2주차 Basic SQL - Schema, Single Table Queries... RDBMS 2022. 3. 13. 9:58
https://blog.naver.com/lsm980515/222671165424

SQL

SQL은 (Structured Query Language)의 약자로 구조화 질의어라고 한다.
즉 user가 필요하고 원하는 것을 RDBMS(관계형 데이터베이스 관리 시스템)에게 요청(쿼리) 하는걸 말한다.

추가적으로 RDBMS에는 Oracle, musql, mariadb, ms-sql 등이 있다.

특징
- 고급 언어로 인간이 이해하기 쉬운 프로그래밍 언어이다.
- 단순한 질의 기능뿐만 아니라 완전한 데이터 정의 기능과 조작 기능을 갖추고 있다.

SQL의 분듀

SQL의 명령어는 3가지로 분듀가 되며 아래와 같다.
DDL - Data Definition Language(데이터 정의어)
DML- Data Manipulation Language(데이터 조작어)
DCL - Data Control Language (데이터 제어어)

DDL은 table 및 attribute(속성)을 **생성, 변경, 제거할때 사용하는 명령어**이다.
DDL의 주요 명령어 - (CREATE, DROP, ALTER, RENAME, TRUNCATE)

DML은 table에서 tuples을 **수정, 삭제, 삽입 등은 할때 사용하는 명령어**이다.
여러 table에서 정보를 가져오는 동작도 할 수 있다.
DML의 주요 명령어 - (INSERT, UPDATE, DELETE, SELECT)

DCL은 데이터를 보호하고 관리하는 목적의 명령어이다.
DCL의 주요 명령어 - (GRANT, REVOKE,SET TRAMSACTION, BEGIN, COMMIT, ROLLBACK, SAVEPOINT, LOCK)

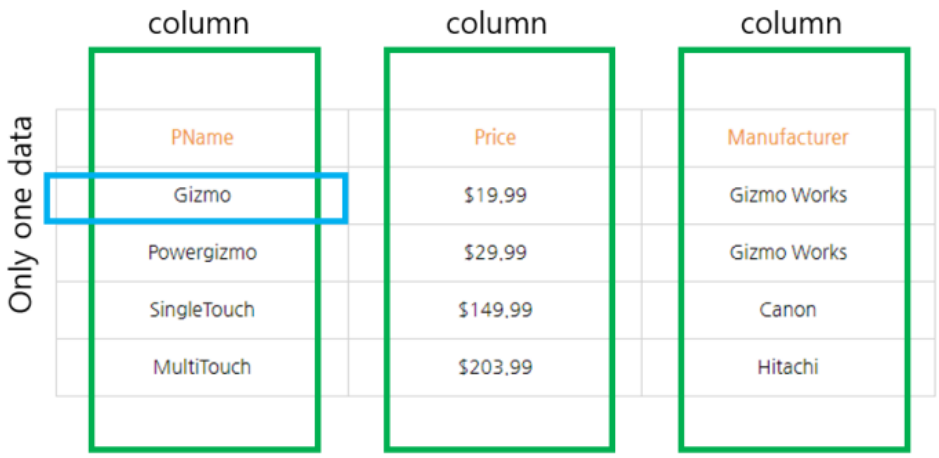
What is the Table in SQL

Product

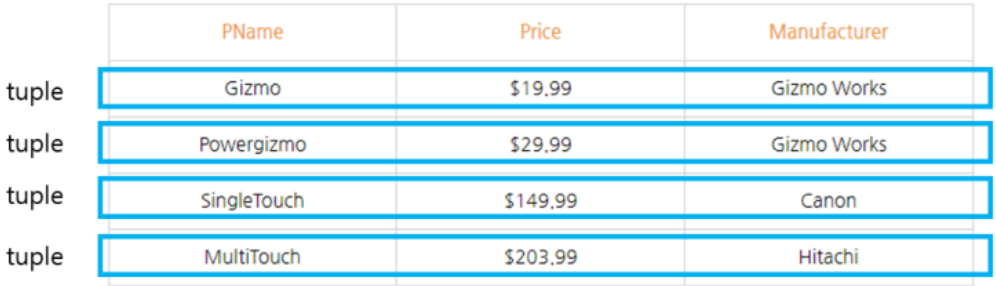
PName	Price	Manufacturer
Gizmo	$19.99	Gizmo Works
Powergizmo	$29.99	Gizmo Works

SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

reation = table = "표" 를 의미한다.
이러한 table의 특징은 일관적인(같이 묶을 수 있는) 특징을 가진 중복되지 않는 데이터를 담기위한 하나의 데이터 집합이다.
여기서 Product는 (entity set)개체 집합이라고도 한다.



filed = attribute = column = "열"을 의미함
위 그림에서 3개의 column이 존재하며, 다른말로 3개의 attribute가 있다. (PName , Price , Manufacturer)
column의 특징으로는 column아래의 칸에는 *단 하나의 data(정보)만 존재 가능*하다.
위에서 이러한 column들도 entity라고 했는데



tuple = record = row = "행"을 의미함.
위 그림에서는 4개의 tuple이 존재하며, 다른말로 4개의 row 또는 4개의 record가 있다.
tuple의 특징으로는 이 table의 모든 attributes을 가지고 있다.

