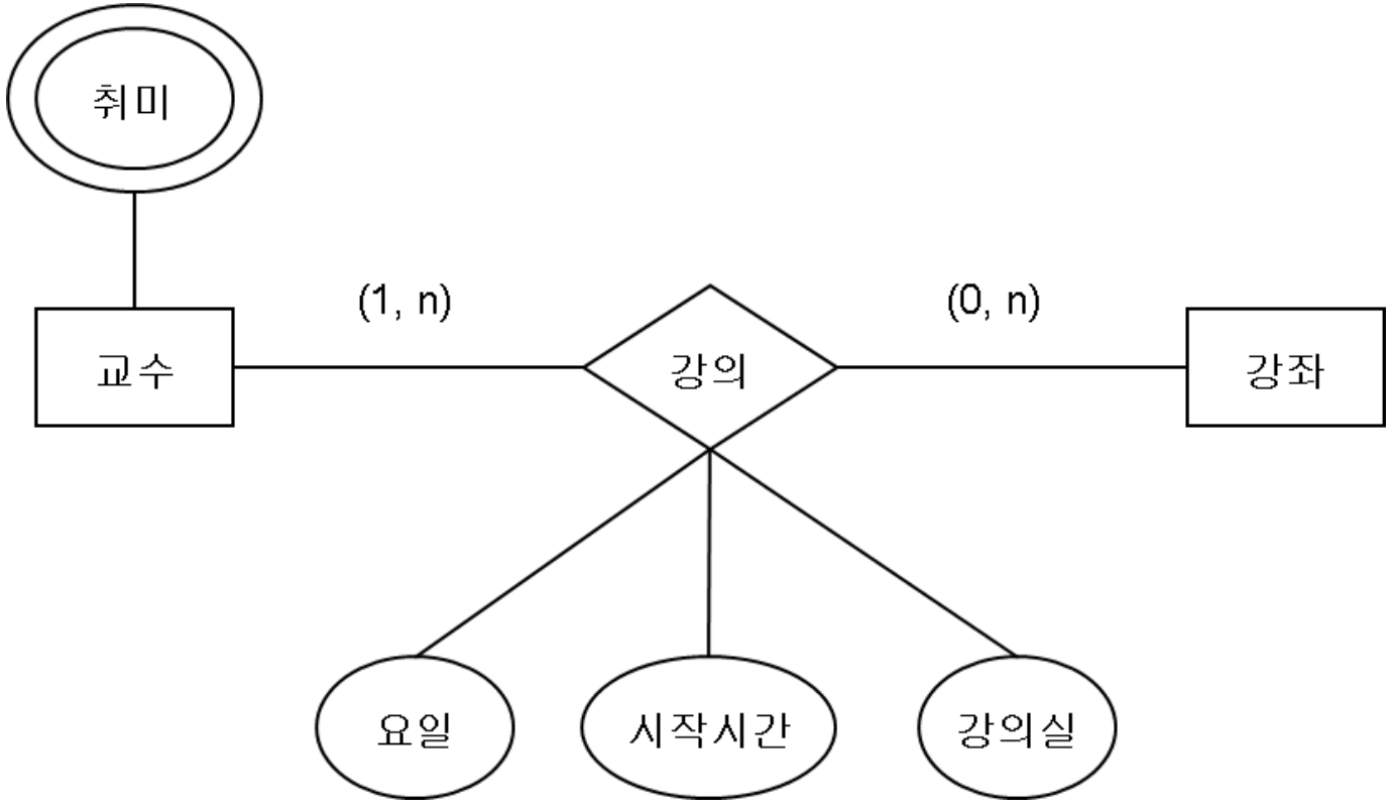
○ 관계 – Mapping Cardinality

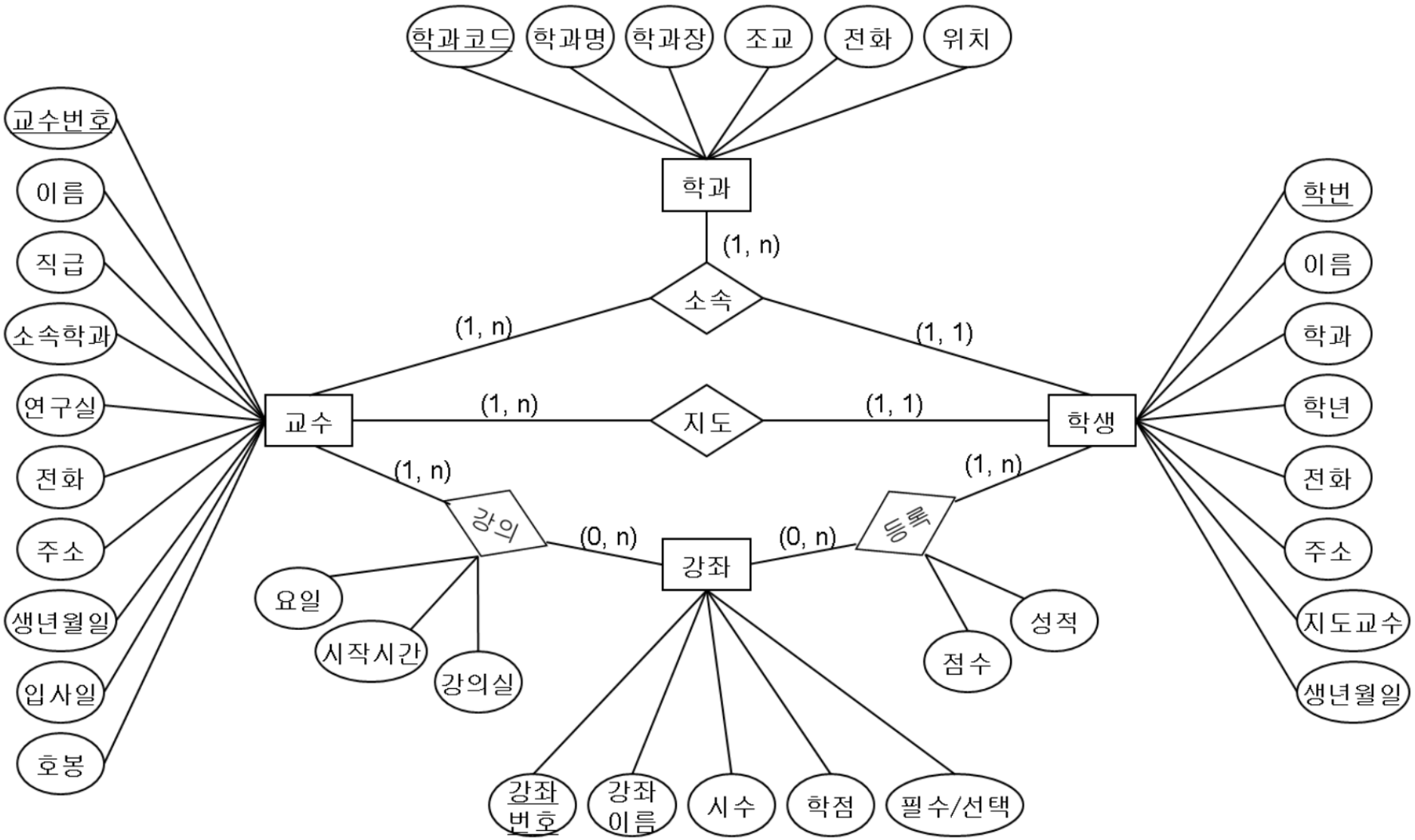
- 조건 1 : 교수는 꼭 학생에 대한 지도를 해야 한다.
  - min-card(교수, 지도) = 1
- 조건 2 : 교수는 여러 명의 학생을 지도할 수 있다.
  - max-card(교수, 지도) = n
- 조건 3 : 학생은 꼭 교수에게 지도를 받아야 한다.
  - min-card(교수, 지도) = 1
- 조건 4 : 학생은 여러 명의 교수에게 지도를 받을 수 없다.
  - max-card(교수, 지도) = 1



○ 속성

- 교수는 어려 개의 취미를 가질 수 있다.
- 강의는 요일, 시작시간, 강의실의 속성을 갖는다.



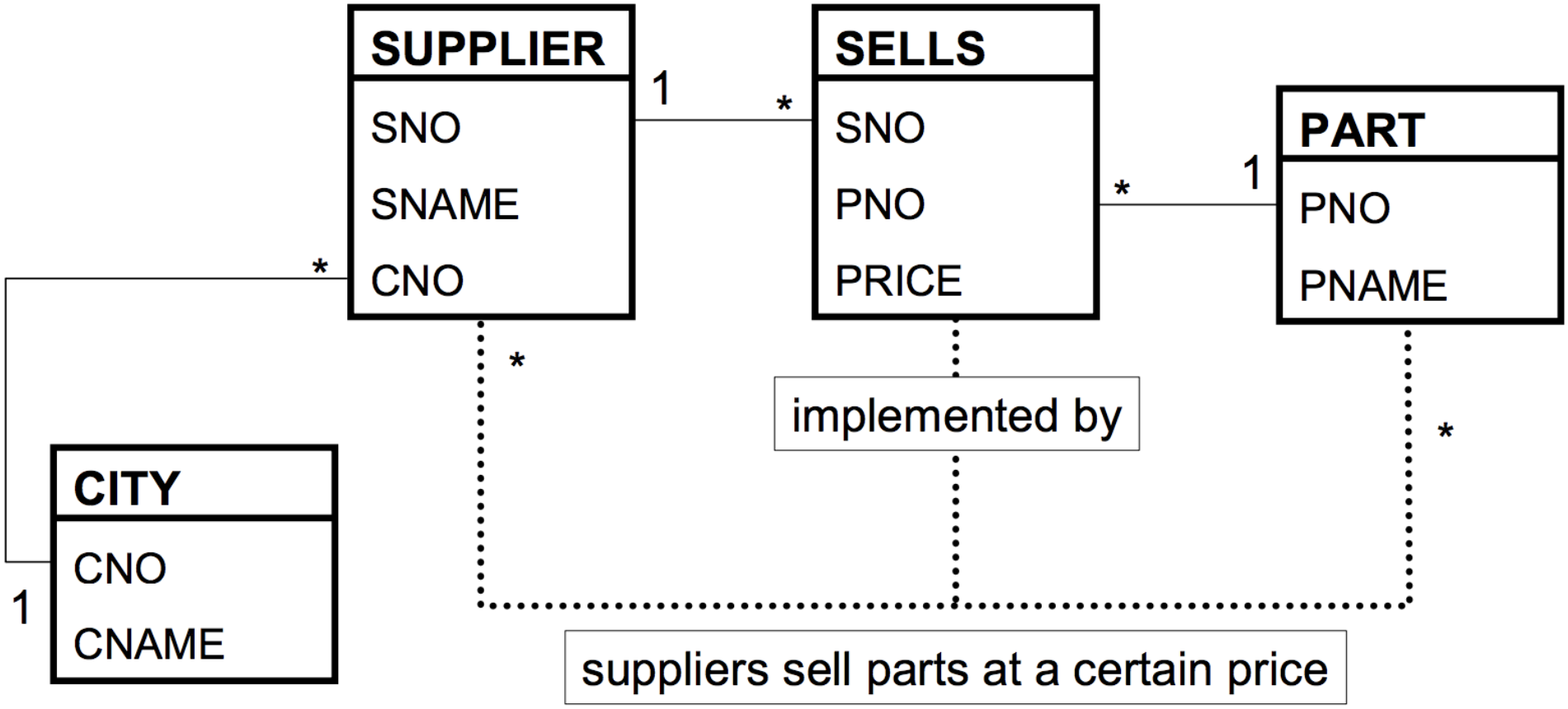

학과코드
학과명
학과장
조교
전화
위치
교수번호
이름
직급
소속학과
연구실
전화
주소
생년월일
입사일
호봉
학과
(1, n)
소속
(1, n)
(1, 1)
교수
(1, n)
지도
(1, 1)
학생
학번
이름
학과
학년
전화
주소
지도교수
생년월일
(1, n)
강의
(0, n)
강좌
(0, n)
등록
(1, n)
요일
시작시간
강의실
성적
점수
강좌번호
강좌이름
시수
학점
필수/선택

■ 예제: 공급자와 부품

- ○ A DB with four tables (*SUPPLIER*, *CITY*, *PART*, *SELLS*)
- ○ *SUPPLIER* has attributes: number (SNO), the name (SNAME) and the city number (CNO)
- ○ *CITY* has attributes number (CNO) and the city name (CNAME)
- ○ *PART* has attributes: number (PNO) the name (PNAME) and the price (PRICE)
- ○ *SELLS* has attributes: part (PNO) and supplier (SNO). SELLS connects SUPPLIER and PART

■ Find Entity / Relationship

■ ER 모델


SUPPLIER
SNO
SNAME
CNO
SELLS
SNO
PNO
PRICE
PART
PNO
PNAME
CITY
CNO
CNAME
1
*
1
*
1
*
*
*
1
*
implemented by
suppliers sell parts at a certain price

■ Entity – City

CITY:

CNO	CNAME
1	London
2	Paris
3	Rome
4	Vienna

○ Row (1, London) in CITY represents a distinct city, London, with city number 1

■ Entity – Supplier

SUPPLIER:

SNO	SNAME	CNO
1	Smith	1
2	Jones	2
3	Adams	1
4	Blake	3

○ Row (1,Smith, 1) in SUPPLIER represents a supplier with supplier number 1 whose name is Smith and who is based in the city numbered 1 (London)

■ Entity – Part

PART:

PNO	PNAME
1	Screw
2	Nut
3	Bolt
4	Cam

○ Row (2,Nut) in PART represents a part with part number 2, and part name Nut

■ Entity – Sells

SELLS:

SNO	PNO	PRICE
1	1	10
1	2	8
2	4	38
3	1	11
3	3	6
4	2	7
4	3	4
4	4	45

○ Row (1,2,8) in SELLS represents the relationship of supplier 1 (Smith) selling part 2 (Nut) for 8 cents

1.3. SQL

■ SQL

○ Structured Query Language

- ➢ RDBMS의 데이터를 관리하기 만들어진 언어 (대부분 ISO 표준을 따름)
- ➢ 자료의 검색과 관리, 데이터베이스 스키마 생성과 수정, 데이터베이스 객체 접근 조정 관리 등을 고안

■ SQL 명령

○ 데이터 정의 언어 DDL(Data Definition Language)

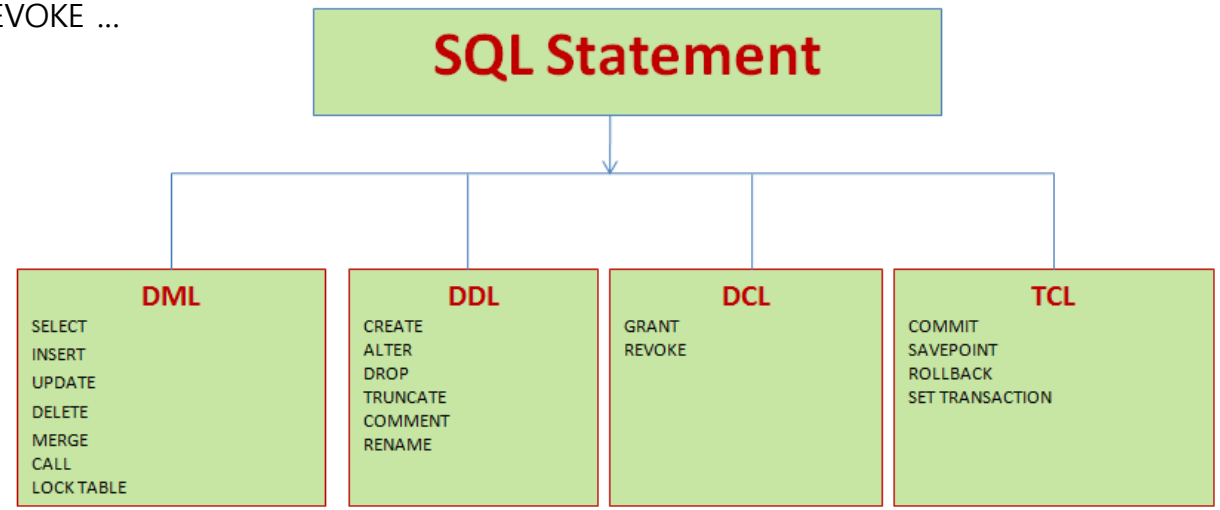
- ➢ CREATE, DROP, ALTER, TRUNCATE ...

○ 데이터 조작 언어 DML(Data Manipulation Language)

- ➢ INSERT, UPDATE, DELETE, SELECT ...

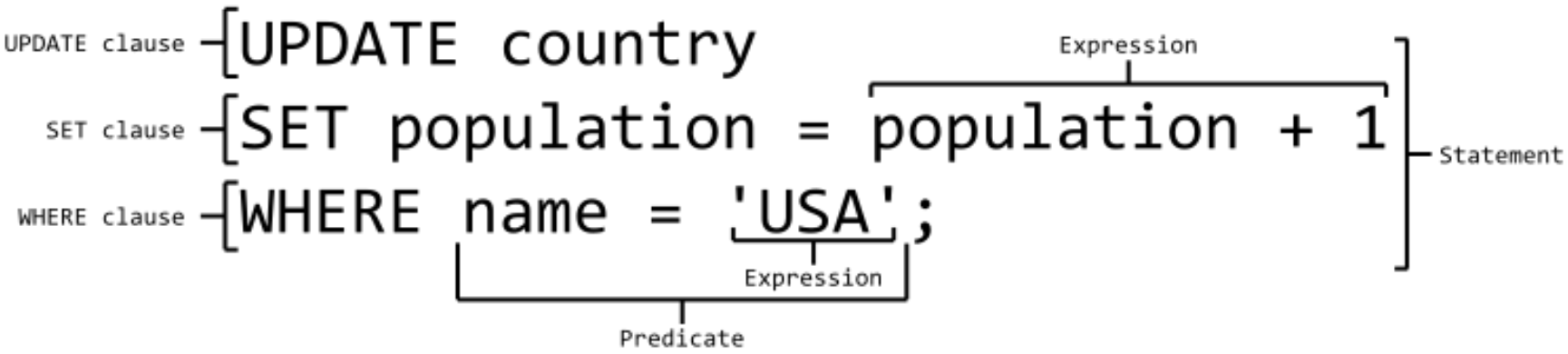
○ 데이터 제어 언어 DCL(Data Control Language)

- ➢ GRANT, REVOKE ...



