CSE 344

Lectures 9: Relational Algebra

Announcements

- Homework 2 due tonight!
- Homework 3 is posted, due next Thursday!
- Webquiz 3 due tomorrow night!
- Reminder about Discussion Board
  - Post your question on HW/WQ here
  - Feel free to answer your friends' questions and discuss concepts (no solution please ☺)
  - You can set alert to get email notification
- Today's lecture: 2.4 and 5.1-5.2

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
  - Relational algebra and query execution
  - Client-server architecture

Relational Algebra

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
- Cartesian product $\times$, join $\bowtie$
- Rename ρ
- Duplicate elimination δ
- Grouping and aggregation γ
- Sorting τ

RA

Extended RA

Why learn RA?

- SQL incorporates RA at its center
- When DBMS processes a query, it is translated into an RA expression internally and is used by the query optimizer
- Why Algebra?

Why learn RA?

- Why Algebra?
  - Has both Operators and Atomic Operands
  - (x+y) * (z - 3)
  - Similarly,

    $\pi_{zip} (\sigma_{disease='heart'}(Patient))$

Union and Difference

$R1 \cup R2$

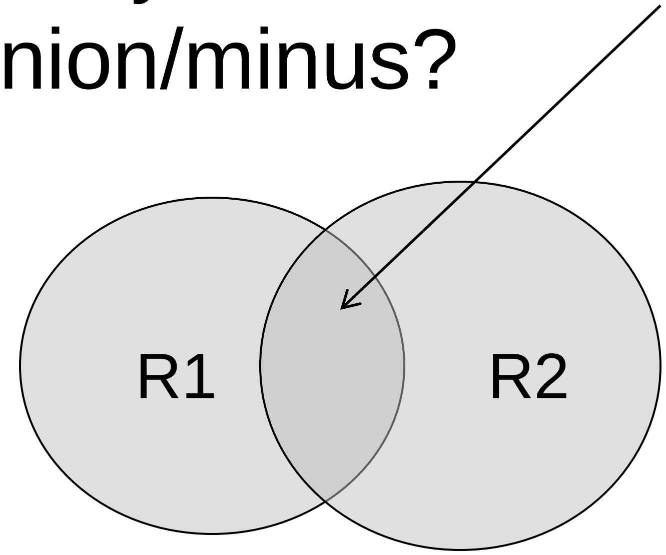
$R1 - R2$

For set operations, R1 and R2

- must have identical schemas
- their attributes must have the same order, i.e. R1(A, B) and R2(B, A) is not allowed

What about Intersection ?

- Can you derive R1 ∩ R2 using union/minus?



R2 – (R2 – R1)

R1 – (R1 – R2)

(R1 U R2) – ((R1 U R2) – R1) - ((R1 U R2) – R2)

What about Intersection ?

- Derived operator using minus

$$\boxed{R1 \cap R2 = R1 - (R1 - R2)}$$

- Derived using join (will explain later)

$$\boxed{R1 \cap R2 = R1 \bowtie R2}$$

Union, Difference, Intersection over Bags

R1 ∪ R2

R1 – R2

R1 ∩ R2

What do they mean over bags ?

R1

A
1
1
1

R2

A
1
1

How many 1's in

- R1 ∪ R2: 5
- R1 – R2: 1
- R2 – R1: 0
- R1 ∩ R2: 2

Selection

- Returns all tuples which satisfy a condition

$$\boxed{\sigma_c(R)}$$

What does Selection Correspond to in SQL?

Ans: WHERE clause

- Examples
  - $\sigma_{\text{Salary > 40000}}$ (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

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

What does Projection Correspond to in SQL?
Ans: SELECT clause

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

Different semantics over sets or bags! Why?

Employee

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

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

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient? Ans: Bag

Checking and removing duplicates is expensive

Composing RA Operators

Patient

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{\text{zip,disease}}(\text{Patient})$

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

$\sigma_{\text{disease='heart'}}(\text{Patient})$

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

$\pi_{\text{zip}}\ (\sigma_{\text{disease='heart'}}(\text{Patient}))$

zip
98120
98125

Cartesian/Cross Product

- Each tuple in R1 with each tuple in R2

$$\boxed{R1 \times R2}$$

- Rare in practice; mainly used to express joins

Cross-Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee Dependent

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

Disambiguate attributes if necessary

Employee.EmpSSN

Dependent.EmpSSN

Renaming

- Changes the schema, not the instance

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

- Example:
  - $\rho_{N, S}$(Employee) → Answer(N, S)
  - Given R(A, B)
    - $\rho_S(R)$: Renamed relation S(A, B)
    - $\rho_{S(X,Y)}(R)$ or S= $\rho_{X,Y}(R)$: Renamed relation S(X, Y)
    - Sometimes written as S = $\rho_{A\text{->}X,\ B\text{->}Y}(R)$

Not really used by systems, but needed on paper

Natural Join

$$\boxed{R1 \bowtie R2}$$

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

Natural Join Example

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 Example 2

AnonPatient P

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

Voters V

name	**age**	**zip**
p1	54	98125
p2	20	98120

$P \bowtie V$

age	**zip**	**disease**	**name**
54	98125	heart	p1
20	98120	flu	p2

Natural Join

- Given schemas R(A, B, C, D), S(A, C, E), what is the schema of T = R $\bowtie$ S ?
- Ans: T(A, B, C, D, E)

