Introduction to Data Management CSE 344

Lecture 8-9: Relational Algebra and Query Evaluation

Announcements

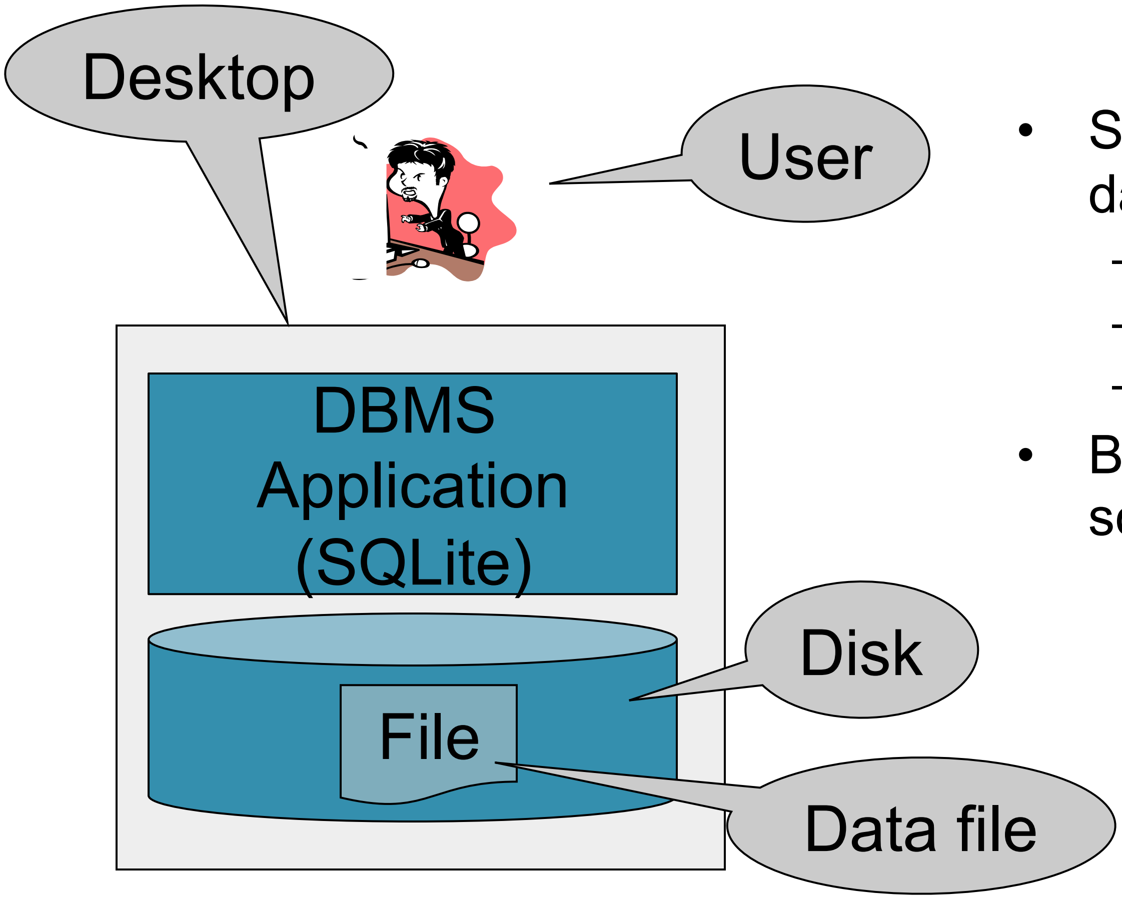
- Webquiz due tomorrow
- Makeup lecture (not mandatory):
  Tuesday, Jan 31st, 3:30 – 4:30, Room TBA
- Homework 3 due on Wednesday
  - IISQLSRV
  - SQL Azure
- Midterm:
  Monday, Feb 6th, 9:30-10:20, in class

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)
  - Tuning queries (CREATE INDEX)

