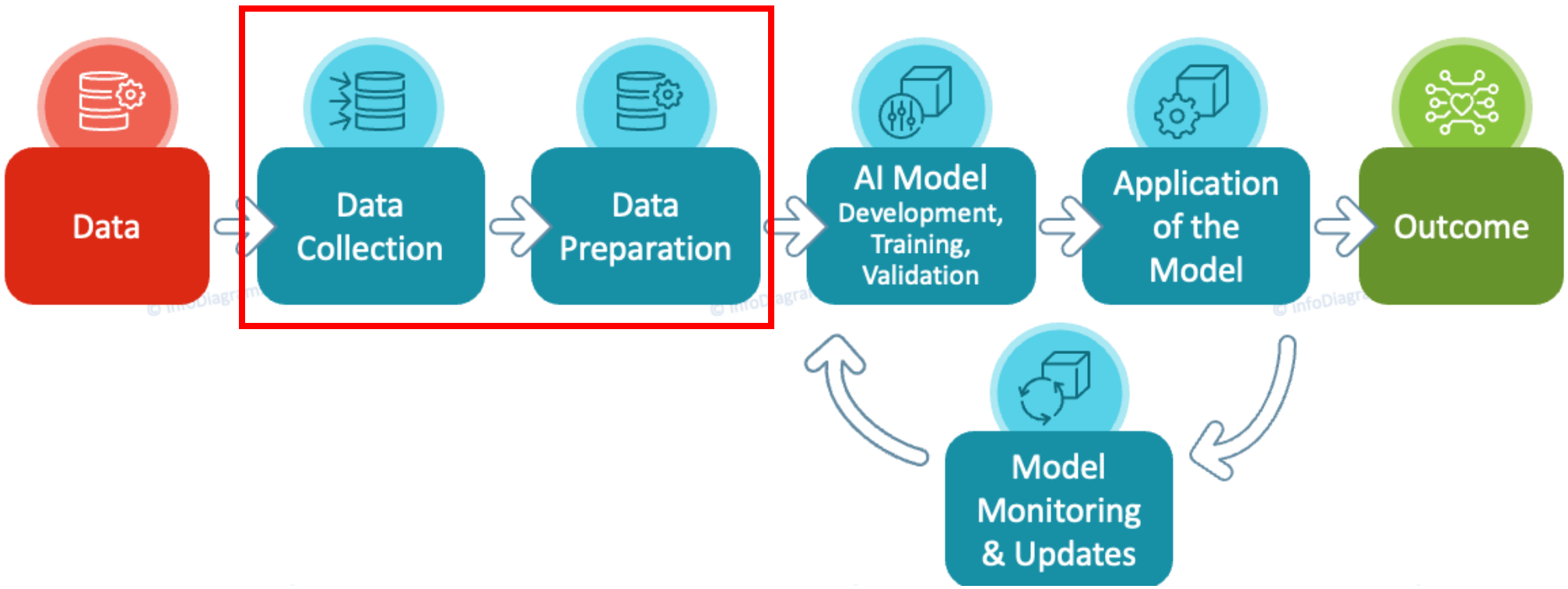
IMEN891K/IDSC501 Data Engineering

8. **SQL II**: Joins & Multisets

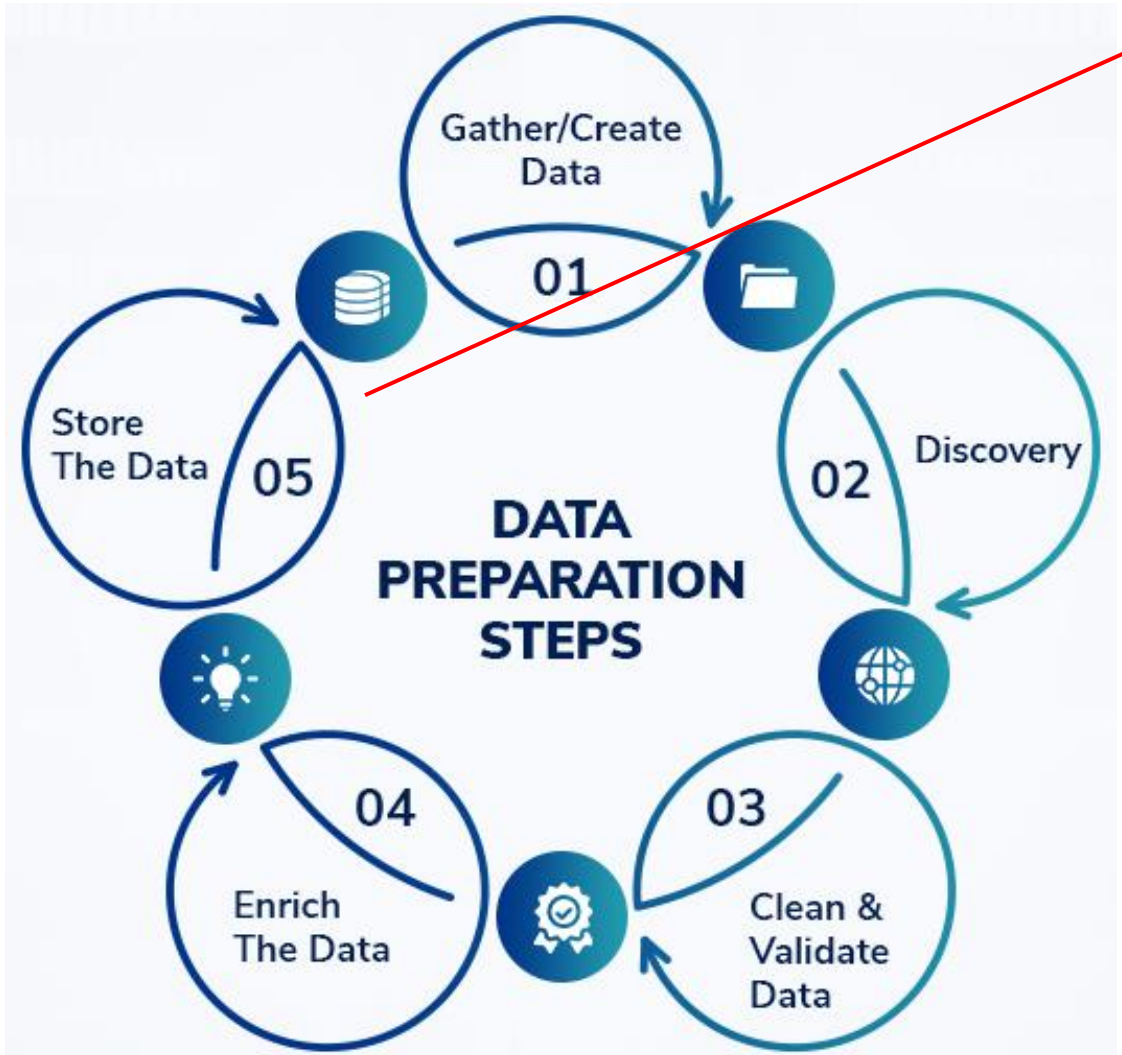
Fall 2025

Instructor: **Sangwoo Mo**

Recap: Course Overview



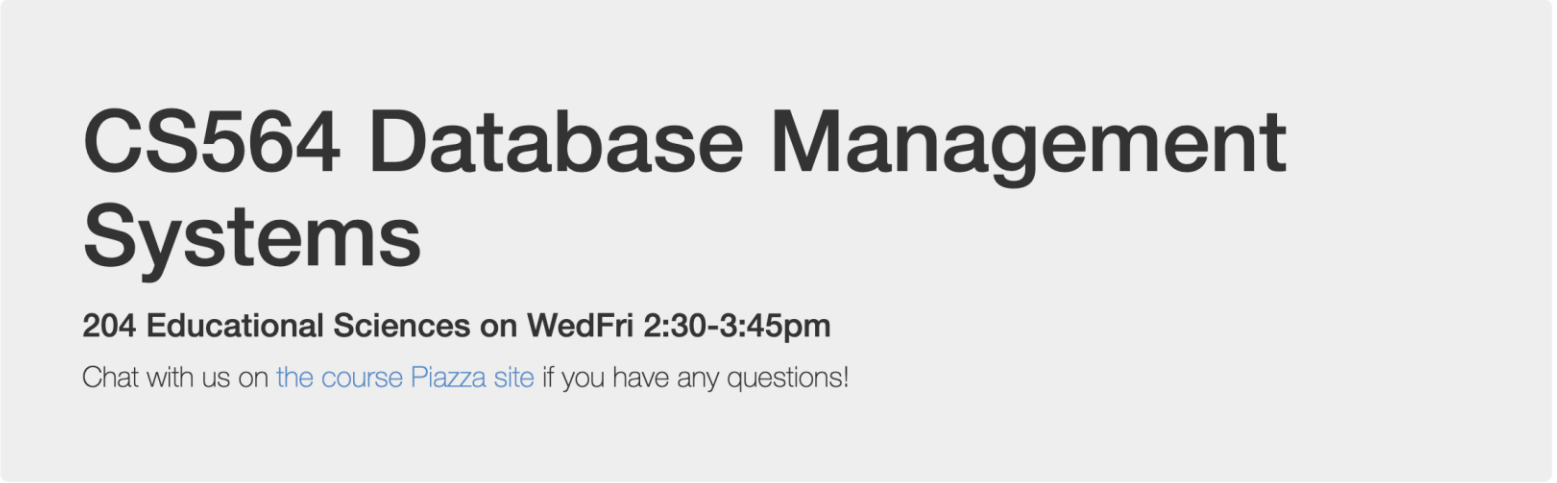
Course Overview



Lecture 7-15

Disclaimer

The SQL part was heavily copy-pasted from Wisconsin CS564



Recap: SQL

- SQL is a standard language for querying and manipulating data
- SQL is a **very high-level** programming language
  - This works because it is optimized well!
- Many standards out there:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
  - Vendors support various subsets

SQL stands for
Structured Query Language

Recap: Python + SQL

기존



ipython-sql 사용



Recap: SQL Query

- Basic form (there are many many more bells and whistles)

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

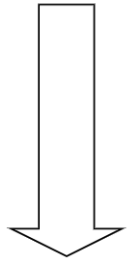
Call this a SFW query.