■ SQL 구문

○ 문(Statement), 절(Clause), 식(Expression), 술어(Predicate)



■ Data Type

○ Boolean (BOOLEAN)

- ➢ to represent value TRUE or FALSE

○ Character (CHAR, VARCHAR)

- ➢ to represent character data for columns such as names of persons or cities

○ Exact numeric (NUMERIC, DECIMAL, INTEGER, SMALLINT, BIGINT)

- ➢ to represent number such as 1, 2, 3, … and so on
- ➢ used for numbers with fractions such as 123.45

○ Approximate numeric (REAL, FLOAT, DOUBLE)

- ➢ to represent numbers with fractions such as 123.45
- ➢ precision of the fraction may not be preserved across manipulation of such column values

○ Datetime (DATE, TIME, TIMESTAMP) / Large Object (CLOB, BLOB)

Data Type Family	Data Types
Character String	CHARACTER, VARCHAR, CLOB
Boolean	BOOLEAN
Binary String	BLOB
Date Time	DATE, TIME, TIMESTAMP
Number	SMALLINT, INTEGER, DECIMAL, NUMERIC, REAL, FLOAT, DOUBLE

■ BOOLEAN

○ values of BOOLEAN column : **TRUE** or **FALSE**

○ Syntax : **BOOLEAN**

○ Example : *definition of a column **is_manager** in a table*

is_manager ***BOOLEAN***

■ CHARACTER

○ a **sequence of characters** such as names of persons, cities, etc

○ Syntax : **{ CHARACTER | CHAR } [(length)]**

○ Length : length of the string, 1 if not specified / a fixed-length data type

○ Example :

book_title *CHARACTER(50)*

book_category *CHAR*

door_type *CHAR(3)*

Inserted Value	Actual Value Inserted
'IN'	'IN '
'OUT'	'OUT'
'INOUT'	Nothing is inserted due to error.

■ VARCHAR

○ varying character data type

○ size of data stored in such a column in each row can vary according to the actual size of the data

○ Syntax : **{ VARCHAR | CHARACTER VARYING | CHAR VARYING } [(length)]**

○ Length : if the data size is less than the size specified, data is not padded with blanks

○ Example :

book_synopsis VARCHAR(500)

■ INTEGER

○ integers in the range of -2147483648 to 2147483647

○ Syntax : **INTEGER**

○ Length : number of bytes occupied in a column by integer is **4bytes**(32bits)

○ Example :

book_sno	***INTEGER***
order_no	***INTEGER***

■ SMALLINT

○ integers in the range of -32768 to 32767

○ Syntax : **SMALLINT**

○ Length : number of bytes occupied in a column by smallint is **2bytes**(16bits)

○ Example :

roll_no	***SMALLINT***
quantity	***SMALLINT***

■ NUMERIC

○ numbers that can have fractions

○ Syntax : { **NUMERIC** } [(**precision** [, **scale**])]

○ Precision : total number of digits including number of decimal places, default is RDBMS specific

○ Scale : number of decimal places, default is 0

○ Length : one byte per digit

○ Example :

price	*NUMERIC(8,2)*
discount	*NUMERIC(4,1)*
interest	*NUMERIC(4,2)*

■ REAL

○ approximate numerical data type

○ Syntax : **REAL**

○ Length : 4bytes(32bits)

○ Example :

salary	***REAL***
width	***REAL***

■ FLOAT

○ approximate numerical data type

○ Syntax : **FLOAT [precision]**

○ Precision : precision of mantissa

○ Length : 4bytes(32bits)

○ Example :

distance ***FLOAT***

■ DOUBLE

○ approximate numerical data type

○ Syntax : **DOUBLE**

○ Length : 8bytes(64bits)

○ Example :

distance ***DOUBLE***

■ DATE

○ calendar date

○ consists of datetime fields : **YEAR**, **MONTH** and **DAY**

○ Syntax : **DATE**

○ Example :

date_of_birth ***DATE***

join_data ***DATE***

■ TIME

○ time of a day

○ consists of datetime fields : **HOUR**, **MINUTE** and **SECOND**

○ Syntax : **TIME [(precision)]**

○ Precision : seconds value, default is 0 / 3 means milliseconds / 6 means microseconds

○ Example :

stat_titme	***TIME***
event_time	***TIME(6)***

■ CLOB

○ large amount of character data

○ Syntax : **{ CHARACTER LARGE OBJECT | CHAR LARGE OBJECT | CLOB } [(large-object-length)]**

○ Large object length : length **[K | M | G]**

○ Example :

part_description ***CLOB(5M)***

emp_resume ***CLOB(10K)***

■ BLOB

○ large amount of binary data

○ Syntax : **{ BINARY LARGE OBJECT | BLOB } [(large-object-length)]**

○ Large object length : length **[K | M | G]**

○ Example :

part_image ***BLOB(2M)***

emp_photograph ***CLOB(150K)***

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

■ DDL

○ CREATE

```
CREATE TABLE [table name] ( [column definitions] ) [table parameters]
```

➢ column definitions

A comma-separated list consisting of any of the following

Column definition: *[column name] [data type] {NULL | NOT NULL} {column options}*

Primary key definition: *PRIMARY KEY ([comma separated column list])*

Constraints: *{CONSTRAINT} [constraint definition]*

RDBMS specific functionality

➢ constraints

NOT NULL - Ensures that a column cannot have a NULL value
UNIQUE - Ensures that all values in a column are different
PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
FOREIGN KEY - Uniquely identifies a row/record in another table
CHECK - Ensures that all values in a column satisfies a specific condition
DEFAULT - Sets a default value for a column when no value is specified
INDEX - Used to create and retrieve data from the database very quickly

➢ 예제

```
CREATE TABLE employees (
    id              INTEGER         PRIMARY KEY,
    first_name      VARCHAR(50)     not null,
    last_name       VARCHAR(75)     not null,
    fname           VARCHAR(50)     not null,
    dateofbirth     DATE            not null
);
```

➢ 예제

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

PersonID	LastName	FirstName	Address	City

○ DROP

```
DROP objecttype objectname.
```

The *DROP* statement destroys an existing database, table, index, or view.

➢ 예제

```
DROP TABLE employees;
```

The *DROP* statement is distinct from the *DELETE* and *TRUNCATE* statements, in that *DELETE* and *TRUNCATE* do not remove the table itself.

○ TRUNCATE

```
TRUNCATE TABLE table_name;
```

The *TRUNCATE* statement is used to delete all data from a table. It's much faster than *DELETE*.

○ ALTER

```
ALTER objecttype objectname parameters.
```

The *ALTER* statement modifies an existing database object.

➢ parameters

```
ALTER TABLE table_name
ADD column_name datatype;
```

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

➢ 예제

– 이미 만들어진 *sink* 테이블에 *bubbles* 열(필드) 추가

```
ALTER TABLE sink ADD bubbles INTEGER;
ALTER TABLE sink DROP COLUMN bubbles;
```

➢ 예제

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

```
ALTER TABLE Persons
MODIFY COLUMN DateOfBirth year;
```

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

```
ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

■ DML

○ INSERT

➢ 기본 방법

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ])
```

The number of columns and values must be the same.

➢ 빠른 방법

```
INSERT INTO table VALUES (value1, [value2, ... ])
```

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.

➢ 예제

```
INSERT INTO phone_book (name, number) VALUES ('John Doe', '555-1212');
```

```
INSERT INTO phone_book VALUES ('John Doe', '555-1212');
```

➢ 예제

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

➢ 예제

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	null	null	Stavanger	null	Norway

○ SELECT

➢ 빠른 방법

```
SELECT column1, column2, ...
FROM table_name;
```

A **SELECT** statement retrieves zero or more rows from one or more database tables or database views.
In most applications, `SELECT` is the most commonly used data query language (DQL) command.

➢ 예제

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

```
SELECT CustomerName, City FROM Customers;
```

```
SELECT * FROM Customers;
```

➢ 기본 방법

```
SELECT [ALL | DISTINCT] 컬럼명 [,컬럼명...]
FROM 테이블명 [,테이블명...]
[WHERE 조건식]
[GROUP BY 컬럼명 [HAVING 조건식]]
[ORDER BY 컬럼명]
GROUP BY 컬럼명[,컬럼명...]
ORDER BY 컬럼명[,컬럼명...]
```

➢ parameters

`WHERE` specifies which rows to retrieve.

`GROUP BY` groups rows sharing a property so that an aggregate function can be applied to each group.

`HAVING` selects among the groups defined by the GROUP BY clause.

`ORDER BY` specifies an order in which to return the rows.

`AS` provides an alias which can be used to temporarily rename tables or columns.

➢ 예제

Table "T"	Query	Result
C1 C2 1 a 2 b	`SELECT * FROM T;`	C1 C2 1 a 2 b
C1 C2 1 a 2 b	`SELECT C1 FROM T;`	C1 1 2
C1 C2 1 a 2 b	`SELECT * FROM T WHERE C1 = 1;`	C1 C2 1 a
C1 C2 1 a 2 b	`SELECT * FROM T ORDER BY C1 DESC;`	C1 C2 2 b 1 a

○ WHERE

- ➢ used to filter records.
- ➢ used to extract only those records that fulfill a specified condition

- ➢ 연산자

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

➢ 예제

– AND, OR

```
DELETE
FROM   mytable
WHERE  mycol > 100 AND item = 'Hammer'
```

– IN (find any values existing in a set of candidates)

```
SELECT ename WHERE ename IN ('value1', 'value2', ...)
```

```
SELECT ename WHERE ename='value1' OR ename='value2'
```

– BETWEEN (find any values within a range)

```
SELECT ename WHERE ename BETWEEN 'value1' AND 'value2'
```

```
SELECT salary from emp WHERE salary BETWEEN 5000 AND 10000
```

- LIKE (find a string fitting a certain description / % wildcard)
  - » '**a%**' / '**%a**' / '**%or%**' / '**_r%**' / '**a_%_%**' / '**a%o**'

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

- CustomerName
  - » '**a%**' / '**%a**' / '**%or%**' / '**_r%**' / '**a_%_%**'

- ContactName
  - » '**a%o**'

- CustomerName NOT LIKE
  - » '**a%**'

○ UPDATE

➢ 기본방법

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

➢ changes the data of one or more records in a table

➢ Either all the rows can be updated, or a subset may be chosen using a condition.

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

➢ 예제

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

➢ Update table

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

➢ Multiple Records

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

➢ Update Warning

```
UPDATE Customers
SET ContactName='Juan';
```

○ DELETE

➢ 기본방법

```
DELETE FROM table_name
WHERE condition;
```

➢ removes one or more records from a table

➢ A subset may be defined for deletion using a condition, otherwise all records are removed

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

➢ 예제

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

➢ Delete

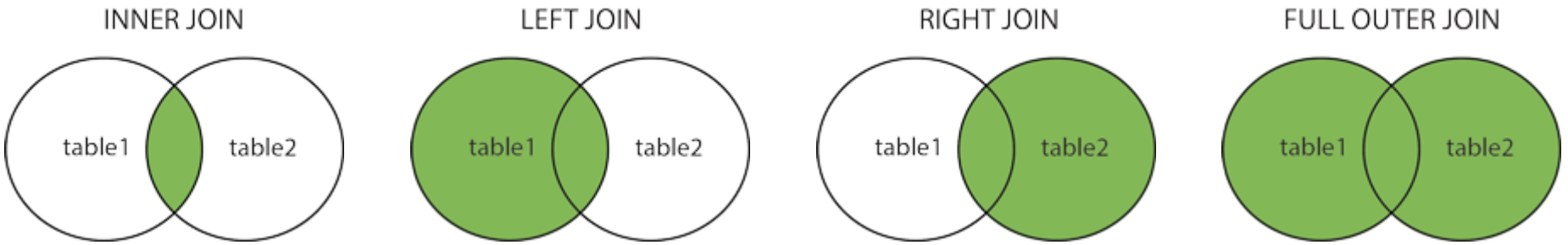
```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste';
```

➢ Delete all records

```
DELETE FROM table_name;
```

○ Joins

- combines columns from one or more tables in a relational database, based on a related column between them
- JOIN: ***INNER***, ***LEFT OUTER***, ***RIGHT OUTER***, ***FULL OUTER***, ***CROSS***



- (INNER) **JOIN**
  - returns records that have matching values in both tables

- **LEFT** (OUTER) JOIN
  - return all records from the left table, and the matched records from the right table

- **RIGHT** (OUTER) JOIN
  - return all records from the right table, and the matched records from the left table

- **FULL** (OUTER) JOIN
  - return all records when there is a match in either left or right table

➢ 예제

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996

○ SQL 예제

➢ CREATE, INSERT

```
CREATE TABLE department
(
 DepartmentID INT Primary key,
 DepartmentName VARCHAR(20)
);

CREATE TABLE employee
(
 LastName VARCHAR(20),
 DepartmentID INT references department(DepartmentID)
);

INSERT INTO department VALUES(31, 'Sales');
INSERT INTO department VALUES(33, 'Engineering');
INSERT INTO department VALUES(34, 'Clerical');
INSERT INTO department VALUES(35, 'Marketing');

INSERT INTO employee VALUES('Rafferty', 31);
INSERT INTO employee VALUES('Jones', 33);
INSERT INTO employee VALUES('Heisenberg', 33);
INSERT INTO employee VALUES('Robinson', 34);
INSERT INTO employee VALUES('Smith', 34);
INSERT INTO employee VALUES('Williams', NULL);
```

➢ Table

Employee table

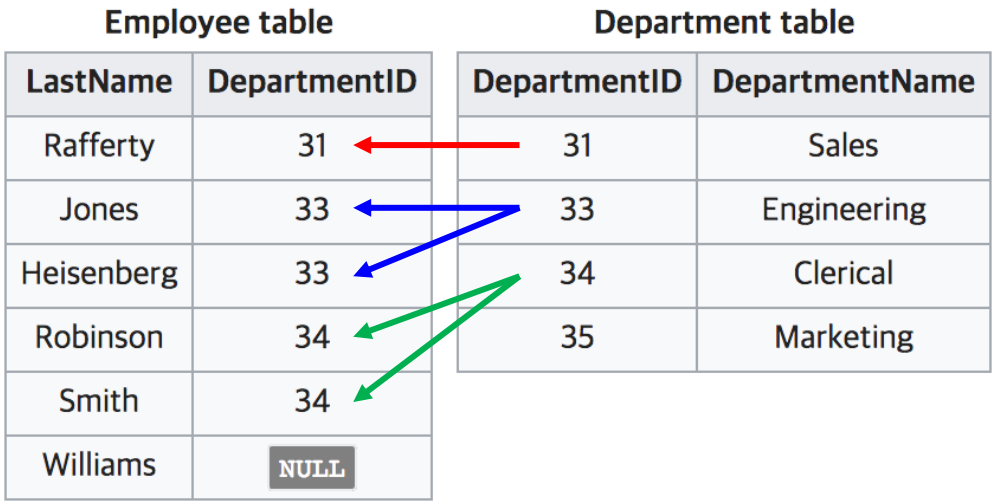
LastName	DepartmentID
Rafferty	31
Jones	33
Heisenberg	33
Robinson	34
Smith	34
Williams	NULL

Department table

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

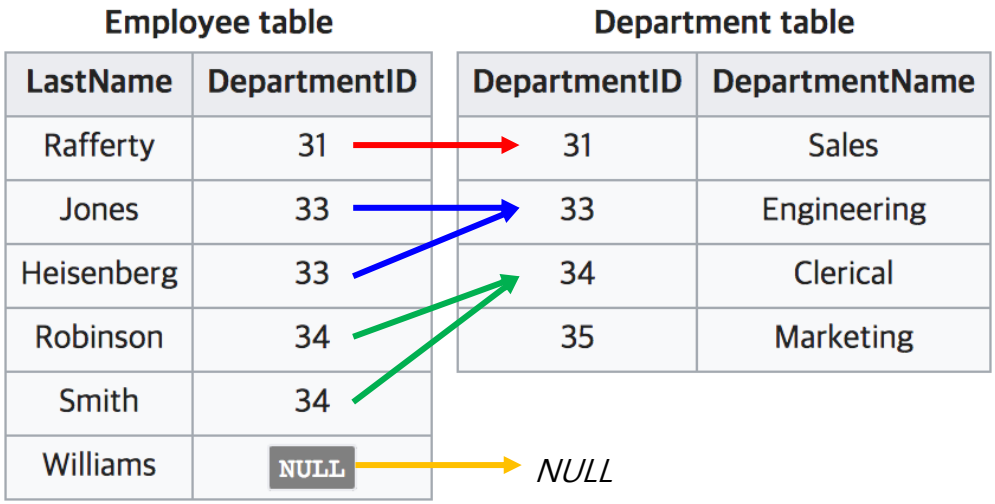
➢ CROSS JOIN

```
SELECT *
FROM employee CROSS JOIN department;
```



