Introduction to Data Management CSE 344

Lecture 7: SQL Wrap-up

Relational Algebra

Announcements

- Webquiz 3 is open, due next Tuesday
- Homework 3 has been posted, due on Wednesday, 2/1
  - Azure can take significant time to set up
  - Don't wait until the last minute to start
  - We support Windows

Using Electronics in Class

- Opened laptops create disturbances to your neighbors
- Please sit in the back if you use your laptop to take notes
- OK if you use surfaces

- And please don't check your email / sms / youtube / fb / etc during class
  – If people are doing this we will have to ban all laptops ☹

Semantics of SQL With Group-By

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

FWGHOS

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
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

What We Learned Yesterday

- Subqueries can occur in every clause:
  - `SELECT`
  - `FROM`
  - `WHERE`

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

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200

Universal quantifiers

Universal quantifiers are hard! ☹

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

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

1. Find *the other* companies that make some product $\geq$ 200

```
SELECT DISTINCT  C.cname
FROM   Company C
WHERE  C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price >= 200)
```

2. Find all companies s.t. all their products have price < 200

```
SELECT DISTINCT  C.cname
FROM   Company C
WHERE  C.cid NOT IN (SELECT P.cid
                     FROM Product P
                     WHERE P.price >= 200)
```

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

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM  Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE P.cid = C.cid and P.price >= 200)
```

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

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

Using ALL:

```
SELECT DISTINCT  C.cname
FROM   Company C
WHERE 200 > ALL (SELECT price
                 FROM Product P
                 WHERE P.cid = C.cid)
```

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

3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Universal quantifiers

∀

Using ALL:

```
SELECT DISTINCT  C.cname
FROM  Company C
WHERE 200 > ALL (SELECT price
                 FROM Product P
                 WHERE P.cid = C.cid)
```

Not supported in sqlite

Question for Database Theory Fans and their Friends

- Can we unnest the *universal quantifier* query?

- We need to first discuss the concept of *monotonicity*

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

Monotone Queries

- Definition A query Q is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

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

Monotone Queries

- Definition A query Q is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q ⇒

pname	city
Gizmo	Lyon
Camera	Lodtz

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

Monotone Queries

- Definition A query Q is monotone if:
  - Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003
iPad	499.99	c001

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz
iPad	Lyon

Monotone Queries

- Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

Monotone Queries

- Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

- Proof. We use the nested loop semantics: if we insert a tuple in a relation R_i, this will not remove any tuples from the answer

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

```
for x1 in R1 do
  for x2 in R2 do
    …
      for xn in Rn do
        if Conditions
          output (a1,…,ak)
```

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

Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200

is not monotone

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

Monotone Queries

- The query:

Find all companies s.t. all their products have price < 200

is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname
Sunworks

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

Monotone Queries

- The query:

Find all companies s.t. <u>all</u> their products have price < 200

is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

⇨

cname
Sunworks

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c001

cid	cname	city
c001	Sunworks	Bonn

⇨

cname

- <u>Consequence</u>: we cannot write it as a SELECT-FROM-WHERE query without nested subqueries

Queries that must be nested

- Queries with universal quantifiers or with negation

Queries that must be nested

- Queries with universal quantifiers or with negation
- Queries that use aggregates in certain ways
  - `sum(..)` and `count(*)` are NOT monotone, because they do not satisfy set containment
  - `select count(*) from R` is not monotone!

```
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

```sql
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)
```

