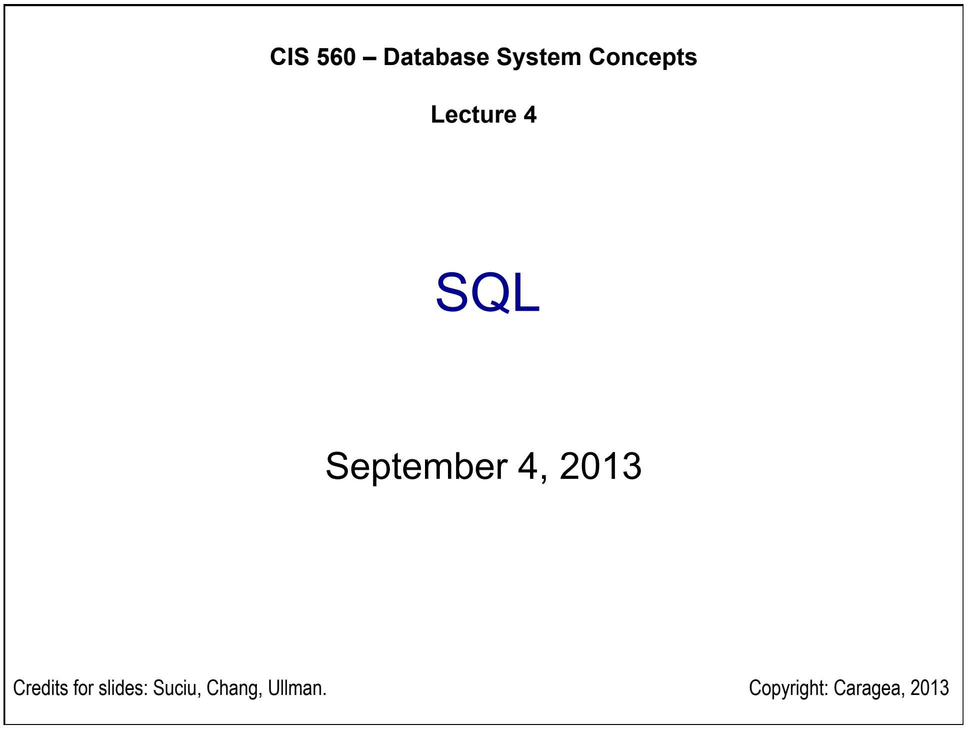


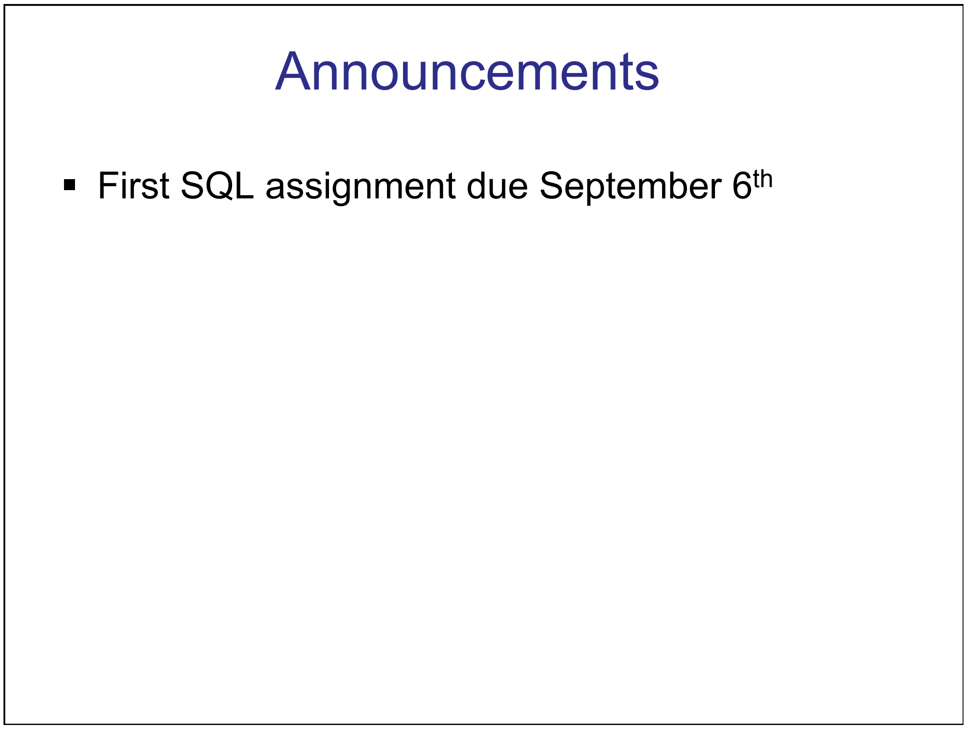
CIS 560 – Database System Concepts

Lecture 4

SQL

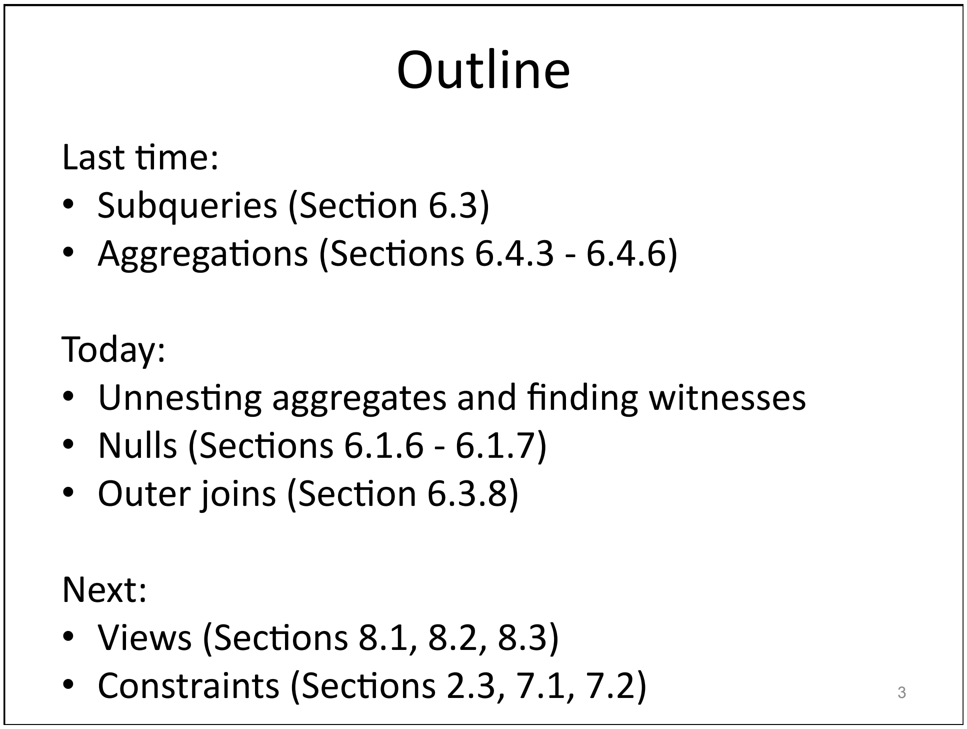
September 4, 2013

Credits for slides: Suciu, Chang, Ullman.



Announcements

- First SQL assignment due September 6th



Outline

Last time:

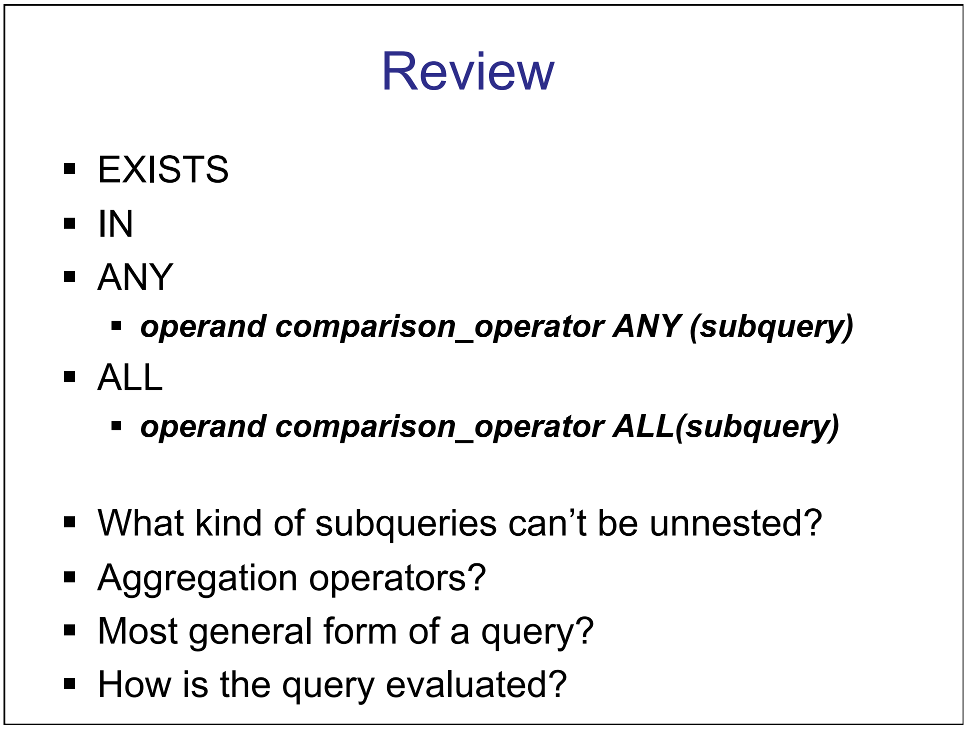
- Subqueries (Section 6.3)
- Aggregations (Sections 6.4.3 - 6.4.6)

Today:

- Unnesting aggregates and finding witnesses
- Nulls (Sections 6.1.6 - 6.1.7)
- Outer joins (Section 6.3.8)

Next:

- Views (Sections 8.1, 8.2, 8.3)
- Constraints (Sections 2.3, 7.1, 7.2)



Review

- EXISTS
- IN
- ANY
  - ***operand comparison_operator ANY (subquery)***
- ALL
  - ***operand comparison_operator ALL(subquery)***

- What kind of subqueries can’t be unnested?
- Aggregation operators?
- Most general form of a query?
- How is the query evaluated?