- Given R(A, B, C), S(D, E), what is R $\bowtie$ S ?
- Ans: R X S

- Given R(A, B), S(A, B), what is R $\bowtie$ S ?
- Ans: R $\cap$ S

Theta Join

- A join that involves a predicate

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

- Here θ can be any condition
- For our voters/disease example:

$$P \bowtie_{P.zip = V.zip \text{ and } P.age < V.age + 5 \text{ and } P.age > V.age - 5} V$$

Equijoin

- A theta join where θ is an equality

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

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

Equijoin Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie_{P.age=V.age} V$

age	P.zip	disease	name	V.zip
54	98125	heart	p1	98125
20	98120	flu	p2	98120

Join Summary

- **Theta-join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_\theta S = \pi_A (\sigma_\theta(R \times S))$
  - Join condition θ consists only of equalities
  - Projection π_A drops all redundant attributes
- **Natural join**: $R \bowtie S = \pi_A (\sigma_\theta(R \times S))$
  - Equijoin
  - Equality on **all** fields with same name in R and in S

So Which Join Is It ?

- When we write R $\bowtie$ S we usually mean an equijoin, 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

Some Examples

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

Q2: Name of supplier of parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}(\text{Part}))$

Q3: Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}(\text{Part}) \cup \sigma_{pcolor='red'}(\text{Part})))$

From SQL to RA

From SQL to RA

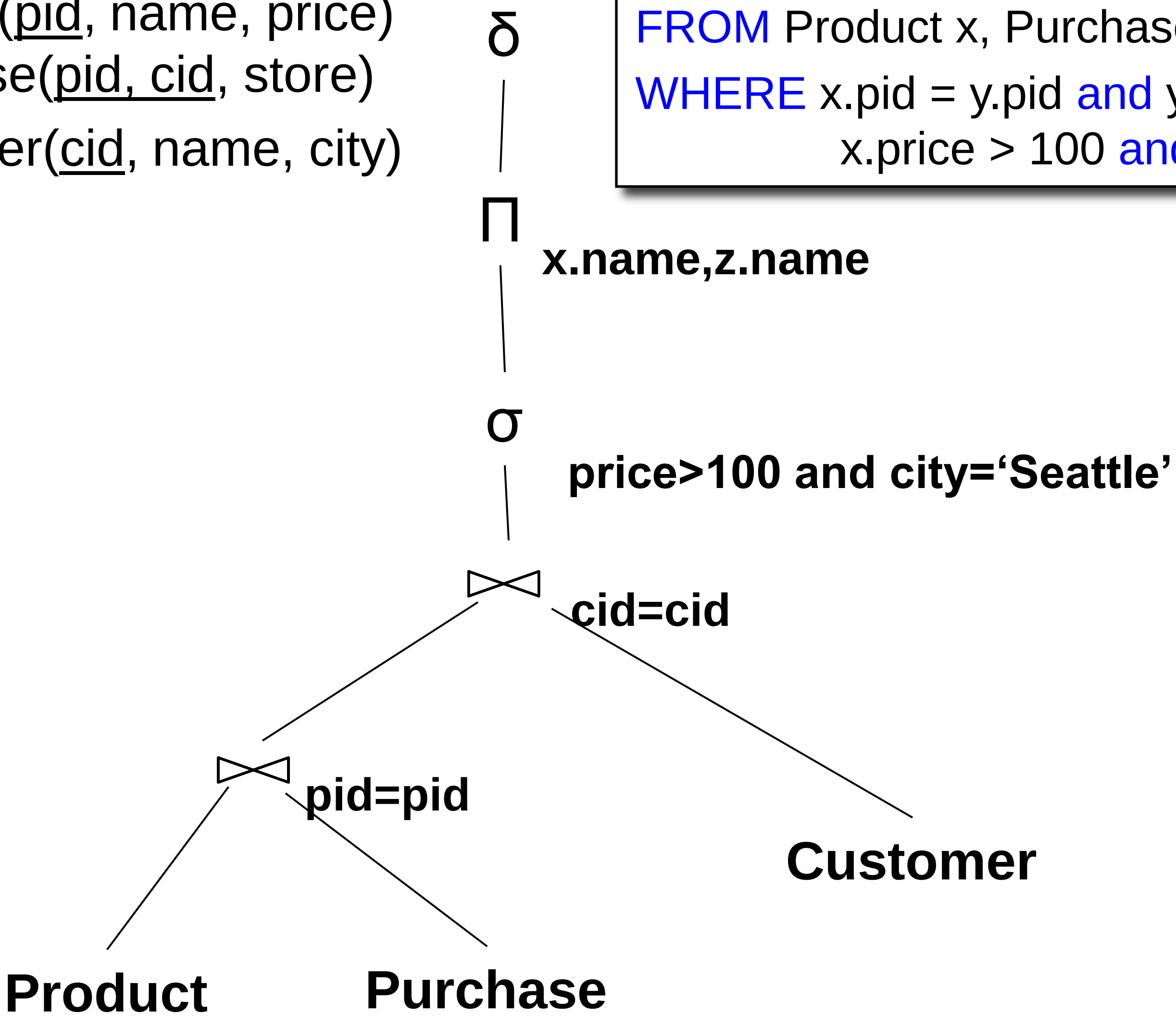
Product(<u>pid</u>, name, price)
Purchase(<u>pid, cid</u>, store)
Customer(<u>cid</u>, name, city)

```
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’
```