In-class Exercise

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:

```sql
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

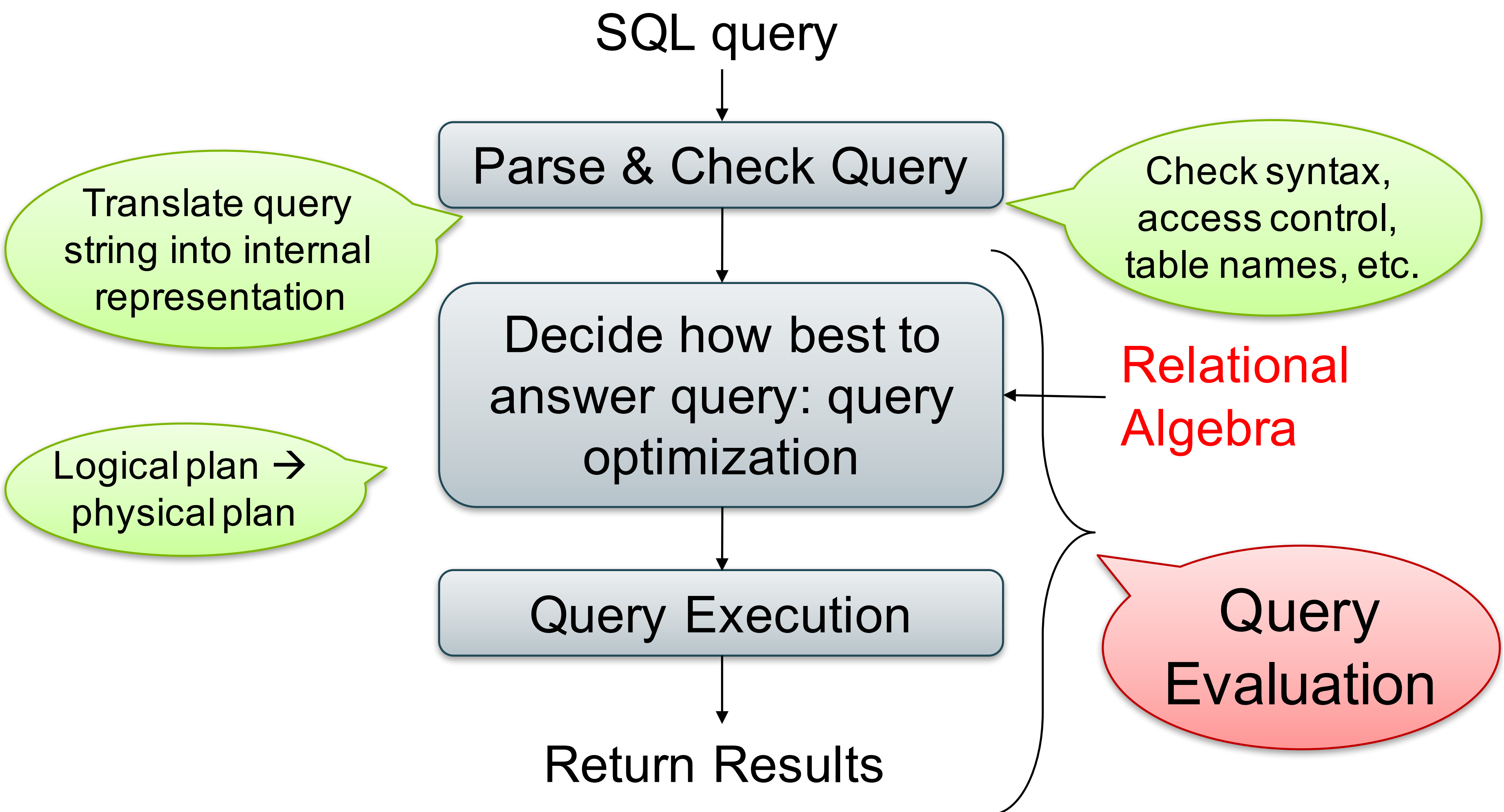
- Data models
- SQL, SQL, SQL
  - Declaring the schema for our data (CREATE TABLE)
  - Inserting data one row at a time or in bulk (INSERT/.import)
  - Querying the data (SELECT)
  - Modifying the schema and updating the data (ALTER/UPDATE)

- Next step: More knowledge of how DBMSs work
  - Relational algebra, query execution, and physical tuning
  - Client-server architecture

The Relational Model

- Instance
  - Organized as “table” or “relation”
- Schema
