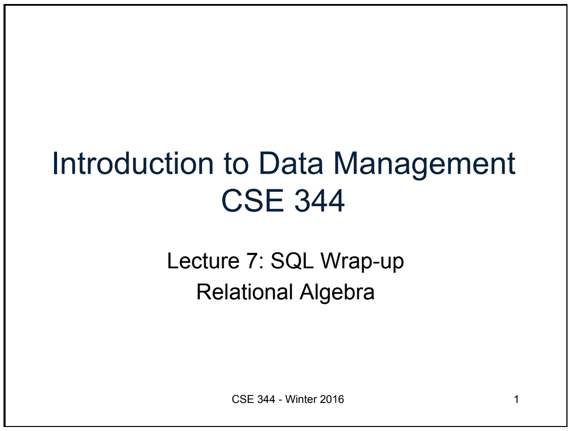
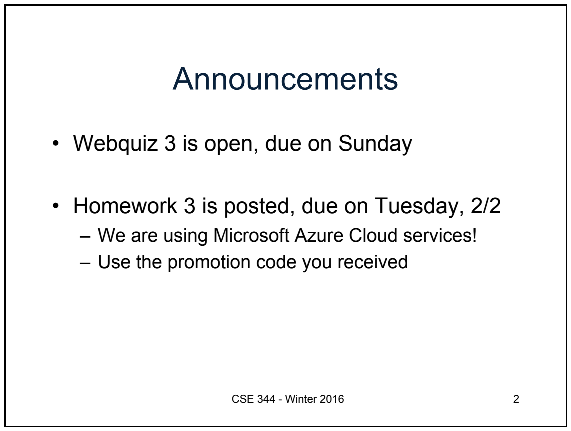


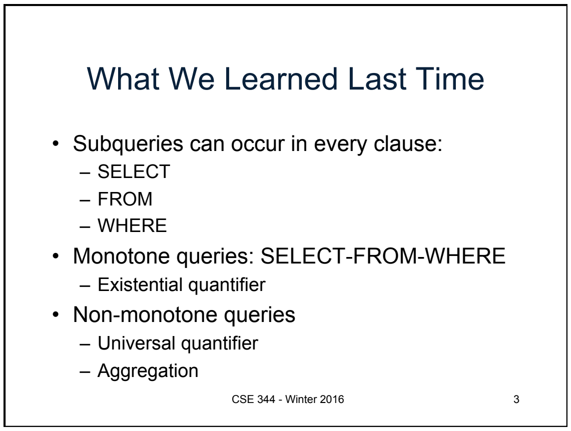
Introduction to Data Management CSE 344

Lecture 7: SQL Wrap-up
Relational Algebra



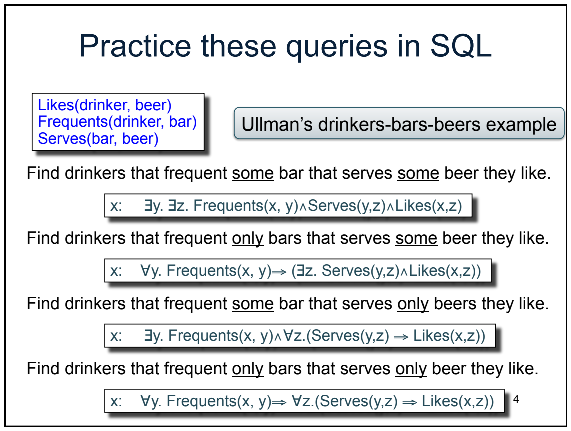
Announcements

- Webquiz 3 is open, due on Sunday
- Homework 3 is posted, due on Tuesday, 2/2
  - We are using Microsoft Azure Cloud services!
  - Use the promotion code you received



What We Learned Last Time

- Subqueries can occur in every clause:
  - SELECT
  - FROM
  - WHERE
- Monotone queries: SELECT-FROM-WHERE
  - Existential quantifier
- Non-monotone queries
  - Universal quantifier
  - Aggregation



Practice these queries in SQL

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

Ullman's drinkers-bars-beers example

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

$x:\quad \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$

Find drinkers that frequent <u>only</u> bars that serves <u>some</u> beer they like.