From SQL to RA

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

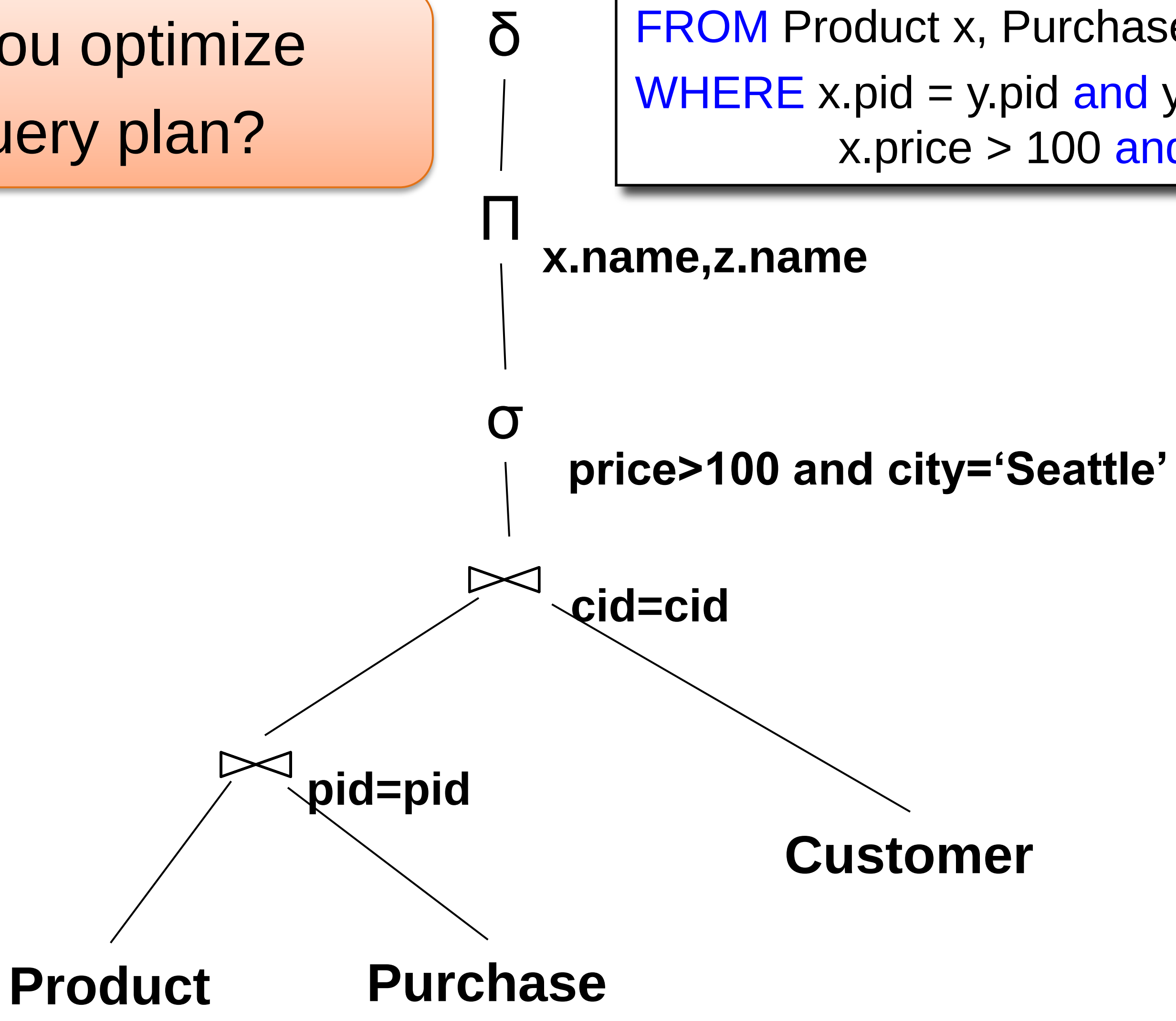
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
          x.price > 100 and z.city = 'Seattle'
```



From SQL to RA

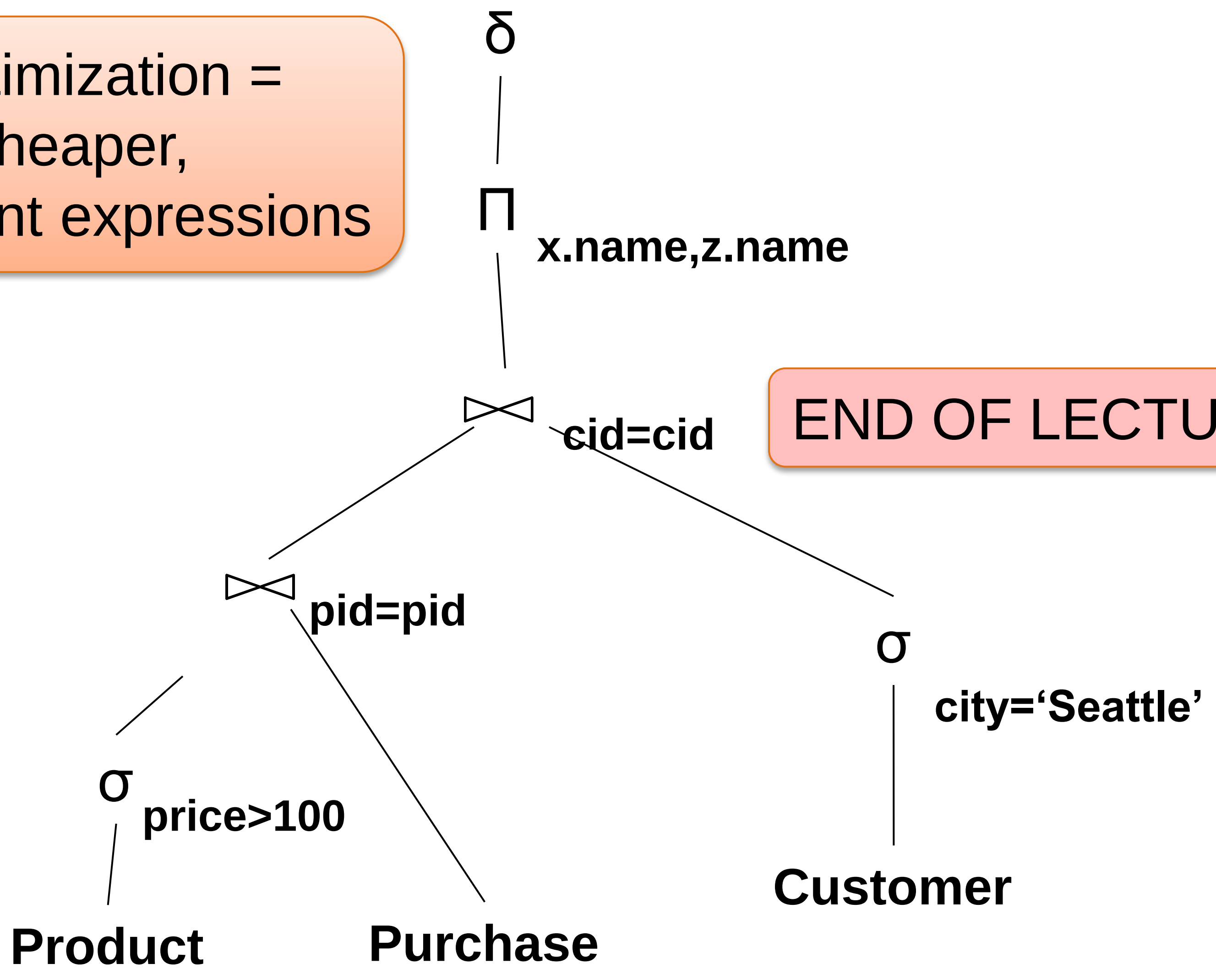
Can you optimize this query plan?