Recap: Selection

Selection is the operation of filtering a relation's tuples on some condition

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

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



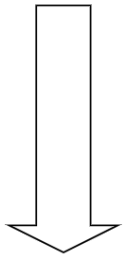
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Recap: Projection

Projection is the operation of producing an output table with tuples that have a subset of their prior attributes

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

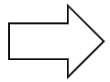
```
SELECT Pname, Price, Manufacturer
FROM   Product
WHERE  Category = 'Gadgets'
```



PName	Price	Manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks

Recap: DISTINCT (Eliminating Duplicates)

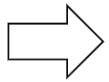
```
SELECT DISTINCT Category
FROM   Product
```



Category
Gadgets
Photography
Household

Versus

```
SELECT Category
FROM   Product
```



Category
Gadgets
Gadgets
Photography
Household

Recap: Key Constraints

A key is a minimal subset of attributes that acts as a unique identifier for tuples in a relation

- A key is an implicit constraint on which tuples can be in the relation
  - i.e. if two tuples agree on the values of the key, then they must be the same tuple!

```
Students(sid:string, name:string, gpa: float)
```

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?

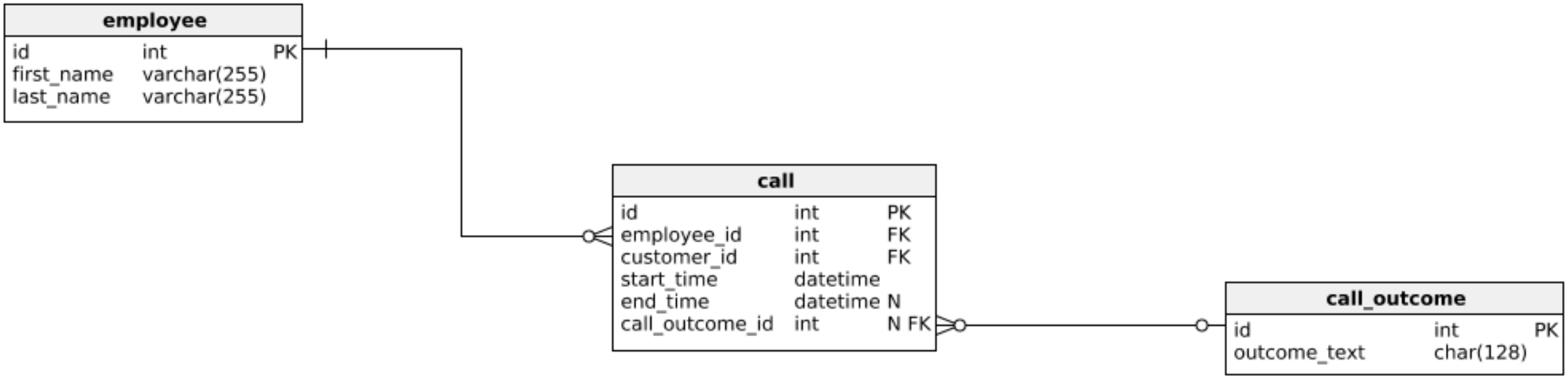
Recap: Declaring Keys

```
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),

)
```

Multi-table Queries



Foreign Key Constraints

- Suppose we have the following schema:

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

- And we want to impose the following constraint:
  - 'Only genuine students may enroll in courses' i.e. a student must appear in the Students table to enroll in a class

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8

Enrolled

student_id	cid	grade
123	564	A
123	537	A+



student_id alone is not a key- what is?

