CSE544
Data Management

Lecture 6: Data Models, Relational Algebra

Announcements

- Project teams due Friday
- HW2 is posted
- Review of “What goes around…” today

Where We Are

- We are done with SQL; Please continue to read and learn on your own
- Today: data models, and why the relational model wins; then RA
- Next lectures: DBMS internals

References

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

Outline

- Early data models
  - IMS
  - CODASYL
- Relational Model in some detail
- Data models that followed the relational model

Early Proposal 1: IMS*

- What is it?

* IBM Information Management System

Early Proposal 1: IMS*

- **Hierarchical data model**

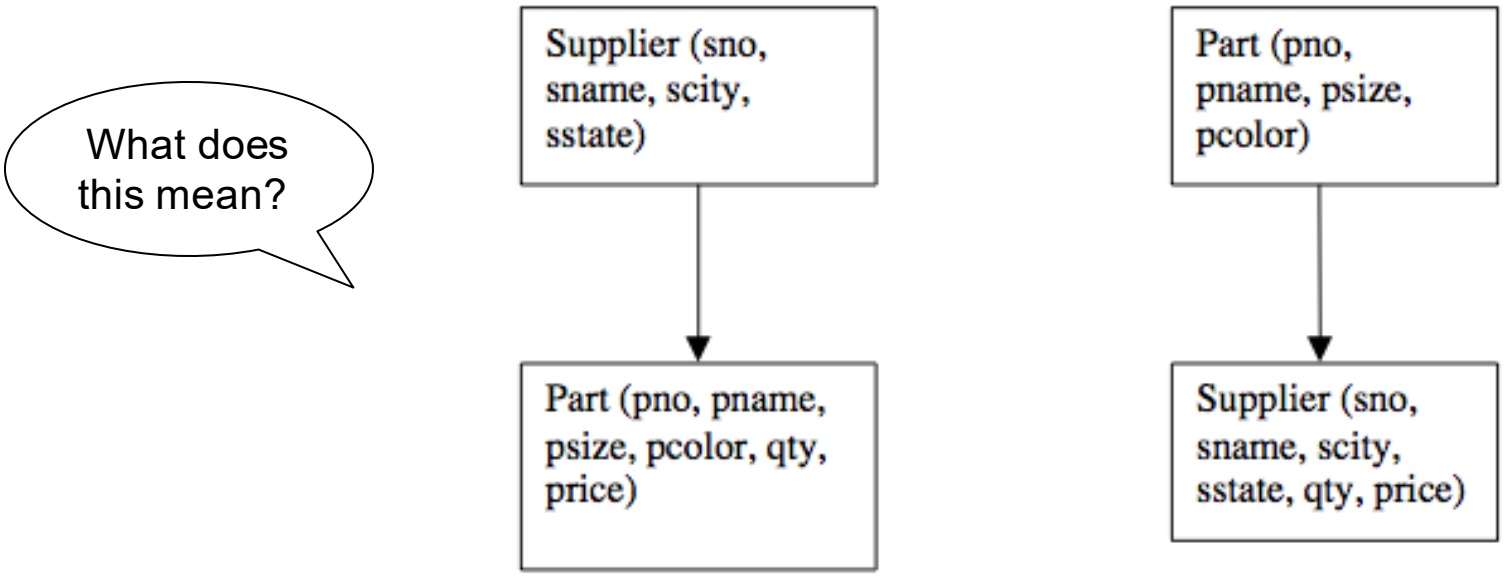
- **Record**
  - **Type**: collection of named fields with data types
  - **Instance**: must match type definition
  - Each instance has a **key**
  - Record types arranged in a **tree**

- **IMS database** is collection of instances of record types organized in a tree