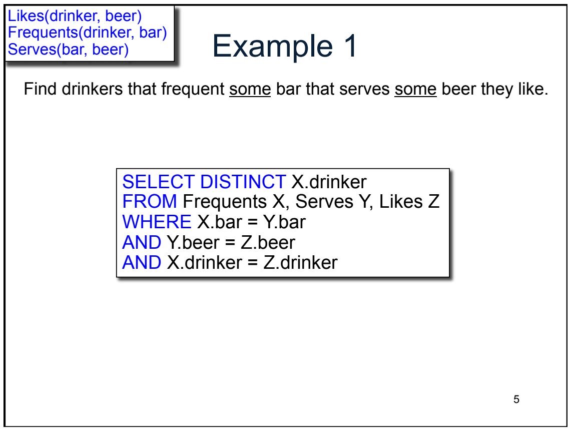
$x:\quad \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they like.

$x:\quad \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

Find drinkers that frequent <u>only</u> bars that serves <u>only</u> beer they like.

$x:\quad \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

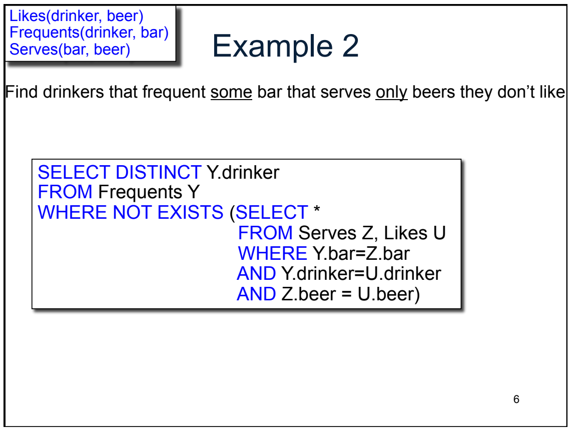


Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Example 1

Find drinkers that frequent <u>some</u> bar that serves <u>some</u> beer they like.

```
SELECT DISTINCT X.drinker
FROM Frequents X, Serves Y, Likes Z
WHERE X.bar = Y.bar
AND Y.beer = Z.beer
AND X.drinker = Z.drinker
```



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

Example 2

Find drinkers that frequent <u>some</u> bar that serves <u>only</u> beers they don't like

```
SELECT DISTINCT Y.drinker
FROM Frequents Y
WHERE NOT EXISTS (SELECT *
                  FROM Serves Z, Likes U
                  WHERE Y.bar=Z.bar
                  AND Y.drinker=U.drinker
                  AND Z.beer = U.beer)
```

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

Example 3

Find drinkers that frequent only bars that serves some beer they like.

(Recall: In example 2, we found drinkers that frequent some bar that serves only beers they don't like)

```
SELECT X.drinker
FROM Frequents X
WHERE X.drinker
NOT IN (SELECT Y.drinker
          FROM Frequents Y
          WHERE NOT EXISTS ( SELECT *
                             FROM Serves Z, Likes U
                             WHERE Y.bar=Z.bar
                             AND Y.drinker=U.drinker
                             AND Z.beer = U.beer))
```

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

Unnesting Aggregates

Find the number of companies in each city

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

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

Equivalent queries

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

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

Unnesting Aggregates

Find the number of products made in each city

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

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

NOT equivalent !
You should know why!

Purchase(pid, product, quantity, price)

GROUP BY v.s. Nested Queries

```
SELECT      product, Sum(quantity) AS TotalSales
FROM        Purchase
WHERE       price > 1
GROUP BY    product
```

```
SELECT DISTINCT  x.product, (SELECT Sum(y.quantity)
                             FROM    Purchase y
                             WHERE x.product = y.product
                             AND y.price > 1)
                             AS TotalSales
FROM        Purchase x
WHERE       x.price > 1
```

Why twice ?