We say that student_id is a foreign key that refers to Students

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)
)
```

Foreign Keys and Update Operations

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

- What if we insert a tuple into Enrolled, but no corresponding student?
  - INSERT is rejected (foreign keys are constraints)!

- What if we delete a student?

DBA chooses (syntax in the book)

  1. Disallow the delete
  2. Remove all of the courses for that student
  3. *SQL allows a third via NULL (not yet covered)*

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is a foreign key vs. a key here?

Product

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

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Product

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

What is a foreign key vs. a key here?

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key

Foreign Key referencing

Product

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

What is a foreign key vs. a key here?

Keys and Foreign Keys

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

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

```
CREATE TABLE Company (
    CName VARCHAR(50) PRIMARY KEY,
    StockPrice INT,
    Country VARCHAR(50)
);

CREATE TABLE Product (
    PName VARCHAR(50) PRIMARY KEY,
    Price DECIMAL(10,2),
    Category VARCHAR(50),
    Manufacturer VARCHAR(50),
    FOREIGN KEY (Manufacturer) REFERENCES Company(CName)
);
```

Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

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

Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

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

A join between tables returns all unique combinations of their tuples which meet some specified join condition

Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

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

A join between tables returns all unique combinations of their tuples which meet some specified join condition

Table Size = |Product| X |Company|

Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

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

A join between tables returns all unique combinations of their tuples which meet some specified join condition

Table Size = |Product| (if FK holds)

Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Ex: Find all products under $200 manufactured in Japan; return their names and prices.

```
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
```

Joins

Product

PName	Price	Category	Manuf
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
Hitachi	15	Japan



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

PName	Price
SingleTouch	$149.99

Tuple Variable Ambiguity in Multi-Table

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

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

Which “address” does this refer to?

Which “name”s??

Tuple Variable Ambiguity in Multi-Table

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

Both equivalent ways to resolve variable ambiguity

```
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
```

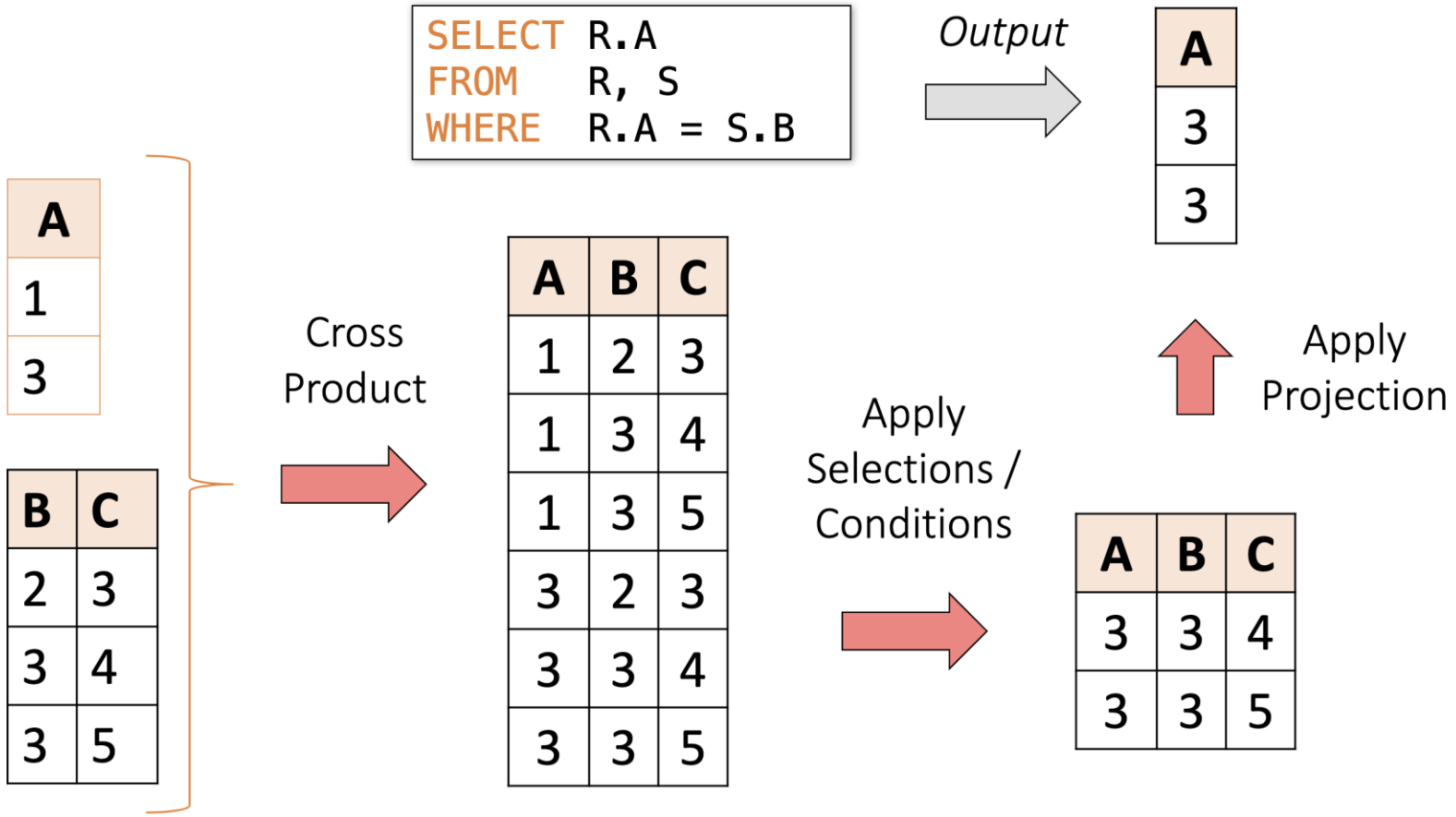
Semantics of SQL Queries

```
SELECT x1.a1, x1.a2, …, xn.ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions(x1,…, xn)
```

Almost never the *fastest* way to compute it!