List, Set, Multiset의 차이

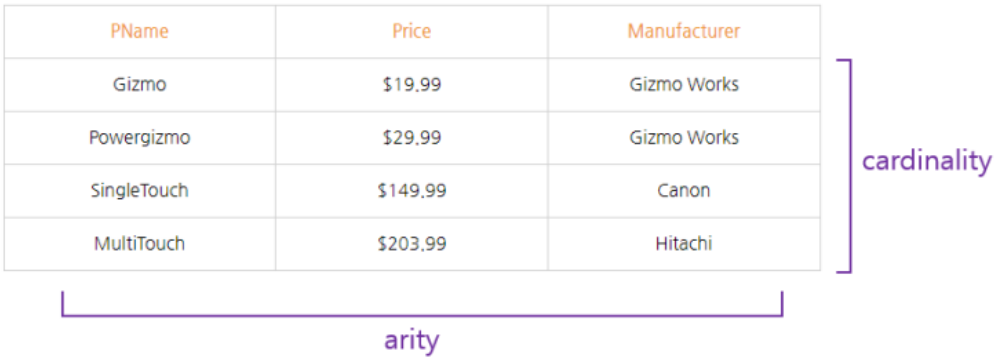
PName	Price	Manufacturer
Gizmo	$19.99	Gizmo Works
Powergizmo	$29.99	Gizmo Works
SingleTouch	$149.99	Canon

MultiTouch	$203.99	Hitachi

List의 특징은 중복을 허용하는 집합이다.
Set의 특징은 중복을 허용하지 않으며, 순서가 상관이 없는 집합이다.
Multiset의 특징은 중복을 허용하면서, 순서가 상관이 없는 집합이다.

위의 그림 중 column of Manufacturer을 보면 Manufacturer의 요소들로[Gizmo Works, Gizmo Works, Canon, Hitachi]순으로 가지는 것을 알 수 있고
list로 표현하면 -> [Gizmo Works, Gizmo Works, Canon, Hitachi]
Set로 표현하면 -> {Gizmo Works, Canon, Hitachi}
Multiset로 표현하면 -> {Gizmo Works, Gizmo Works, Canon, Hitachi}

Arity & Cardinality



degree = arity는 attribute의 개수이다.
즉 여기서는 3개의 attributes(PName, Price, Manufacturer)가 있으므로 arity of the relation는 3개이다.
cardinality는 tuple의 개수이다.
즉 여기서는 4개의 tuples가 있으므로 cardinality of the relation는 4개 이다.

Data Types in SQL

Atomic types:
characters: CHAR(20), VARCHAR(50)
Numbers : INT, GIGINT, SMALLINT, FLOAT
Others: MONEY, DATETIE, ...

atomic의 뜻은 **유일 함**을 뜻하며, 각 attributes의 values의 정보가 한개만 존재해야 된다는 뜻이다.

즉. 그림과 같이 칸 마다 정보가 한개만 있어야 된다는 뜻이다.

각 attributes의 표현되는 값의 형식(data type)은 지정하는 것을 **domain**이라고 함.
ex) 문자형, 정수형, 소수형, 등...

Table Schema & Key

Schema란 **데이터베이스의 관한 전반적인 structure를 기술**하는 것.
즉 schema = 구조 이다.

> DB를 구성하는 데이터 개체(entity), 속성(attribute), 관계(relation)에 대한 정의와 이들이 유지해야 될 제약(constraint) 조건 등에 관해 전박적으로 정의 하는 것을 말한다.

ex) Product(Pname: string, Price: float, Category:string, Manufacturer: string)

따라서 이 한줄이 schema가 될 수 있는데 풀어서 해석하면 아래와 같다.

Product라는 개체를 가진 table이 있고
그 안에 속성은 Pname, Price, Category 등이 있으며,
Pname에는 회원정보를, Price에는 가격을, Category에는 분듀항목의 데이터값을 저장할 것이며,
각 속성의 제약 조건은 Pname은 string, Price는 float type ...의 값을 가져야 된다.

Key는 한 **relation에서 각각의 tuple(행)은 유일하게 식별하기 위해 사용**되는 하나 or 그 이상의 attribute의 집합이다.

위의 Product에서의 key는 하나의 attribute가 될 수 없고, 두개 이상의 attribute가 모여야지만 하나의 key를 형성할 수 있다.
왜냐하면 Product의 존재하는 attributes 중 동일한 값이 안들어 가는 attribute가 없기 때문이다.
즉 Product에서는 한개의 attribute로는 유일성을 충족 시키지 못하기 때문에 유일성을 충족시키기 위해 두개 이상의 attributes을 하나의 key로 만들어야 한다.
여기서 Product에서는 두개의 attribute만 있으면 유일성과 최소성을 충족시키는 key를 만들 수 있으므로 3개 그 이상의 attributes들을 모아 하나의 키를 만들 필요가 없다.

※최소성 : tuple을 식별하는데 꼭 필요한 attribute로만 구성되어 있는 성질
※유일성 : 하나의 키로 tuple을 유일하게 식별 해낼 수 있는 성질

Student

Sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Student라는 table을 보면 모든 attributes가 key가 될 수 있는 것처럼 보인다.
왜냐하면 현재 중복되는 값을 가진 column이 하나도 없기 때문인데 실제로 key가 될 수 있는 attribute는 Sid 뿐이다.
왜냐하면 잠재적으로 Name과 Gpa는 다든 학생들의 데이터가 들어오면서 충분히 중복이 될 수 있는 정보이지만 Sid는 앞으로 들어올 데이터에 대해서도 **절대 중복이 일어 날 수 없는 atrribute이기 때문**이다.