Author(login,name)
Wrote(login,url)

More Unnesting

Find authors who wrote ≥ 10 documents:

Author(login,name)
Wrote(login,url)

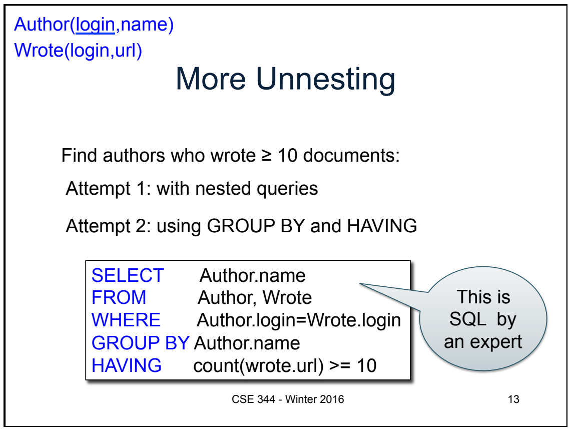
More Unnesting

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

This is SQL by a novice

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



Author(login,name)
Wrote(login,url)

More Unnesting

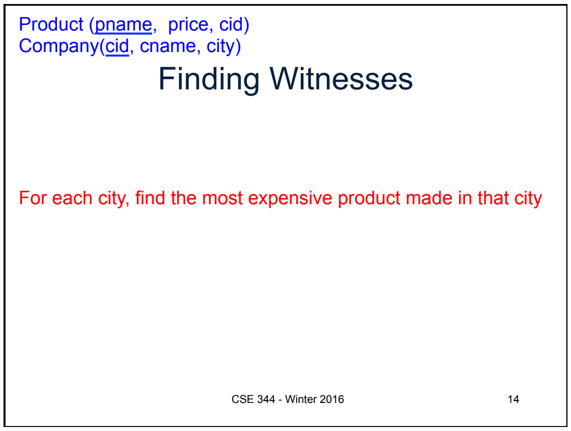
Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

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

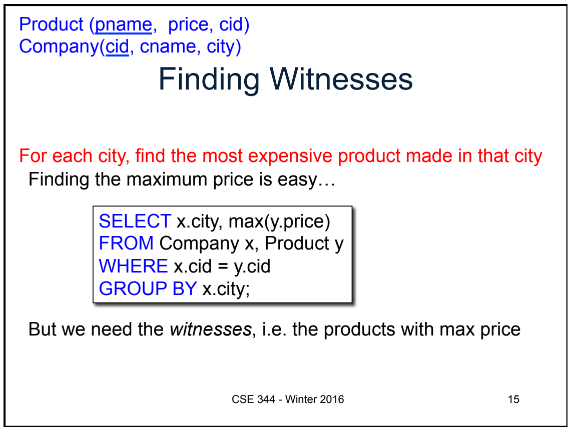
This is SQL by an expert



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

Finding Witnesses

For each city, find the most expensive product made in that city



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

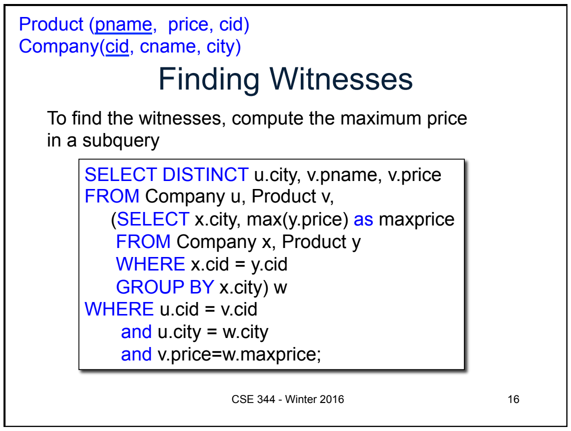
Finding Witnesses

For each city, find the most expensive product made in that city

Finding the maximum price is easy…

```
SELECT x.city, max(y.price)
FROM Company x, Product y
WHERE x.cid = y.cid
GROUP BY x.city;
```

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

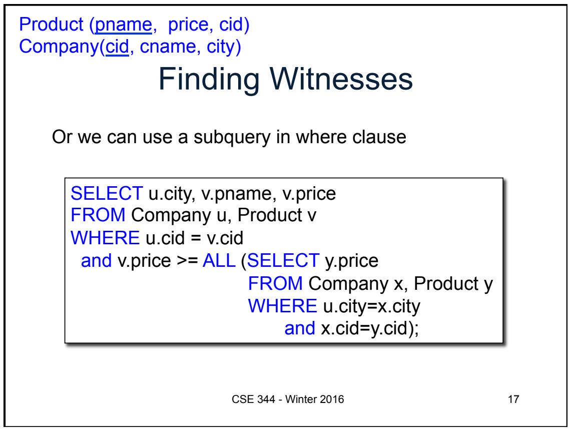


Product (pname, price, cid)
Company(cid, cname, city)