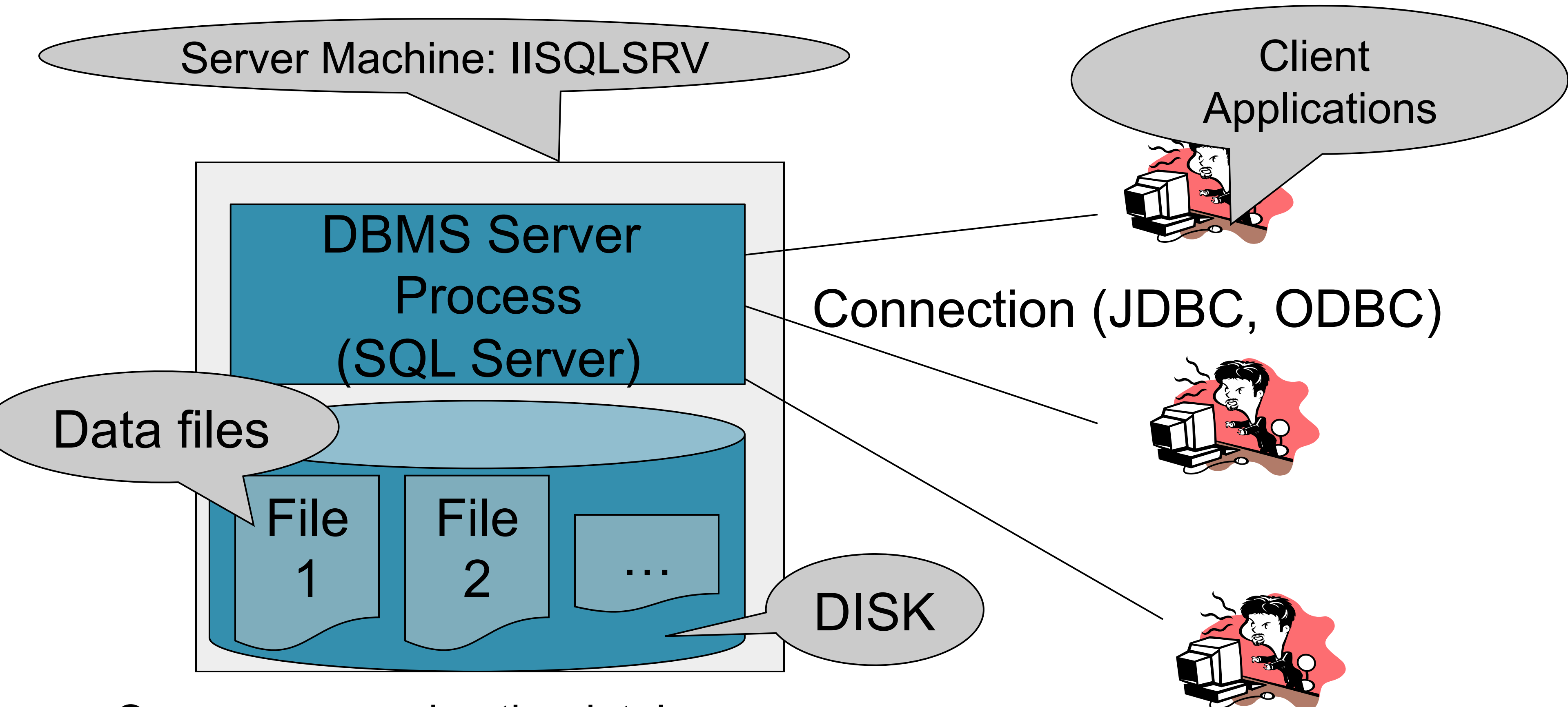
- Next step: More knowledge of how DBMSs work
  - Client-server architecture
  - Relational algebra and query execution

Data Management with SQLite



- So far, we have been managing data with SQLite as follows:
  - One data file
  - One user
  - One DBMS application
- But only a limited number of scenarios work with such model

Client-Server Architecture



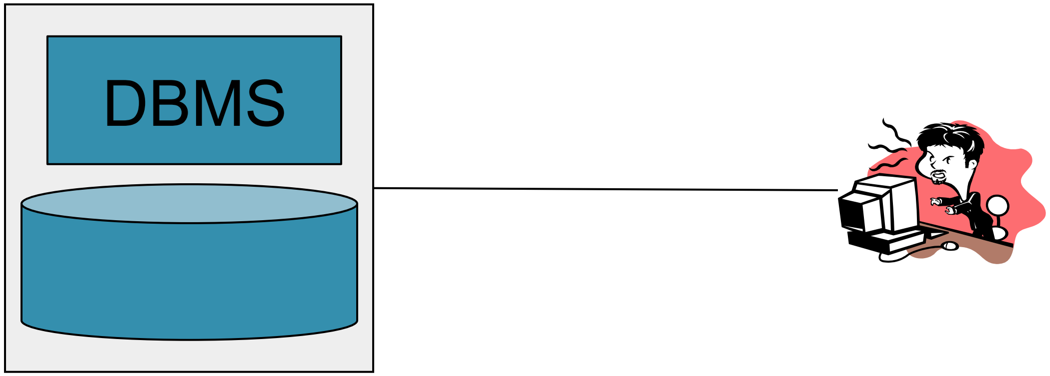
- One server running the database
- Many clients, connecting via the JDBC (Java Database Connectivity Protocol)