* IBM Information Management System

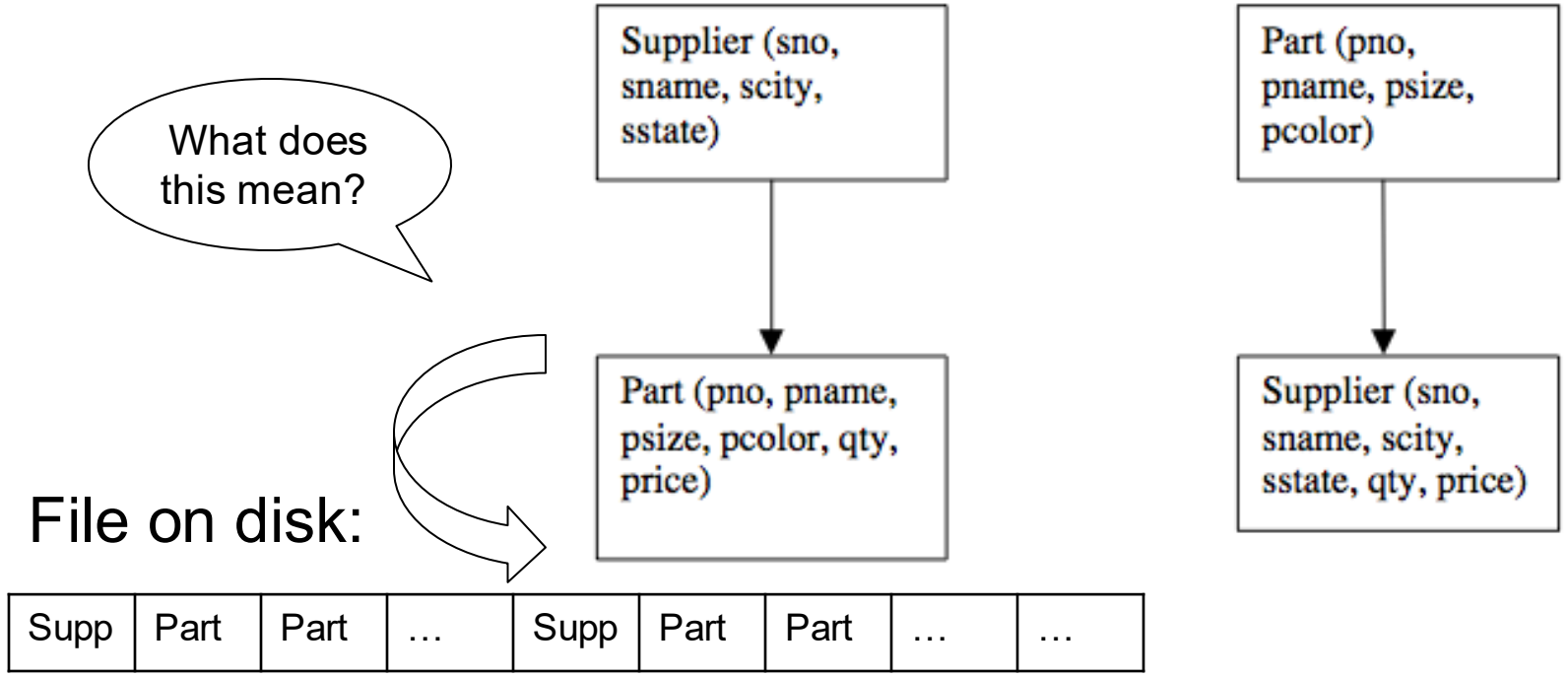
IMS Example

- Figure 2 from “What goes around comes around”