Finding Witnesses

To find the witnesses, compute the maximum price in a subquery

```
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
    (SELECT x.city, max(y.price) as maxprice
     FROM Company x, Product y
     WHERE x.cid = y.cid
     GROUP BY x.city) w
WHERE u.cid = v.cid
    and u.city = w.city
    and v.price=w.maxprice;
```

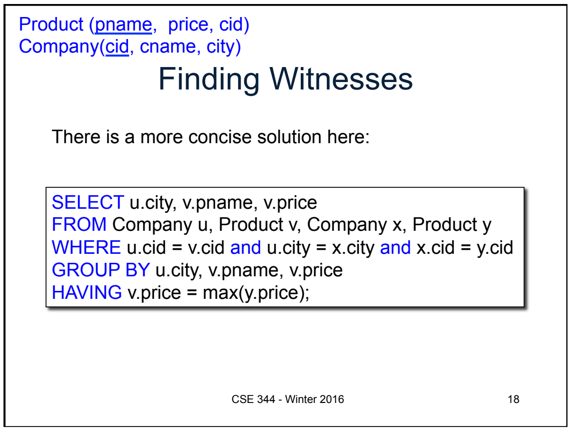


Product (pname, price, cid)
Company(cid, cname, city)

Finding Witnesses

Or we can use a subquery in where clause

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                      FROM Company x, Product y
                      WHERE u.city=x.city
                          and x.cid=y.cid);
```

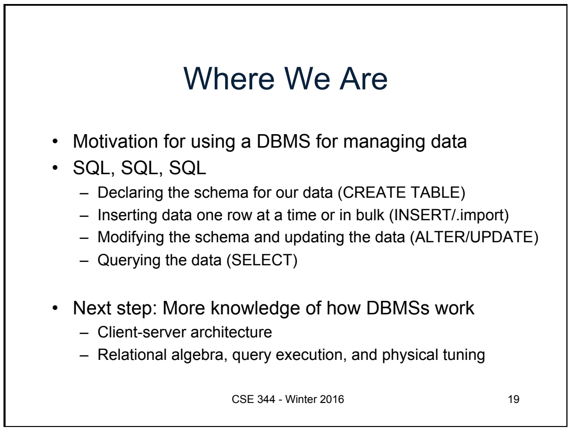


Product (pname, price, cid)
Company(cid, cname, city)

Finding Witnesses

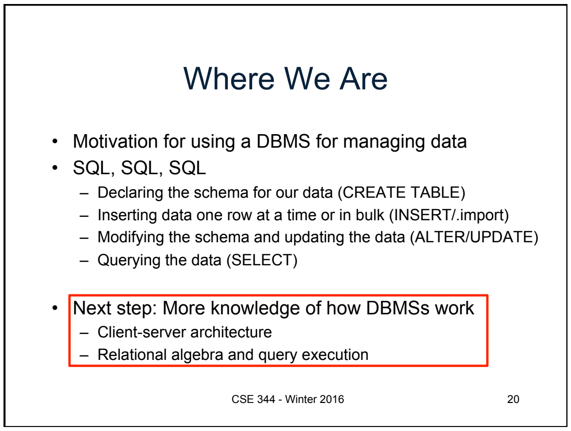
There is a more concise solution here:

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid and u.city = x.city and x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price);
```



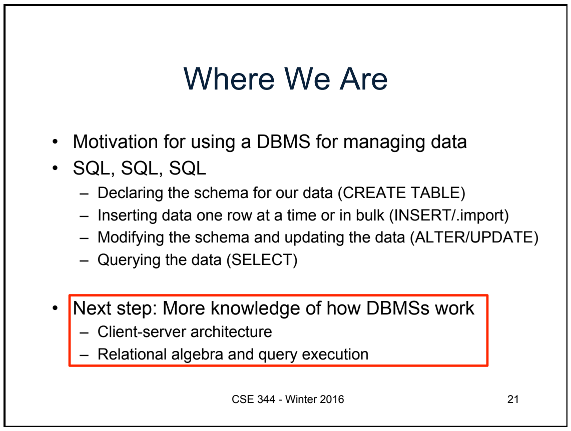
Where We Are

- Motivation for using a DBMS for managing data
- SQL, SQL, SQL
  - Declaring the schema for our data (CREATE TABLE)
  - Inserting data one row at a time or in bulk (INSERT/.import)
  - Modifying the schema and updating the data (ALTER/UPDATE)
  - Querying the data (SELECT)
- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra, query execution, and physical tuning