Client-Server Architecture

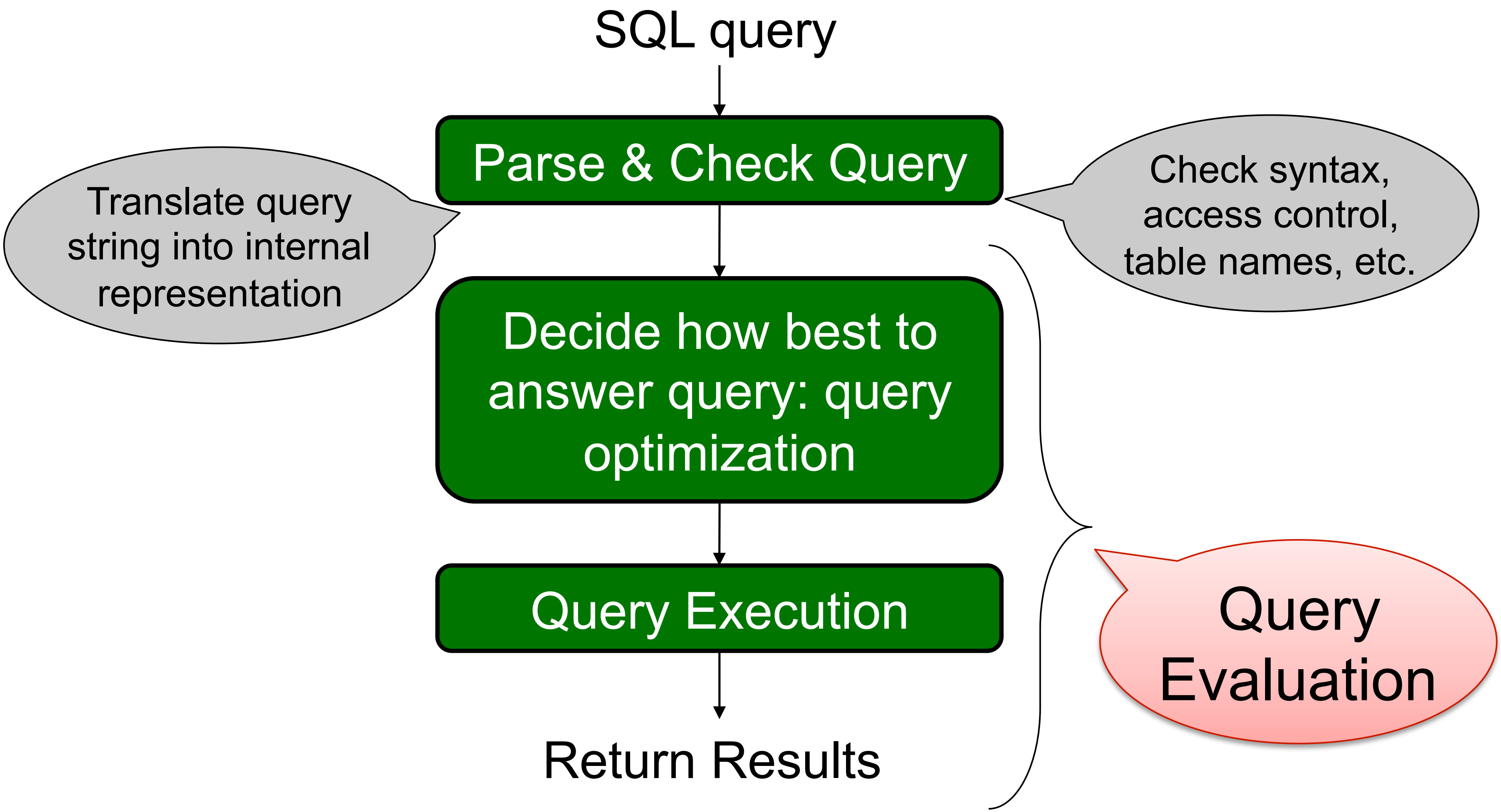
- One *server* that stores the database (called DBMS or RDBMS):
  - Your own desktop, or
  - Some beefy system (IISQLSRV1), or
  - A cloud service (SQL Azure)
- Many *clients* run apps and connect to DBMS
  - Microsoft's Management Studio (for SQL Server), or
  - psql (for postgres)
  - Some Java program or some C++ program
- Clients "talk" to server using JDBC protocol

Using a DBMS Server

1. Client application establishes connection to server
2. Client must authenticate self
3. Client submits SQL commands to server
4. Server executes commands and returns results



Query Evaluation Steps Review



Question: How does Query Evaluation Work?