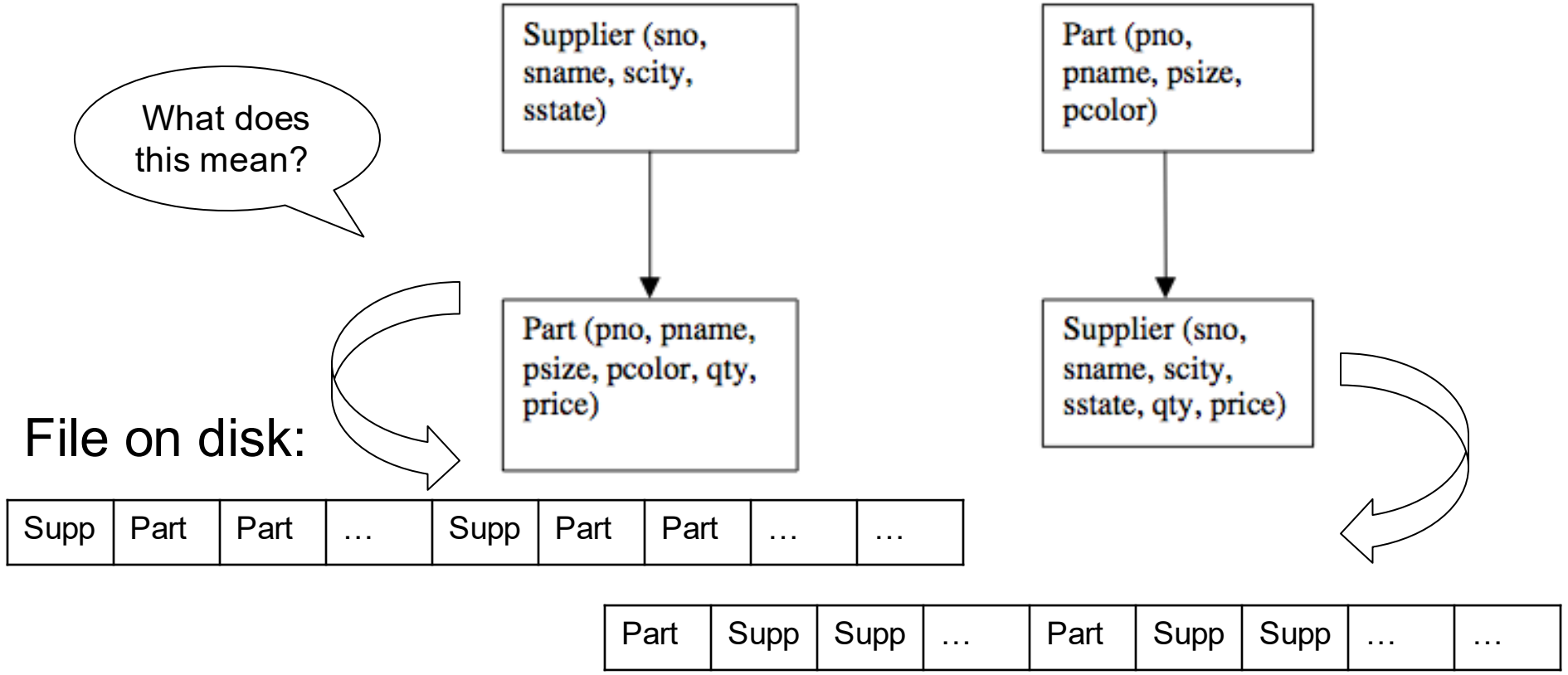
IMS Example

- Figure 2 from “What goes around comes around”



IMS Example

- Figure 2 from “What goes around comes around”



IMS Limitations

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent
- **Record-at-a-time** user interface
  - User must specify algorithm to access data

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent
- **Record-at-a-time** user interface
  - User must specify algorithm to access data
- **Very limited physical independence**
  - Phys. organization limits possible operations
  - Application programs break if organization changes
- **Some logical independence but limited**

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

- Each record has a hierarchical sequence key (HSK)
- HSK defines semantics of commands:
  - get_next; get_next_within_parent
- **DL/1 is a record-at-a-time language**
  - Programmers construct algorithm, worry about optimization

Data storage

How is data physically stored in IMS?

Data storage

How is data physically stored in IMS?