```
Answer = {}
for x1 in R1 do
    for x2 in R2 do
        .....
            for xn in Rn do
                if Conditions(x1,…, xn)
                    then Answer = Answer ∪ {(x1.a1, x1.a2, …, xn.ak)}
return Answer
```

Semantics of SQL Queries



Semantics of SQL Queries

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

2. Apply **selections / conditions**:

$$Y = \{(r, s) \in X \mid r.A == r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,) \; for \; y \in Y$$

= Returning only *some* attributes

Remembering this order is critical to understanding the output of certain queries (see later on…)

“Semantics” of SQL Queries

≠ “Execution Order”

(Not actually how DBMs work)

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

2. Apply **selections / conditions**:

$$Y = \{(r, s) \in X \mid r.A == r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,) \; for \; y \in Y$$

= Returning only *some* attributes

Remembering this order is critical to understanding the output of certain queries (see later on…)

Caveats in Declaring Foreign Keys

```
CREATE TABLE Company(
        cname           VARCHAR(100),
        stockprice      FLOAT,
        country                 VARCHAR(100),
        PRIMARY KEY (cname),
        FOREIGN KEY (cname) REFERENCES Product(pname, manufacturer)
)

CREATE TABLE Product(
        pname           VARCHAR(100),
        price           FLOAT,
        category                VARCHAR(100),
        manufacturer VARCHAR(100),
        PRIMARY KEY (pname, manufacturer)
)
```

Can we do this? What would be the problem?

Caveats in Declaring Foreign Keys

```
CREATE TABLE Company(
        cname           VARCHAR(100),
        stockprice      FLOAT,
        country                 VARCHAR(100),
        PRIMARY KEY (cname),
        FOREIGN KEY (cname) REFERENCES Product(pname, manufacturer)
)

CREATE TABLE Product(
        pname           VARCHAR(100),
        price           FLOAT,
        category                VARCHAR(100),
        manufacturer VARCHAR(100),
        PRIMARY KEY (pname, manufacturer)
)
```

If the **primary key** is a set of columns (a **composite key**), then the **foreign key** also must be a set of columns that corresponds to the **composite key**.

Caveats in Declaring Foreign Keys

```
CREATE TABLE Company(
        cname           VARCHAR(100),
        stockprice      FLOAT,
        country                 VARCHAR(100),
        PRIMARY KEY (cname),

)

CREATE TABLE Product(
        pname           VARCHAR(100),
        price           FLOAT,
        category                VARCHAR(100),
        manufacturer VARCHAR(100),              FOREIGN KEY (manufacturer)
        PRIMARY KEY (pname, manufacturer)       REFERENCES Company(cname)
)
```

If the **primary key** is a set of columns (a **composite key**), then the **foreign key** also must be a set of columns that corresponds to the **composite key**.

Caveats in Declaring Foreign Keys

```
CREATE TABLE Company(
       cname        VARCHAR(100),
       stockprice   FLOAT,
       country            VARCHAR(100),
       PRIMARY KEY (cname),

)

CREATE TABLE Product(
       pname        VARCHAR(100),
       price        FLOAT,
       category           VARCHAR(100),
       manufacturer VARCHAR(100),          FOREIGN KEY (manufacturer)
       PRIMARY KEY (pname, manufacturer)   REFERENCES Company(cname)
)
```

More natural direction of reference

If the **primary key** is a set of columns (a **composite key**), then the **foreign key** also must be a set of columns that corresponds to the **composite key**.

A Subtlety about Joins

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

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

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

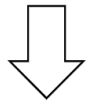
A Subtlety about Joins

Product

PName	Price	Category	Manuf
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
Hitachi	15	Japan



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

Country
USA
USA

A Subtlety about Joins

Product

PName	Price	Category	Manuf
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
Hitachi	15	Japan

DISTINCT

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

Country
USA

Practice: Foreign Key Constraints

Let's illustrate some more complex queries that join two tables together.

- Consider a table of companies, stock price, and HQ country.

  > company(cname, stockprice, country)

- we'll then revist products and introduce some consistency requirements

```
%%sql
drop table if exists product; -- This needs to be dropped if exists, see why further down!
drop table if exists company;
create table company (
    cname varchar primary key, -- company name uniquely identifies the company.
    stockprice money, -- stock price is in money
    country varchar); -- country is just a string
insert into company values ('GizmoWorks', 25.0, 'USA');
insert into company values ('Canon', 65.0, 'Japan');
insert into company values ('Hitachi', 15.0, 'Japan');
```