  - tables and columns / relations and attributes
- Query languages
  - SQL
  - Relational algebra (RA)
- We will learn RA by studying the internals of DBMS

Query Evaluation Steps



The WHAT and the HOW

- SQL = WHAT we want to get from 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

Relation Algebra

- Relations and attributes
- Functions that are applied to relations
  - Return relations
  - Can be composed together
  - Often displayed using a tree rather than linearly
  - Uses Greek symbols: σ, π, δ, etc

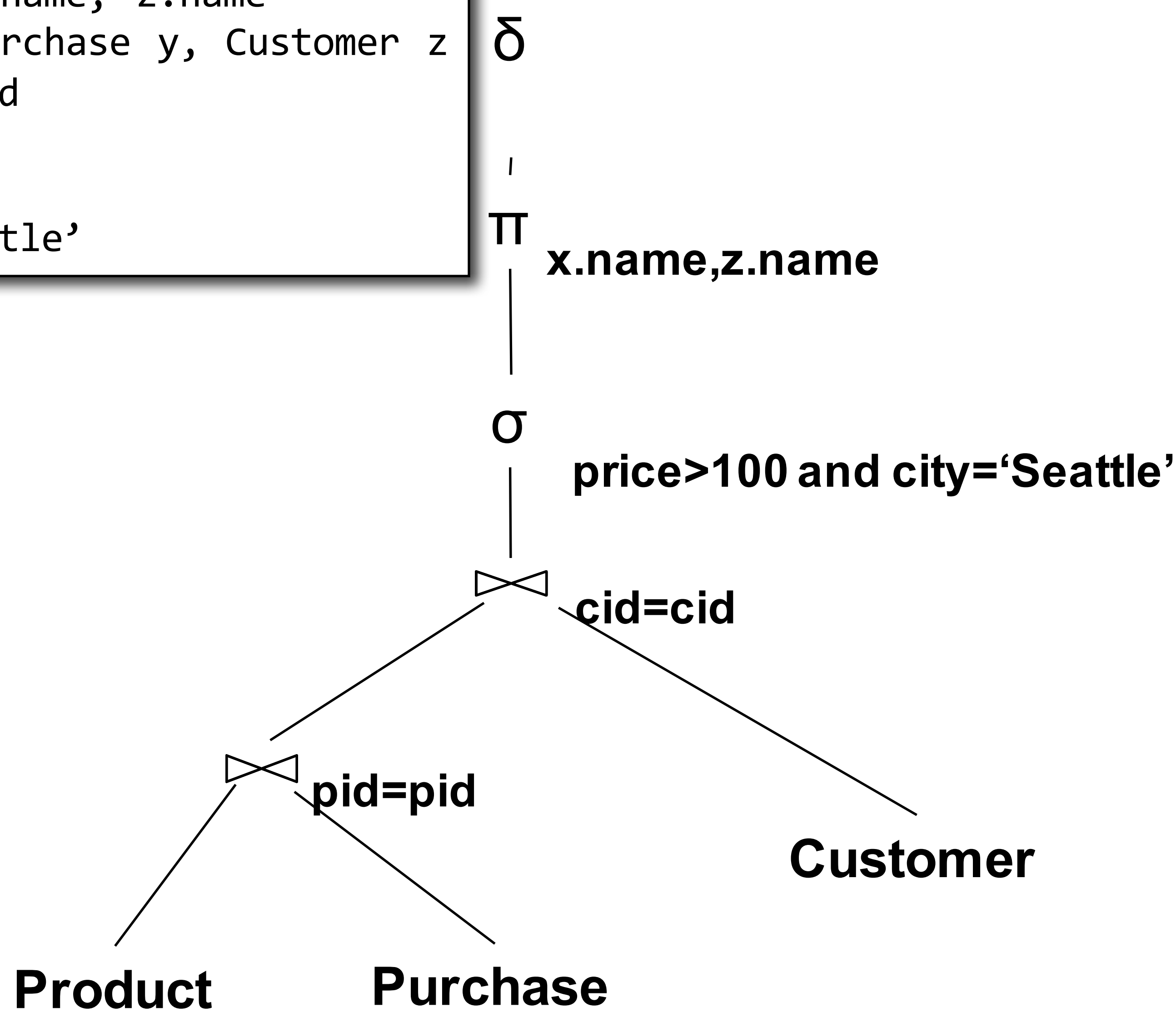
- Language for describing query plans

Overview: Relational Algebra = HOW