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

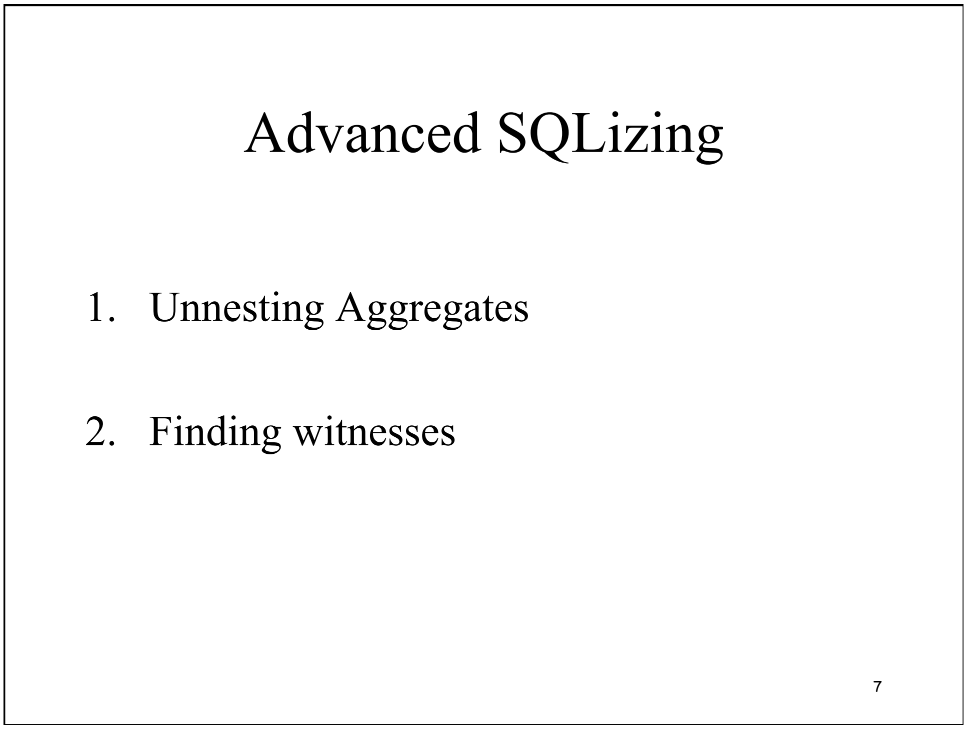
C2 = is any condition on aggregate expressions

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

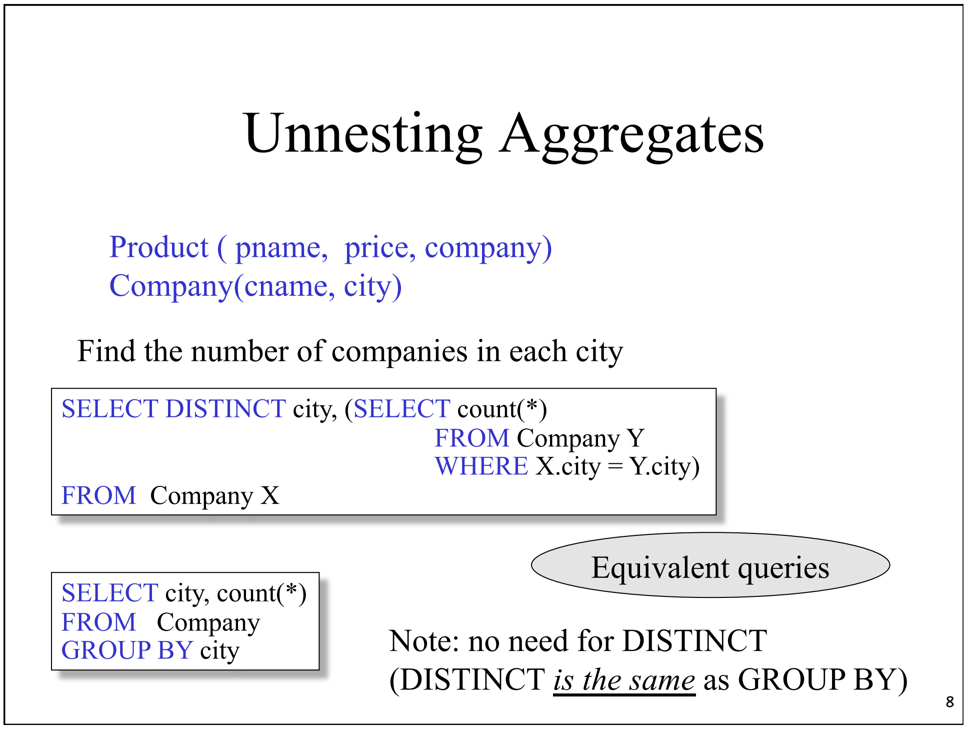
Evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result