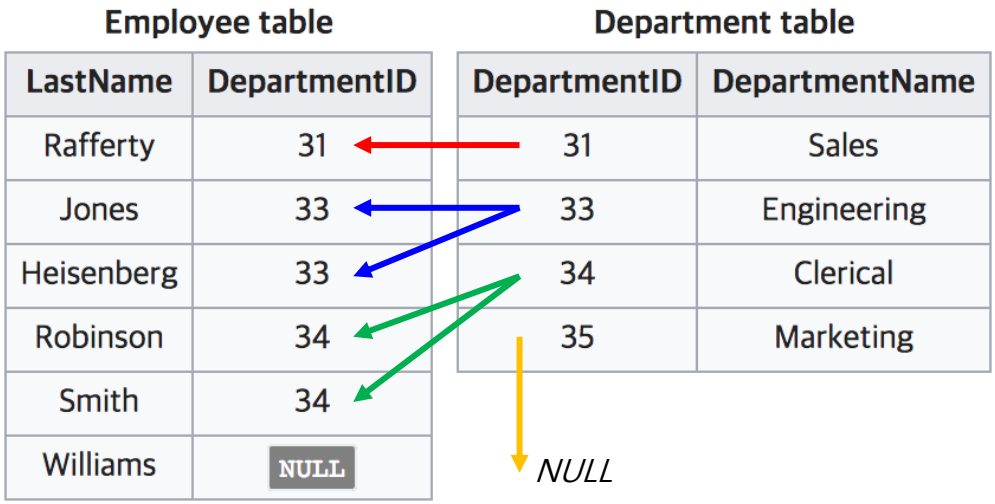
➢ INNER JOIN

```
SELECT employee.LastName, employee.DepartmentID, department.DepartmentName
FROM employee
INNER JOIN department ON
employee.DepartmentID = department.DepartmentID
```



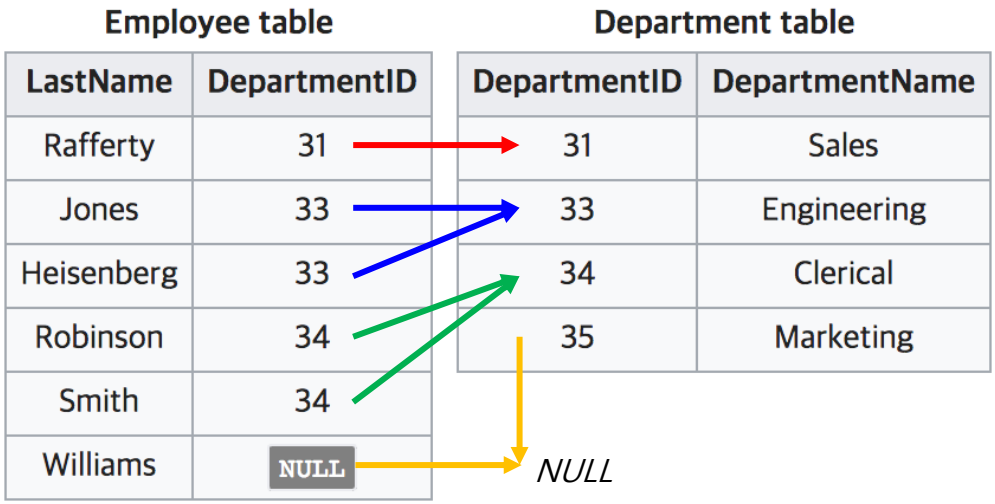
➢ LEFT OUTER JOIN

```
SELECT *
FROM employee
LEFT OUTER JOIN department ON employee.DepartmentID = department.DepartmentID;
```



➢ RIGHT OUTER JOIN

```
SELECT *
FROM employee RIGHT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```



➢ FULL OUTER JOIN

```
SELECT *
FROM employee FULL OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

○ SQL 예제

CITY:

CNO	CNAME
1	London
2	Paris
3	Rome
4	Vienna

SUPPLIER:

SNO	SNAME	CNO
1	Smith	1
2	Jones	2
3	Adams	1
4	Blake	3

PART:

PNO	PNAME
1	Screw
2	Nut
3	Bolt
4	Cam

SELLS:

SNO	PNO	PRICE
1	1	10
1	2	8
2	4	38
3	1	11
3	3	6
4	2	7
4	3	4
4	4	45

➢ QUERY

- SELECT * FROM PART;
- SELECT * FROM SELLS WHERE PRICE > 11;
- SELECT SNO, PRICE FROM SELLS WHERE PRICE > 11;
- SELECT PNO, PRICE FROM SELLS WHERE PNO = 1 AND PRICE <= 10;

- SELECT * FROM SUPPLIER, PART; *(CROSS JOIN – 4*4)*
- SELECT S.SNAME, C.CNAME FROM SUPPLIER AS S, CITY AS C WHERE S.CNO = C.CNO;
- SELECT SNAME AS LondonSuppliers FROM SUPPLIER, CITY *(CROSS JOIN – 4*4)*
  WHERE SUPPLIER.CNO = CITY.CNO AND CNAME = 'London';
- SELECT SNAME, PNAME, PRICE FROM SELLS, SUPPLIER, PART *(CROSS JOIN – 8*4*4)*
  WHERE SELLS.SNO = SUPPLIER.SNO AND SELLS.PNO = PART.PNO ORDER BY SNAME, PNAME;

- INSERT INTO SUPPLIER VALUES (1, 'Smith', 1);
- UPDATE SELLS SET PRICE = 15 WHERE SNO = 1 AND PNO = 1;
- DELETE FROM SUPPLIER WHERE SNAME = 'Smith';

1.4. SQLite

■ What is SQLite?

- ○ Open Source Database(Free)
- ○ Serverless(Direct I/O)
- ○ Self-contained(Embedded)
- ○ Single Disk File(Cross-platform)
- ○ Zero-configuration(No setup, No server)
- ○ Supports RDBMS Features(ACID, SQL Syntax, Transactions, etc)

http://sqlitebrowser.org

■ SQLite 확인

```
import sqlite3
```

`sqlite3.`**`version`**

The version number of this module, as a string. This is not the version of the SQLite library.

`sqlite3.`**`version_info`**

The version number of this module, as a tuple of integers. This is not the version of the SQLite library.

`sqlite3.`**`sqlite_version`**

The version number of the run-time SQLite library, as a string.

`sqlite3.`**`sqlite_version_info`**

The version number of the run-time SQLite library, as a tuple of integers.

■ Database 연결(생성)

```
conn = sqlite3.connect('경로/이름' or ':memory:')
```

sqlite3.**connect**(*database*[, *timeout*, *detect_types*, *isolation_level*, *check_same_thread*, *factory*, *cached_statements*, *uri*])

Opens a connection to the SQLite database file *database*. You can use `":memory:"` to open a database connection to a database that resides in RAM instead of on disk.

When a database is accessed by multiple connections, and one of the processes modifies the database, the SQLite database is locked until that transaction is committed. The *timeout* parameter specifies how long the connection should wait for the lock to go away until raising an exception. The default for the timeout parameter is 5.0 (five seconds).

■ Cursor 생성

```
cur = conn.cursor()
```

cursor(*factory=Cursor*)

The cursor method accepts a single optional parameter *factory*. If supplied, this must be a callable returning an instance of `Cursor` or its subclasses.

```
type(cur)
dir(cur)
```

```
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'arraysize',
'close',
'connection',
'description',
'execute',
'executemany',
'executescript',
'fetchall',
'fetchmany',
'fetchone',
'lastrowid',
'row_factory',
'rowcount',
'setinputsizes',
'setoutputsize']
```

■ Data Type

SQLite natively supports the following types: `NULL`, `INTEGER`, `REAL`, `TEXT`, `BLOB`.

Sr.No.	Storage Class & Description
1	**NULL** The value is a NULL value.
2	**INTEGER** The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
3	**REAL** The value is a floating point value, stored as an 8-byte IEEE floating point number.
4	**TEXT** The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
5	**BLOB** The value is a blob of data, stored exactly as it was input.

execute

```
cur.execute('SQL')
```

execute(*sql*[, *parameters*])

Executes an SQL statement. The SQL statement may be parameterized (i. e. placeholders instead of SQL literals). The `sqlite3` module supports two kinds of placeholders: question marks (qmark style) and named placeholders (named style).

```
cur.execute("create table people (name_last, age)")

who = "Yeltsin"
age = 72

# This is the qmark style:
cur.execute("insert into people values (?, ?)", (who, age))

# And this is the named style:
cur.execute("select * from people where name_last=:who and age=:age", {"who": who, "age": age})

print(cur.fetchone())
```

executemany

```
cur.executemany('SQL', params)
```

executemany(*sql*, *seq_of_parameters*)

Executes an SQL command against all parameter sequences or mappings found in the sequence *seq_of_parameters*. The `sqlite3` module also allows using an iterator yielding parameters instead of a sequence.

```
sql = "insert into people values (?, ?)"
curData = [('A', 1), ('B', 2), ('C', 3)]

cur.executemany(sql, curData)
```

executescript

```
cur.executescript('''SQL1; SQL2; ...''')
```

executescript(*sql_script*)

This is a nonstandard convenience method for executing multiple SQL statements at once. It issues a `COMMIT` statement first, then executes the SQL script it gets as a parameter.

```
cur.executescript("""
    create table person (
        first_name text primary key,
        last_name  text not null
    );

    insert into person values ('name', 'kim');
""")
```

```
cur.execute('select * from person')
print(cur.fetchall())
```

fectchone

```
cur.fetchone()
```

fetchone()

Fetches the next row of a query result set, returning a single sequence, or `None` when no more data is available.

fetchmanay

```
cur.fetchmany(size)
```

fetchmany(*size=cursor.arraysize*)

Fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available.

The number of rows to fetch per call is specified by the *size* parameter. If it is not given, the cursor's arraysize determines the number of rows to be fetched. The method should try to fetch as many rows as indicated by the size parameter. If this is not possible due to the specified number of rows not being available, fewer rows may be returned.

Note there are performance considerations involved with the *size* parameter. For optimal performance, it is usually best to use the arraysize attribute. If the *size* parameter is used, then it is best for it to retain the same value from one `fetchmany()` call to the next.

■ fectchall

```
cur.fetchall()
```

fetchall()

Fetches all (remaining) rows of a query result, returning a list. Note that the cursor's arraysize attribute can affect the performance of this operation. An empty list is returned when no rows are available.

■ DDL

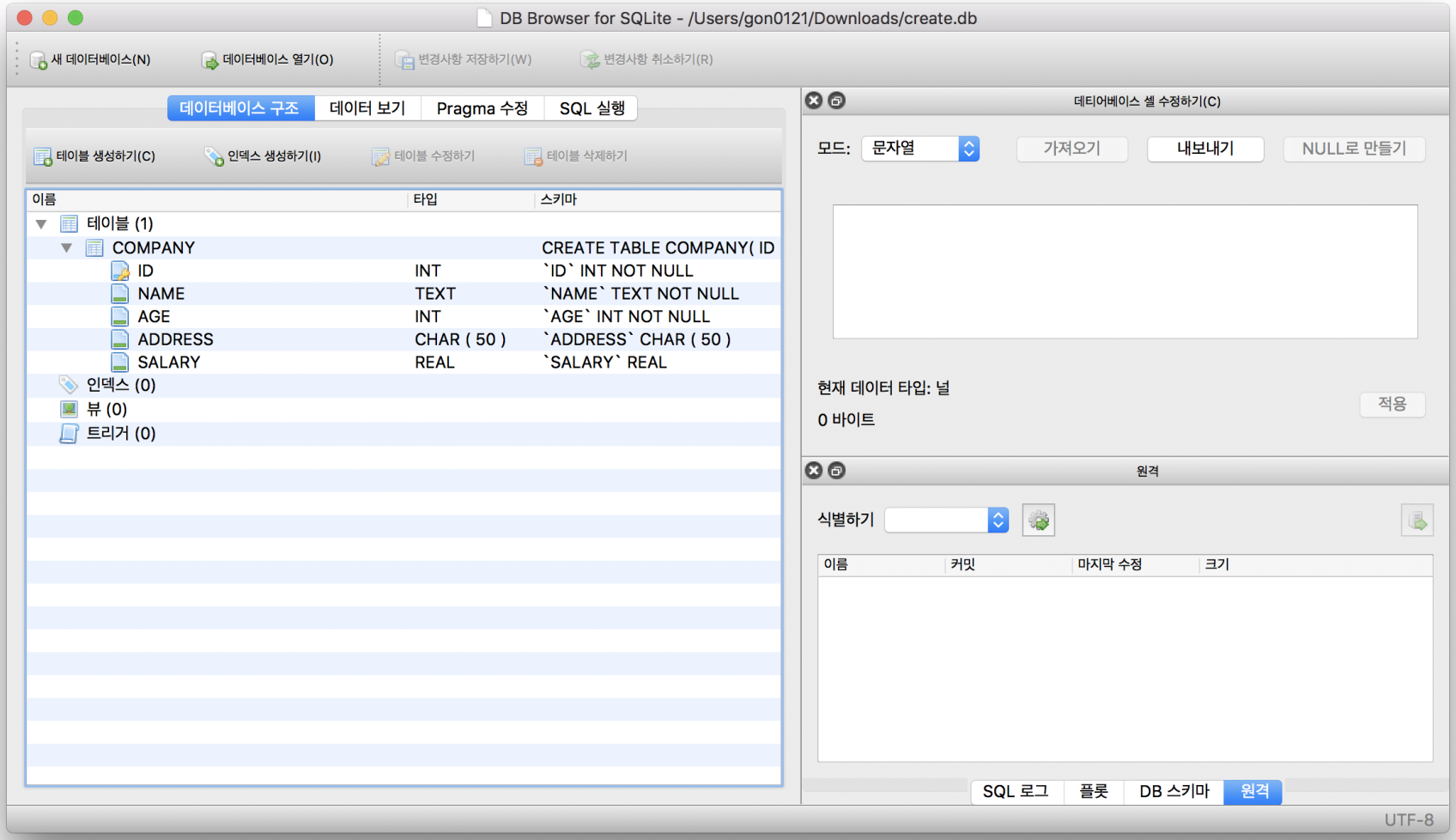
○ CREATE

```
conn = sqlite3.connect('create.db')
print("Opened database successfully")

conn.execute('''
    CREATE TABLE COMPANY(
        ID INT PRIMARY KEY     NOT NULL,
        NAME           TEXT    NOT NULL,
        AGE            INT     NOT NULL,
        ADDRESS        CHAR(50),
        SALARY         REAL);
    ''')

print("Table created successfully")
```

○ DROP

```
conn.execute('drop table company')
```

○ CREAT 확인

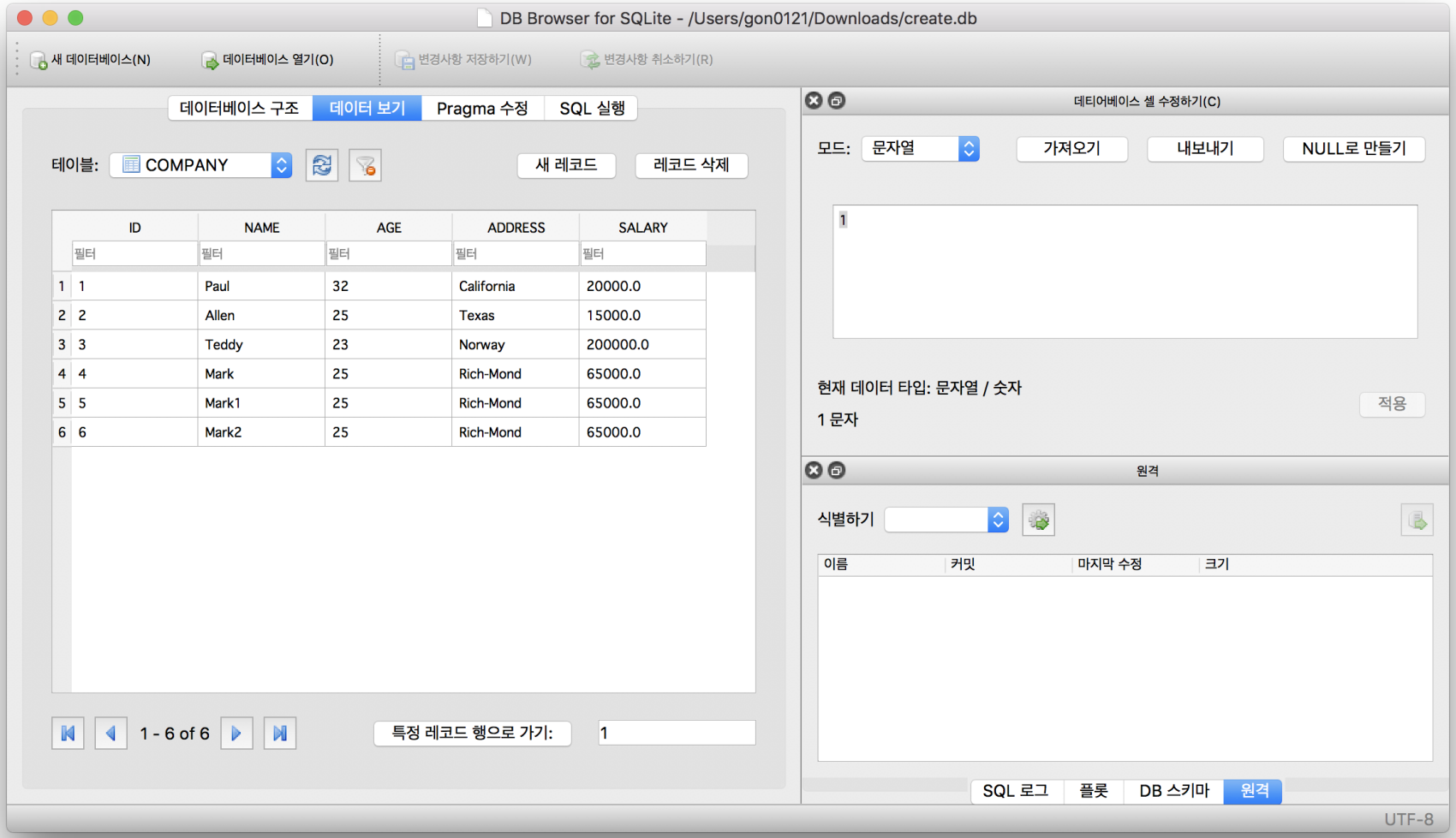

DB Browser for SQLite - /Users/gon0121/Downloads/create.db
새 데이터베이스(N)
데이터베이스 열기(O)
변경사항 저장하기(W)
변경사항 취소하기(R)
데이터베이스 구조
데이터 보기
Pragma 수정
SQL 실행
테이블 생성하기(C)
인덱스 생성하기(I)
테이블 수정하기
테이블 삭제하기
이름
타입
스키마
테이블 (1)
COMPANY
CREATE TABLE COMPANY(ID
ID
INT
`ID` INT NOT NULL
NAME
TEXT
`NAME` TEXT NOT NULL
AGE
INT
`AGE` INT NOT NULL
ADDRESS
CHAR (50)
`ADDRESS` CHAR (50)
SALARY
REAL
`SALARY` REAL
인덱스 (0)
뷰 (0)
트리거 (0)
데티어베이스 셀 수정하기(C)
모드:
문자열
가져오기
내보내기
NULL로 만들기
현재 데이터 타입: 널
0 바이트
적용
원격
식별하기
이름
커밋
마지막 수정
크기
SQL 로그
플롯
DB 스키마
원격
UTF-8