```sql
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'
```

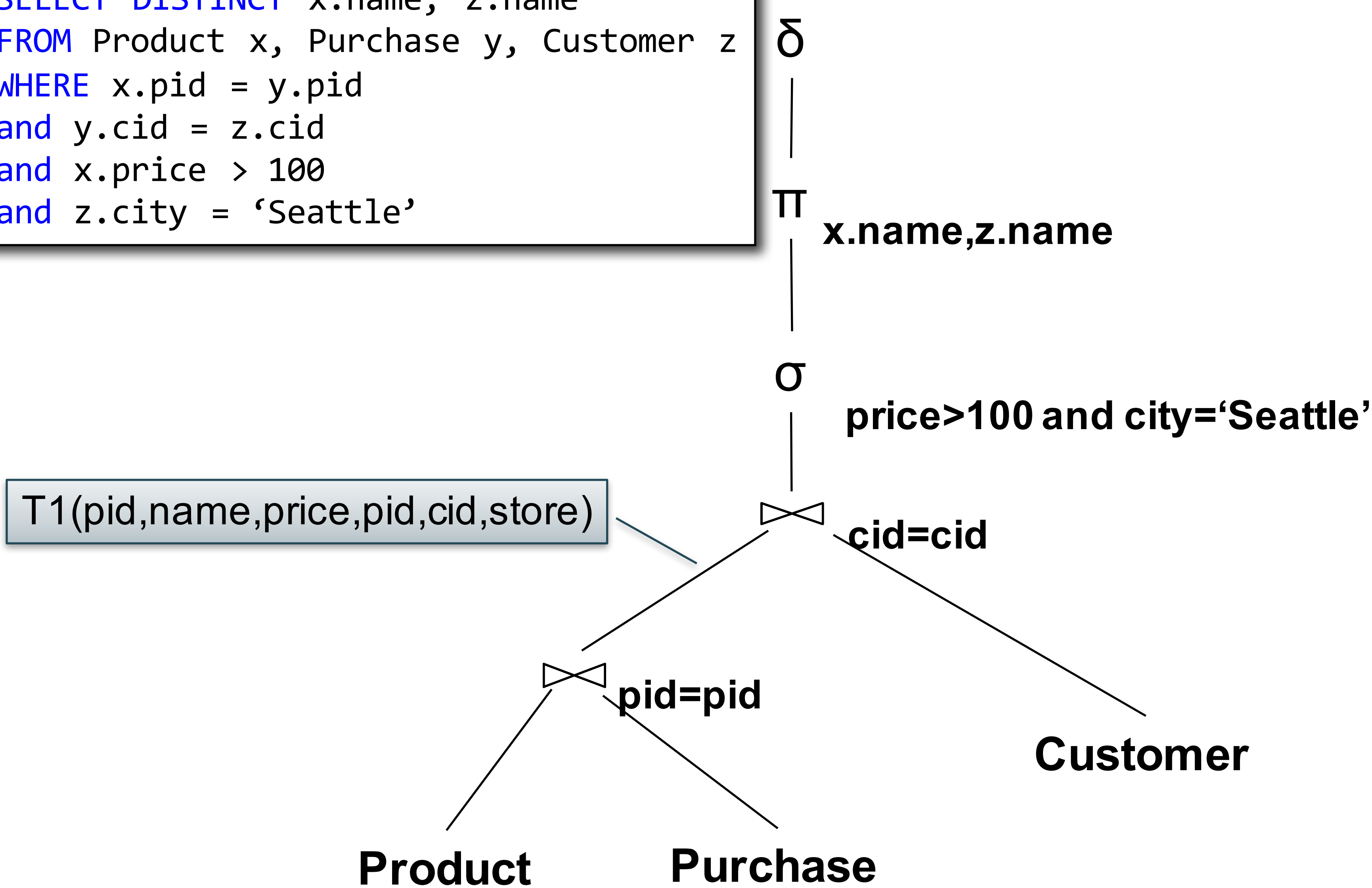
Overview: Relational Algebra = HOW