```
Done.
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
```

Practice: Foreign Key Constraints

- Suppose that we want to create a products table

> Product(pname, price, category, manufacturer)

- Something is odd here: We can have manufacturers that sell products but don't occur in our company table!
- To protect against, this we introduce *foreign keys*

We say the company name in products *refers* to cname in company. Let's do it in SQL! the key statement below is:

> foreign key (manufacturer) references company(cname)

- Note that cname must be a key in company!
- Keys and Foreign keys come up *all* the time.
  - PKs and FKs are common (others, less so)

```
%%sql drop table if exists product;
pragma foreign_keys = ON; -- WARNING by default off in sqlite
create table product(
       pname varchar primary key, -- name of the product
       price money, -- price of the product
       category varchar, -- category
       manufacturer varchar, -- manufacturer
       foreign key (manufacturer) references company(cname));

insert into product values('Gizmo', 19.99, 'Gadgets', 'GizmoWorks');
insert into product values('PowerGizmo', 29.99, 'Gadgets', 'GizmoWorks');
insert into product values('SingleTouch', 149.99, 'Photography', 'Canon');
insert into product values('MultiTouch', 203.99, 'Household', 'Hitachi');
```

```
Done.
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

Practice: Foreign Key Constraints

Indeed foreign keys are a *constraint*

> What happens if we introduce a company name not in our table?

```
try:
    %sql insert into product values('MultiTouch', 203.99, 'Household', 'Google');
except Exception as e:
    print e
    print "Rejected!"
```

```
(IntegrityError) UNIQUE constraint failed: product.pname u"insert into product values('MultiTouch', 203.99, 'Household', 'Google');" ()
Rejected!
```

Practice: Foreign Keys and Delete

- What happens if we delete a company? Three options:
  - Disallow the delete. (default)
  - Remove all products (add "`on delete cascade`")
  - A third variant due to NULL

First option (default)- delete is disallowed

```
try:
    %sql delete from company where cname = 'Hitachi';
except Exception as e:
    print e
    print "Disallowed!"
```

```
(IntegrityError) FOREIGN KEY constraint failed u"delete from company where cname = 'Hitachi';" ()
Disallowed!
```

Practice: Foreign Keys and Delete

- What happens if we delete a company? Three options:
  - Disallow the delete. (default)
  - Remove all products (add "`on delete cascade`")
  - A third variant due to NULL

First option (default)- delete is disallowed

```
try:
    %sql delete from company where cname = 'Hitachi';
except Exception as e:
    print e
    print "Disallowed!"
```

```
(IntegrityError) FOREIGN KEY constraint failed u"delete from company where cname = 'Hitachi';" ()
Disallowed!
```

Second option: remove all products belonging to the company we delete

Try adjusting the foreign key constraint clause when you create the products table as follows:

> foreign key (manufacturer) references company(cname) on delete cascade

Now, when a company row is deleted, all of the products linked by the foreign key constraint will be deleted as well.

Practice: Joins

> Product (pname, price, category, manufacturer)
> Company (cname, stockPrice, country)

The query we want to answer is:

> Find all products under $200 manufactured in Japan; return their names and prices.

Notice products don't have a location and manufacturers don't have price. Need info in *each* of the tables.

```
%%sql
SELECT pname, price
FROM product, company
where manufacturer=cname and country='Japan' and price <= 200;
```

```
Done.
```

pname	price
SingleTouch	149.99

Practice: Joins

Let's see how to write this join in a modular way.

```
%%sql -- Part 1: Set of Japanese companies.
SELECT distinct cname -- do we need distinct?
from company where country='Japan';
```

Done.

cname
Canon
Hitachi

```
%%sql -- Part 2: Cheap Products (under $200)
select distinct pname, price, manufacturer
from product
where price <= 200;
```

Done.

pname	price	manufacturer
Gizmo	19.99	GizmoWorks
PowerGizmo	29.99	GizmoWorks
SingleTouch	149.99	Canon

Practice: Joins

```
%%sql -- combine them with nested SFW queries... this is a cross product?
SELECT *
FROM
  (SELECT DISTINCT pname, price, manufacturer
   FROM product
   WHERE price <= 200) CheapProducts,
  (SELECT DISTINCT cname
   FROM company
   WHERE country='Japan') JapaneseProducts;
```

```
Done.
```

pname	price	manufacturer	cname
Gizmo	19.99	GizmoWorks	Canon
Gizmo	19.99	GizmoWorks	Hitachi
PowerGizmo	29.99	GizmoWorks	Canon
PowerGizmo	29.99	GizmoWorks	Hitachi
SingleTouch	149.99	Canon	Canon
SingleTouch	149.99	Canon	Hitachi