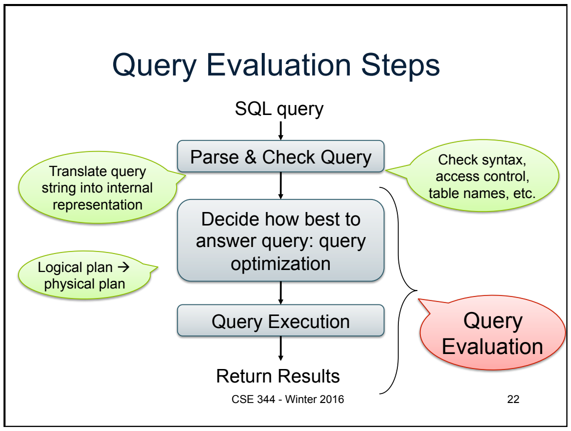
Where We Are

- Motivation for using a DBMS for managing data
- SQL, SQL, SQL
  - Declaring the schema for our data (CREATE TABLE)
  - Inserting data one row at a time or in bulk (INSERT/.import)
  - Modifying the schema and updating the data (ALTER/UPDATE)
  - Querying the data (SELECT)
- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra and query execution



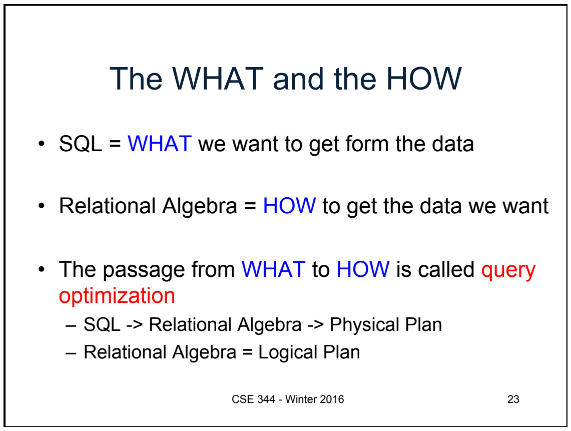
Where We Are

- Motivation for using a DBMS for managing data
- SQL, SQL, SQL
  - Declaring the schema for our data (CREATE TABLE)
  - Inserting data one row at a time or in bulk (INSERT/.import)
  - Modifying the schema and updating the data (ALTER/UPDATE)
  - Querying the data (SELECT)
- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra and query execution



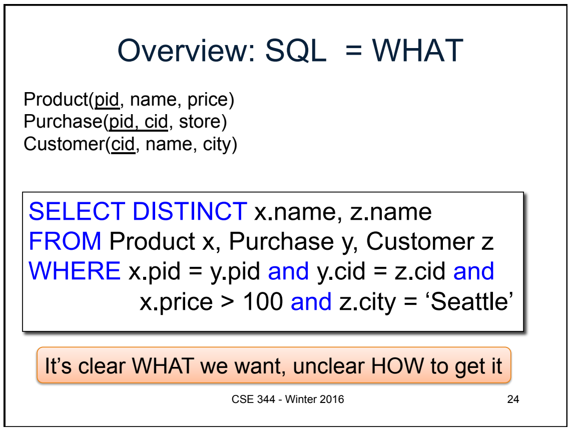
Query Evaluation Steps

SQL query
Parse & Check Query
Decide how best to answer query: query optimization
Query Execution
Return Results
Translate query string into internal representation
Check syntax, access control, table names, etc.
Logical plan → physical plan
Query Evaluation



The WHAT and the HOW

- SQL = WHAT we want to get form the data
- Relational Algebra = HOW to get the data we want
- The passage from WHAT to HOW is called query optimization
  - SQL -> Relational Algebra -> Physical Plan
  - Relational Algebra = Logical Plan