```
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'
```



An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions



END OF LECTURE09

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
- Sorting τ

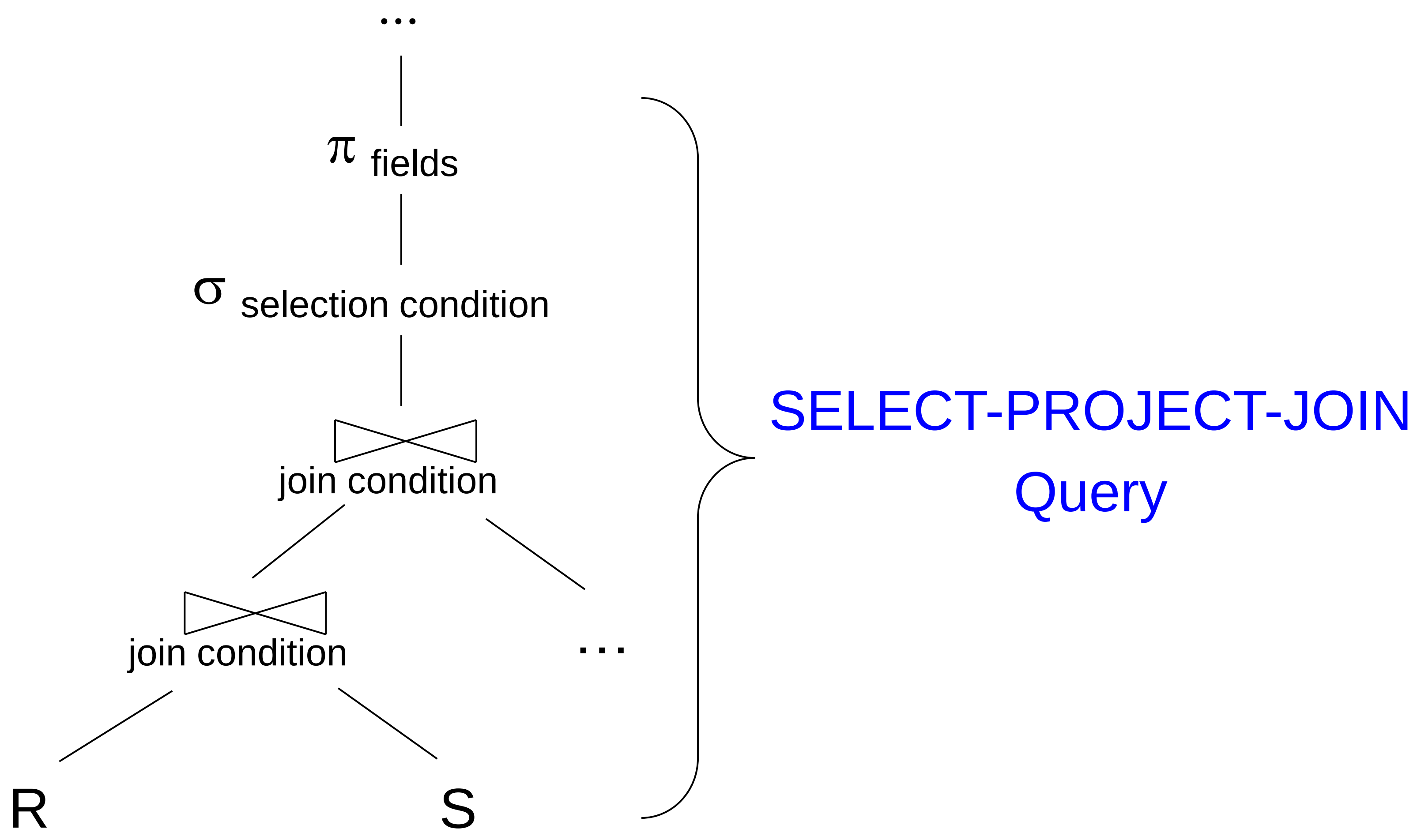
Logical Query Plan

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

$$\begin{tikzcd}
 & T3(\text{city, c}) \\
\Pi_{\text{city, c}} \arrow[d, dash] & \\
 & T2(\text{city,p,c}) \\
\sigma_{p > 100} \arrow[d, dash] & \\
 & T1(\text{city,p,c}) \\
\gamma_{\text{city, sum(price)} \rightarrow p,\ \text{count(*)} \rightarrow c} \arrow[d, dash] & \\
\text{sales(product, city, price)} &
\end{tikzcd}$$

T1, T2, T3 = temporary tables

Typical Plan for Block (1/2)



Typical Plan For Block (2/2)

$$\begin{tikzcd}
& \text{having}_{\text{condition}} \arrow[d, dash] & \\
& \gamma_{\text{fields, sum/count/min/max(fields)}} \arrow[d, dash] & \\
& \pi_{\text{fields}} \arrow[d, dash] & \\
& \sigma_{\text{selection condition}} \arrow[d, dash] & \\
& \bowtie_{\text{join condition}} \arrow[dl, dash] \arrow[dr, dash] & \\
\ldots & & \ldots
\end{tikzcd}$$

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)
```