```
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'
```



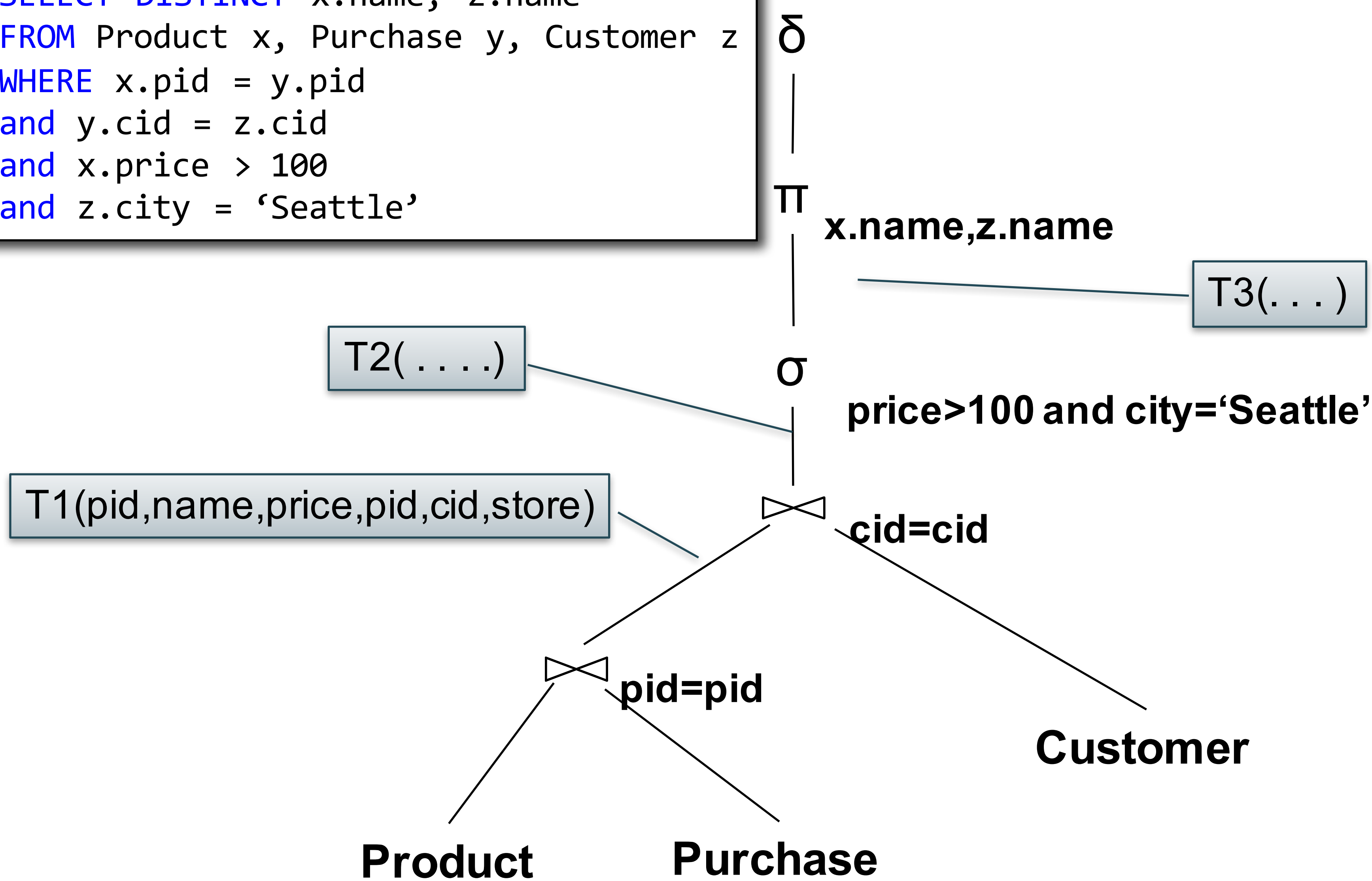
Overview: Relational Algebra = HOW