Advanced SQLizing

1. Unnesting Aggregates
2. Finding witnesses



Unnesting Aggregates

Product (pname, price, company)
Company(cname, city)

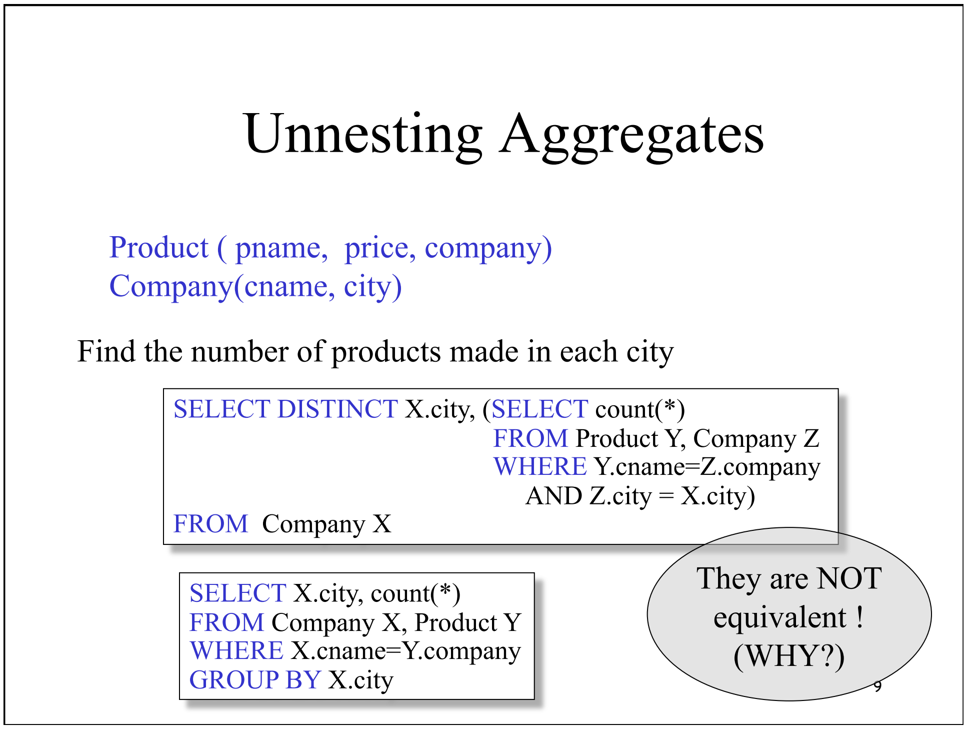
Find the number of companies in each city

```
SELECT DISTINCT city, (SELECT count(*)
                                     FROM Company Y
                                     WHERE X.city = Y.city)
FROM  Company X
```

Equivalent queries

```
SELECT city, count(*)
FROM   Company
GROUP BY city
```

Note: no need for DISTINCT
(DISTINCT *is the same* as GROUP BY)



Unnesting Aggregates

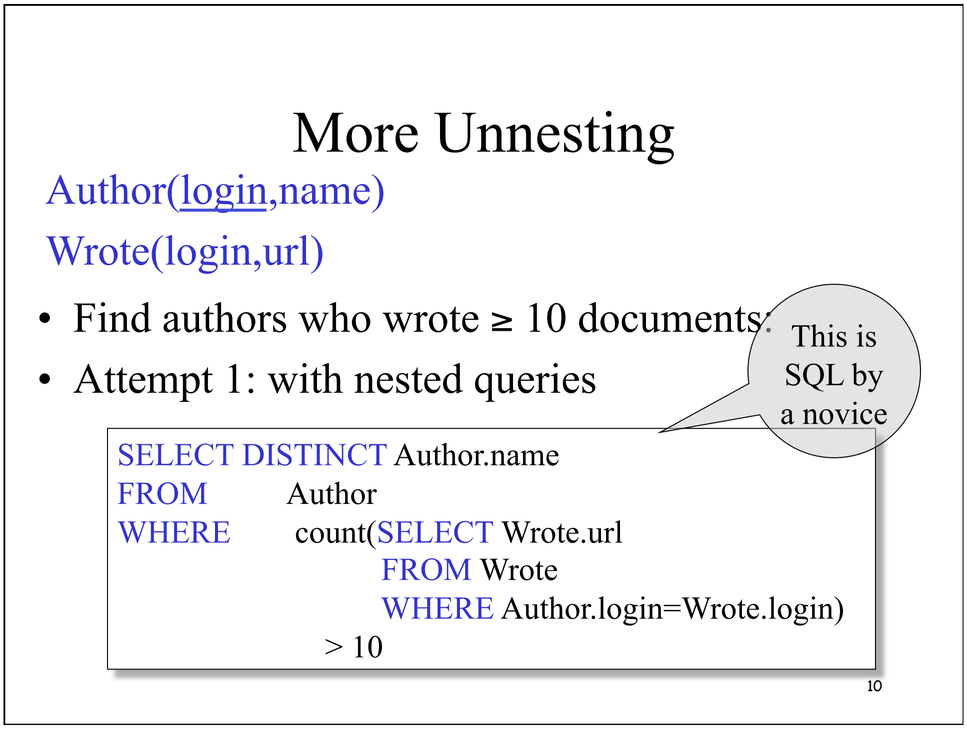
Product (pname, price, company)
Company(cname, city)