The WHAT and the HOW

- In SQL we write WHAT we want to get form the data
- The database system needs to figure out HOW to get the data we want
- The passage from WHAT to HOW goes through the Relational Algebra

Physical Data Independence

Overview: SQL = WHAT

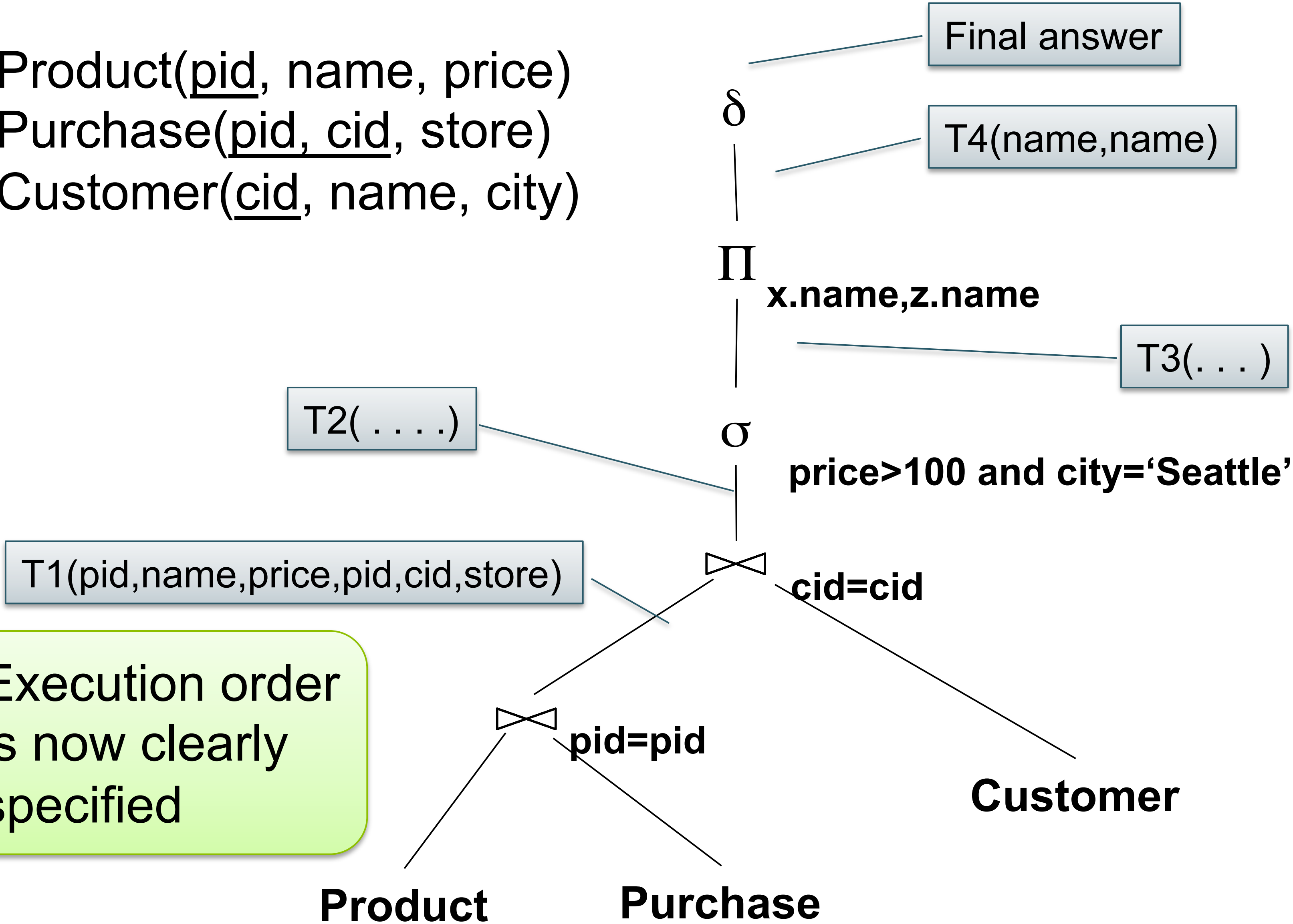
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

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

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

Overview: Relational Algebra = HOW

Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)



Execution order is now clearly specified

Sets v.s. Bags

- Sets: {a,b,c}, {a,d,e,f}, { }, . . .
- Bags: {a, a, b, c}, {b, b, b, b, b}, . . .