■ DML

○ INSERT

```
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
              VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
              VALUES (:id, :name, :age, :address, :salary)",
             {'id':2, 'name':'Allen', 'age':25, 'address':'Texas', 'salary':15000.00});

data = [(3, 'Teddy', 23, 'Norway ', 200000.00 ),
        (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )]

conn.executemany("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
                  VALUES (?, ?, ?, ?, ?)", data);

conn.executescript("""
    INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
    VALUES (5, 'Mark1', 25, 'Rich-Mond ', 65000.00 );

    INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
    VALUES (6, 'Mark2', 25, 'Rich-Mond ', 65000.00 );
""");

conn.commit()
print("Records created successfully")
```

○ INSERT 확인


DB Browser for SQLite - /Users/gon0121/Downloads/create.db
새 데이터베이스(N)
데이터베이스 열기(O)
변경사항 저장하기(W)
변경사항 취소하기(R)
데이터베이스 구조
데이터 보기
Pragma 수정
SQL 실행
테이블: COMPANY
새 레코드
레코드 삭제
ID NAME AGE ADDRESS SALARY
1 Paul 32 California 20000.0
2 Allen 25 Texas 15000.0
3 Teddy 23 Norway 200000.0
4 Mark 25 Rich-Mond 65000.0
5 Mark1 25 Rich-Mond 65000.0
6 Mark2 25 Rich-Mond 65000.0
1 - 6 of 6
특정 레코드 행으로 가기:
데티어베이스 셀 수정하기(C)
모드: 문자열
가져오기
내보내기
NULL로 만들기
현재 데이터 타입: 문자열 / 숫자
1 문자
적용
원격
식별하기
이름 커밋 마지막 수정 크기
SQL 로그
플롯
DB 스키마
원격
UTF-8

○ SELECT

```
cursor = conn.execute("SELECT id, name, address, salary from COMPANY")

for row in cursor:
    print("ID = ", row[0])
    print("NAME = ", row[1])
    print("ADDRESS = ", row[2])
    print("SALARY = ", row[3], end='\n\n')

print("Operation done successfully")
```

```
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  20000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  200000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

ID =  5
NAME =  Mark1
ADDRESS =  Rich-Mond
SALARY =  65000.0

ID =  6
NAME =  Mark2
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

○ UPDATE

```
cid = 1

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = :id", {'id':cid})
conn.commit()

print("Total number of rows updated :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
    print("ID = ", row[0])
    print("NAME = ", row[1])
    print("ADDRESS = ", row[2])
    print("SALARY = ", row[3], "\n")
```

```
Total number of rows updated : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0
```

○ DELETE

```
conn.execute("DELETE from COMPANY where ID = 2;")

print("Total number of rows deleted :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
    print("ID = ", row[0])
    print("NAME = ", row[1])
    print("ADDRESS = ", row[2])
    print("SALARY = ", row[3], "\n")
```

```
Total number of rows deleted : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  200000.0
```

■ 예제

○ Database 생성(열기), Cursor 생성

```
import sqlite3 as sql

con = None

try:
    con = sql.connect('test.db')

    cur = con.cursor()

    cur.execute('SELECT SQLITE_VERSION()')

    data = cur.fetchone()

    print("SQLite Version: ", data)

except Exception as e:
    print("Error: ", e)

finally:
    if con:
        con.close()
```

○ DDL – CREATE, DML – INSERT(execute)

```
con = sql.connect('test.db')

with con:
    cur = con.cursor()
    cur.execute("CREATE TABLE Cars(Id INT, Name TEXT, Price INT)")
    cur.execute("INSERT INTO Cars VALUES(1,'Audi',52642)")
    cur.execute("INSERT INTO Cars VALUES(2,'Mercedes',57127)")
    cur.execute("INSERT INTO Cars VALUES(3,'Skoda',9000)")
    cur.execute("INSERT INTO Cars VALUES(4,'Volvo',29000)")
    cur.execute("INSERT INTO Cars VALUES(5,'Bentley',350000)")
    cur.execute("INSERT INTO Cars VALUES(6,'Citroen',21000)")
    cur.execute("INSERT INTO Cars VALUES(7,'Hummer',41400)")
    cur.execute("INSERT INTO Cars VALUES(8,'Volkswagen',21600)")
```

○ DDL – CREATE, DML – INSERT(executemany)

```
cars = [
    (1, 'Audi', 52642),
    (2, 'Mercedes', 57127),
    (3, 'Skoda', 9000),
    (4, 'Volvo', 29000),
    (5, 'Bentley', 350000),
    (6, 'Hummer', 41400),
    (7, 'Volkswagen', 21600)
]

con = sql.connect('test.db')

with con:
    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS Cars")
    cur.execute("CREATE TABLE Cars(Id INT PRIMARY KEY, Name TEXT, Price INT)")
    cur.executemany("INSERT INTO Cars VALUES(?, ?, ?)", cars)
```

○ DDL – DROP, CREATE, DML – INSERT(executescript)

```
try:
    con = sql.connect('test.db')

    cur = con.cursor()

    cur.executescript("""
        DROP TABLE IF EXISTS Cars;
        CREATE TABLE Cars(Id INT, Name TEXT, Price INT);
        INSERT INTO Cars VALUES(1,'Audi',52642);
        INSERT INTO Cars VALUES(2,'Mercedes',57127);
        INSERT INTO Cars VALUES(3,'Skoda',9000);
        INSERT INTO Cars VALUES(4,'Volvo',29000);
        INSERT INTO Cars VALUES(5,'Bentley',350000);
        INSERT INTO Cars VALUES(6,'Citroen',21000);
        INSERT INTO Cars VALUES(7,'Hummer',41400);
        INSERT INTO Cars VALUES(8,'Volkswagen',21600);
        """)

    con.commit()

except Exception as e:
    if con:
        con.rollback()

    print("Error: ", e)

finally:
    if con:
        con.close()
```

○ DDL – DROP, CREATE, DML – INSERT(lastrowid)

```
con = sql.connect(':memory:')

with con:
    cur = con.cursor()
    cur.execute("CREATE TABLE Friends(Id INTEGER PRIMARY KEY, Name TEXT);")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Tom');")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Rebecca');")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Jim');")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Robert');")

    lid = cur.lastrowid
    print("The last Id of the inserted row is", lid)
```

○ DML – SELECT(fetchone)

```
con = sql.connect('test.db')

with con:
    cur = con.cursor()
    cur.execute("SELECT * FROM Cars")

    while True:
        row = cur.fetchone()

        if row == None:
            break

        print(row[0], row[1], row[2])
```

○ DML – SELECT(dictionary, fetchall)

```
con = sql.connect('test.db')

with con:
    con.row_factory = sql.Row

    cur = con.cursor()
    cur.execute("SELECT * FROM Cars")

    rows = cur.fetchall()

    for row in rows:
        print(row["Id"], row["Name"], row["Price"])
```

○ DML – UPDATE(qmark)

```
uId = 1
uPrice = 62300

con = sql.connect('test.db')

with con:
    cur = con.cursor()

    cur.execute("UPDATE Cars SET Price=? WHERE Id=?", (uPrice, uId))

    print("Number of rows updated:", cur.rowcount)
```

○ DML – UPDATE(named)

```
uId = 4

con = sql.connect('test.db')

with con:

    cur = con.cursor()

    cur.execute("SELECT Name, Price FROM Cars WHERE Id=:Id", {"Id": uId})

    row = cur.fetchone()
    print(row[0], row[1])
```

○ Dump(backup)

iterdump()

Returns an iterator to dump the database in an SQL text format. Useful when saving an in-memory database for later restoration. This function provides the same capabilities as the `.dump` command in the **sqlite3** shell.

```
# Convert file existing_db.db to SQL dump file dump.sql
import sqlite3

con = sqlite3.connect('existing_db.db')
with open('dump.sql', 'w') as f:
    for line in con.iterdump():
        f.write('%s\n' % line)
```

○ Export / Import

```
cars = (
    (1, 'Audi', 52643),
    (2, 'Mercedes', 57642),
    (3, 'Skoda', 9000),
    (4, 'Volvo', 29000),
    (5, 'Bentley', 350000),
    (6, 'Hummer', 41400),
    (7, 'Volkswagen', 21600)
)

def writeData(data):
    f = open('cars.sql', 'w')

    with f:
        f.write(data)


con = sql.connect(':memory:')

with con:
    cur = con.cursor()

    cur.execute("DROP TABLE IF EXISTS Cars")
    cur.execute("CREATE TABLE Cars(Id INT, Name TEXT, Price INT)")
    cur.executemany("INSERT INTO Cars VALUES(?, ?, ?)", cars)
    cur.execute("DELETE FROM Cars WHERE Price < 30000")

    data = '\n'.join(con.iterdump())

    writeData(data)
```

○ Export / Import

```
def readData():
    f = open('cars.sql', 'r')

    with f:
        data = f.read()
        return data


con = sql.connect(':memory:')

with con:
    cur = con.cursor()

    rowData = readData()
    cur.executescript(rowData)

    cur.execute("SELECT * FROM Cars")

    rows = cur.fetchall()

    for row in rows:
        print(row)
```

○ Transaction(Commit)

```
try:
    con = sql.connect('test.db')
    cur = con.cursor()
    cur.execute("DROP TABLE IF EXISTS Friends")
    cur.execute("CREATE TABLE Friends(Id INTEGER PRIMARY KEY, Name TEXT)")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Tom')")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Rebecca')")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Jim')")
    cur.execute("INSERT INTO Friends(Name) VALUES ('Robert')")

    #con.commit()

except Exception as e:

    if con:
        con.rollback()

    print("Error:", e)