Find the number of products made in each city

```
SELECT DISTINCT X.city, (SELECT count(*)
                         FROM Product Y, Company Z
                         WHERE Y.cname=Z.company
                            AND Z.city = X.city)
FROM  Company X
```

```
SELECT X.city, count(*)
FROM Company X, Product Y
WHERE X.cname=Y.company
GROUP BY X.city
```

They are NOT equivalent ! (WHY?)



More Unnesting

Author(login,name)

Wrote(login,url)

- Find authors who wrote ≥ 10 documents:
- Attempt 1: with nested queries

This is SQL by a novice

```
SELECT DISTINCT Author.name
FROM        Author
WHERE        count(SELECT Wrote.url
                   FROM Wrote
                   WHERE Author.login=Wrote.login)
                > 10
```

More Unnesting

- Find all authors who wrote at least 10 documents:
- Attempt 2: SQL style (with GROUP BY)

```
SELECT      Author.name
FROM        Author, Wrote
WHERE       Author.login=Wrote.login
GROUP BY Author.name
HAVING      count(wrote.url) > 10
```

This is SQL by an expert

Finding Witnesses

Store(<u>sid</u>, sname)
Product(<u>pid</u>, pname, price, sid)

For each store,
find its most expensive products

Finding Witnesses

Finding the maximum price is easy…

```
SELECT Store.sid, max(Product.price)
FROM    Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid
```

But we need the *witnesses*, i.e. the products with max price

Finding Witnesses

Store(sid, sname)
Product(pid, pname, price, sid)

```
SELECT Store.sname, x.pname
FROM    Store, Product x
WHERE  Store.sid = x.sid and
            x.price >=
                 ALL (SELECT y.price
                        FROM Product y
                        WHERE Store.sid = y.sid)
```

NULLS in SQL

- Whenever we don’t have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Value is withheld
  - Etc.
- The schema specifies for each attribute if can be null (*nullable* attribute) or not
- How does SQL cope with tables that have NULLs ?

Null Values

- If x is NULL then 4*(3-x)/7 is still NULL
- If x is NULL then x=‘Joe’   is UNKNOWN
- In SQL there are three Boolean values:

FALSE	=	0
UNKNOWN	=	0.5
TRUE	=	1

Null Values

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

```
SELECT *
FROM Person
WHERE  (age < 25) AND
            (height > 6 OR weight > 190)
```

E.g.
age=20
heigth=NULL
weight=200

Rule in SQL: include only tuples that yield TRUE

Null Values

Unexpected behavior:

```
SELECT *
FROM     Person
WHERE  age < 25  OR  age >= 25
```

Some Persons are not included !

Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM     Person
WHERE  age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Persons

Patterns

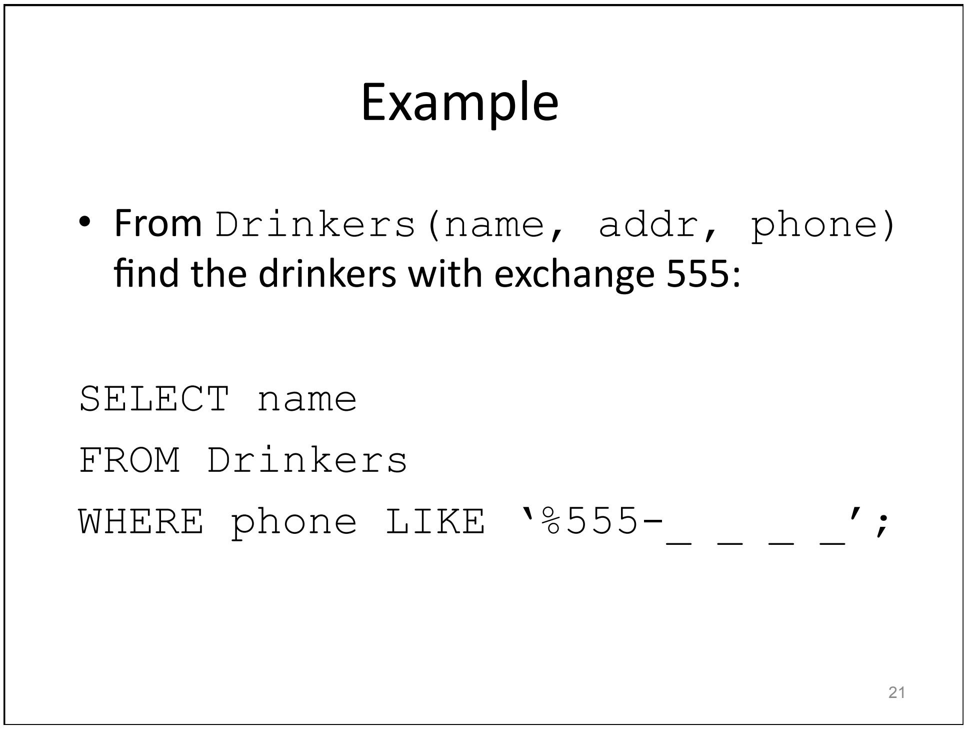
- WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.
- General form:

  <Attribute> LIKE <pattern> or

  <Attribute> NOT LIKE <pattern>
- Pattern is a quoted string with

  % = any string

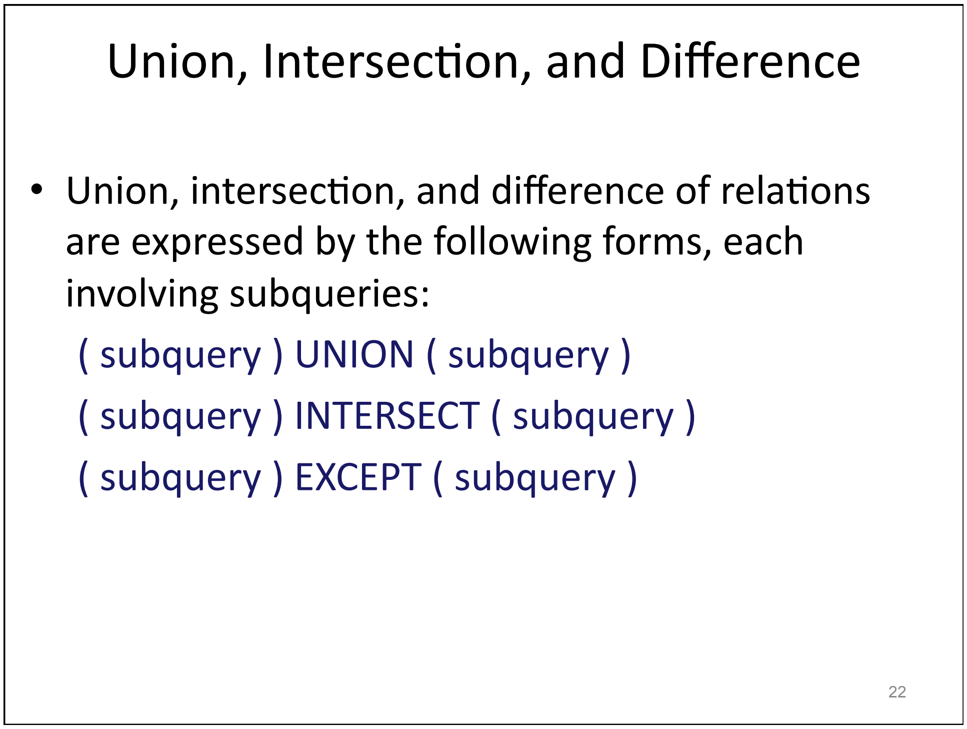
  _ = any character



Example

- From `Drinkers(name, addr, phone)` find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE ‘%555-_ _ _ _’;
```



Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:

  (subquery) UNION (subquery)

  (subquery) INTERSECT (subquery)