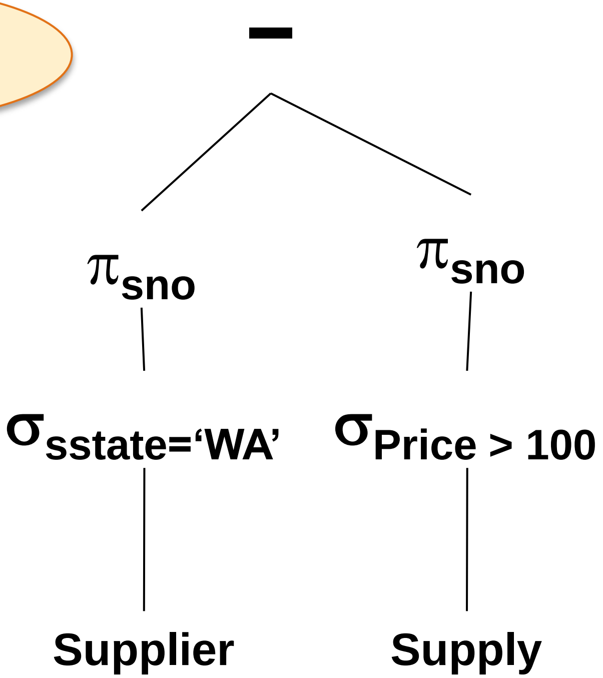
EXCEPT = set difference

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…



From Logical Plans to Physical Plans

Supplier(<u>sid</u>, sname, scity, sstate)

Supply(<u>sid, pno</u>, quantity)

Example

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

Give a relational algebra expression for this query

Supplier(<u>sid</u>, sname, scity, sstate)

Supply(<u>sid, pno</u>, quantity)