Relational Algebra has two semantics:

- Set semantics = standard Relational Algebra
- Bag semantics = extended Relational Algebra

Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection σ
- Projection Π
- Join $\bowtie$
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

RA

Extended RA

Union and Difference

$$R1 \cup R2$$
$$R1 - R2$$

What do they mean over bags ?

What about Intersection ?

- Derived operator using minus

$$R1 \cap R2 = R1 - (R1 - R2)$$

- Derived using join (will explain later)

$$R1 \cap R2 = R1 \bowtie R2$$

Selection

- Returns all tuples which satisfy a condition

- Examples $\sigma_c(R)$
  - $\sigma_{\text{Salary > 40000}}$ (Employee)
  - $\sigma_{\text{name = "Smith"}}$ (Employee)
- The condition c can be =, <, ≤, >, ≥, <>

Employee

SSN	Name	Salary
1234545	John	200000
5423341	Smith	600000
4352342	Fred	500000

$\sigma_{\text{Salary > 40000}}$ (Employee)

SSN	Name	Salary
5423341	Smith	600000
4352342	Fred	500000

Projection

- Eliminates columns

$$\Pi_{A1,\ldots,An}(R)$$

- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)
  - Answer(SSN, Name)

Different semantics over sets or bags! Why?

Employee

SSN	Name	Salary
1234545	John	20000
5423341	John	60000
4352342	John	20000

$\Pi_{Name,Salary}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient?

Cartesian Product

- Each tuple in R1 with each tuple in R2

$$R1 \times R2$$

- Very rare in practice; mainly used to express joins

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee $\times$ Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Renaming

- Changes the schema, not the instance

$$\rho_{B1,\ldots,Bn}(R)$$

- Example:
  - $\rho_{N,\,S}(\text{Employee}) \rightarrow \text{Answer}(N, S)$

Not really used by systems, but needed on paper

Natural Join

$$R1 \bowtie R2$$

- Meaning: $R1 \bowtie R2 = \Pi_A(\sigma(R1 \times R2))$
- Where:
  - The selection σ checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

Natural Join

R

A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

R ⋈ S =
$\Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of R ⋈ S ?
- Given R(A, B, C), S(D, E), what is R ⋈ S ?
- Given R(A, B), S(A, B), what is R ⋈ S ?

Theta Join

- A join that involves a predicate

$$R1 \bowtie_{\theta} R2 \quad = \quad \sigma_{\theta}(R1 \times R2)$$

- Here θ can be any condition

Eq-join

- A theta join where θ is an equality

$$R1 \bowtie_{A=B} R2 \quad = \quad \sigma_{A=B} (R1 \times R2)$$

- This is by far the most used variant of join in practice

So Which Join Is It ?

- When we write R ⋈ S we usually mean an eq-join, but we often omit the equality predicate when it is clear from the context

More Joins

- **Outer join**
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes

- Variants
  - Left outer join
  - Right outer join
  - Full outer join

Outer Join Example

AnonPatient P

age	**zip**	**disease**
54	98125	heart
20	98120	flu
33	98120	lung

AnnonJob J

job	**age**	**zip**
lawyer	54	98125
cashier	20	98120

P ⟕ V

age	**zip**	**disease**	**job**
54	98125	heart	lawyer
20	98120	flu	cashier
33	98120	lung	null