  (subquery) EXCEPT (subquery)

Example

- From relations

  `Likes(drinker, beer)`

  `Sells(bar, beer, price)` and

  `Frequents(drinker, bar)`

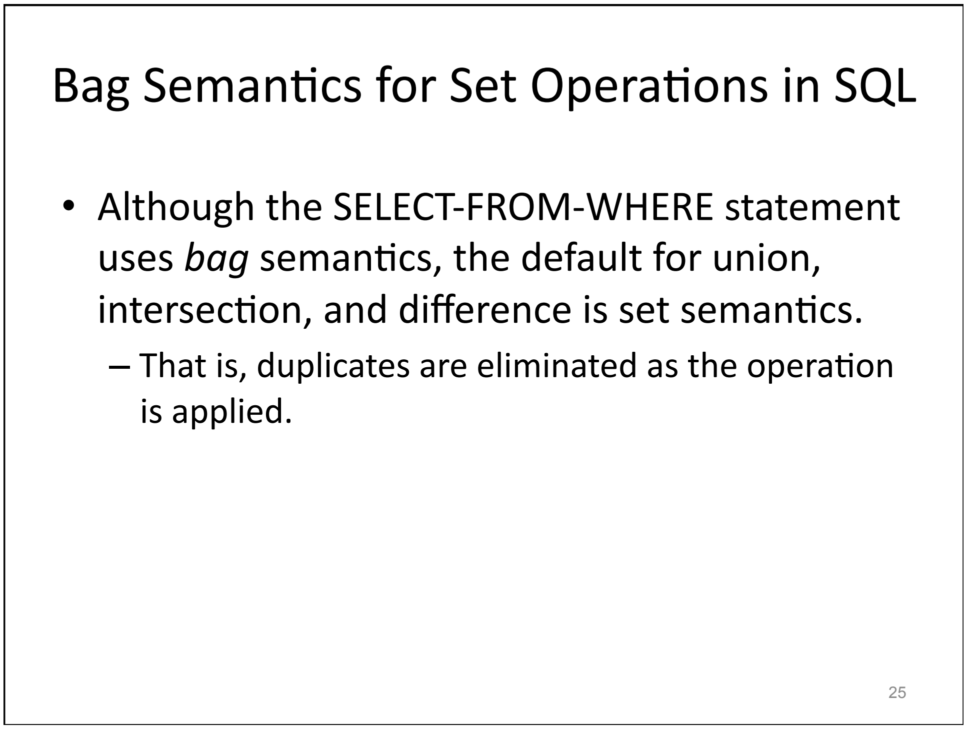
  find the drinkers and beers such that:

  1. The drinker likes the beer, and
  2. The drinker frequents at least one bar that sells the beer.

Solution

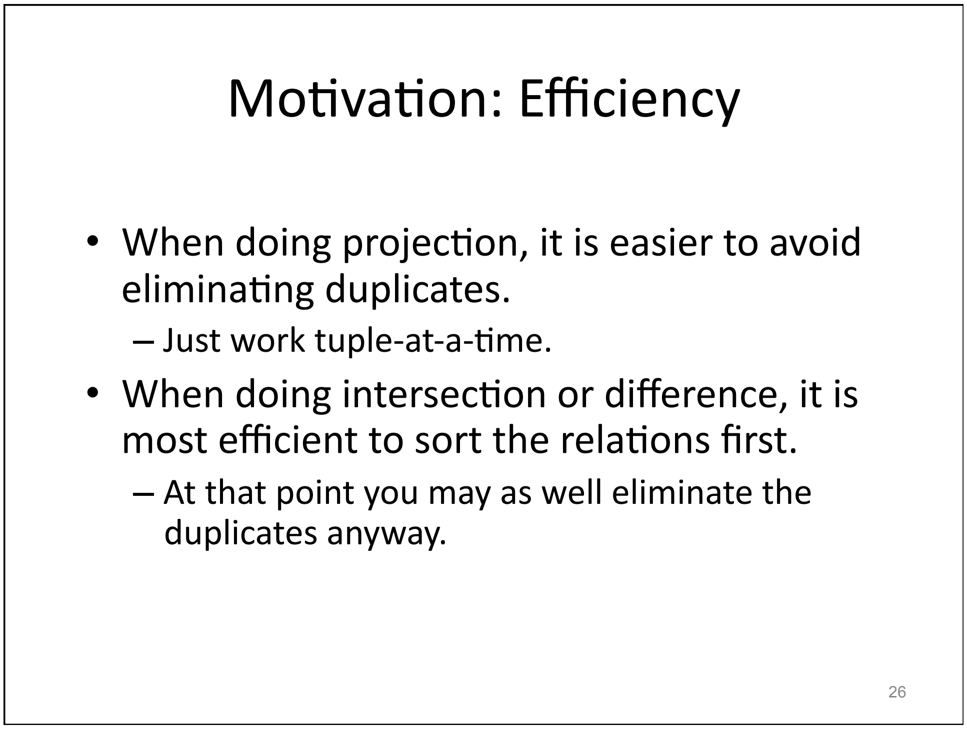
The drinker frequents a bar that sells the beer.