Relational Algebra

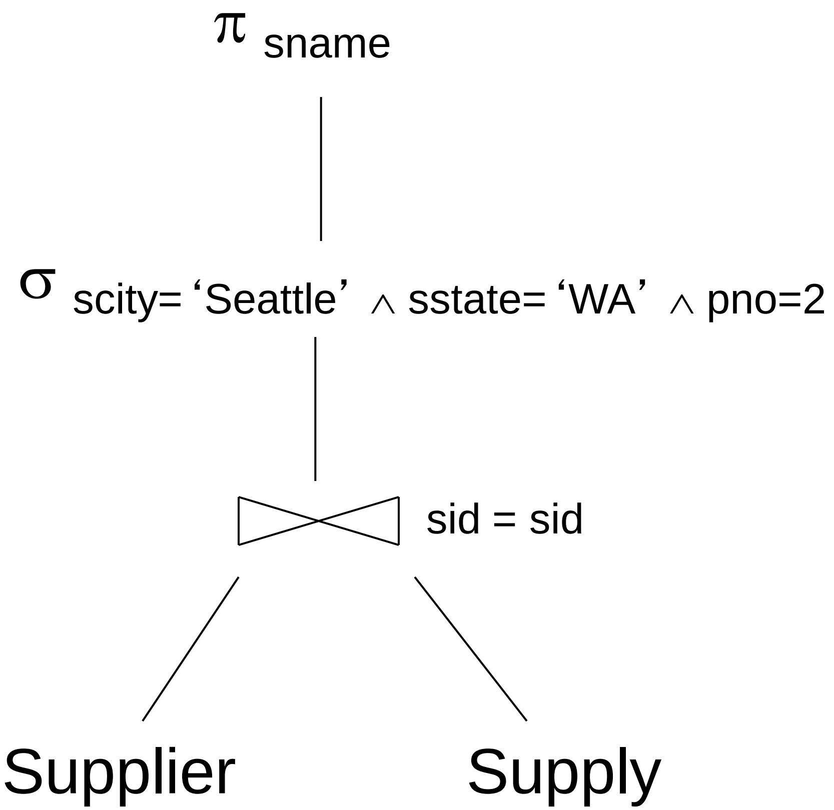
$$\pi_{\text{sname}}(\sigma_{\text{scity}=\text{'Seattle'} \wedge \text{sstate}=\text{'WA'} \wedge \text{pno}=2}(\text{Supplier} \bowtie_{\text{sid = sid}} \text{Supply}))$$

Supplier(<u>sid</u>, sname, scity, sstate)

Supply(<u>sid, pno</u>, quantity)

Relational Algebra

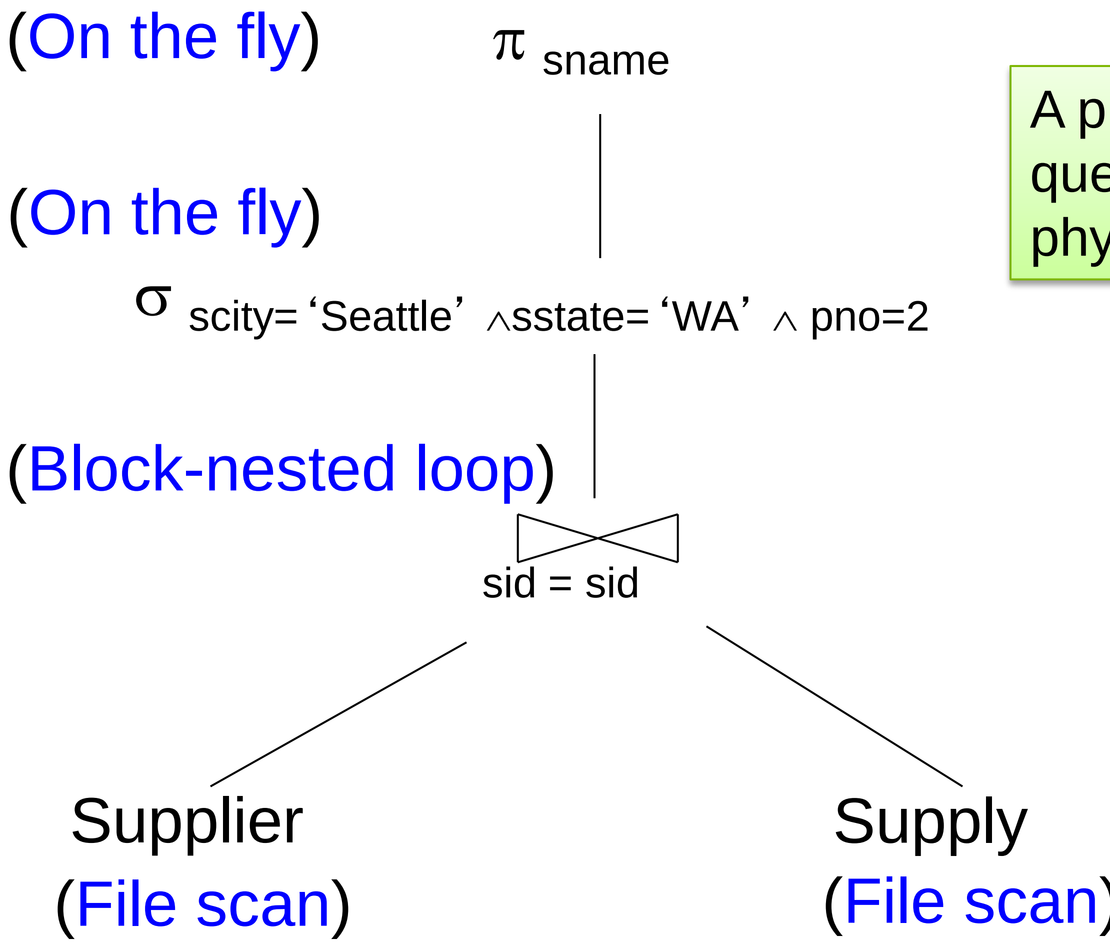
Relational algebra expression is also called the "logical query plan"


