Lecture 4: SQL Part III

Today's Lecture

1. Set operators & nested queries
   - ACTIVITY: Set operator subtleties

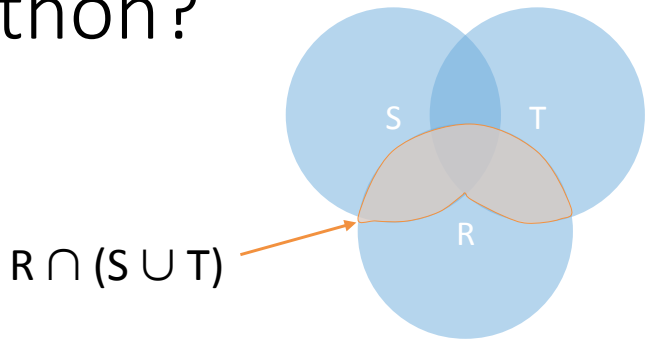
1. Set Operators & Nested Queries

What you will learn about in this section

1. Multiset operators in SQL
2. Nested queries
3. ACTIVITY: Set operator subtleties

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

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

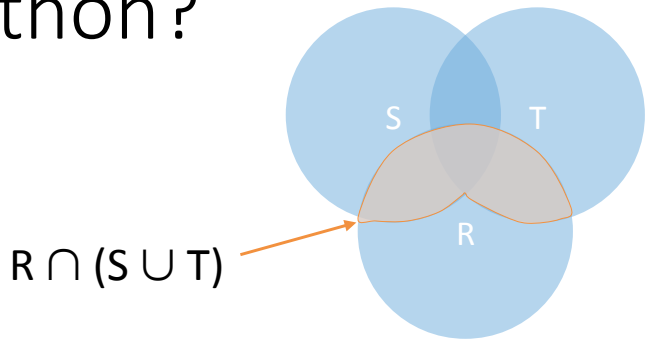
  2. Apply selections / conditions

If-then statements!

  3. Apply projection

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

Recall Multisets

Multiset X

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

Equivalent Representations of a Multiset

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

Multiset X

Tuple	
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

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

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

∩

Multiset Y

Tuple	
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

=

Multiset Z

Tuple	
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0

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

For sets, this is **intersection**

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

∪

Multiset Y

Tuple	
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

=

Multiset Z

Tuple	
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2

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

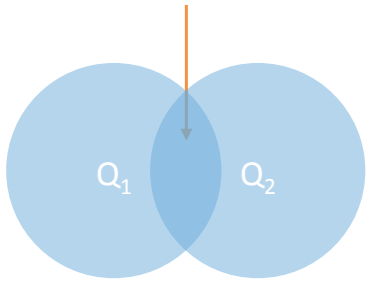
For sets, this is **union**

Multiset Operations in SQL

Explicit Set Operators: INTERSECT

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

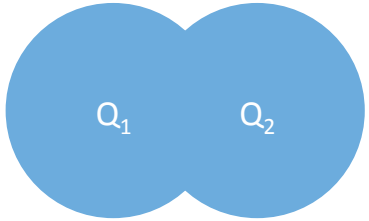
$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$



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?

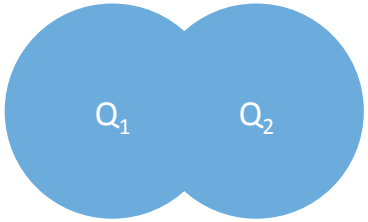
By default:
SQL uses set semantics!

What if we want duplicates?

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
```

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

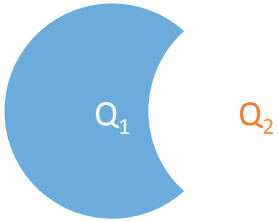


ALL indicates Multiset operations

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?

INTERSECT: 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?**

INTERSECT: Remember the semantics!

```
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

INTERSECT: Remember the semantics!

```
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 factory 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?

High-level note on nested queries

- We can do nested queries because SQL is ***compositional:***
    - Everything (inputs / outputs) is represented as multisets- the output of one query can thus be used as the input to another (nesting)!
- This is extremely powerful!

Nested queries: Sub-queries Return Relations

Another example:

```
Company(name, city)
Product(name, maker)
Purchase(id, product, buyer)
```

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
     SELECT pr.maker
      FROM   Purchase p, Product pr
      WHERE  p.product = pr.name
        AND p.buyer = 'Joe Blow')
```

"Cities where one can find companies that manufacture products bought by Joe Blow"

Nested Queries

Are these queries equivalent?