finally:
    if con:
        con.close()
```

■ 예제 (ER Model – SQL)

Track	Length	Artist	Album	Genre	Rating	Count
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23
Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	★★★★★	24
Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	★★★★★	26
Track 02	2:45	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	22
Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	★★★★★	21
War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Metal	★★★★★	25
Paranoid	2:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Planet Caravan	4:35	Black Sabbath	Paranoid	Metal	★★★★★	25
Iron Man	5:59	Black Sabbath	Paranoid	Metal	★★★★★	26
Electric Funeral	4:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Hand of Doom	7:10	Black Sabbath	Paranoid	Metal	★★★★★	23
Rat Salad	2:30	Black Sabbath	Paranoid	Metal	★★★★★	31
Jack the Stripper/Fairies Wear ...	6:14	Black Sabbath	Paranoid	Metal	★★★★★	24
Bomb Squad (TECH)	3:28	Brent	Brent's Album			1
clay techno	4:36	Brent	Brent's Album			2
Heavy	3:08	Brent	Brent's Album			1
Hi metal man	4:20	Brent	Brent's Album			1
Mistro	2:58	Brent	Brent's Album			1

○ 개체(Entity) – 관계(Relationship)

Track

Length

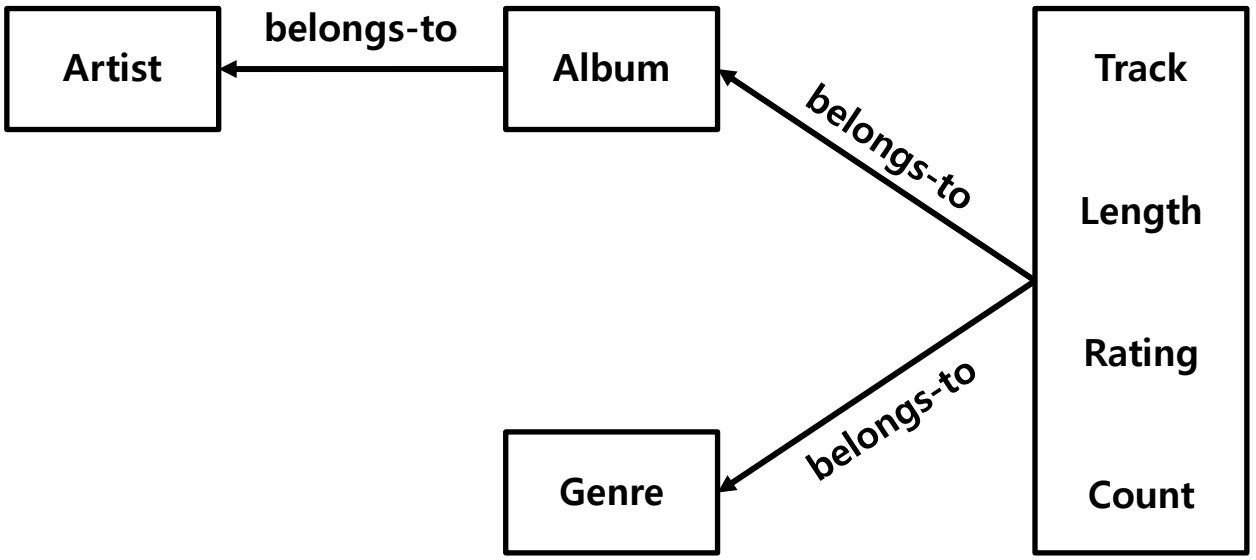
Artist

Album

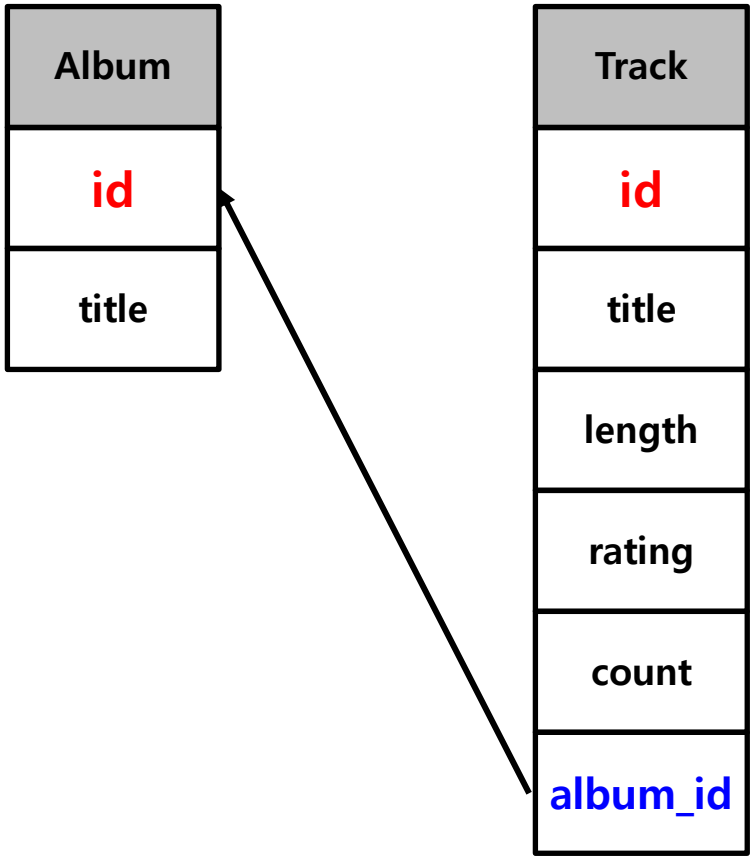
Genre

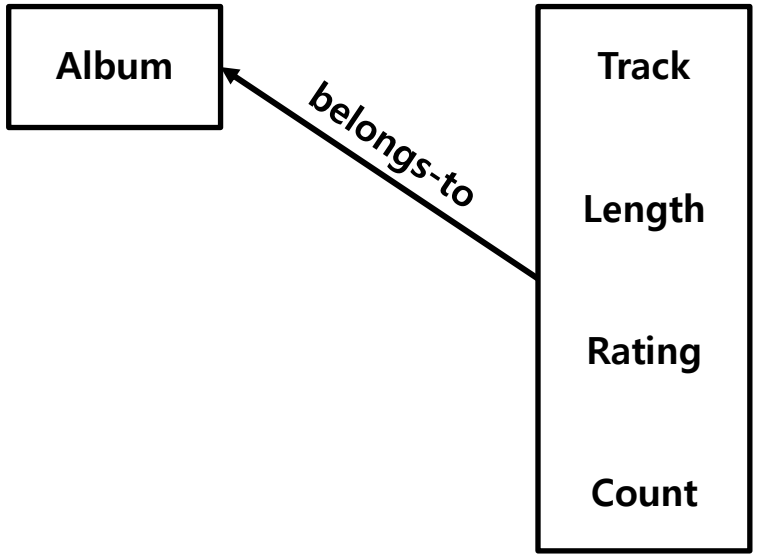
Rating

Count

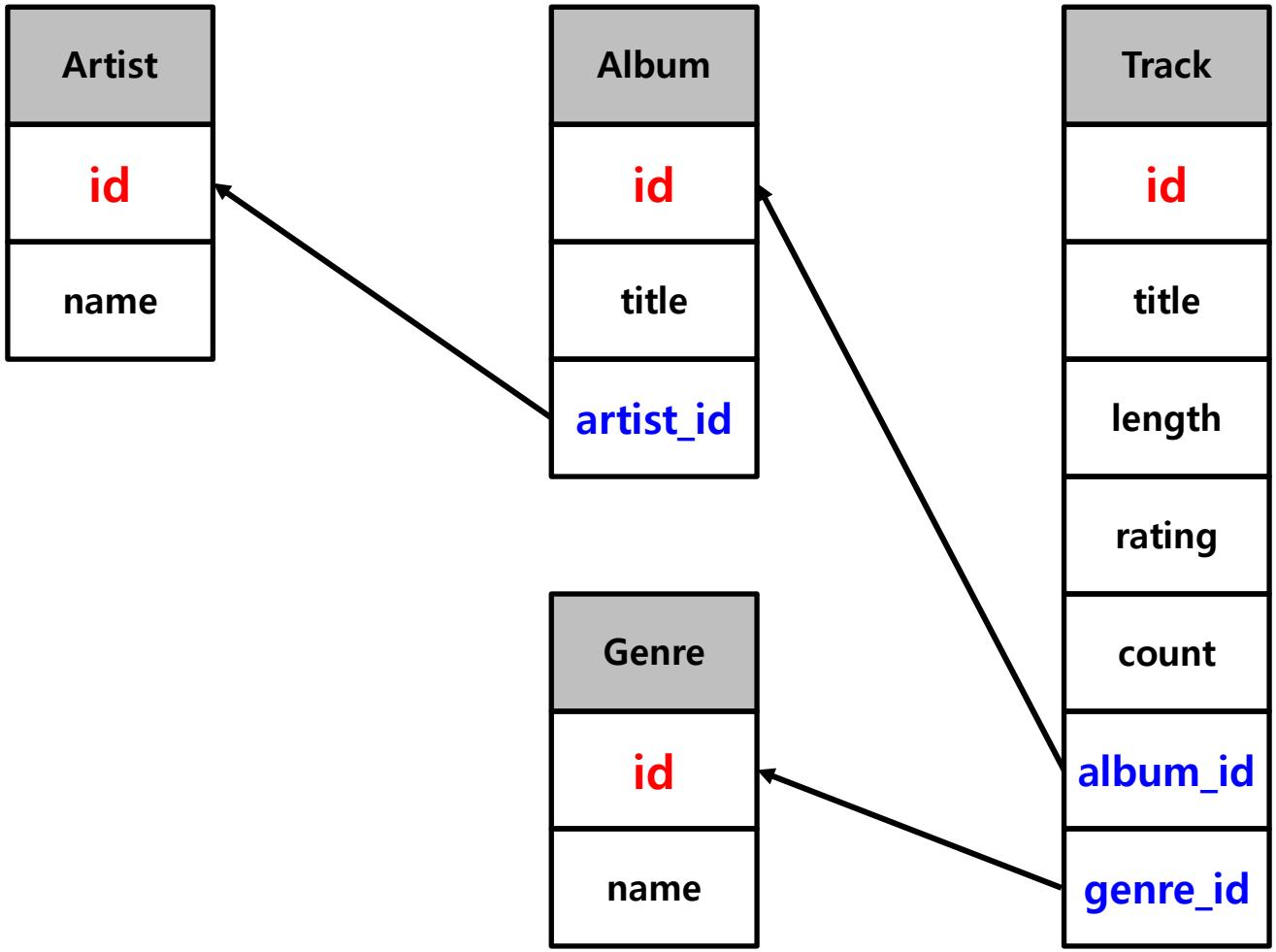


○ 기본키(Primary Key) / 참조키(Foreign Key)





○ 기본키(Primary Key) / 참조키(Foreign Key)



○ TABLE 생성

➢ Artist

– CREATE TABLE Artist (
id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
name TEXT
)

➢ Genre

– CREATE TABLE Genre (
id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
name TEXT
)

➢ Album

– CREATE TABLE Album (
id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
title TEXT,
artist_id INTEGER
)

- Track
  - CREATE TABLE Track (
    id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    title TEXT,
    length INTEGER,
    rating INTEGER,
    count INTEGER,
    album_id INTEGER,
    genre_id INTEGER
)

○ INSERT

- Artist
  - insert into Artist (name) values ('Led Zepplin')
  - insert into Artist (name) values ('AC/DC')

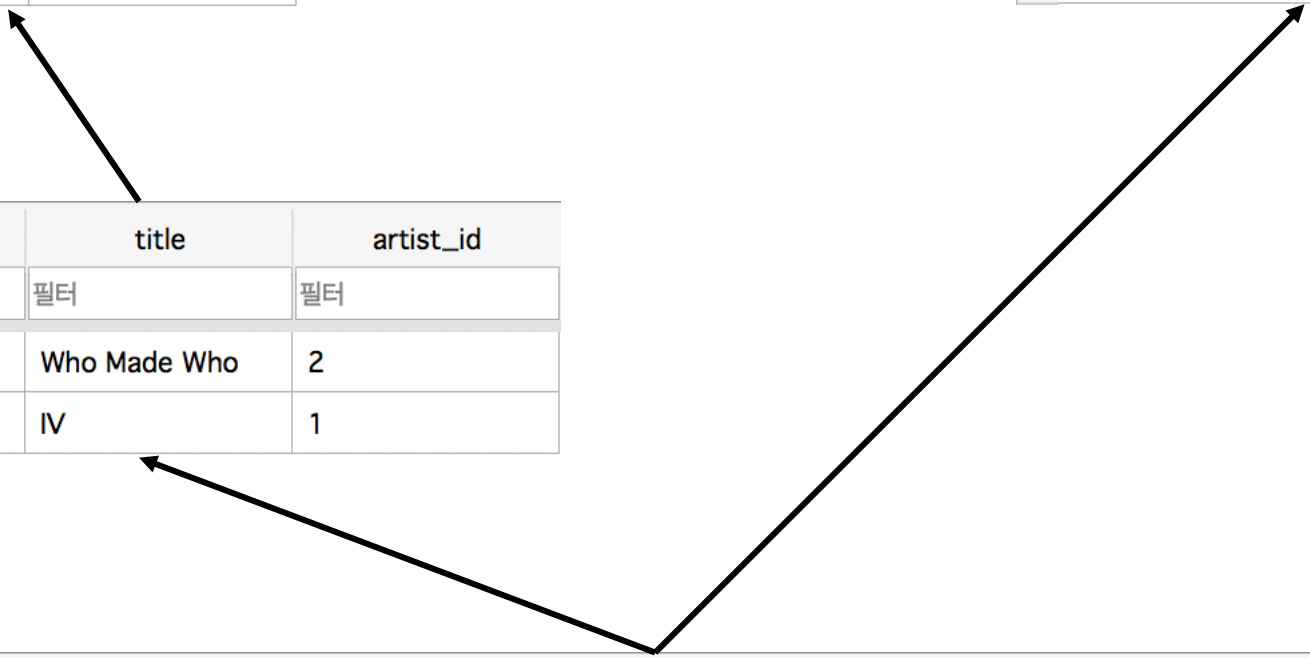
- Genre
  - insert into Genre (name) values ('Rock')
  - insert into Genre (name) values ('Metal')

- Album
  - insert into Album (title, artist_id) values ('Who Made Who', 2)
  - insert into Album (title, artist_id) values ('IV', 1)

- Track
  - insert into Track (title, rating, length, count, album_id, genre_id) values ('Black Dog', 5, 297, 0, 2, 1)
  - insert into Track (title, rating, length, count, album_id, genre_id) values ('Stairway', 5, 482, 0, 2, 1)
  - insert into Track (title, rating, length, count, album_id, genre_id) values ('About to Rock', 5, 313, 0, 1, 2)
  - insert into Track (title, rating, length, count, album_id, genre_id) values ('Who Made Who', 5, 207, 0, 1, 2)

	id	name
	필터	필터
1	1	Led Zepplin
2	2	AC/DC

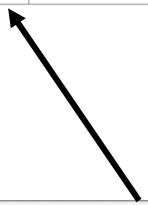
	id	name
	필터	필터
1	1	Rock
2	2	Metal



	id	title	artist_id
	필터	필터	필터
1	1	Who Made Who	2
2	2	IV	1

	id	title	length	rating	count	album_id	genre_id
	필터	필터	필터	필터	필터	필터	필터
1	1	Black Dog	297	5	0	2	1
2	2	Stairway	482	5	0	2	1
3	3	About to Rock	313	5	0	1	2
4	4	Who Made Who	207	5	0	1	2

	id	name
	필터	필터
1	1	Led Zepplin
2	2	AC/DC



	id	title	artist_id
	필터	필터	필터
1	1	Who Made Who	2
2	2	IV	1

○ Join

➢ select Album.title, Artist.name
from Album
join Artist on
Album.artist_id = Artist.id

	title	name
1	Who Made Who	AC/DC
2	IV	Led Zepplin

➢ select Track.title, Genre.name
from Track
join Genre on
Track.genre_id = Genre.id

	title	name
1	Black Dog	Rock
2	Stairway	Rock
3	About to Rock	Metal
4	Who Made Who	Metal

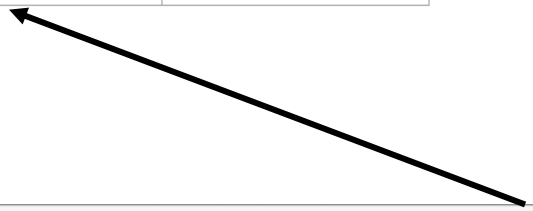
	id	name
	필터	필터
1	1	Rock
2	2	Metal

	id	title	length	rating	count	album_id	genre_id
	필터	필터	필터	필터	필터	필터	필터
1	1	Black Dog	297	5	0	2	1
2	2	Stairway	482	5	0	2	1
3	3	About to Rock	313	5	0	1	2
4	4	Who Made Who	207	5	0	1	2

➢ select Album.title, Track.title
from Track
join Album on
Track.album_id = Album.id

	title	title
1	IV	Black Dog
2	IV	Stairway
3	Who Made Who	About to Rock
4	Who Made Who	Who Made Who

	id	title	artist_id
	필터	필터	필터
1	1	Who Made Who	2
2	2	IV	1



	id	title	length	rating	count	album_id	genre_id
	필터	필터	필터	필터	필터	필터	필터
1	1	Black Dog	297	5	0	2	1
2	2	Stairway	482	5	0	2	1
3	3	About to Rock	313	5	0	1	2
4	4	Who Made Who	207	5	0	1	2

➢ select Track.title, Artist.name, Album.title, Genre.name
from Track
join Artist join Album join Genre on
Track.album_id = Album.id
and
Track.genre_id = Genre.id
and
Album.artist_id = Artist.id

	title	name	title	name
1	Black Dog	Led Zepplin	IV	Rock
2	Stairway	Led Zepplin	IV	Rock
3	About to Rock	AC/DC	Who Made Who	Metal
4	Who Made Who	AC/DC	Who Made Who	Metal

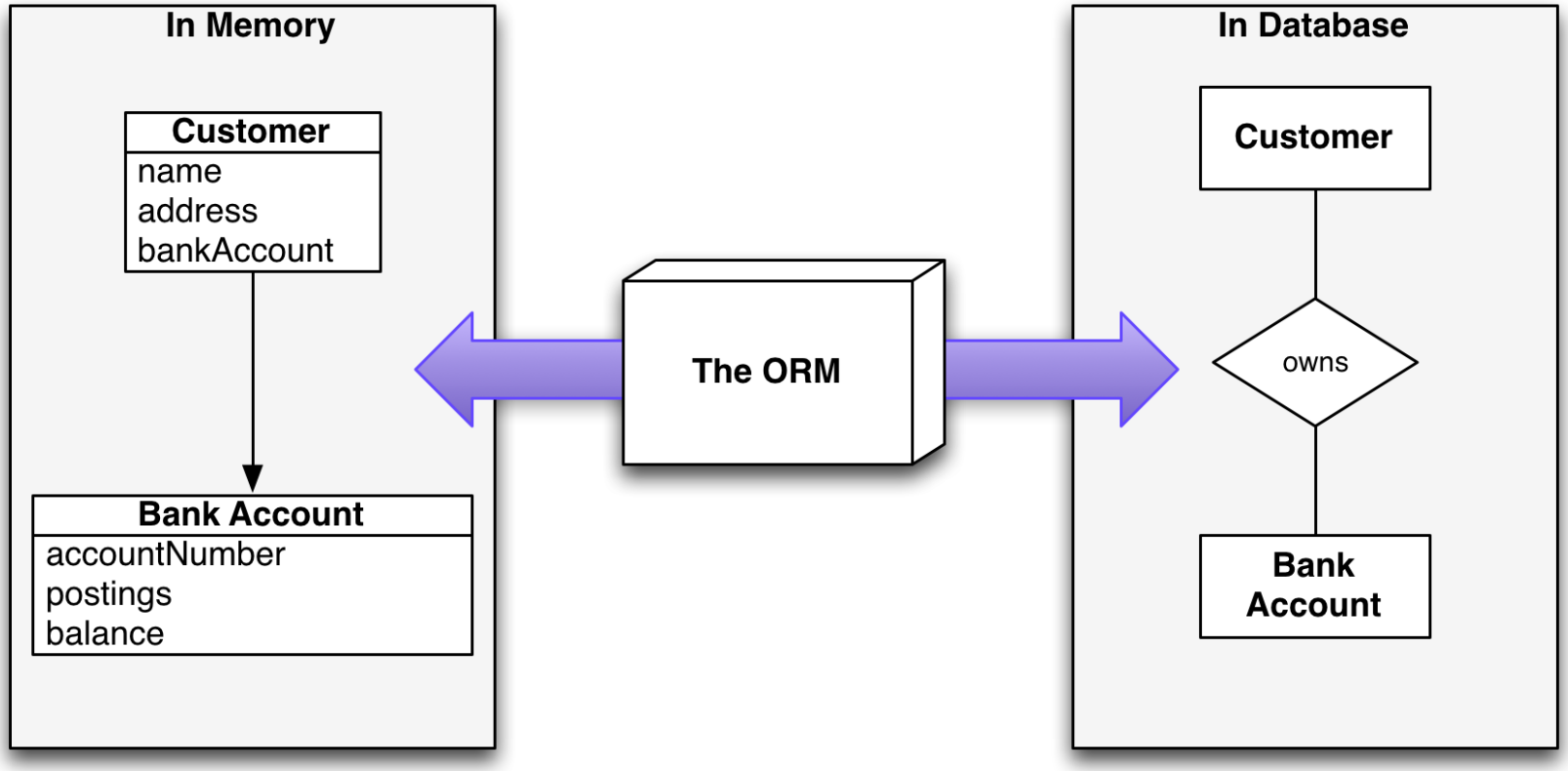
	id	title	length	rating	count	album_id	genre_id
	필터	필터	필터	필터	필터	필터	필터
1	1	Black Dog	297	5	0	2	1
2	2	Stairway	482	5	0	2	1
3	3	About to Rock	313	5	0	1	2
4	4	Who Made Who	207	5	0	1	2

	title	name	title	name
1	Black Dog	Led Zepplin	IV	Rock
2	Stairway	Led Zepplin	IV	Rock
3	About to Rock	AC/DC	Who Made Who	Metal
4	Who Made Who	AC/DC	Who Made Who	Metal

☑	Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
☑	Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
☑	Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
☑	For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
☑	Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
☑	Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
☑	Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
☑	Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23

2. ORM, RE

2.1. ORM 소개



■ What is ORM?

○ ORM stands Object Relational Mapping

○ Programming technique for converting data between incompatible type systems using object-oriented programming languages

■ Why use ORM?

○ Mismatch between the object model and the relational database

- ➢ RDBSs represent data in tabular format
- ➢ Object-Oriented languages represent data as an interconnected graph of objects

○ ORM frees the programmer from dealing with simple repetitive database queries

○ Automatically mapping the database to business objects

○ Programmers focus more on business problems and less with data storage

○ The mapping process can aid in data verification and security before reaching the database

○ ORM can provide a caching layer above the database

■ Disadvantages

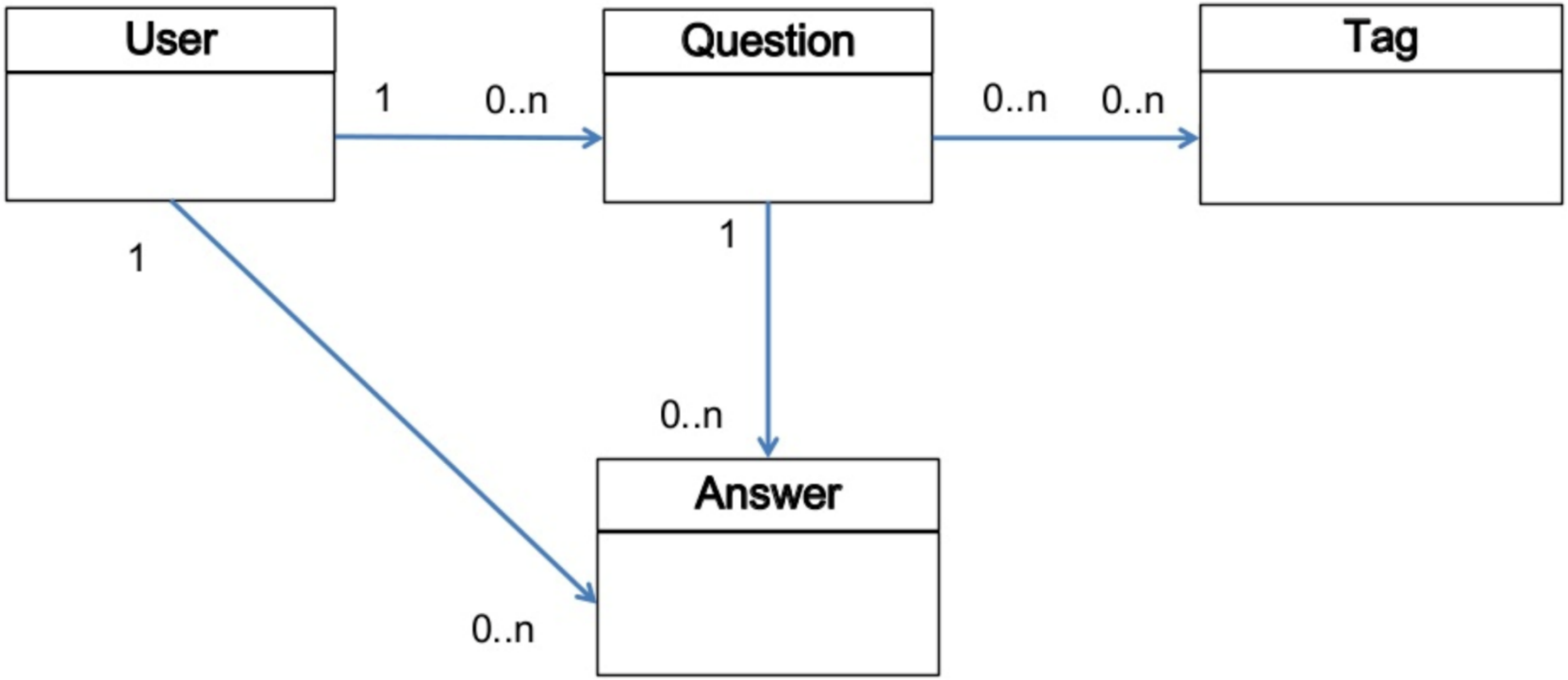
○ Potentially increasing processing overhead

■ 예제

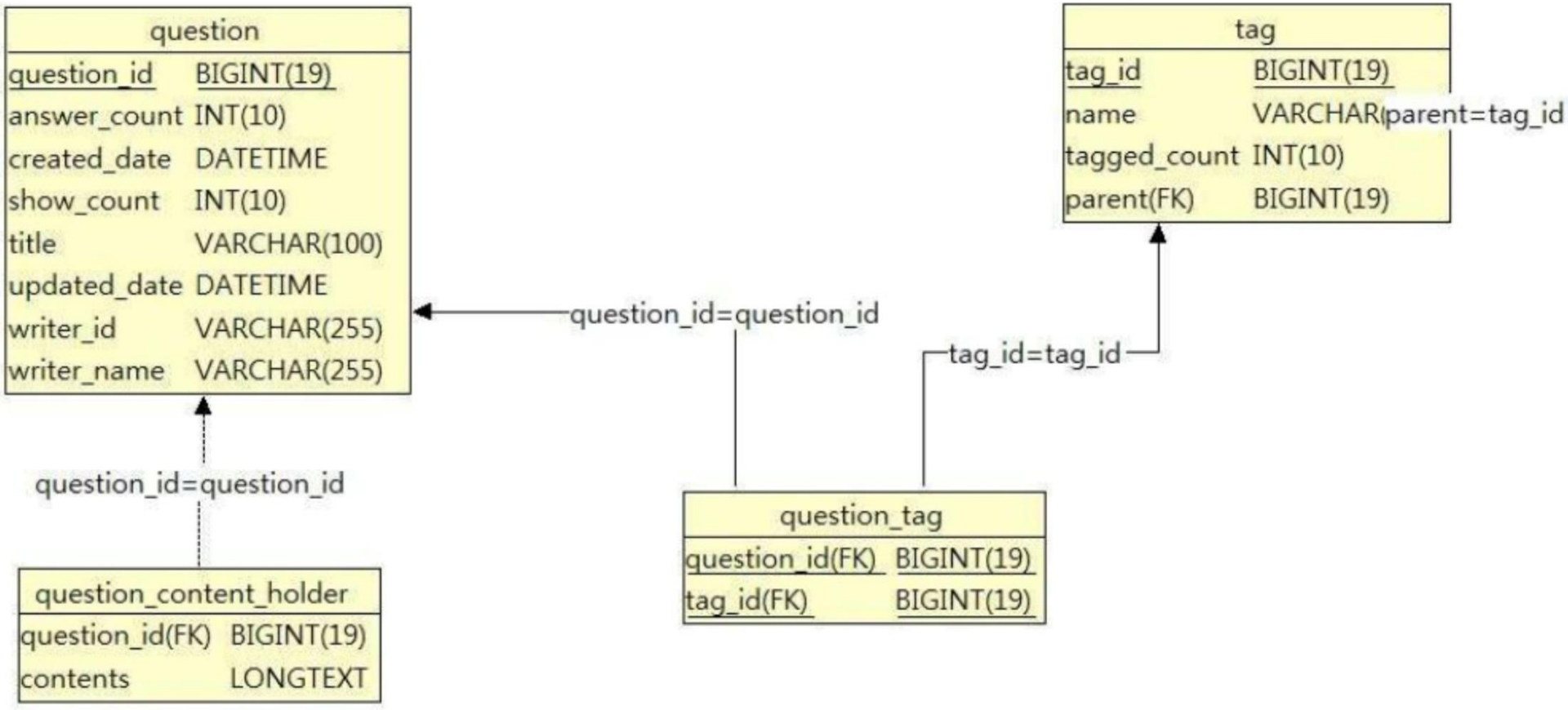
○ 요구사항

- 사용자는 질문 할 수 있어야 한다
- 질문에 대한 답변을 할 수 있어야 한다
- 질문할 때 태그를 추가할 수 있어야 한다
- 태그는 태그 풀에 존재하는 태그만 추가할 수 있다
- 태그가 추가될 경우 해당 태그 수는 +1 증가, 삭제될 경우 해당 태그 수는 -1 감소되어야 한다

○ 개체-관계 모델(ER Model)



○ 데이터베이스 설계


question
question_id BIGINT(19)
answer_count INT(10)
created_date DATETIME
show_count INT(10)
title VARCHAR(100)
updated_date DATETIME
writer_id VARCHAR(255)
writer_name VARCHAR(255)
tag
tag_id BIGINT(19)
name VARCHAR
tagged_count INT(10)
parent(FK) BIGINT(19)
parent=tag_id
question_id=question_id
tag_id=tag_id
question_id=question_id
question_tag
question_id(FK) BIGINT(19)
tag_id(FK) BIGINT(19)
question_content_holder
question_id(FK) BIGINT(19)
contents LONGTEXT

○ 질문 추가 시

- 질문 등록
- 태그 등록
  - 태그 풀에서 해당 태그 ID 가져오기
  - 태그 풀에서 해당 태그 수 증감하기