```
(SELECT * FROM Likes)
   INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar
);
```



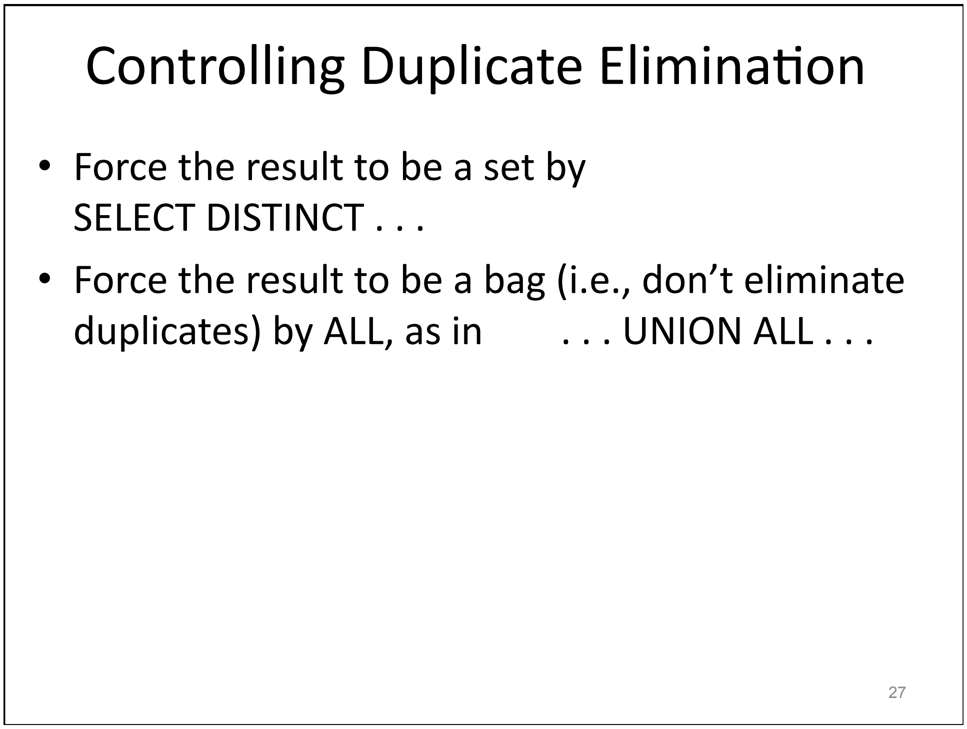
Bag Semantics for Set Operations in SQL

- Although the SELECT-FROM-WHERE statement uses *bag* semantics, the default for union, intersection, and difference is set semantics.
  – That is, duplicates are eliminated as the operation is applied.



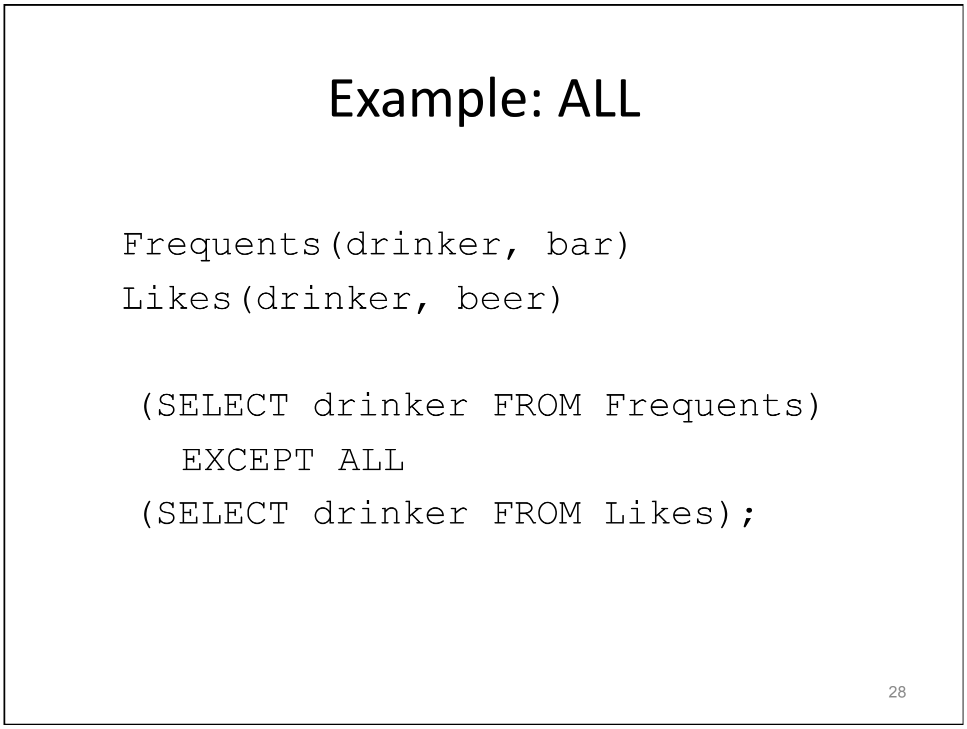
Motivation: Efficiency

- When doing projection, it is easier to avoid eliminating duplicates.
  – Just work tuple-at-a-time.
- When doing intersection or difference, it is most efficient to sort the relations first.
  – At that point you may as well eliminate the duplicates anyway.



Controlling Duplicate Elimination

- Force the result to be a set by SELECT DISTINCT . . .
- Force the result to be a bag (i.e., don’t eliminate duplicates) by ALL, as in . . . UNION ALL . . .



Example: ALL

```
Frequents(drinker, bar)
Likes(drinker, beer)

 (SELECT drinker FROM Frequents)
   EXCEPT ALL
 (SELECT drinker FROM Likes);
```