- Root records
  - Stored sequentially (sorted on key)
  - Indexed in a B-tree using the key of the record
  - Hashed using the key of the record
- Dependent records
  - Physically sequential
  - Various forms of pointers
- Selected organizations restrict DL/1 commands
  - No updates allowed due to sequential organization
  - No “get-next” for hashed organization

Data Independence

What is it?

Data Independence

What is it?

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

Lessons from IMS

- Physical/logical data independence needed
- Tree structure model is restrictive
- Record-at-a-time programming forces user to do optimization

Early Proposal 2: CODASYL

What is it?

Early Proposal 2: CODASYL

What is it?

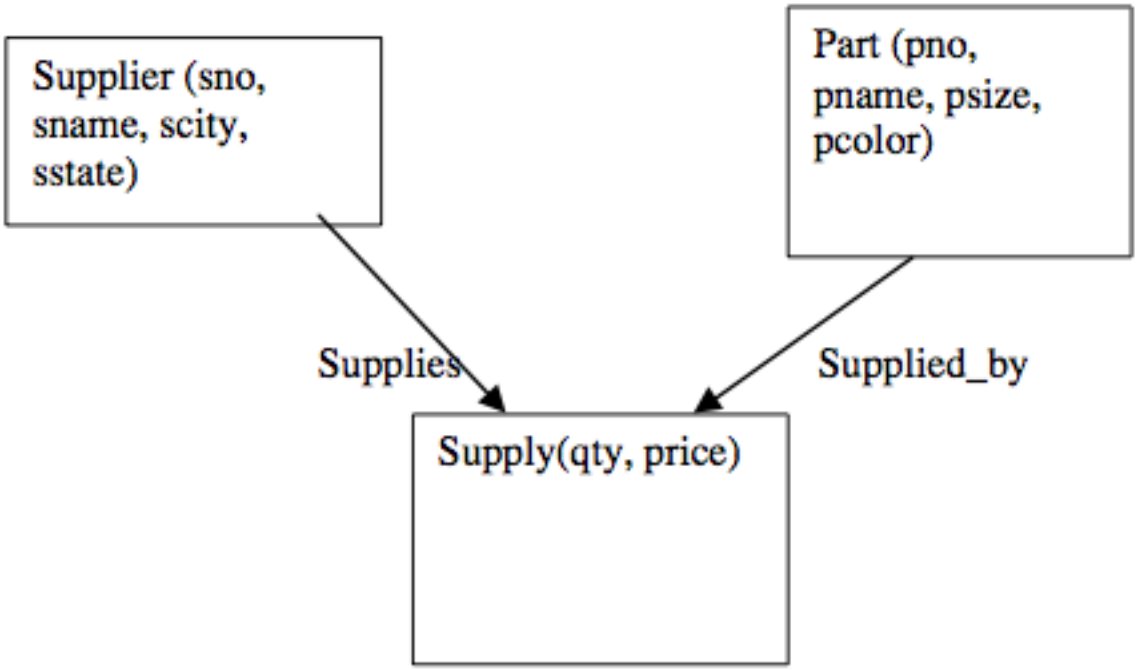
- **Networked data model**

- Primitives are also **record types** with **keys**
- Record types are organized into **network**
- Multiple parents; arcs = “sets”
- More flexible than hierarchy

- **Record-at-a-time** data manipulation language

CODASYL Example

- Figure 5 from “What goes around comes around”



CODASYL Limitations

- No data independence: application programs break if organization changes
- Record-at-a-time: “navigate the hyperspace”

The Programmer as Navigator

by Charles W. Bachman

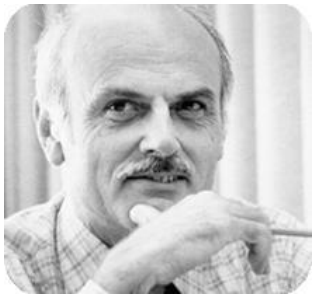
Outline

- Early data models
- Relational Model in some detail
- Data models that followed the relational model

Relational Model Overview

Ted Codd 1970

- What was the motivation? What is the model?