```
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'
```



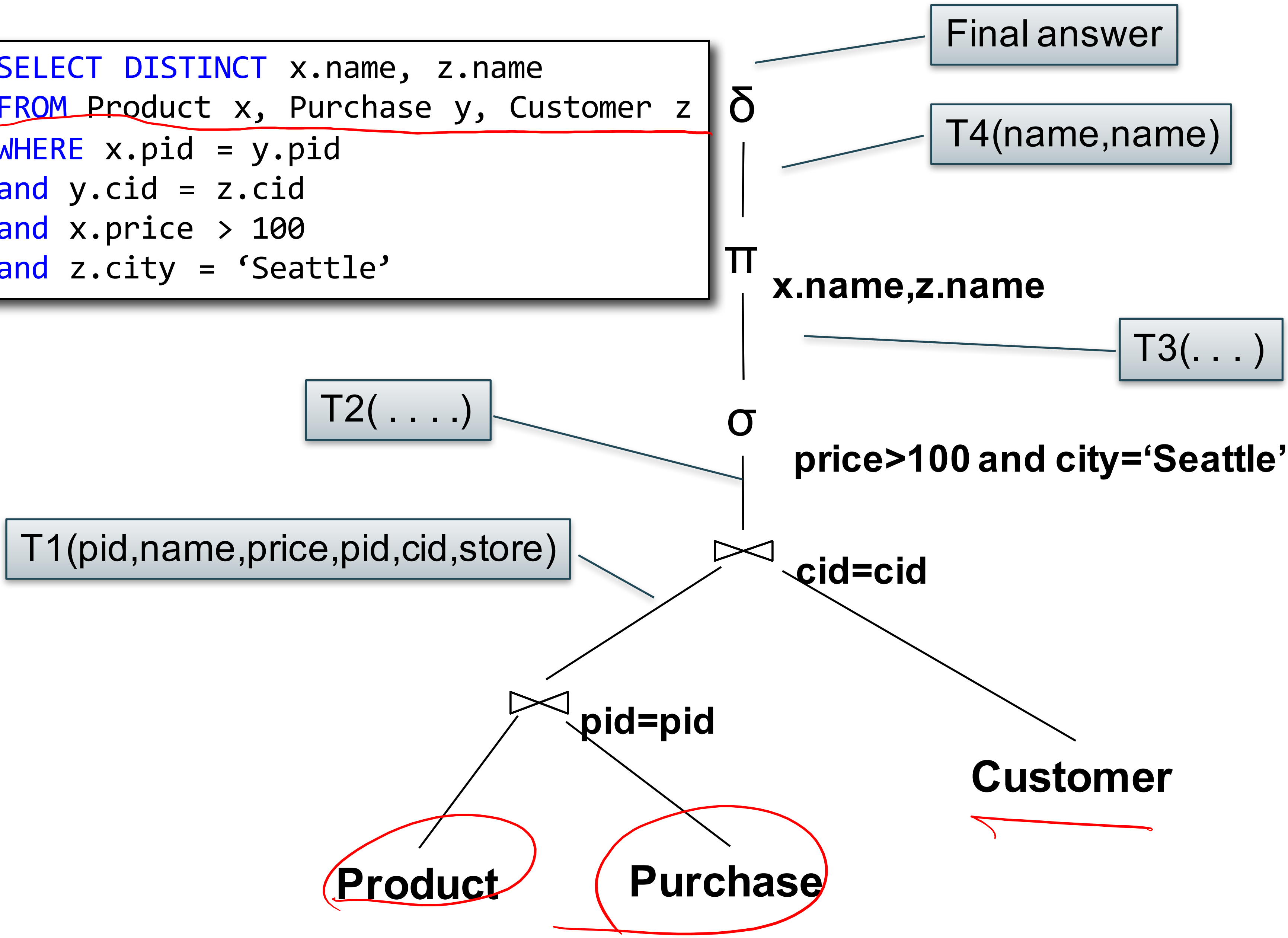
Overview: Relational Algebra = HOW