Key의 종류

후보키(Candidate Key)

- relation을 구성하는 attributes 중에 tuple을 유일하게 식별할 수 있는 attribute의 집합을 의미
- 유일성과 최소성을 만족하는 key
- primary key가 될 수 있는 후보 key이다.
- ex) 학번, 주민번호 ....등

기본키(Primary Key)

- candidate key에서 선택된 Key
- NULL값이 들어 갈 수 없으며, 어떠한 값이라도 가져야함 (무결성)
- 기본키로 정의된 attribute에는 동일한 값이 중복되어 들어가 있으면 안됨.(무결성)

※무결성 : 데이터의 정확성과 일관성을 유지하고, 데이터에 결손과 부정합이 없음을 보증

슈퍼키(Super Key)

- 유일성을 만족하면서, 최소성은 만족시키지 못하는 Key
- Relation에 있는 atrributes들의 집합으로 구성된 키

※최소성을 만족시키지 못한다는 말의 의미는 attributes이 조합되어서 super key를 형성했는데 이 attributes들이 분리되어 각각의 attribute가 되었을때 각 attribute는 유일성이 있는 키로 사용하지 못한다는 말이다.

대체키(Surrogate Key or Alternate Key)

- Candidate Key가 2이상일 때 Candidate Key 중에서 Primary Key로 선택되지 않은 키

외래키(Foreign Key)

- relation들 간의 관계를 나타내기 위해 사용됨.
- 참조되는 relation의 primary key와 대응되어 relation간에 참조 관계를 표현하는 도구
- 외래키로 지정되면 참조 relation의 primary key에 없는 값은 입력할 수 없다.
- 외래키가 되는 attribute와 primary key가 되는 attribute의 이름은 달라도 된다.

ex) Studensts(Sid, string...) , Enrolled(Sid, string...) O
ex) Studensts(Sid, string...) , Enrolled(Second_Sid, string...) O

- 외래키 attribute의 domain과 참조되는 primary key의 domain은 반드시 같아야 한다.

왜냐하면 domain이 같아야 연관성 있는 tuple을 찾기 위한 비교 연산이 가능하기 때문.

- 반드시 외래키가 참조되는 relation의 pimary key가 아닌 다든 attribute를 참조하면 안된다.

왜냐하면 primary key가 아니면 tuple을 유일하게 구별하기 어렵기 때문이다.

Relations

Students

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Courses

cid	cname	credits
564	564-2	4
308	417	2

Enrolled

sid	cid	Grade
123	564	A

여기서 Students와 Enrolled 두개의 table만 일단 보자

두 개의 table이 공통으로 공유하는 attribute는 Sid이며,
sid는 유일성과 최소성을 만족하므로 primary key로 사용될 수 있으며
Enrolled relation이 Students relation을 참조하고 있으므로
students에 있는 Sid가 primary key이며 enrolled의 Sid는 foreign key가 된다.

※Primary key가 아닌데 attribute 값의 중복을 허용하지 않을려면 attribute에 unique라는 제약을 걸면된다.

NULL & NOT NULL

NULL의 의미는 **정보의 부재를 나타내기 위해 사용하는 값**으로, 이론적으로 아무것도 없는 특수한 데이터를 뜻함.
즉. 값이 정해지지 않음을 뜻함. 그렇다고 공집합{}은 아님

Student

Sid	Name	Gpa
123	Bob	3.9
143	Jim	NULL

위와 같은 Student라는 table을 만든다고 가정하면 Name과 Sid의 값은 꼭 필요로 한 값이고 Gpa는 값는 꼭 필요로 하지 않는 값이기 때문에 이 relation을 만들때
constraints를 명시하는데 Sid 와 Name은 null값이 올 수 없다 라는 식으로 cosntraints를 줄 수 있다.
또 우리가 원한다면 atrribute뿐만이 아니라 table 등에도 제약을 걸 수 있다.

※주의할 점으로 너무 많은 제약을 주게 되면 느려질 수 도 있다.

SQL Query

```
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```

위와 같은 것을 SFW query라고 부르며 기본형식이다.

해석 순서: FROM -> WHERE -> SELECT

SELECT 어떤 attribute를 선택할 것인지 입력해야된다.
FROM에는 어떤 relation인지 적어야되며
WHERE에는 conditions을 입력해야 함.
※SELECT와 FROM은 항상 필요하다.

Simple SQL Query: Selection

SQL Query에서 Selection은 어떠 조건에서 **relation의 tuples은 필터링하는 작업** 이다.

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	Gizmo Works
Powergizmo	$29.99	Gadgets	Gizmo Works
SingleTouch	$149.99	Photography	Canon

MultiTouch	$203.99	Household	Hitachi

```
SELECT *
FROM Product
WHERE Category = 'Gadgets'
```

위와 같은 명령어를 입력하면 ↓

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	Gizmo Works
Powergizmo	$29.99	Gadgets	Gizmo Works

지금과 같은 relation을 받아볼 수 있는데
위의 명령어를 해석 하면 SELECT에서는 '*'라고 입력이 되어 있는데
'*'의 의미는 모든 정보를 가져온다는 뜻이고,
SELECT에 *가 있으므로 모든 attributes를 가져온다는 의미가 된다.
FROM에서는 Product라고 입력이 되어 있는데
이 말의 의미는 Product라는 table에서 정보를 가져오겠다는 뜻이다.
WHERE Catergory = 'Gadgets'라는 구절은
Category라는 attribute에서 Gadgets라는 정보를 가지고 있는 애들만 가져오라는 조건이다.
따라서 이 모든 명령어들을 조합해보면
모든 attribute에 있는 정보들을 가져오는데 이 atrributes은 Product라는 table에 속한 attributes이고
attributes중 Catergory라는 attribute에 Gadgets라는 정보를 가지고 있는 정보들만 가져오라는 의미가 된다.

Simple SQL Query: Projection

Projection은 **table을 만드는 작업**인데 **몇몇 attributes들을 선택해서 table을 만든다.**

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	Gizmo Works
Powergizmo	$29.99	Gadgets	Gizmo Works
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Category = 'Gadgets'
```