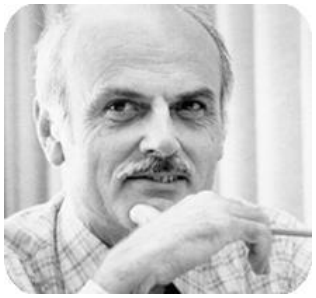
Relational Model Overview

Ted Codd 1970

- Motivation: logical and physical data independence

- Store data in a **simple data structure** (table)
- Access data through **set-at-a-time** language
- **No need for physical storage proposal**

Relational Database: A Practical Foundation for Productivity



Great Debate

- Pro relational
  - What were the arguments?
- Against relational
  - What were the arguments?
- How was it settled?

Great Debate

- Pro relational
  - CODASYL is too complex
  - No data independence
  - Record-at-a-time hard to optimize
  - Trees/networks not flexible enough
- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to implement efficiently
- Ultimately settled by the market place

Outline

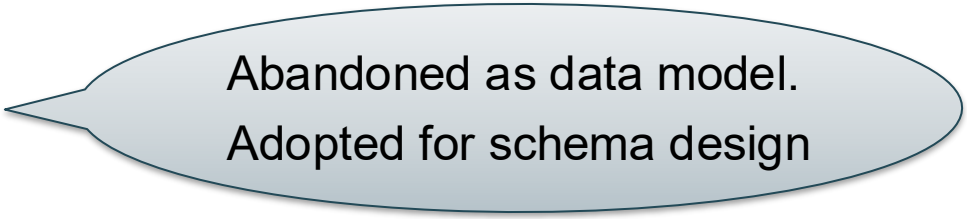
- Early data models
- Relational Model in some detail
- Data models that followed the relational model

Other Data Models

- Entity-relationship
- Object-relational
- Semistructured
- Key-value pairs

Other Data Models

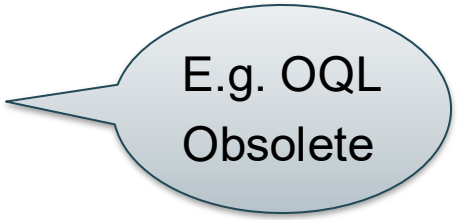
- Entity-relationship



- Object-relational
- Semistructured
- Key-value pairs

Other Data Models

- Entity-relationship
- Object-relational
- Semistructured
- Key-value pairs

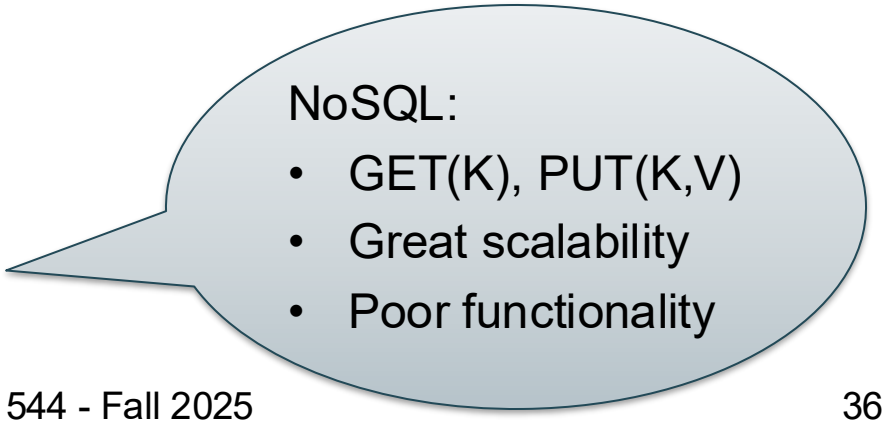


Other Data Models

- Entity-relationship
- Object-relational
- Semistructured
- Key-value pairs

Other Data Models

- Entity-relationship
- Object-relational
- Semistructured
- Key-value pairs



Data Independence in the Relational Model

Data Independence

- Logical data independence:
  - SQL views

- Physical data independence:
  1. Declarative query: FO or SQL
  2. Query plan: Relational Algebra
  3. Query optimization / execution