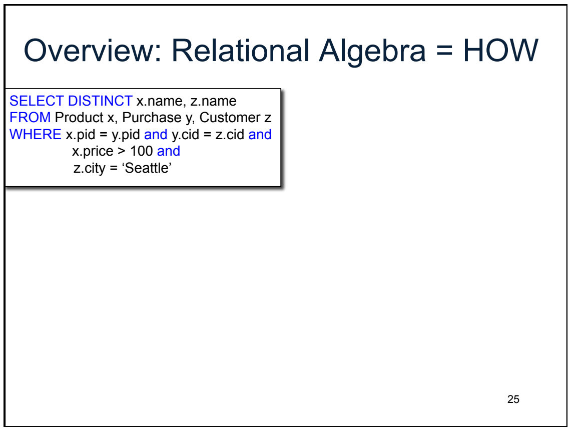


Overview: SQL = WHAT

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
                  x.price > 100 and z.city = 'Seattle'
```

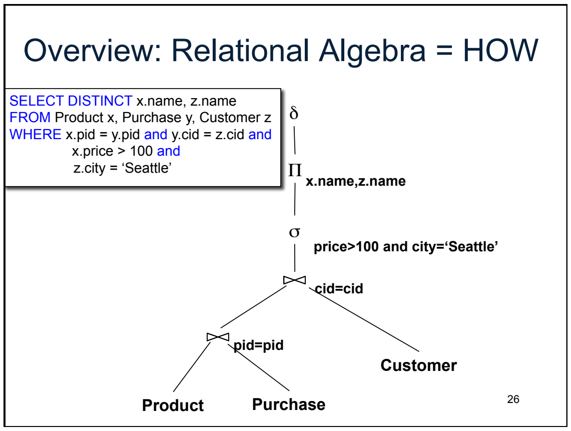
It's clear WHAT we want, unclear HOW to get it



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

25



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

δ

Π x.name,z.name

σ price>100 and city='Seattle'

⋈ cid=cid

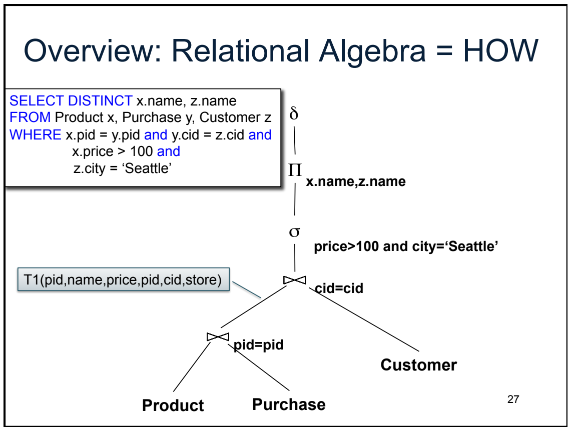
⋈ pid=pid

Product

Purchase

Customer

26



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

δ

Π x.name,z.name

σ price>100 and city='Seattle'

T1(pid,name,price,pid,cid,store)

⋈ cid=cid

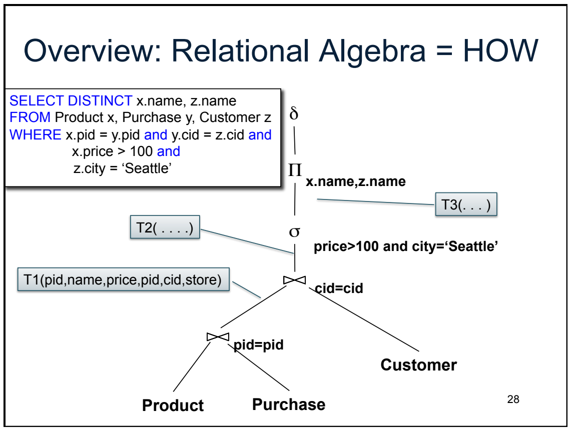
⋈ pid=pid

Product

Purchase

Customer

27



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

δ

Π x.name,z.name

T3(. . .)

T2(. . . .)

σ price>100 and city='Seattle'

T1(pid,name,price,pid,cid,store)

⋈ cid=cid

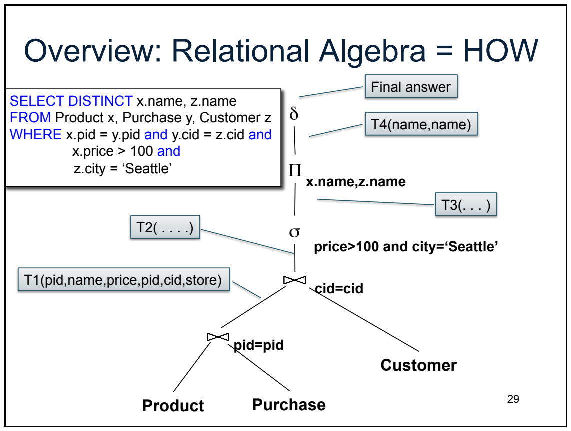
⋈ pid=pid

Product

Purchase

Customer

28



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

Final answer

δ

T4(name,name)

Π x.name,z.name

T3(. . .)

T2(. . . .)

σ price>100 and city='Seattle'

T1(pid,name,price,pid,cid,store)

⋈ cid=cid