π sname
σ scity= 'Seattle' $\wedge$ sstate= 'WA' $\wedge$ pno=2
$\bowtie$ sid = sid
Supplier
Supply

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 1



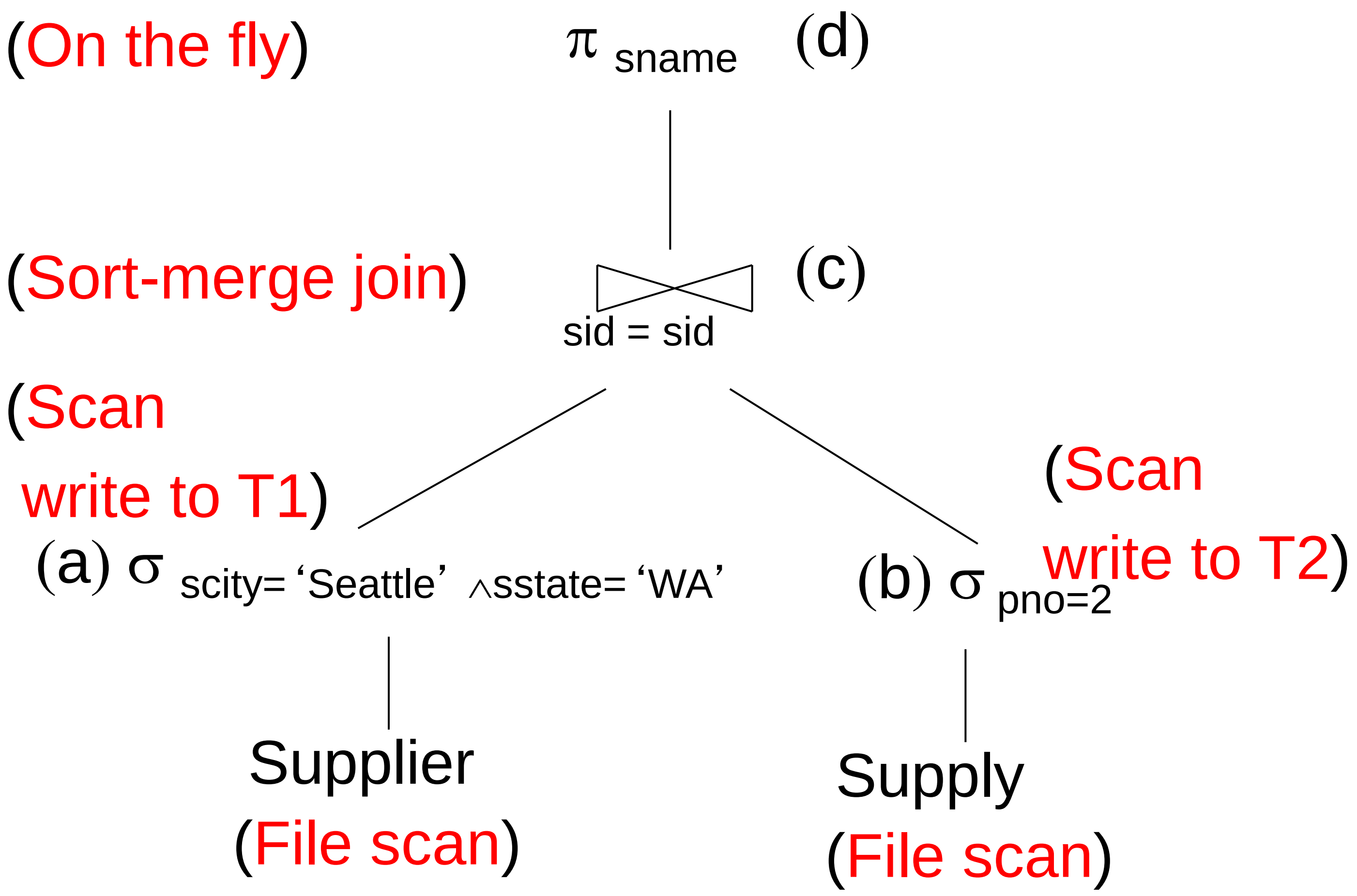
(On the fly)
π_{sname}
(On the fly)
$\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$
(Block-nested loop)
$\bowtie_{\text{sid = sid}}$
Supplier
(File scan)
Supply
(File scan)