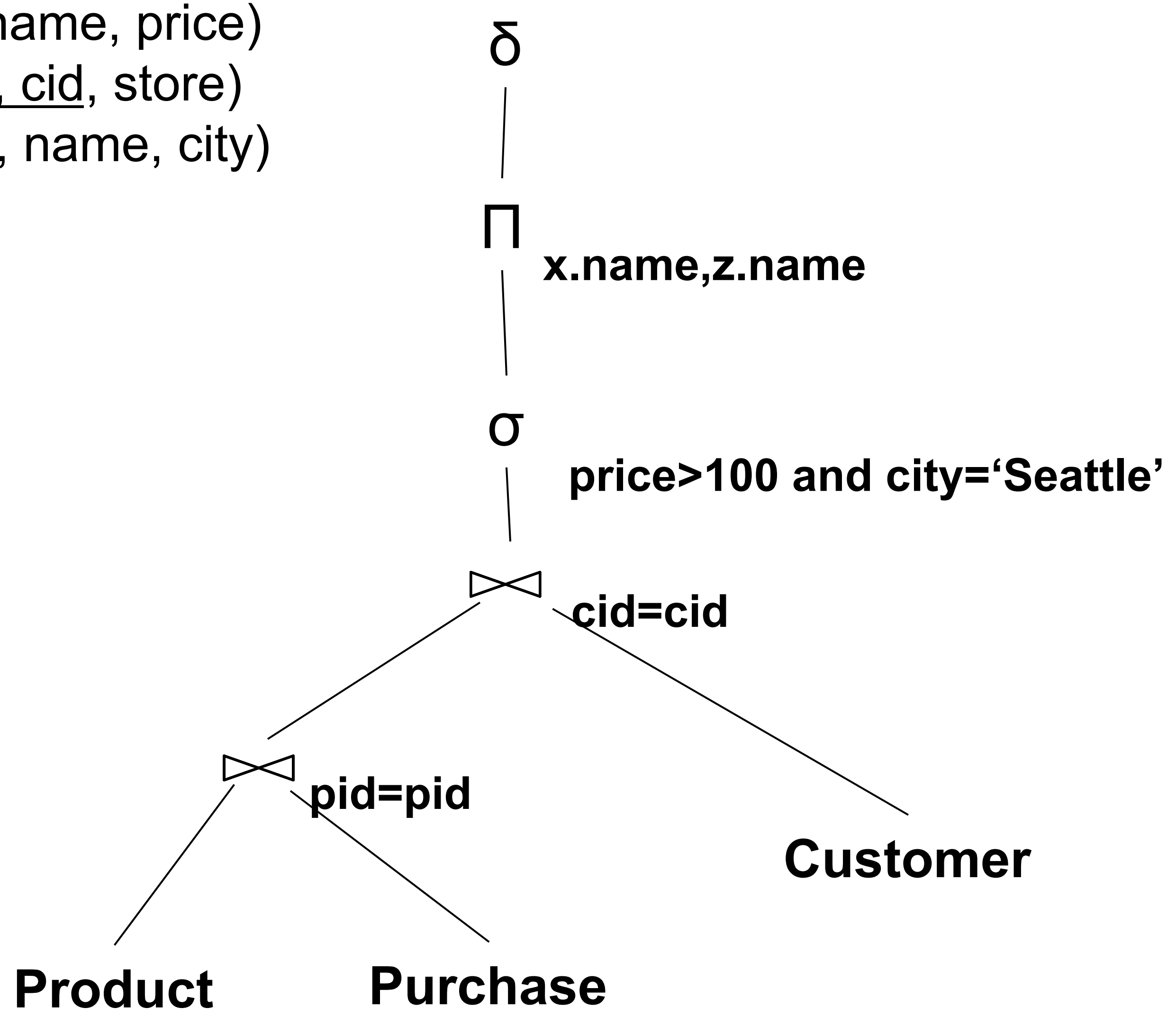
From SQL to RA

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 = y.cid and
          x.price > 100 and z.city = 'Seattle'
```

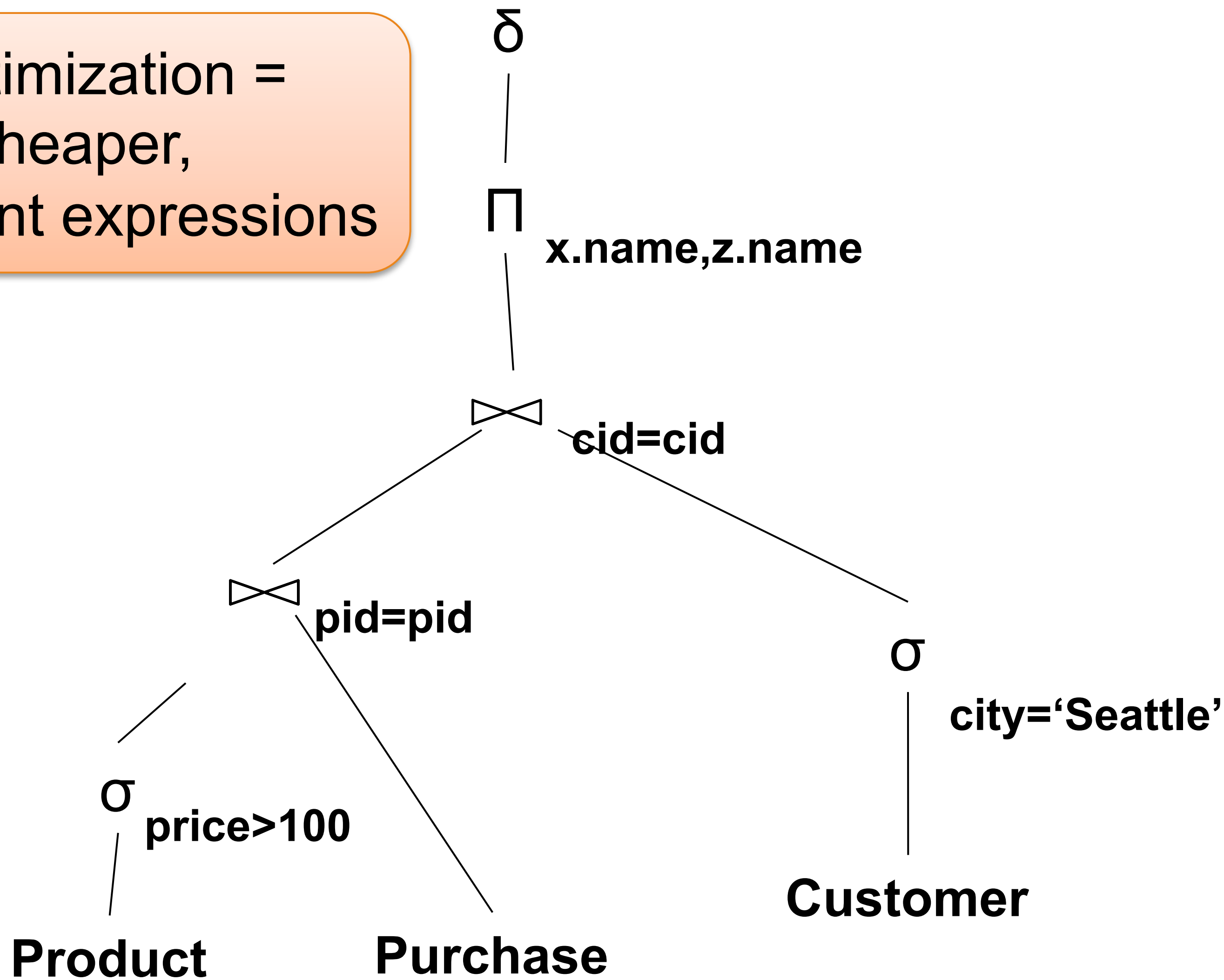
From SQL to RA

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



An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions



Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
- Sorting τ

Logical Query Plan

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```

T3(city, c)

$\Pi_{\text{city, c}}$

T2(city,p,c)

$\sigma_{p > 100}$

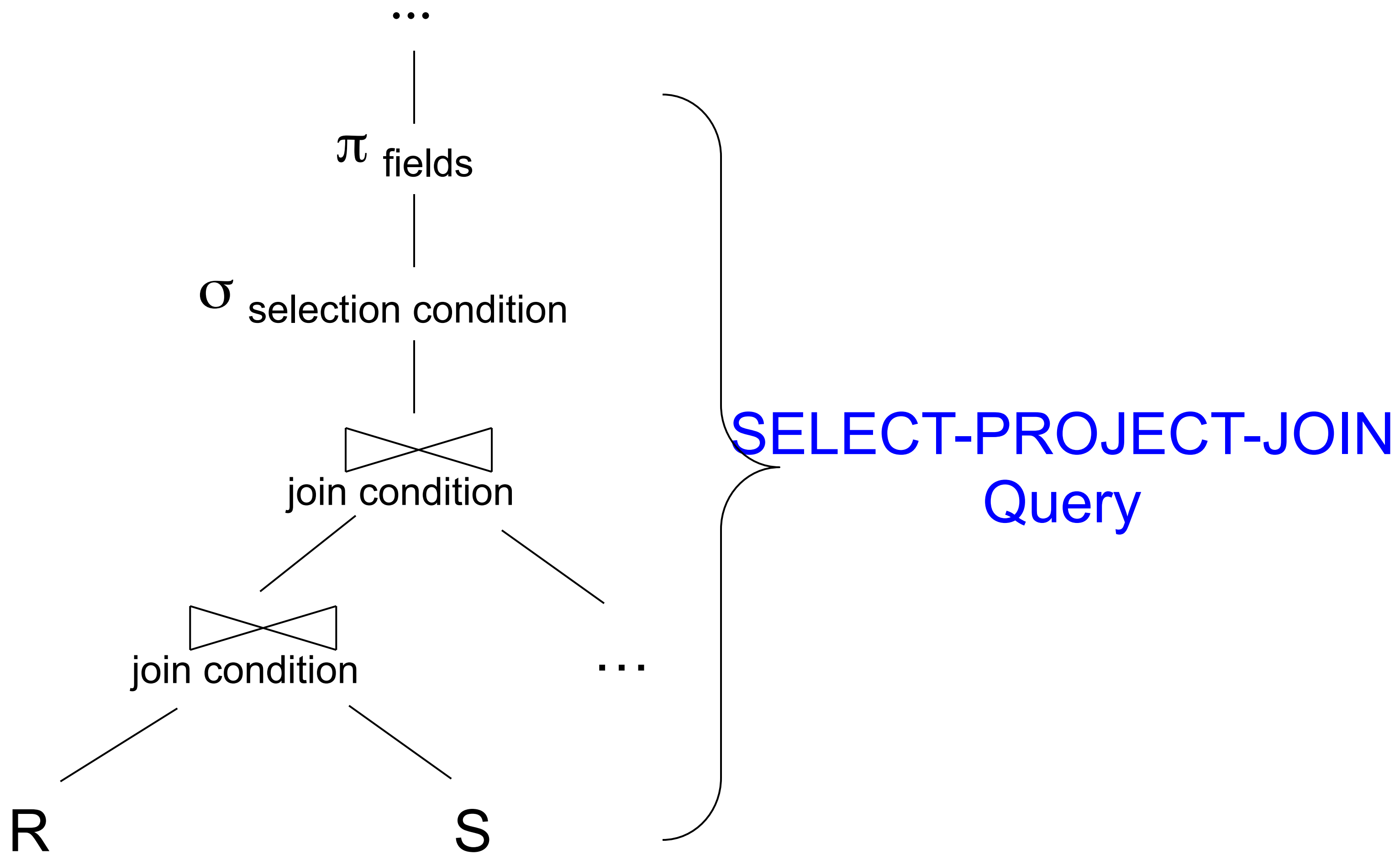
T1(city,p,c)