위와 같은 명령어를 입력하면 ↓

PName	Price	Manufacturer
Gizmo	$19.99	Gizmo Works
Powergizmo	$29.99	Gizmo Works

지금과 같은 table을 새로 생성한다.
위의 명령어를 차례대로 해석해보면
SELECT PName, Price, Manufacturer은 PName, Price, Manufacturer라는 attributes의 정보들만 가져오겠다는 의미이고
FROM Product라는 구절은 위 attributes을 불러오는데 Product라는 table에서 attributes들을 불러오겠다는 의미이다.
WHERE Category = 'Gadgets'라는 구절의 의미는 Category라는 attribute에서 Gadgets라는 정보를 가지고 있는 애들만 가져오라는 조건이다.

만약 SELECT의 attributes의 순서가 바뀌면 바뀐 순서대로 table을 구성한다.
ex) SELECT PName, Manufacturer, Price

PName	Manufacturer	Price
Gizmo	Gizmo Works	$19.99
Powergizmo	Gizmo Works	$29.99

Notation

Product(PName, Price, Category, Manufacturer)라는 input schema가 존재할때

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Category = 'Gadgets'
```

새로 만들게 된 table의 output schema는 Answer(PName, Price, Manufacturer)이고
이 schema의 이름은 아직 정해지지 않았으며, 당연히 input schema와 다른 존재이다.

A Few Details

SQL의 commands(명령어)는 **대소문자를 구분하지 않는다.**
ex) SELEST = Select = select
ex) Product = product

하지만 **values의 대소문자는 구분한다.**
'Seattle' ≠ 'saettle'

그리고 **문자를 입력할때**에는 '' single quotes만 사용해야된다. ""X
ex) 'abc' O
ex) "abc" X

LIKE: Simple string Pattern Matching

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	Gizmo Works
Powergizmo	$29.99	Gadgets	Gizmo Works
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT PNAme, Manufacturer
FROM Product
WHERE PName = '%i%'
```

LIKE라는 키워드는 **Pattern Matching은 하는데 유용한 키워드**이다.
즉. 특정 패턴을 가지고 있는 조건을 검색할때 유용하다.
그리고 LIKE연산자의 특징으로 문자로 구성된 연산자라는 점이다.
문법 형식은 s LIKE p 이다.

p자리에는 두가지의 기호가 올 수 있는데 바로 %와 _이다.
%의 의미는 글자수 제한 없음을 의미함.
_의 의미는 패턴 개수를 조건으로 부여한다.

따라서 위의 코드예시를 보면
WHERE PName = '%i%' 라는 구절을 볼 수 있는데 이 문장을 해석해보면
PName이라는 attribute에서 i 앞뒤로 몇개의 문자가 있던지 상관없이 i라는 문자를 가지고 있는 값들을 가져와라는 뜻이다.
따라서 출력하게 되면

PName	Manufacturer
Gizmo	Gizmo Works
Powergizmo	Gizmo Works
SingleTouch	Canon
MultiTouch	Hitachi

위와 같이 출력된 이유는 SELECT에서 attributes을 PNAme, Manufacturer 두개만 지정했기 때문이고
WHERE PName = '%i%' 구절을 통해 PName이 가지고 있는 값들 중 i를 안가지고 있는 값이 없기 때문에 모든 값들이 출력이 된것을 확인 할 수 있다.

또 다든 예시로

```
SELECT PNAme, Manufacturer
FROM Product
WHERE Manufacturer = '_i%'
```

라고 명령어를 입력하게 되면 아래와 같은 table을 얻어 볼 수 있다.

PName	Manufacturer
Gizmo	Gizmo Works
Powergizmo	Gizmo Works
MultiTouch	Hitachi

위 명령어를 해석해보면
Product 라는 table에서 PNAme, Manufacturer 두개의 attribute를 가져오고, 그 조건 중 Manufactuere의 "vlaues중 i앞에 한글자만 있고, i 뒤로는 글자 개수 상관없다"라는 condition을 만족하는 values을 가져오라는 뜻이다.
따라서 i 앞에 G만 있는 Gimzo Works랑, i앞에 H만 있는 Hitachi를 가져오라는 의미가 된다.

DISTINCT: Eliminating Duplicates

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	Gizmo Works
Powergizmo	$29.99	Gadgets	Gizmo Works
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

DISTINCT는 **중복되는 값은 제거하고 출력하고 싶은때 사용**하며 SELECT 구문에서만 사용한다.
not FROM , not WHERE

```
SELECT DISTINCT Category
FROM Product
```

라는 명령어를 입력하게 되면

Category
Gadgets
Photography
Household

지금과 출력을 받아 볼 수 있다.
여기서 보면 Gadgets라는 value가 한개 사라진것을 확인 할 수 있는데 그 이유는 DISTINCT가 중복되는 value를 제거 했기 때문이다.