A physical query plan is a logical query plan annotated with physical implementation details

Supplier(<u>sid</u>, sname, scity, sstate)

Supply(<u>sid, pno</u>, quantity)

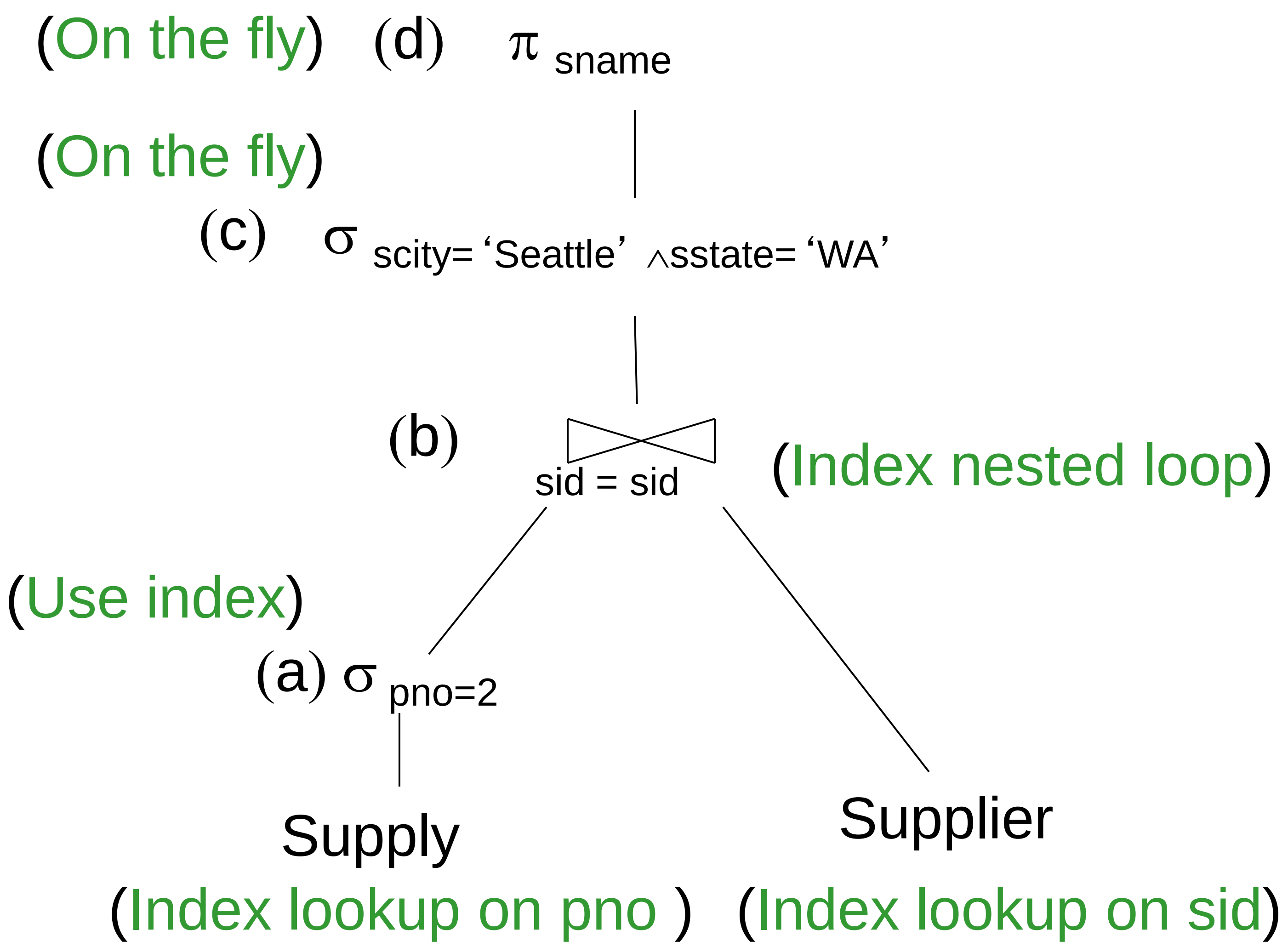
Physical Query Plan 2



Supplier(<u>sid</u>, sname, scity, sstate)

Supply(<u>sid, pno</u>, quantity)

Physical Query Plan 3


(On the fly) (d) π_{sname}
(On the fly)
(c) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$
(b) $\bowtie_{\text{sid = sid}}$
(Index nested loop)
(Use index)
(a) $\sigma_{\text{pno=2}}$
Supply
Supplier
(Index lookup on pno)
(Index lookup on sid)

Assume: clustered

Doesn't matter if clustered or not

Physical Data Independence

- Means that applications are insulated from changes in physical storage details
  - E.g., can add/remove indexes without changing apps
  - Can do other physical tunings for performance

- SQL and relational algebra facilitate physical data independence because both languages are “set-at-a-time”: Relations as input and output