```
INSERT INTO question VALUES(?, ?, ?, ?, ?); → question_id = 1

SELECT tag_id, name FROM tag WHERE name="python"; → tag_id = 1
SELECT tag_id, name FROM tag WHERE name="alchemy"; → tag_id = 2

INSERT INTO question_tag VALUES(1, 1);
INSERT INTO question_tag VALUES(1, 2);

UPDATE tag SET tagged_count = tagged_count + 1 where name="python";
UPDATE tag SET tagged_count = tagged_count + 1 where name="alchemy";
```

○ 질문 수정 시

- 질문 수정
- 태그 수정
  - 태그 풀에서 해당 태그 ID 가져오기
  - 태그 풀에서 해당 태그 수 증감하기

```
UPDATE question SET title=?, contents=? WHERE question_id = 1;

SELECT tag_id, name FROM tag WHERE name="python"; → tag_id = 1
SELECT tag_id, name FROM tag WHERE name="alchemy"; → tag_id = 2
SELECT tag_id, name FROM tag WHERE name="orm"; → tag_id = 3

INSERT INTO question_tag VALUES(1, 3);
DELETE FROM question_tag where question_id=1 and tag_id=2;

UPDATE tag SET tagged_count = tagged_count - 1 where name="alchemy"
UPDATE tag SET tagged_count = tagged_count + 1 where name="orm"
```

*테이블 간의 관계보다 **데이터베이스 대한 처리**에 집중*

○ 객체-관계로 접근

```
public class Question {
    private long qid;
    private string title;
    [ … ]

    public processTags(string tag) {
    }
}
```

```
public class Tag {
    private long tid;
    private string tag;
    [ … ]

    public add(string tag) {
    }
}
```

INSERT INTO question VALUES(?, ?, ?, ?, ?); → question(1)
SELECT tag_id, name FROM tag WHERE name="python"; → tag.getByName("python")

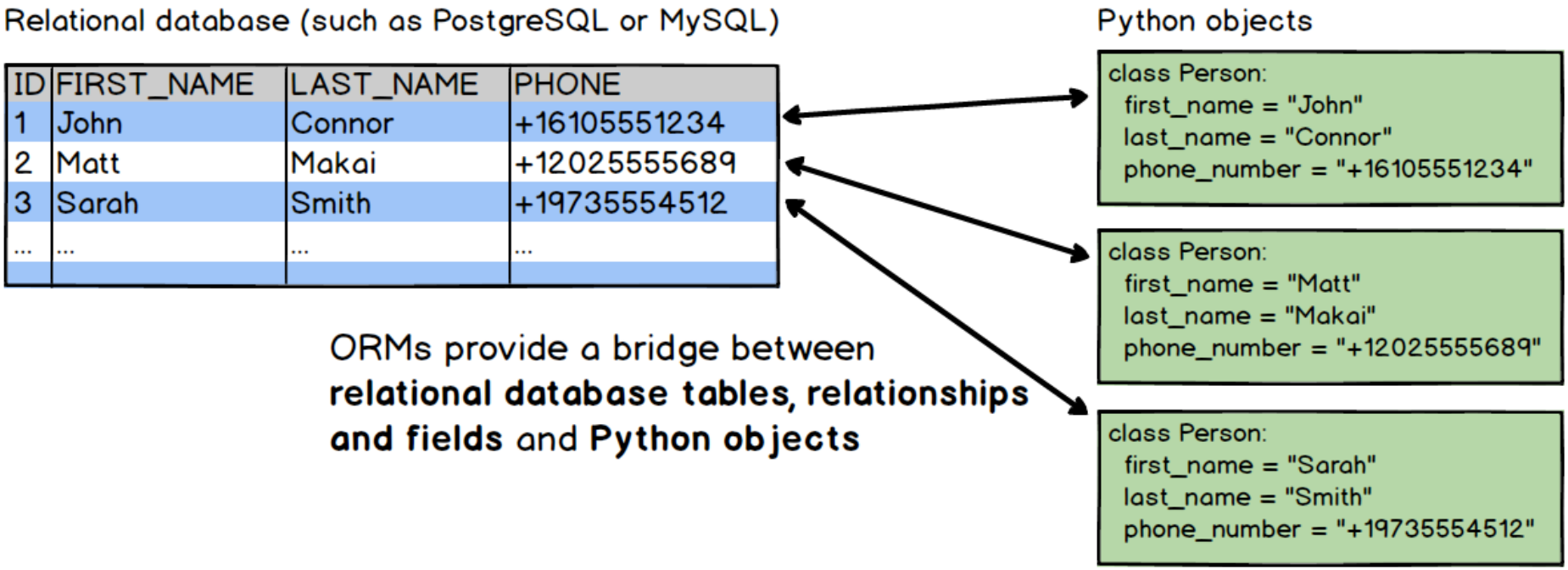
■ ORM in Python

○ ORM allows a developer to write Python code instead of SQL to create, read, update and delete

- ➢ Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures

○ ORMs make it theoretically possible to switch an application between various relational databases

- ➢ In practice however, it's best to use the same database for local development as is used in production



■ ORMs

○ Python ORM libraries are not required for accessing relational databases

➢ In fact, the low-level access is typically provided by another library called a *database connector*

web framework	None	Flask	Flask	Django
ORM	SQLAlchemy	SQLAlchemy	SQLAlchemy	Django ORM
database connector	(built into Python stdlib)	MySQL-python	psycopg	psycopg
relational database	SQLite	MySQL	PostgreSQL	PostgreSQL

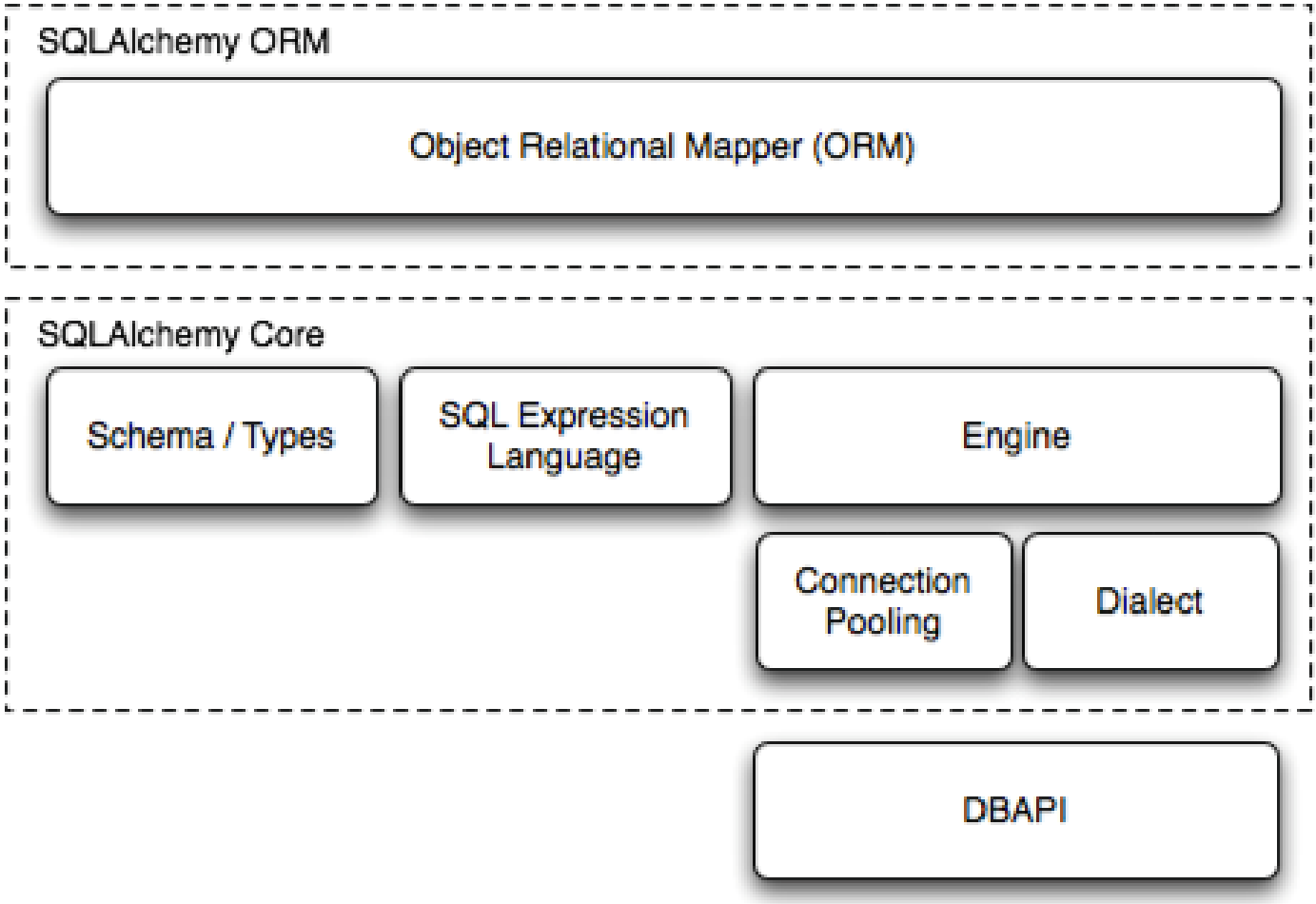
2.2. SQLAlchemy

■ What is SQLAlchemy?

○ SQLAlchemy is a well-regarded database toolkit and ORM implementation written in Python

○ SQLAlchemy provides a generalized interface for creating and executing database-agnostic code without needing to write SQL statements.

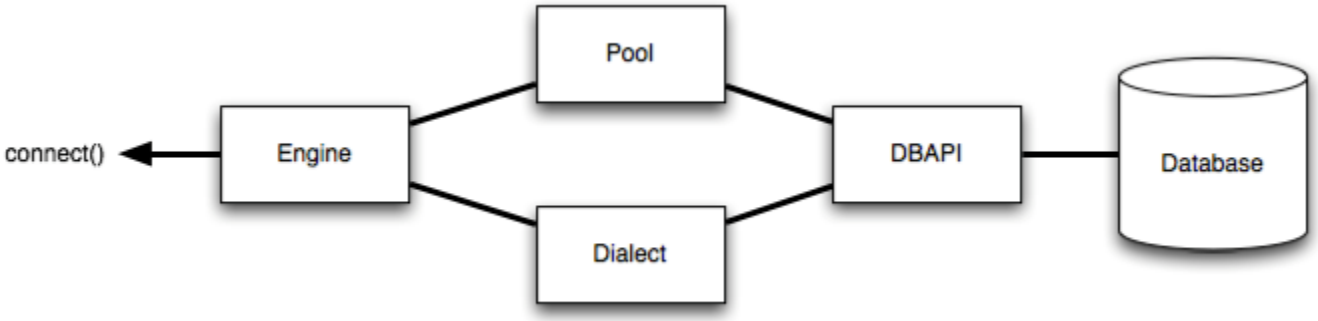
■ SQLAlchemy architecture



■ SQLAlchemy CORE

○ Engine

- ➢ starting point for any SQLAlchemy application
- ➢ a registry which provides connectivity to a particular database server



○ Dialect

- ➢ communicate with various types of DBAPI implementations and databases
- ➢ interprets generic SQL and database commands in terms of specific DBAPI and database backend
  - – Firebird, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SQLite, Sybase

○ Connection Pool

- ➢ holds a collection of database connections in memory for fast re-use

■ Connecting

○ create_engine