$\gamma_{\text{city, sum(price)} \rightarrow \text{p, count(*)} \rightarrow \text{c}}$

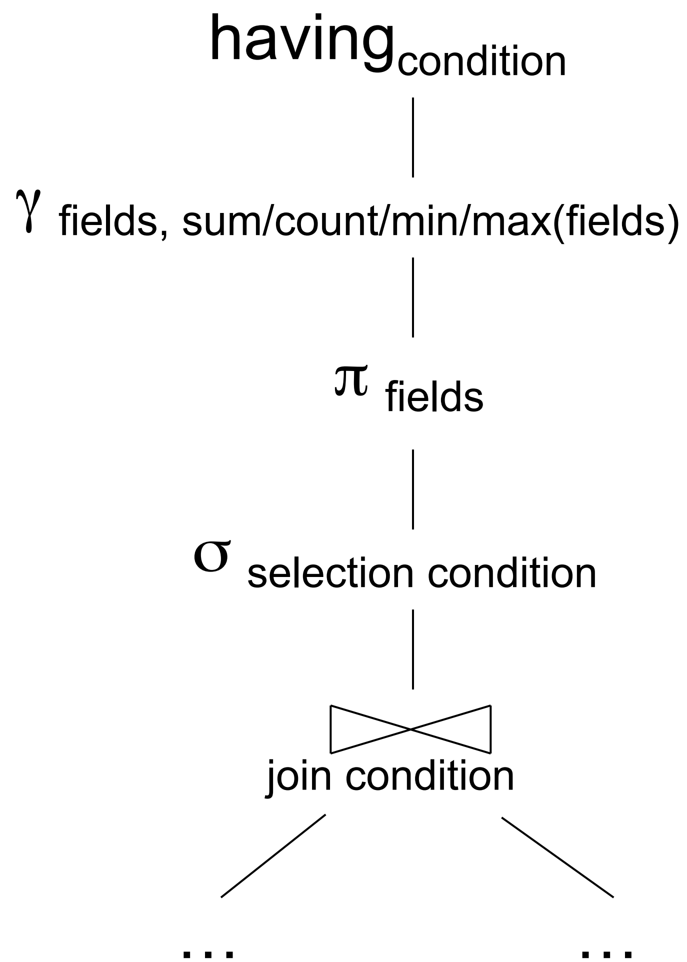
sales(product, city, price)

T1, T2, T3 = temporary tables

Typical Plan for Block (1/2)



Typical Plan For Block (2/2)



Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
   and not exists
     (SELECT *
      FROM Supply P
      WHERE P.sno = Q.sno
            and P.price > 100)
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
            and P.price > 100)
```

Correlation !

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
             and P.price > 100)
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and Q.sno not in
      (SELECT P.sno
       FROM Supply P
       WHERE P.price > 100)
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
   EXCEPT
(SELECT P.sno
  FROM Supply P
  WHERE P.price > 100)
```

←

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = 'WA'
    and Q.sno not in
       (SELECT P.sno
        FROM Supply P
        WHERE P.price > 100)
```

EXCEPT = set difference

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

How about Subqueries?

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
   EXCEPT
 (SELECT P.sno
  FROM Supply P
  WHERE P.price > 100)
```

Finally…