ORDER BY: Sorting the Results

ORDERD BY는 WHERE문 다음에 올 수 있으며 WHERE문이 없을 경우 FROM문 다음에 올 수 있다.

ORDERD BY는 **오름차순으로 정렬하고 싶은때 사용한다.**

※사실상 query문의 제일 마지막에 항상 위치함

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	Gizmo Works
Powergizmo	$29.99	Gadgets	Gizmo Works
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT PName, Price
FROM Product
WHERE price > 10
ORDER BY PName
```

PName	Price
Gizmo	$19.99
MultiTouch	$203.99
Powergizmo	$29.99
SingleTouch	$149.99

위의 명령어를 해석해 보면 Product table에서 Price가 10이상인 values을 출력하는데 출력하는 attribute는 PName과 Price 이고, PName은 오름차순으로 정렬하라 이다.
따라서 a의 가장 가까운 G가 1행에 위치하게 되고 PName 기준으로 오름차순으로 정렬됨을 볼 수 있다.

만약 내림차순으로 하고 싶다면 ORDER BY PName 뒤에 ORDER BY PName DESC를 붙여 주면 된다.

복수의 열인 경우 ORDER BY

복수의 열을 정렬하고 싶을때 ORDER BY구에서 ,로 열명을 구분해서 지정하면 된다.

Integer

First	Second
1	1
1	2
2	2

1	3

```
SELECT *
FROM Integer
ORDER BY First, Second DESC
```

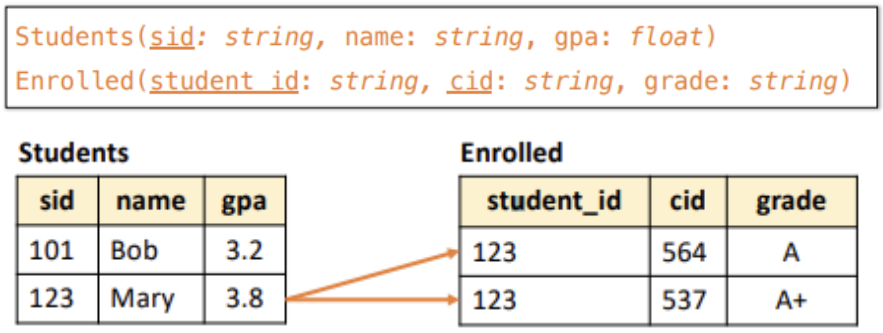
라는 명령어를 보면 Integer라는 relation에서 모든 attribute를 출력하는데 First는 오름차순, Second는 내림차순으로 정렬하라는 명령어이다.

First	Second
1	3
1	2
1	1
2	2

Second를 First의 하위호환이며 위 명령어를 통해 "1-3, 1-2, 1-1, 2-1"로 정렬할 수 있음.
그래서 먼저 First의 값들이 오름차순으로 정렬되고 Second의 값이 정렬이 되었는데
3번째 tuple을 보면 1-1인데 느낌적으로 1-2가 와야된다고 느끼는데 결과값은 1-1이다.
그이유는 "값이 같아 순서를 결정할 수 없는 경우에는 다음으로 지정한 열명을 기준으로 정렬하기 때문이다."

※만약 value값에 NULL값이 있다면 가장 먼저 나오거 가장 마지막에 나온다.

Foreign key



위에 key부분에 어느정도 자세히 설명해 놔서 개념적인 설명보다는 해당 이미지를 예시로 설명만 해보면

현재 Students, Enrolled라는relations이 있고, Enrolled는 Students relation에서 sid라는 attribute를 참조해서 만들어진 student_id라는 attribute를 가지고 있다.
그래서 Students의 isd를 그대로 참조하는 속성의 집합인 student_id가 Foreign key가 되어야 되지만,
사진을 보면 student_id는 중복을 허용해서 student_id혼자만으로는 foreign key를 형성할 수 없어서
Enrolled의 primary key인 cid와 student_id라는 attribute를 합쳐서 하나의 foreign key를 형성한다.

Client

Client-ID	Name	Age	grade	job	point
abcd	choi	40	platinum	doctor	5000

dcba	kang	35	gold	teacher	10000

primary key = Clinet-ID

Order

order_numb	order_ID	product	unit	price	order-date
103	abcd	orenge	5	20000₩	2022.1.01
150	dcba	apple	8	20000₩	2022.1.05

primary key = order_numb

이렇게 foreign key를 만드는 이유는 주문을 예시로 들면 쉽게 이해가 간다.

외래키는 relation간의 관계를 표현할때 유용하게 쓰이기 때문에 사용한다고 위에서 언급한적이 있다.
Client의 primary key는 Client-ID이고 이것을 참조해서 만들어진 attribute는 Order relation의 order_ID이므로 foreign key는 order_ID가 되는데
만약 Client relation에서 primary key인 Client-ID를 참조 안하고 Order relation을 만들었다고 생각해보자

order_numb	product	unit	price	order-date
103	orenge	5	20000₩	2022.1.01
150	apple	8	20000₩	2022.1.05

그러면 foreign key는 현재 없는 상태이고 Order relation에서 2022.1.01에 주문한 고객의 정보를 알고 싶을 때 해당 고객의 정보를 알 수 없는 불상사가 생긴다.
하지만 foreign key로 Client relation에서 Client-ID의 attribute 값들을 참조해 order_ID를 만들면 추후 특정 날짜에 주문한 고객의 정보를 알고 싶을때 손 쉽게 알 수 있다.

Declaring Foreign Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)

CREATE TABLE Enrolled(
     student_id CHAR(20),
     cid        CHAR(20),
     grade      CHAR(10),
     PRIMARY KEY (student_id, cid),
     FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

해당 그림은 students라는 table은 우선 만들어졌다고 가정하고, Enrolled라는 table을 출력하는 코드이다.

우선 구조를 살펴보면