Practice: Joins

```
%%sql
-- Combine them as a join!
SELECT DISTINCT pname, price
FROM
  (SELECT DISTINCT pname, price, manufacturer
   FROM product
   WHERE price <= 200) CheapProducts,
  (SELECT distinct cname
   FROM company
   WHERE country='Japan') JapaneseProducts
WHERE cname = manufacturer;
```

Done.

pname	price
SingleTouch	149.99

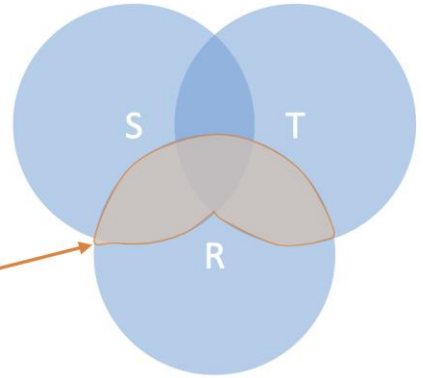
An Unintuitive Query

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

What does it compute?

An Unintuitive Query

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



Computes $R \cap (S \cup T)$

But what if $S = \phi$?

Go back to the semantics!

An Unintuitive Query

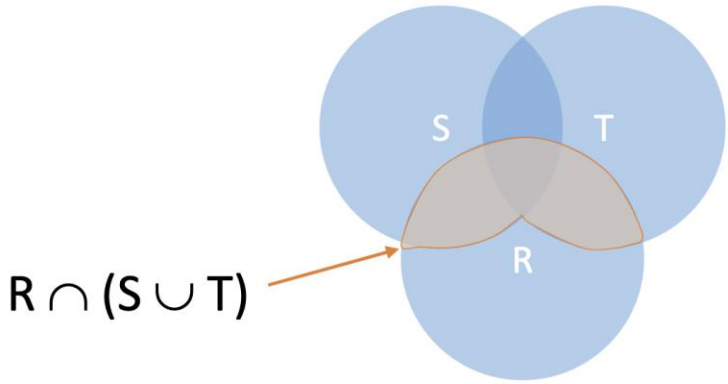
```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

- Recall the semantics!
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection
- If S = {}, then the cross product of R, S, T = {}, and the query result = {}!

Must consider semantics here.
Are there more explicit way to do set operations like this?

What does this look like in Python?

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



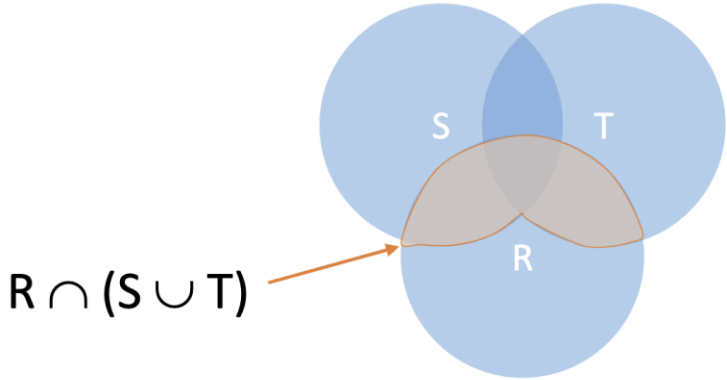
- Semantics:
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection

Joins / cross-products are just **nested for loops** (in simplest implementation)!

If-then statements!

What does this look like in Python?

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