```
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'
```



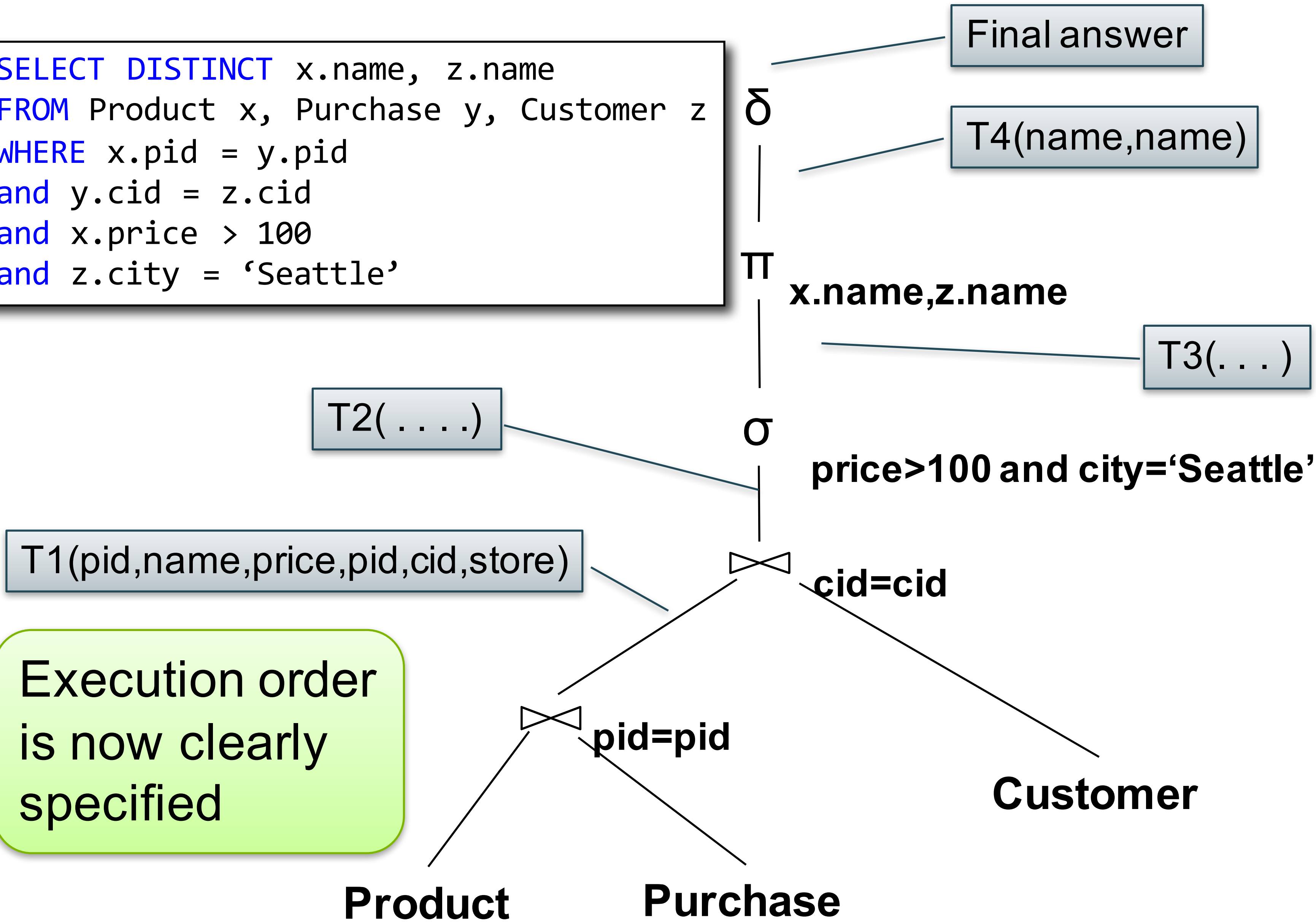
Overview: Relational Algebra = HOW