```
CREATE TABLE [TABLE이름[(
    attributs_name1 [type()],
    attributs_name2 [type()],
    ...
    PRIMARY KEY (attribute, attribute, ....),
    FOREIGN KEY (primary key of the relation) REFERENCES ([the_relation_name(primary key
)
```

위와 같은 구조를 가지고 처음으로 테이블을 생성할때에는 당연히 FOREIGN KEY 문장은 생략해도 된다.

PRIMARY KEY부분에 (,)로 두개의 attribute가 있는 것을 확인 할 수 있는데 두개의 attribute가 합쳐져 한개의 primary key를 만든다는 의미이다. 두개의 primary key가 존재한다는 의미가 아님.
두개이상의 primary key가 될수있는 attribute가 있을 수는 있지만 그 중 하나만 primary key가 되는 것이다.

만약 foreign key에 새로운 값을 insert하고 싶으면 primary key에 그 value가 있어야지만 insert가능하다.

만약 참조된 relation(여기선 Students)의 primary key에 있는 value값들 중 하나를 삭제할려고 하면 다른 relations 중에 해당 value를 참조하고 있는지 유효성 검사를 해야하며
만약 참조하고 있다면 참조하는 relation의 foreign key에 삭제하고 싶은 value를 먼저 삭제한 후 참조된 relation의 value를 삭제하면 된다.

CROSS JOIN

	PName	Price	Category	Manufacturer
1	Gizmo	$19	Gadgets	GWorks
2	Powergizmo	$29	Gadgets	GWorks
3	SingleTouch	$149	Photography	Canon
4	MultiTouch	$203	Household	Hitachi

	Cname	Stock	Country
a	GWorks	25	USA
b	Canon	65	Japan
c	Hitach	15	Japan

교차 조인(CROSS JOIN - CARTESIN PRODUCT)

라는 Join 방법이 있는데 이 방법은 **두 table의 곱집합은 한 table은 만드는데 사용**하는 방법이다.
이 방법은 다른특별한 조건없이 table Product와 table Company의 각 행을 다 조합한 결과이다.

따라서 그 결과 아래와 같은 표가 만들어지며

PName	Price	Category	Manufacturer	Cname	Stock	Country
Gizmo	$19	Gadgets	GWorks	GWorks	25	USA
Gizmo	$19	Gadgets	Gworks	Canon	65	Japan
...	...	...	...	...	...	...

MultiTouch	$203	Household	Hitach	Hitach	15	Japna

각 행을 조합하는 경우의 수는 [(1-A), (1-B), (1-C), (2-A), ..., (4-C)]로 총12개 이다.

Joins

join은 RDBMS에서 **관련 있는 column을 기준으로 row를 합쳐주는 연산**이다.
또 두 개 이상의 table에 정보가 나뉘어져 있을 때 사용하며 table과 table을 connect하여 특정한 data를 얻고자 할 때 사용하는 연산자 이다.
join을 할때는 두 table을 **공통으로 묶어준 attribute가 필요함.**

Product

PName	Price	Category	Manufacturer
Gizmo	$19	Gadgets	GWorks
Powergizmo	$29	Gadgets	GWorks
SingleTouch	$149	Photography	Canon
MultiTouch	$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitach	15	Japan

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country='Japan'
       AND Price <= 200
```

```
SELECT PName, Price
FROM   Product
JOIN   Company ON Manufacturer = Cname
               AND Country='Japan'
WHERE  Price <= 200
```

Ex)일본에서 만들어진 $200 이하의 제품을 모두 찾고, 제품의 이름과 가격들을 출력하자 !라고 하면
위와 같이 query를 작성할 수 있는데 왼쪽은 암묵적 표현방법이고, 오른쪽은 명시적 표현방법이다.

먼저 암묵적 표현방법을 해석해보면
Product와 Company라는 두 개의 relation에서 데이터 합쳐서 하나의 table을 만들고 싶은데
Manufacture = Cnamed의 values이 일치해야하고,
Country라는 attribute에 "japan"이라는 데이터가 있고,
Price라는 attribute에서 200달러 이하인 데이터가 있으면
PName과 Price를 attributes로 가지는 table을 만들어라
라고 해석 가능하다.

명시적인 표현방법을 해석해보면
Poduct라는 relation을 기준으로 잡고 Company라는 relation을 서로 join하고 싶다는 의미가 된다.
Join할 때에는 두 relation을 공통으로 묶어줄 attribute가 필요한데 그게 Product relation에서는 Manufacturer 이고, Company relation에서는 Cname column이 되며 두 column을 비교해서 서로 값이 같은 row끼리 가로방

향으로 연결하라는 뜻이다.
그리고 Country 라는 attribute에서 'Japan'이라는 값을 가지고 있어야 한다.
또 다른 조건으로 Price attribute의 values중 200달러 이하인 value값을 가지고 있어야 한다.