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
   AND p.buyer = 'Joe Blow')
```

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe Blow'
```

Nested Queries

Are these queries equivalent?

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
  SELECT pr.maker
  FROM   Purchase p, Product pr
  WHERE  p.name = pr.product
   AND p.buyer = 'Joe Blow')
```

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe Blow'
```

Beware of duplicates!

Nested Queries

```
SELECT DISTINCT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe Blow'
```

```
SELECT DISTINCT c.city
FROM   Company c
WHERE  c.name  IN (
  SELECT pr.maker
  FROM   Purchase p, Product pr
  WHERE  p.product = pr.name
      AND p.buyer = 'Joe Blow')
```

Now they are equivalent (both use set semantics)

Subqueries Return Relations

You can also use operations of the form:

- s > ALL R
- s < ANY R
- EXISTS R

ANY and ALL not supported by SQLite.

Ex: `Product(name, price, category, maker)`

```
SELECT name
FROM   Product
WHERE  price > ALL(
     SELECT price
     FROM   Product
     WHERE  maker = 'Gizmo-Works')
```

Find products that are more expensive than all those produced by "Gizmo-Works"

Subqueries Returning Relations

You can also use operations of the form:

- s > ALL R
- s < ANY R
- EXISTS R

Ex:

```
Product(name, price, category, maker)
```

```
SELECT p1.name
FROM   Product p1
WHERE  p1.maker = 'Gizmo-Works'
   AND EXISTS(
      SELECT p2.name
      FROM   Product p2
      WHERE  p2.maker <> 'Gizmo-Works'
         AND p1.name = p2.name)
```

<> means !=

Find 'copycat' products, i.e. products made by competitors with the same names as products made by "Gizmo-Works"

Nested queries as alternatives to INTERSECT and EXCEPT

INTERSECT and EXCEPT not in some DBMSs!

```
(SELECT R.A, R.B
 FROM   R)
INTERSECT
(SELECT S.A, S.B
 FROM   S)
```

```
SELECT R.A, R.B
FROM   R
WHERE EXISTS(
       SELECT *
       FROM S
        WHERE R.A=S.A AND R.B=S.B)
```

```
(SELECT R.A, R.B
 FROM   R)
EXCEPT
(SELECT S.A, S.B
 FROM   S)
```

```
SELECT R.A, R.B
FROM   R
WHERE NOT EXISTS(
       SELECT *
        FROM S
        WHERE R.A=S.A AND R.B=S.B)
```

If R, S have no duplicates, then can write without sub-queries (HOW?)

Correlated Queries Using External Vars in Internal Subquery

```
Movie(title, year, director, length)
```

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
          SELECT  year
          FROM    Movie
          WHERE   title =  m.title)
```

Find movies whose title appears more than once.

Note the scoping of the variables!

Note also: this can still be expressed as single SFW query…

```
SELECT DISTINCT title
FROM   Movie AS m1, Movie As m2
WHERE m1.title = m2.title and
m1.year <> m2.year
```

Internal Subquery

```
Movie(title, year, director, length)
```

Find movies whose title appears more than once.

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
          SELECT  year
          FROM    Movie
          WHERE   title =  m.title)
```

Note the scoping of the variables!

Note also: this can still be expressed as single SFW query…

Complex Correlated Query

```
Product(name, price, category, maker, year)
```

```
SELECT DISTINCT  x.name, x.maker
FROM   Product AS x
WHERE  x.price > ALL(
          SELECT y.price
          FROM   Product AS y
          WHERE  x.maker = y.maker
             AND y.year < 1972)
```

Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

Can be very powerful (also much harder to optimize)

Basic SQL Summary

- SQL provides a high-level declarative language for manipulating data (DML)
- The workhorse is the SFW block
- Set operators are powerful but have some subtleties
- Powerful, nested queries also allowed.

Activity-4-1.ipynb