```
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'
```



Overview: Relational Algebra = HOW

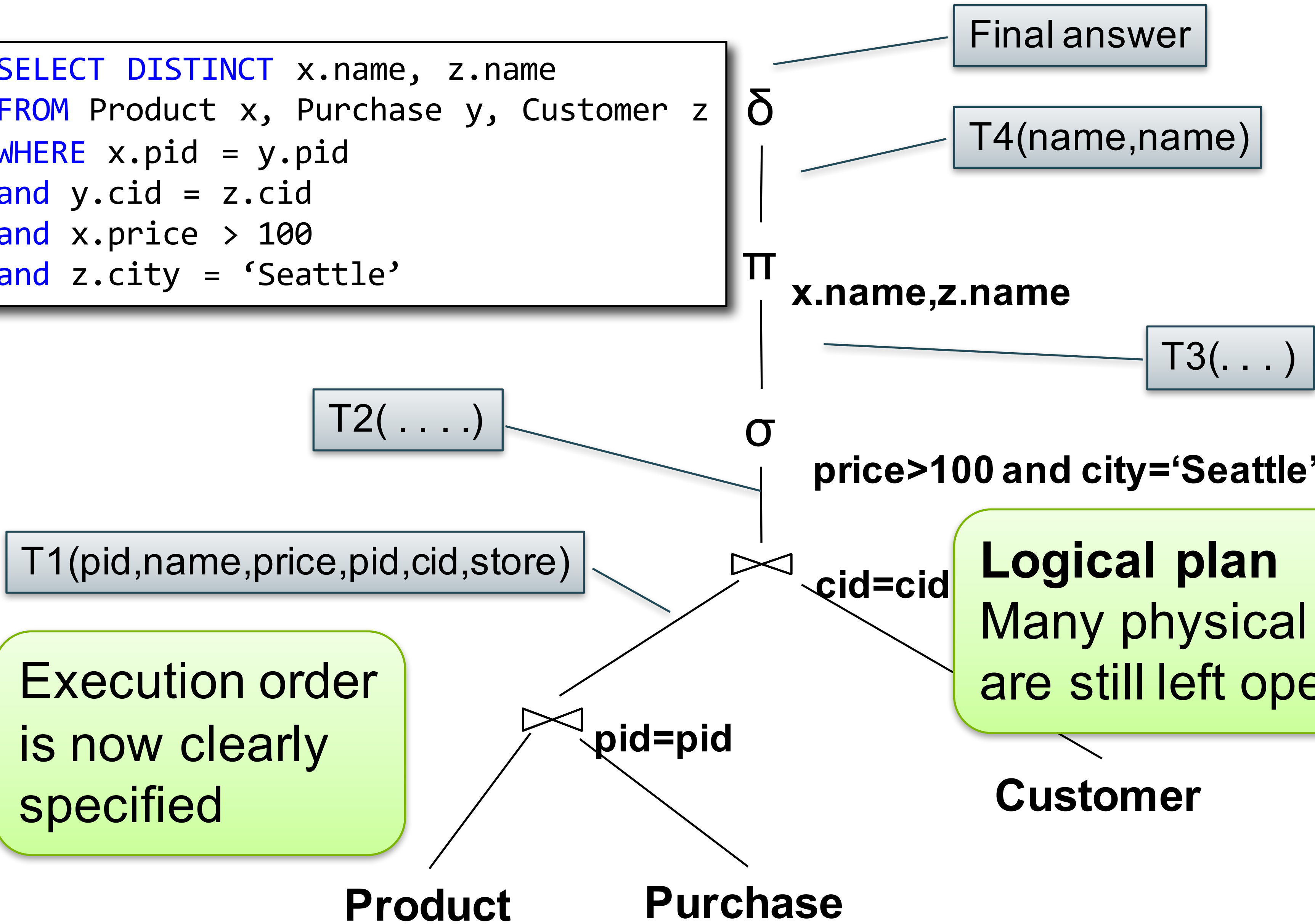
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid
and y.cid = z.cid
and x.price > 100
and z.city = 'Seattle'
```



Execution order is now clearly specified

Overview: Relational Algebra = HOW

```sql
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'
```



Execution order is now clearly specified

Logical plan
Many physical details are still left open!

Overview: Relational Algebra = HOW

```sql
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'
```