그 결과 아래와 같은 table을 얻게 된다.
이 결과는 벤 다이어그램으로 치면 table A와 table B의 교집합과 같은 결과이다. (A ∩ B)

PName	Price
SingleTouch	$149

Tuple Variable Ambiguity in Multi-Table

만약 tables를 **join하고 싶은데** 각 table의 attribute 중 **attribute의 이름이 같은 경우**아래의 그림과 같은 두 가지 방법으로 해결가능하다.

```
Person(name, address, worksfor)
Company(name, address)
```

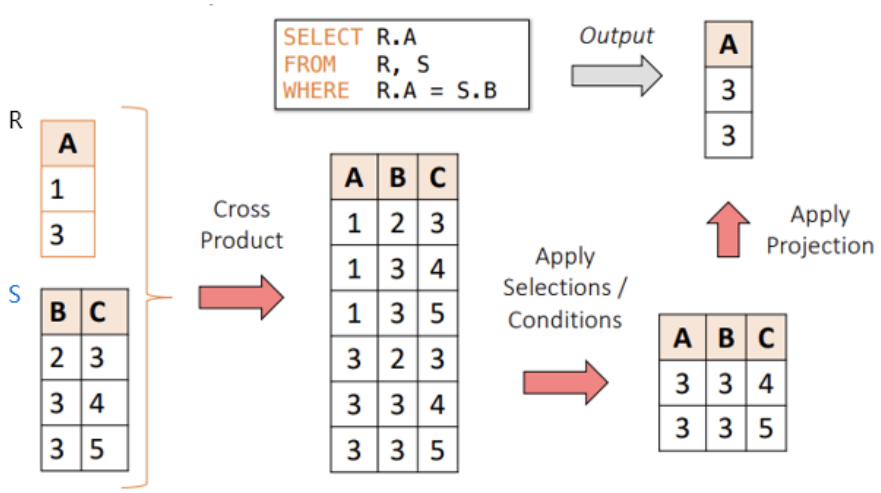
```
SELECT DISTINCT Person.name, Person.address
FROM            Person, Company
WHERE           Person.worksfor = Company.name
```

```
SELECT DISTINCT p.name, p.address
FROM            Person p, Company c
WHERE           p.worksfor = c.name
```

우선 SELECT라는 명령어를 사용하고 dot(.)을 이용해서 Person.name으로 표현가능한데
이 명령어의 의미는 Person이라는 table의 name attribute를 출력 하는데
중복되는 값은 제거하고 출력 하라는 뜻이다. <- ※이 말이 있는 이유는 DISTIMCT와 같이 사용했기 때문
또 WHERE문을 보면 dot(.)으로 위와 같이 어느 table의 어떤 attribute인지 명시하는 방법으로 위의 문제를 해결할 수 있다.

마지막 방법은 FROM문에 Person p라는 구절을 통해 Person이라는 table은 p와 동일하게 쓰일 수 있게 해준다는 뜻으로 이 방법을 이용해 명시적으로 나타내어 해결가능하다.

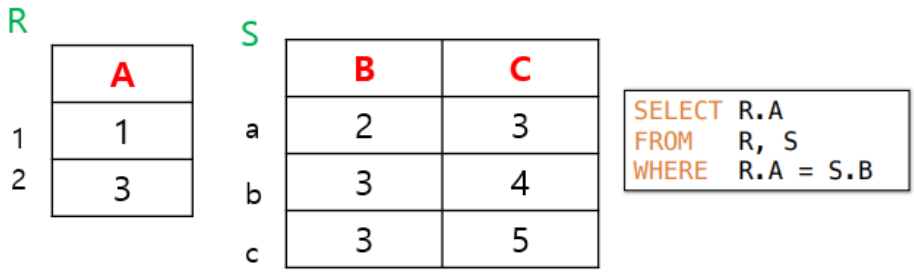
An example of SQL semantics



이 구조를 잘 보면 SQL의 문법의 의미를 파악하는데 도움이 된다.

가지고 있는 것 : R,S라는 table이 존재
-> Cross Product라는 말의 의미는 Cross join이라는 말과 의미가 같다.
따라서 R,S table의 있는 각 row들을 모두 합친 경우의 table을 만들라는 말과 같다.
-> 그리고나서 조건을 보면 R.A = S.B 인데 이말의 뜻은
R table의 A attribute의 values과 , S table의 B attribute의 values가 같은 경우 라는 뜻이다.
같은 경우는 (3,3,4), (3,3,5)이다.
-> 그리고 SELECT문을 보면 R.A라고 적혀있고 이 뜻은
R table의 A attribute만 선택하여 보겠다는 뜻이다.
따라서 A attribute와 value 3을 가진 두개의 row가 합쳐진 table이 생성이 된다.

The semantics of a join



1. Take cross product:

$$X = R \times S = \{1,2\} \times \{a,b,c\} = \{\{1,a\},\{1,b\}....\{2,c\}\}$$

X = R x S의 의미는 곱집합(Cartesian product)을 만들어라는 의미

2. Apply selections / conditions:

$$Y = \{(R, S) \in X \mid R.A == S.B\} = \{\{2,b\},\{2,c\}\}$$

R.A == S.B가 성립하는 경우에 R과 S의 곱집합의 부분집합들 중 X의 속하는 부분집합이 Y이다.

3. Apply projections to get final output:

$$Z = (y.A,) \text{ for } y \in Y$$

Z는 A attribute에 속하는 y 원소를 가지는 집합이다. Z = {{2,b.A},{2,c.A}}
정확한지는 모르지만 {2,b.A}라는 표현이 2,b라는 부분집합 중 A attribute 값만을 추출해라 라는 뜻인것 같다.