```
sqlalchemy.create_engine(*args, **kwargs)
```

```
dialect+driver://username:password@host:port/database
```

```
# sqlite://<nohostname>/<path>
# where <path> is relative:
engine = create_engine('sqlite:///foo.db')
```

○ 예제

```
import sqlalchemy

sqlalchemy.__version__
```

```
from sqlalchemy import create_engine

engine = create_engine("sqlite://", echo=True)
#engine = create_engine("sqlite:///:memory:", echo=True)
#engine = create_engine("sqlite:///test.db", echo=True)

print(engine)
```

- Lazy connecting
- The echo flag is a shortcut to setting up SQLAlchemy logging

■ Create

○ Table

```
class sqlalchemy.schema.Table(*args, **kw)
```

- Table object constructs a unique instance of itself based on its name and optional schema name within the given MetaData object
- name, metadata, columns, constraint, …

○ Column

```
class sqlalchemy.schema.Column(*args, **kwargs)
```

- Represents a column in a database table
- name, type, constraint, authoincreament, default, nullable, …

○ MetaData

```
class sqlalchemy.schema.MetaData
```

- A collection of Table objects and their associated schema constructs
- Holds a collection of Table objects as well as an optional binding to an Engine or Connection. If bound, the Table objects in the collection and their columns may participate in implicit SQL execution
- Table objects themselves are stored in the MetaData.tables dictionary

○ 예제

```
from sqlalchemy import Table, Column, Integer, String, MetaData, ForeignKey

metadata = MetaData()
users = Table('users', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String),
    Column('fullname', String),
)

addresses = Table('addresses', metadata,
    Column('id', Integer, primary_key=True),
    Column('user_id', None, ForeignKey('users.id')),
    Column('email_address', String, nullable=False)
)

metadata.create_all(engine)
```

■ Insert

○ insert

insert(*values=None, inline=False, **kwargs*)

- Represents an INSERT construct
- generate an insert() construct against this TableClause

○ compile

compile(*bind=None, dialect=None, **kw*)

- Compile this SQL expression
- Return value is a Compiled object
- Compiled object also can return a dictionary of bind parameter names and values using the params accessor

○ 예제

```
insert = users.insert()
print(insert)

insert = users.insert().values(name='kim', fullname='Anonymous, Kim')
print(insert)

insert.compile().params
```

■ Executing

○ Connection

```
class sqlalchemy.engine.Connection
```

- Provides high-level functionality for a wrapped DB-API connection
- Provides execution support for string-based SQL statements as well as ClauseElement, Compiled and DefaultGenerator objects

○ execute

```
execute(object, *multiparams, **params)
```

- Executes a SQL statement construct and returns a ResultProxy

○ ResultProxy

```
class sqlalchemy.engine.ResultProxy(context)
```

- Wraps a DB-API cursor object to provide easier access to row columns

○ 예제

```
conn = engine.connect()
conn

insert.bind = engine
str(insert)

result = conn.execute(insert)

result.inserted_primary_key
```

➢ execute의 params 사용

```
insert = users.insert()

result = conn.execute(insert, name="lee", fullname="Unknown, Lee")

result.inserted_primary_key
```

➢ DBAPI의 executemany() 사용

```
conn.execute(addresses.insert(), [
    {"user_id":1, "email_address":"anonymous.kim@test.com"},
    {"user_id":2, "email_address":"unknown.lee@test.com"}
])
```

■ Select

○ select

`sqlalchemy.sql.expression.select`

- Construct a new Select
- columns, whereclause, from_obj, group_by, order_by, ...

○ 예제

```
from sqlalchemy.sql import select

query = select([users])
result = conn.execute(query)

for row in result:
    print(row)
```

```
result = conn.execute(select([users.c.name, users.c.fullname]))

for row in result:
    print(row)
```

■ ResultProxy

○ fetchone

fetchone()

➢ Fetch one row, just like DB-API cursor.fetchone()

○ fetchall

fetchall()

➢ Fetch all rows, just like DB-API cursor.fetchall()

○ 예제

```
result = conn.execute(query)

row = result.fetchone()
print("id -", row["id"], ", name -", row["name"], ", fullname -", row["fullname"])

row = result.fetchone()
print("id -", row[0], ", name -", row[1], ", fullname -", row[2])

result = conn.execute(query)
rows = result.fetchall()

for row in rows:
    print("id -", row[0], ", name -", row[1], ", fullname -", row[2])

result.close()
```

■ Conjunctions

○ 예제

```
from sqlalchemy import and_, or_, not_

print(users.c.id == addresses.c.user_id)

print(users.c.id == 1)

print((users.c.id == 1).compile().params)

print(or_(users.c.id == addresses.c.user_id, users.c.id == 1))

print(and_(users.c.id == addresses.c.user_id, users.c.id == 1))

print(and_(
        or_(
            users.c.id == addresses.c.user_id,
            users.c.id == 1
        ),
        addresses.c.email_address.like("a%")
    )
)

print((
        (users.c.id == addresses.c.user_id) |
        (users.c.id == 1)
     ) & (addresses.c.email_address.like("a%")))
```

■ Selecting

○ 예제

```
result = conn.execute(select([users]).where(users.c.id==1))
for row in result:
    print(row)

result = conn.execute(select([users, addresses]).where(users.c.id==addresses.c.user_id))
for row in result:
    print(row)

result = conn.execute(select([users.c.id, users.c.fullname, addresses.c.email_address])
                      .where(users.c.id==addresses.c.user_id))
for row in result:
    print(row)

result = conn.execute(select([users.c.id, users.c.fullname, addresses.c.email_address])
                      .where(users.c.id==addresses.c.user_id)
                      .where(addresses.c.email_address.like("un%")))
for row in result:
    print(row)
```

■ Join

○ join

```
join(right, onclause=None, isouter=False, full=False)
```

➢ Return a Join from this FromClause to another FromClause

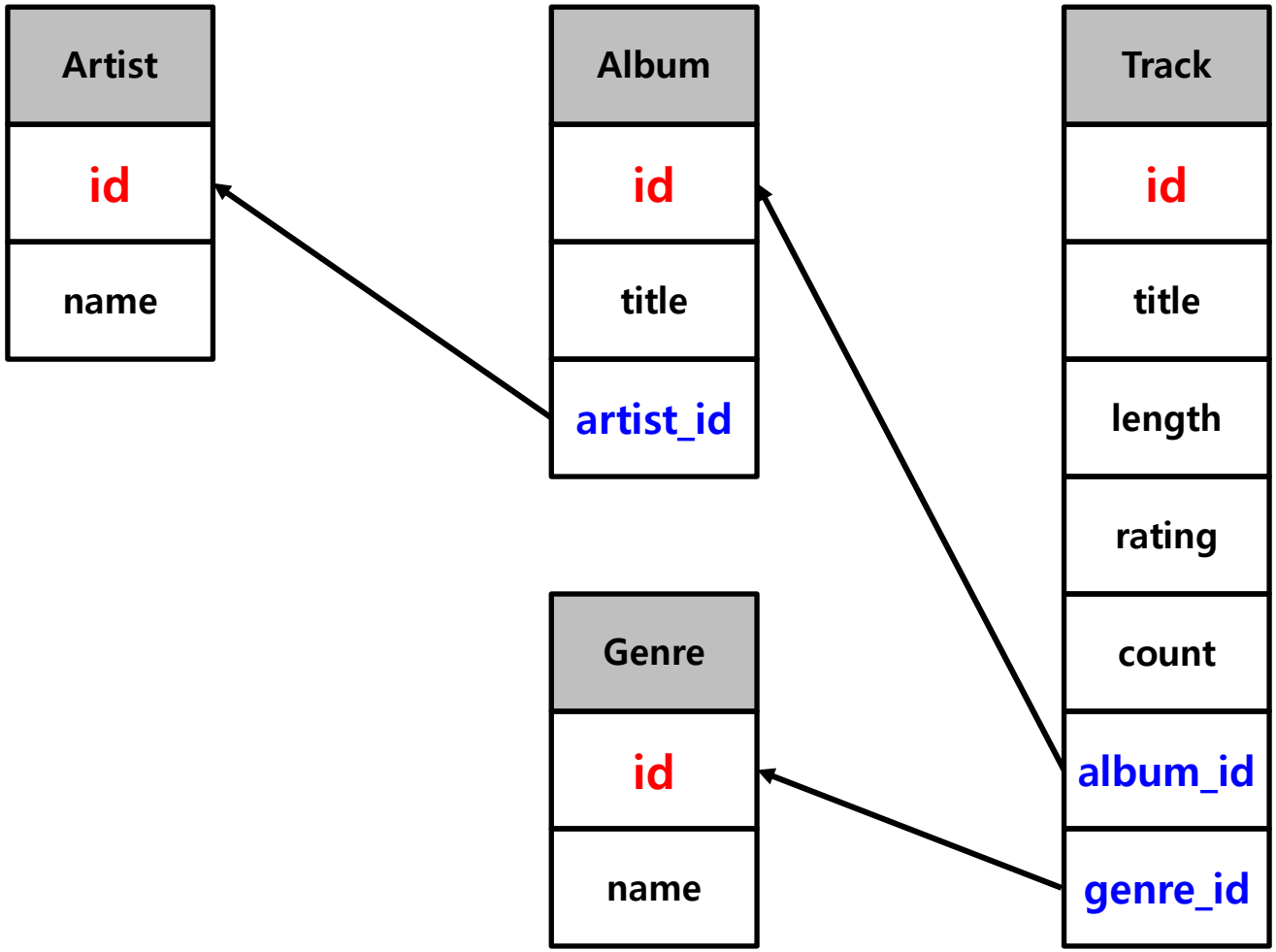
○ 예제

```
from sqlalchemy import join

print(users.join(addresses))

print(users.join(addresses, users.c.id == addresses.c.user_id))
```

➢ ON condition of the join, as it's called, was automatically generated based on the ForeignKey object

```
query = select([users.c.id, users.c.fullname, addresses.c.email_address]) \
        .select_from(users.join(addresses))

result = conn.execute(query).fetchall()
for row in result:
    print(row)
```

■ 예제


Artist
id
name
Album
id
title
artist_id
Track
id
title
length
rating
count
album_id
genre_id
Genre
id
name

○ Create

```
artist = Table("Artist", metadata,
               Column("id", Integer, primary_key=True),
               Column("name", String, nullable=False),
               extend_existing=True)

album = Table("Album", metadata,
              Column("id", Integer, primary_key=True),
              Column("title", String, nullable=False),
              Column("artist_id", Integer, ForeignKey("Artist.id")),
              extend_existing=True)

genre = Table("Genre", metadata,
              Column("id", Integer, primary_key=True),
              Column("name", String, nullable=False),
              extend_existing=True)

track = Table("Track", metadata,
              Column("id", Integer, primary_key=True),
              Column("title", String, nullable=False),
              Column("length", Integer, nullable=False),
              Column("rating", Integer, nullable=False),
              Column("count", Integer, nullable=False),
              Column("album_id", Integer, ForeignKey("Album.id")),
              Column("genre_id", Integer, ForeignKey("Genre.id")),
              extend_existing=True)

metadata.create_all(engine)
```

○ SHOW TABLES

```
tables = metadata.tables
for table in tables:
    print(table)

for table in engine.table_names():
    print(table)
```

○ Insert

```
conn.execute(artist.insert(), [
    {"name":"Led Zepplin"},
    {"name":"AC/DC"}
])

conn.execute(album.insert(), [
    {"title":"IV", "artist_id":1},
    {"title":"Who Made Who", "artist_id":2}
])

conn.execute(genre.insert(), [
    {"name":"Rock"},
    {"name":"Metal"}
])

conn.execute(track.insert(), [
    {"title":"Black Dog", "rating":5, "length":297, "count":0, "album_id":1, "genre_id":1},
    {"title":"Stairway", "rating":5, "length":482, "count":0, "album_id":1, "genre_id":1},
    {"title":"About to rock", "rating":5, "length":313, "count":0, "album_id":2, "genre_id":2},
    {"title":"Who Made Who", "rating":5, "length":297, "count":0, "album_id":2, "genre_id":2}
])
```

○ Select

```
artistResult = conn.execute(artist.select())
for row in artistResult:
    print(row)

albumResult = conn.execute(album.select())
for row in albumResult:
    print(row)

genreResult = conn.execute(genre.select())
for row in genreResult:
    print(row)

trackResult = conn.execute(track.select())
for row in trackResult:
    print(row)
```

○ Where

```
trackResult = conn.execute(select([track])
                           .where(
                               and_(track.c.album_id == 1, track.c.genre_id == 1)
                           )
                          )
for row in trackResult:
    print(row)
```

○ Update

```
from sqlalchemy import update

conn.execute(track.update().values(genre_id=2).where(track.c.id==2))
conn.execute(track.update().values(genre_id=1).where(track.c.id==3))
```

○ Where

```
trackResult = conn.execute(select([track])
                           .where(
                               and_(track.c.album_id == 1,
                                    or_(track.c.genre_id == 1,
                                        track.c.genre_id == 2,)
                                   )
                           )
                          )
for row in trackResult:
    print(row)
```

○ Join

```
print(track.join(album))

result = conn.execute(track
                      .select()
                      .select_from(track.join(album)))

for row in result.fetchall():
    print(row)

result = conn.execute(track
                      .select()
                      .select_from(track.join(album))
                      .where(album.c.id==1))

for row in result.fetchall():
    print(row)
```

○ Multiple Join

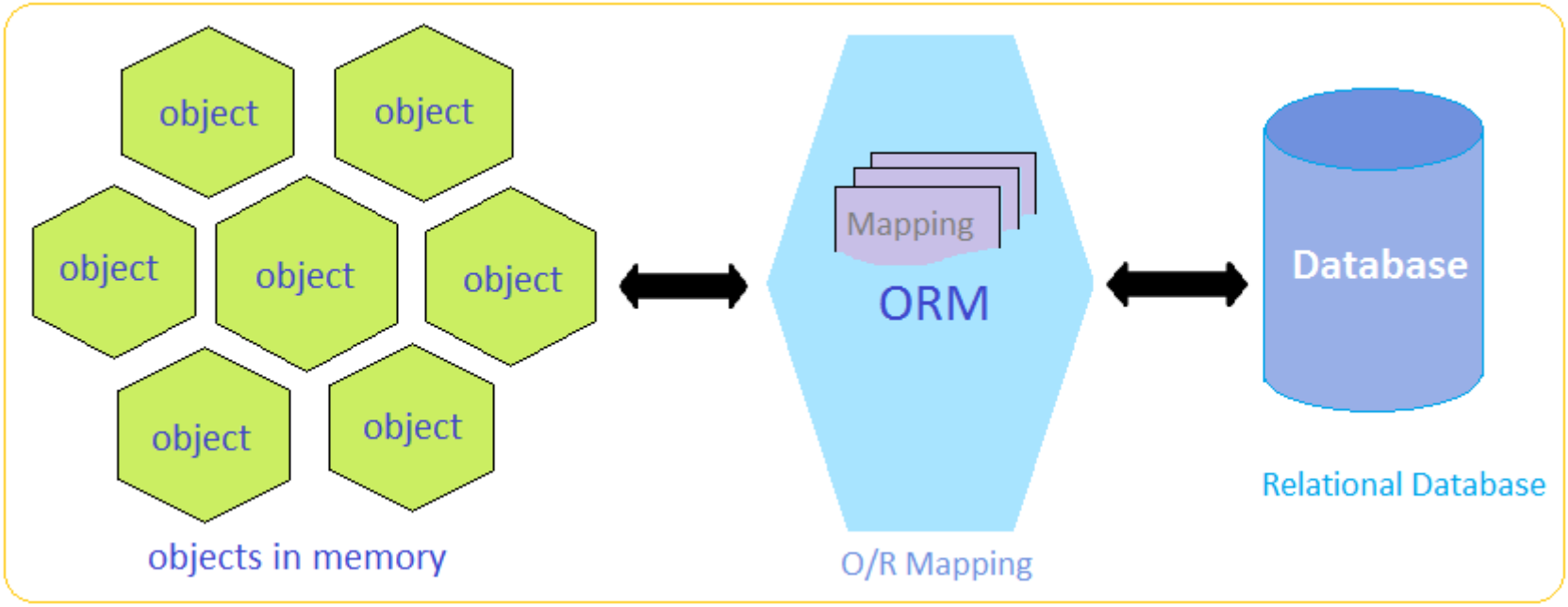
```
print(track.join(album))
print(track.join(album).join(genre))
print(track.join(album).join(artist))
print(track.join(album).join(genre).join(artist))

result = conn.execute(select([track.c.title, album.c.title, genre.c.name, artist.c.name])
                      .select_from(track.join(album).join(genre).join(artist)))

for row in result.fetchall():
    print(row)


result = conn.execute(track
                      .select()
                      .select_from(track.join(album).join(genre).join(artist))
                      .where(
                          and_(
                              genre.c.id==1,
                              artist.c.id==1,
                          )
                      )
                     )

for row in result.fetchall():
    print(row)
```

○ Open/Close

```
from sqlalchemy import create_engine, MetaData

engine = create_engine("sqlite:///alchemy_core.db", echo=True)
conn = engine.connect()

metadata = MetaData(bind=engine, reflect=True)
metadata.reflect(bind=engine)

for row in metadata.tables:
    print(row)
```

```
tables = metadata.tables
for table in tables:
    print(table)