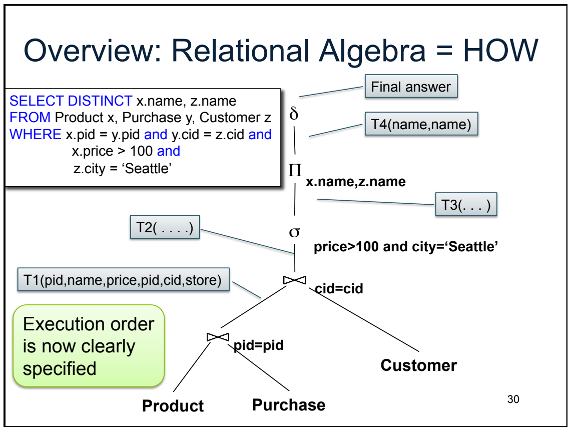
⋈ pid=pid

Product

Purchase

Customer

29



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

Final answer

δ

T4(name,name)

Π x.name,z.name

T3(. . .)

T2(. . . .)

σ price>100 and city='Seattle'

T1(pid,name,price,pid,cid,store)

⋈ cid=cid

Execution order is now clearly specified

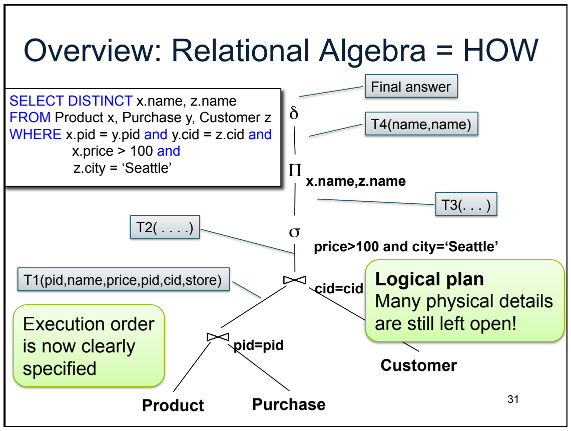
⋈ pid=pid

Product

Purchase

Customer

30



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

Final answer

δ

T4(name,name)

Π x.name,z.name

T3(. . .)

T2(. . . .)

σ price>100 and city='Seattle'

T1(pid,name,price,pid,cid,store)

⋈ cid=cid

Logical plan Many physical details are still left open!

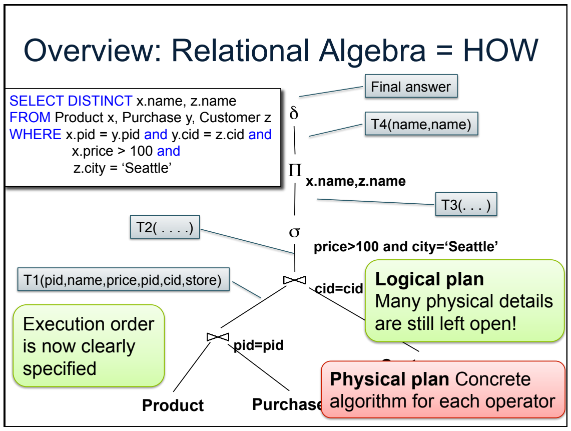
Execution order is now clearly specified

⋈ pid=pid

Customer

Product

Purchase



Overview: Relational Algebra = HOW

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and
          z.city = 'Seattle'
```

Final answer

δ

T4(name,name)

Π x.name,z.name

T3(. . .)

T2(. . . .)

σ price>100 and city='Seattle'

T1(pid,name,price,pid,cid,store)

⋈ cid=cid

Logical plan Many physical details are still left open!

Execution order is now clearly specified

⋈ pid=pid

Product

Purchase

Physical plan Concrete algorithm for each operator