별개로 Semantics라는건 not "excution order"이다.
즉.우리가 만든 구절의 의미(문법)들이 실행순서를 의미하지는 않는다는 뜻이다.

A Subtlety about Joins

Product

PName	Price	Category	Manufacturer
Gizmo	$19	Gadgets	GWorks
Powergizmo	$29	Gadgets	GWorks
SingleTouch	$149	Photography	Canon
MultiTouch	$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitach	15	Japan

ex) Find all countries that manufacture some product in the 'Gadgets' category.

문제를 해석해보면 Gadget을 value로 가지는 Category에서 생산품을 팔고있는 모든 나라를 찾아라.

이 문제를 보면 하나의 table로는 구성할 수 없다는 것을 알수 있다 왜냐하면 Category라는 attribute는 Product table에 있고, country attribute는 Company table에 있기 때문이다.

```
SELECT Country
FROM   Product, Company
WHERE  Manufacturer=CName AND Category='Gadgets'
```

위 문제를 Query문으로 만들면 이렇게 되고 해석해보면
1. Product table을 기준으로 Company table을 합쳐라 라는 뜻이다.
2.합치는 조건은 Manufacturer column과 CName column의 값을 비교해서 서로 값이 같은 row끼리 연결하라는 뜻이고
또 다른 조건으로 Category의 value값으로 Gadgets를 가지는 row여야 된다.
3. 그리고 Country라는 attribute를 가지는 table을 만들어라는 뜻이다.

순서대로 차근차근 해보면

1번 결과

PName	Price	Category	Manufacturer	Cname	Stock	Country
Gizmo	$19	Gadgets	GWorks	GWorks	25	USA
Gizmo	$19	Gadgets	Gworks	Canon	65	Japan
...	...	...	...	...	...	...
MultiTouch	$203	Household	Hitach	Hitach	15	Japna

2번 결과

PName	Price	Category	Manufacturer	Cname	Stock	Country
Gizmo	$19	Gadgets	GWorks	GWorks	25	USA
Powergizmo	$29	Gadgets	GWorks	GWorks	25	USA
SingleTouch	$149	Photography	Canon	Canon	65	Japan
MultiTouch	$203	Household	Hitach	Hitach	15	Japan

▼

PName	Price	Category	Manufacturer	Cname	Stock	Country
Gizmo	$19	Gadgets	GWorks	GWorks	25	USA
Powergizmo	$29	Gadgets	GWorks	GWorks	25	USA

3번 결과

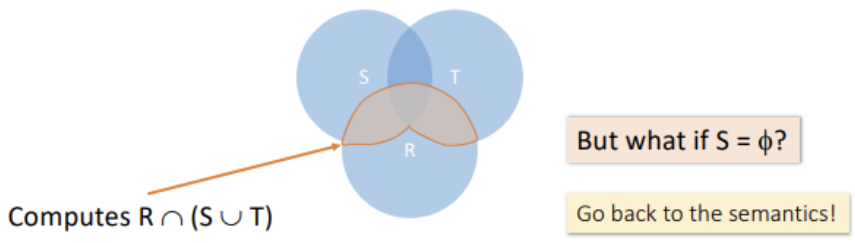
Country
USA
USA

위와 같은 table이 만들어 진다.

하지만 지금 보면 2개의 row의 value값이 같아 중복이 일어 나는데 이를 방지 하기 위해서는
위의 코드에 SELET 구절에 DISTINCT를 넣으면 된다. !!
ex) SELECT DISTINCT Country

An Unintuitive Query

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```



위 그림을 봤을때 해당 query를 만족할려면 하나의 table이라도 {}이면 안된다.
즉. 하나의 table이라도 공집합이면 안된다.

왜냐하면 From R, S, T에서 CROSS JOIN을 하는데 이 결과가 {}이 나오기 때문이다.
따라서 해당 query문의 결과는 {}이 된다.

Join의 종듀

▶내부 조인 (INNER JOIN)
Ⅰ. 교차 조인 (CROSS JOIN , CARTESIN PRODUCT)
Ⅱ. 등가/동등/동일 조인 (EQUI JOIN)
Ⅲ. 비등가 조인 (NON-EQUI JOIN)
Ⅳ. 자연 조인 (NATURAL JPIN)

▶외부 조인 (OUTER JOIN)
Ⅰ. 교차 조인 (FULL OUTER JOIN)
Ⅱ. 왼쪽 (LEFT OUTER JOIN)
Ⅲ. 오른쪽 (RIGHT OUTER JOIN)

▶셀프 조인 (SELF JOIN)
▶안티 조인 (ANTI JOIN)
▶세미 조인 (SEMI JOIN)

※나중에 정리예정

여러 블로그, 영상 및 수업자료를 참고해서 만들었습니다. (↓↓↓링크는 아래쪽에↓↓↓)

https://wkdtjsgur100.github.io/database-terms/
https://blog.naver.com/julie0427/222607810421
https://blog.naver.com/zircorn7/221650177002
https://limkydev.tistory.com/108
https://blog.naver.com/dsz08082/222537027551?isInf=true
https://terms.naver.com/entry.naver?docId=1180957&cid=40942&categoryId=32838
https://blog.naver.com/wltjsaltjs93/222667005698
https://advenoh.tistory.com/23
https://blog.naver.com/jerrypoiu/221488150548

Prof. Lee Seungjin님이 제공해준 Data Processing Systems의 자료를 활용하여 제작하였습니다.