```
output = {}

for r in R:
   for s in S:
      for t in T:
         if r['A'] == s['A'] or r['A'] == t['A']:
            output.add(r['A'])
return list(output)
```

Can you see now what happens if S = []?

Multiset Operations

Multiset X

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)



Equivalent Representations of a Multiset

$\boldsymbol{\lambda(X)}$= *"Count of tuple in X" (Items not listed have implicit count 0)*

Multiset X

Tuple	$\boldsymbol{\lambda(X)}$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

Note: In a set all counts are {0,1}.

Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cap$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0

$$\lambda(Z) = min(\lambda(X), \lambda(Y))$$

For sets, this is intersection

Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cup$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	5
(1, b)	1
(2, c)	3
(1, d)	2

$$\lambda(Z) = max(\lambda(X), \lambda(Y))$$

For sets,
this is union

Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\sqcup$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2

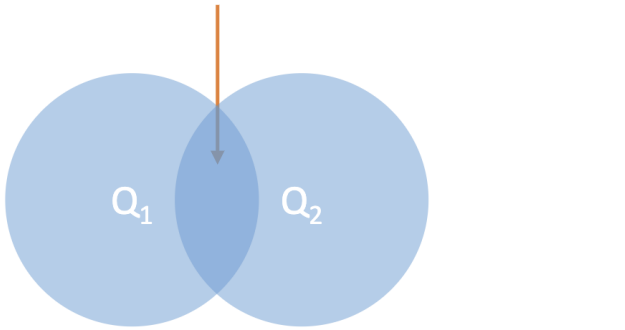
$$\lambda(Z) = \lambda(X) + \lambda(Y)$$

For sets,
this is disjoint
union

Multiset Operations in SQL – INTERSECT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
INTERSECT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

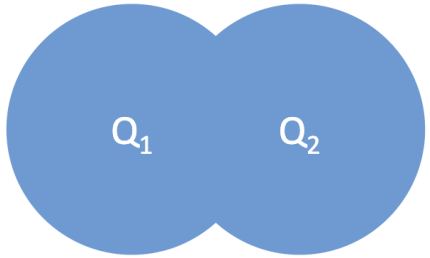
$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$



Multiset Operations in SQL – UNION

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$



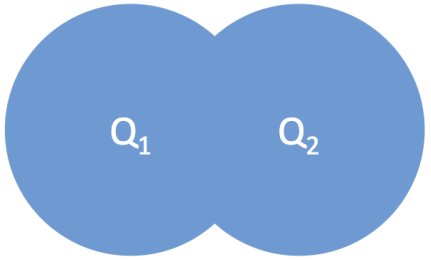
Why aren't there duplicates?

What if we want duplicates?

Multiset Operations in SQL – UNION ALL

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION ALL
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$

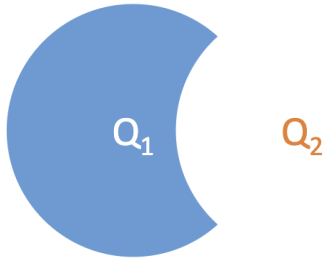


ALL indicates the Multiset disjoint union operation

Multiset Operations in SQL – EXCEPT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
EXCEPT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \backslash \{r.A | r.A = t.A\}$$



What is the multiset version?

$$\boldsymbol{\lambda(Z) = \lambda(X) - \lambda(Y)}$$

For elements that are in X

Still some subtle problems…

```
Company(name, hq_city)
Product(pname, maker, factory_loc)
```

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'US'
INTERSECT
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'China'
```

"Headquarters of companies which make gizmos in US AND China"

What if two companies have HQ in US: BUT one has factory in China (but not US) and vice versa? **What goes wrong?**

Still some subtle problems...

```
Company(name, hq_city) AS C
Product(pname, maker,
factory_loc) AS P
```

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='US'
INTERSECT
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
AND factory_loc='China'
```

Example: C JOIN P on maker = name

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	X	Y Inc.	China

Still some subtle problems...

```
Company(name, hq_city) AS C
Product(pname, maker,
factory_loc) AS P
```

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='US'
INTERSECT
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
AND factory_loc='China'
```

Example: C JOIN P on maker = name

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	X	Y Inc.	China

X Co has a factory in the US (but not China)
Y Inc. has a factor in China (but not US)

But Seattle is returned by the query!

We did the INTERSECT
on the wrong attributes!

One Solution: Nested Queries

```
Company(name, hq_city)
Product(pname, maker, factory_loc)
```

```
SELECT DISTINCT hq_city
FROM   Company, Product
WHERE  maker = name
       AND name IN (
            SELECT maker
            FROM   Product
            WHERE  factory_loc = 'US')
       AND name IN (
            SELECT maker
            FROM   Product
            WHERE  factory_loc = 'China')
```

"Headquarters of companies which make gizmos in US AND China"

Note: If we hadn't used DISTINCT here, how many copies of each hq_city would have been returned?

Practice: Multisets

```
%sql DROP TABLE IF EXISTS A; DROP TABLE IF EXISTS B;
%sql CREATE TABLE A (x int, y int); CREATE TABLE B (x int, y int);
for i in range(1,6):
    %sql INSERT INTO A VALUES (:i, :i+1)
for i in range(1,11,3):
    %sql INSERT INTO B VALUES (:i, :i+2)
```

Practice: Multisets

Consider the two relations A and B below:

```
%sql SELECT * FROM A;
```

```
 * sqlite://
Done.
```

x	y
1	2
2	3
3	4
4	5
5	6

```
%sql SELECT * FROM B;
```

```
 * sqlite://
Done.
```

x	y
1	3
4	6
7	9
10	12

Practice: Multisets

Assuming no duplicates, can you write an `INTERSECT` query, **just over the x attribute**, without using `INTERSECT` OR nested queries? Write your query here:

```
%sql SELECT A.x FROM A, B WHERE A.x = B.x;
```

```
 * sqlite://
Done.
```

x
1
4

Practice: Multisets

Next, using set operators again as well, can you return all the *full* tuples in A and B that overlap in x attributes? Write your query here:

```
%%sql
SELECT x, y FROM (
    SELECT A.x, A.y FROM A, B WHERE A.x = B.x
    UNION
    SELECT B.x, B.y FROM A, B WHERE A.x = B.x
);
```

```
 * sqlite://
Done.
```

x	y
1	2
1	3
4	5
4	6