#album

track = metadata.tables["Track"]
track

for row in conn.execute(track.select()).fetchall():
    print(row)

conn.close()
metadata.clear()
```

■ SQLAlchemy ORM



Data Mapping to Classes

■ Declare

○ declarative_base

```
sqlalchemy.ext.declarative.declarative_base
```

➢ Construct a base class for declarative class definitions

○ 예제

```
from sqlalchemy import create_engine

engine = create_engine("sqlite:///:memory:", echo=True)
```

```
from sqlalchemy.ext.declarative import declarative_base

base = declarative_base()
```

➢ declarative_base() callable returns a new base class from which all mapped classes should inherit

■ Create

○ declarative_base 상속 class 선언

```
class User(base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column("passwd", String)

    def __repr__(self):
        return "<T'User(name='%s', fullname='%s', password='%s')>" \
                % (self.name, self.fullname, self.password)
```

- a new Table and mapper() will have been generated
- table and mapper are accessible via __table__ and __mapper__ attributes

○ Schema

```
User.__table__
```

○ Create Table

```
base.metadata.create_all(engine)
```

○ Create Instance

```
kim = User(name="kim", fullname="anonymous, Kim", password="kimbap heaven")

print(kim)
print(kim.id)
```

■ Session

○ Session

class sqlalchemy.orm.session.**Session**

- Session establishes all conversations with the database and represents all the objects
- Manages persistence operations for ORM-mapped objects

○ sessionmaker

class sqlalchemy.orm.session.**sessionmaker**

- sessionmaker factory generates new Session objects when called
- sessionmaker class is normally used to create a top level Session configuration

○ 예제

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
session = Session()
```

■ Insert

○ add

```
add(instance, _warn=True)
```

- ➢ Place an object in the Session, persisted to the database on the next flush operation.
- ➢ Repeated calls to add() will be ignored

○ addall

```
add_all(instances)
```

- ➢ Add the given collection of instances to this Session

○ 예제

```
session.add(kim)

session.add_all([
    User(name="lee", fullname="unknown, Lee", password="123456789a"),
    User(name="park", fullname="nobody, Park", password="Parking in Park")
])
```

- ➢ Pending. no SQL has yet been issued and the object is not yet represented by a row in the database

■ Update

○ dirty

```
dirty
```

- The set of all persistent instances considered dirty
- Instances are considered dirty when they were modified but not deleted

○ is_modified

```
is_modified(instance, include_collections=True, passive=True)
```

- Return True if the given instance has locally modified attributes

○ 예제

```
kim.password = "password"

session.dirty

session.is_modified(kim)
```

■ Commit

○ commit

```
commit()
```

- ➢ Flush pending changes and commit the current transaction
- ➢ If no transaction is in progress, this method raises an InvalidRequestError

■ Select

○ query

```
class sqlalchemy.orm.query.Query(entities, session=None)
```

- ➢ ORM-level SQL construction object
- ➢ Query is the source of all SELECT statements generated by the ORM

○ 예제

```
for row in session.query(User):
    print(type(row))
    print(row.id, row.name, row.fullname, row.password)
```

○ filter

filter(**criterion*)

- Apply the given filtering criterion to a copy of this Query, using SQL expressions
- Allow you to use regular Python operators with the class-level attributes on your mapped class

○ filter_by

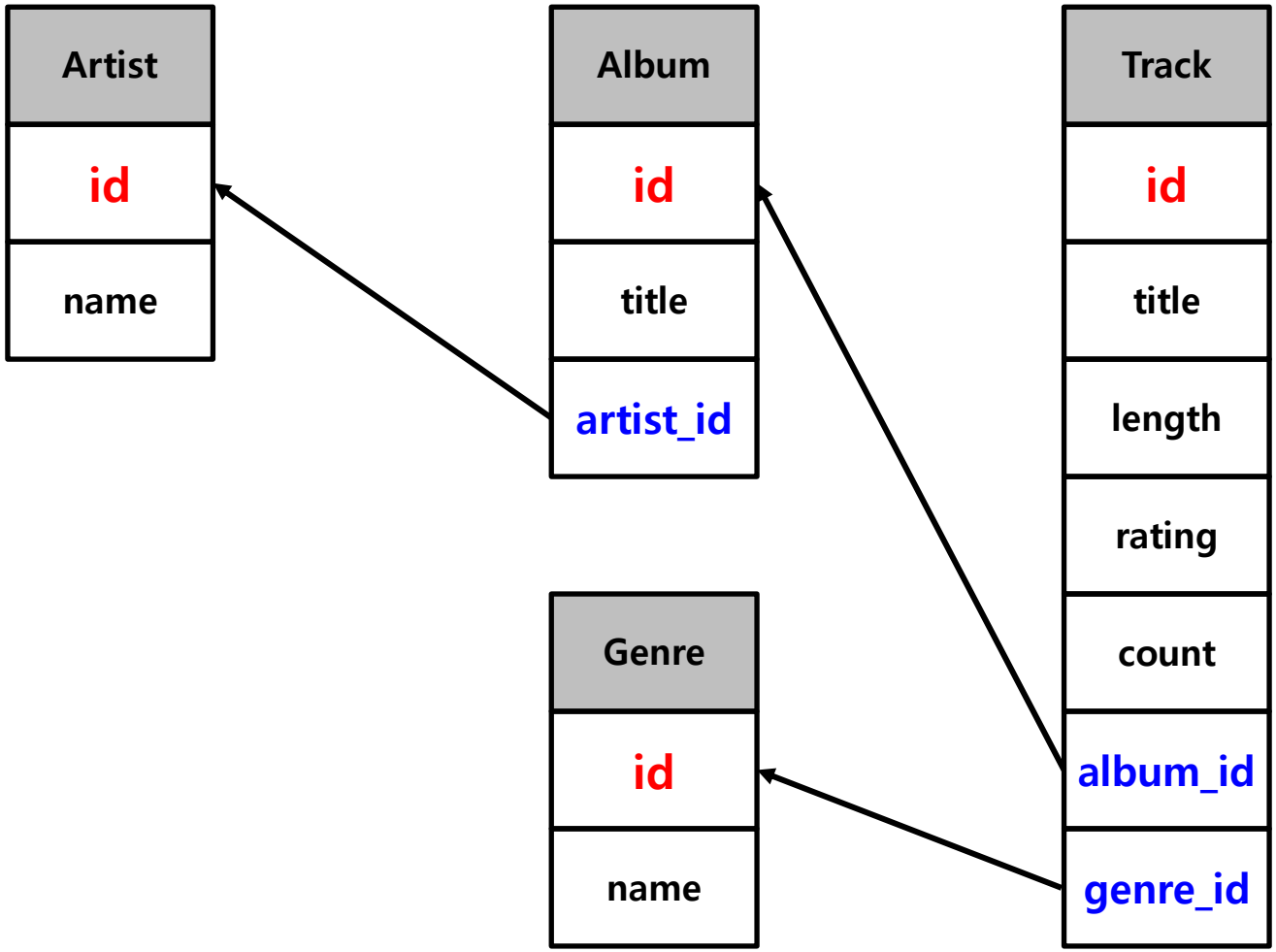
filter_by(***kwargs*)

- apply the given filtering criterion to a copy of this Query, using keyword expressions

○ 예제

```
for row in session.query(User.id, User.fullname).filter(User.name == "lee"):
    print(type(row))
    print(row.id, row.fullname)

for row in session.query(User.id, User.fullname).filter_by(name = "lee"):
    print(type(row))
    print(row.id, row.fullname)
```

■ 예제


Artist
id
name
Album
id
title
artist_id
Track
id
title
length
rating
count
album_id
genre_id
Genre
id
name

○ Create

```
class Artist(Base):
    __tablename__ = "Artist"

    id = Column(Integer, primary_key=True)
    name = Column(String)

class Album(Base):
    __tablename__ = "Album"

    id = Column(Integer, primary_key=True)
    title = Column(String)
    artist_id = Column(Integer, ForeignKey("Artist.id"))

class Genre(Base):
    __tablename__ = "Genre"

    id = Column(Integer, primary_key=True)
    name = Column(String)

class Track(Base):
    __tablename__ = "Track"

    id = Column(Integer, primary_key=True)
    title = Column(String)
    length = Column(Integer)
    rating = Column(Integer)
    count = Column(Integer)
    album_id = Column(Integer, ForeignKey("Album.id"))
    genre_id = Column(Integer, ForeignKey("Genre.id"))
```

○ Insert

```
artist1 = Artist(name="Led Zepplin")
artist2 = Artist(name="AC/DC")

session.add_all([artist1, artist2])
session.commit()

album = [Album(title="IV", artist_id=artist1.id),
         Album(title="Who Made Who", artist_id=artist2.id)]

session.add_all(album)
session.commit()

session.add_all([Genre(name="Rock"), Genre(name="Metal")])
session.commit()

album1 = session.query(Album).filter(Album.artist_id==artist1.id).one()
album2 = session.query(Album).filter(Album.artist_id==artist2.id).one()

genre1 = session.query(Genre).filter(Genre.name=="Rock").filter(Genre.id==1).one()
genre2 = session.query(Genre).filter(Genre.name=="Metal").filter(Genre.id==2).one()

track = [Track(title="Black Dog", rating=5, length=297, count=0, album_id=album1.id, genre_id=genre1.id),
         Track(title="Stairway", rating=5, length=482, count=0, album_id=album1.id, genre_id=genre2.id),
         Track(title="About to rock", rating=5, length=313, count=0, album_id=album2.id, genre_id=genre1.id),
         Track(title="Who Made Who", rating=5, length=297, count=0, album_id=album2.id, genre_id=genre2.id)]
session.add_all(track)
session.commit()
```

○ Join

```
result = session.query(Track.title, Album.title, Genre.name, Artist.name) \
         .select_from(Track) \
         .join(Album)\
         .join(Genre)\
         .join(Artist).all()

for row in result:
    print(row)

songList = [dict(Track=row[0], Album=row[1], Genre=row[2], Artist=row[3]) for row in result]

songList
```

■ Relationship

○ relationship

```
sqlalchemy.orm.relationship
```

- ➢ Provide a relationship between two mapped classes
- ➢ This corresponds to a parent-child or associative table relationship

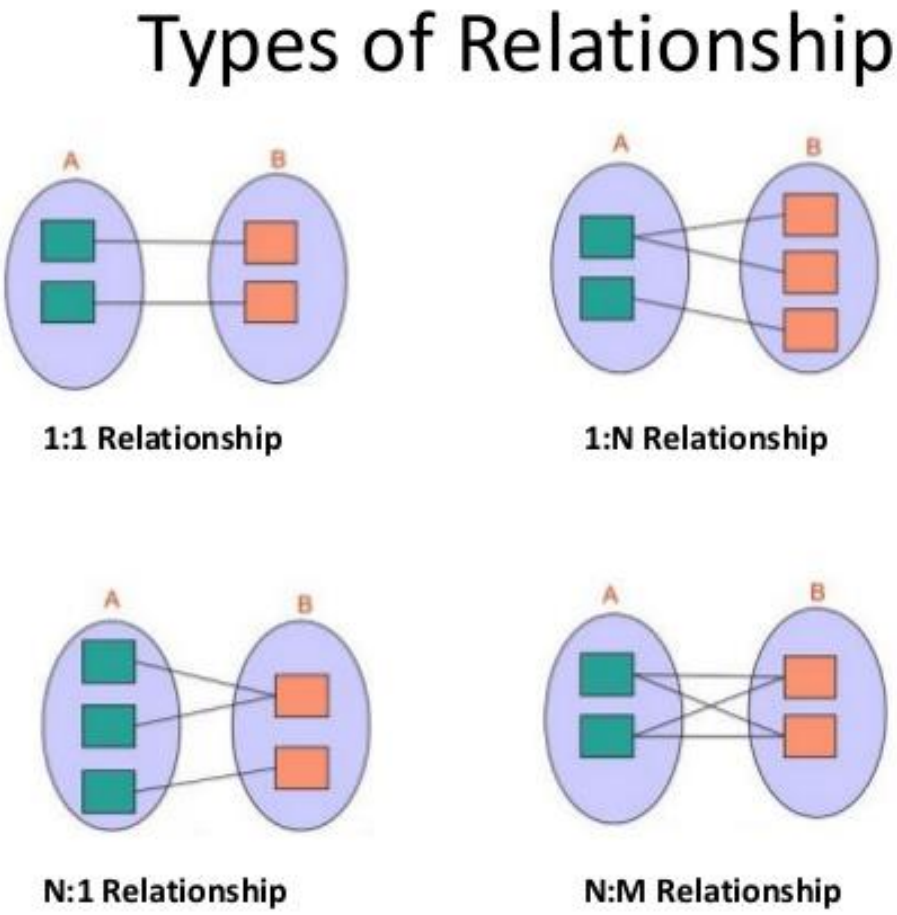
○ back_populates / backref

- ➢ indicates the string name of a property to be placed on the related mapper's class that will handle this relationship in the other direction

○ uselist

- ➢ a boolean that indicates if this property should be loaded as a list or a scalar

■ Relationship Model



■ 예제

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column("passwd", String)

    #addresses = relationship("Address", back_populates="user")
    #addresses = relationship("Address", back_populates="user", uselist=False)

    def __repr__(self):
        return "<User(name='%s', fullname='%s', password='%s')>" % (self.name, self.fullname, self.password)
```

```
class Address(Base):
    __tablename__ = "addresses"

    id = Column(Integer, primary_key=True)
    email_address = Column(String, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"))

    #user = relationship("User", back_populates="addresses", uselist=False)
    #user = relationship("User", back_populates="addresses")
    #user = relationship("User")

    def __repr__(self):
        return "<Address(email_address='%s')>" % self.email_address
```

■ 예제

```
class Artist(Base):
    __tablename__ = "Artist"

    id = Column(Integer, primary_key=True)
    name = Column(String)

    albumList = relationship("Album", back_populates="artist")

class Album(Base):
    __tablename__ = "Album"

    id = Column(Integer, primary_key=True)
    title = Column(String)
    artist_id = Column(Integer, ForeignKey("Artist.id"))

    artist = relationship("Artist", back_populates="albumList", uselist=False)
    trackList = relationship("Track", back_populates="album")

class Genre(Base):
    __tablename__ = "Genre"

    id = Column(Integer, primary_key=True)
    name = Column(String)

    trackList = relationship("Track", back_populates="genre")

class Track(Base):
    __tablename__ = "Track"

    id = Column(Integer, primary_key=True)
    title = Column(String)
    length = Column(Integer)
    rating = Column(Integer)
    count = Column(Integer)
    album_id = Column(Integer, ForeignKey("Album.id"))
    genre_id = Column(Integer, ForeignKey("Genre.id"))

    album = relationship("Album", back_populates="trackList", uselist=False)
    genre = relationship("Genre", back_populates="trackList", uselist=False)
```

Insert

```
track1 = Track(title="Black Dog", rating=5, length=297, count=0)
track2 = Track(title="Stairway", rating=5, length=482, count=0)
track3 = Track(title="About to rock", rating=5, length=313, count=0)
track4 = Track(title="Who Made Who", rating=5, length=297, count=0)
```

```
track1.album = track2.album = Album(title="IV")
track3.album = track4.album = Album(title="Who Made Who")
```

```
track1.genre = track3.genre = Genre(name="Rock")
track2.genre = track4.genre = Genre(name="Metal")
```

```
track1.album.artist = track2.album.artist = Artist(name="Led Zepplin")
track3.album.artist = track4.album.artist = Artist(name="AC/DC")
```

Select

```
print("Title: %s, Album: %s, Genre: %s, Artist: %s" %
      (track1.title, track1.album.title, track1.genre.name, track1.album.artist.name))
print("Title: %s, Album: %s, Genre: %s, Artist: %s" %
      (track3.title, track3.album.title, track3.genre.name, track3.album.artist.name))
```

```
print("TrackID: %d, AlbumID: %d, GenreID: %d, Artist: %d" %
      (track1.id, track1.album.id, track1.genre.id, track1.album.artist.id))

print("TrackID: %d, AlbumID: %d, GenreID: %d, Artist: %d" %
      (track3.id, track3.album.id, track3.genre.id, track3.album.artist.id))
```