Data Independence

- Logical data independence:
  - SQL views

- Physical data independence:
  1. Declarative query: FO or SQL
  2. Query plan: Relational Algebra
  3. Query optimization / execution

SQL Views

- CREATE VIEW RedSuppliers AS …
- Creates a new relation name
- The content of that relation is defined by a SELECT-FROM-WHERE query

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

SQL Views

```
CREATE VIEW RedSuppliers AS
  SELECT DISTINCT x.*
  FROM Supplier x, Supply y, Part z
  WHERE x.sno=y.sno and y.pno=z.pno and z.pcolor='Red';
```

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

SQL Views

```
CREATE VIEW RedSuppliers AS
  SELECT DISTINCT x.*
  FROM Supplier x, Supply y, Part z
  WHERE x.sno=y.sno and y.pno=z.pno and z.pcolor='Red';
```

RedSuppliers is added to the database schema

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

SQL Views

```
CREATE VIEW RedSuppliers AS
  SELECT DISTINCT x.*
  FROM Supplier x, Supply y, Part z
  WHERE x.sno=y.sno and y.pno=z.pno and z.pcolor='Red';
```

RedSuppliers is added to the database schema
Later we can use it, like any table:

```
SELECT DISTINCT x.*
FROM RedSupplier x, Supply y, Part z
WHERE x.sno=y.sno and y.pno=z.pno and z.pcolor='Blue';
```

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

SQL Views

```
CREATE VIEW RedSuppliers AS
  SELECT DISTINCT x.*
  FROM Supplier x, Supply y, Part z
  WHERE x.sno=y.sno and y.pno=z.pno and z.pcolor='Red';
```

RedSuppliers is added to the database schema
Later we can use it, like any table:

What does this query compute?

```
SELECT DISTINCT x.*
FROM RedSupplier x, Supply y, Part z
WHERE x.sno=y.sno and y.pno=z.pno and z.pcolor='Blue';
```

Discussion

- CREATE VIEW…
  - Add view name permanently to the schema
  - To delete, type DROP VIEW …
- CREATE TEMPORAY VIEW…
  - Supported in postgres
  - Add view name to the schema…
  - …remove at the end of the session
- WITH …. Only local to one query

Virtual v.s. Materialized Views

- Virtual view – the default
  - The query defining the view is evaluated every time when it is used
  - The data is always up to date!
  - But we re-compute it every time: inefficient
- Materialized view
  - CREATE MATERALIZED VIEW …
  - Supported by some systems
  - Computed only once: efficient
  - But may not be up to date

Data Independence

- Logical data independence:
  - SQL views

- Physical data independence:
  1. Declarative query: FO or SQL
  2. Query plan: Relational Algebra
  3. Query optimization / execution

Data Independence

- Logical data independence:
  - SQL views

- Physical data independence:
  1. Declarative query: FO or SQL
  2. Query plan: Relational Algebra
  3. Query optimization / execution

We saw this

Next

Next lectures

Relational Algebra

Executing SQL Queries

- User writes in SQL
- System: SQL → Relational Algebra
- Query Plan: is optimized, the executed

Relational Algebra

Five operators:

- Selection σ
- Projection Π
- Join or cartesian product $\bowtie$, $\times$
- Union $\cup$
- Difference $-$

Selection

$\sigma_{condition}(T)$

Returns tuples in T
that satisfy the condition

```
SELECT *
FROM T
WHERE condition;
```

Selection

$\sigma_{condition}(T)$

Returns tuples in T
that satisfy the condition

$\sigma_{C=20 \wedge D \geq 66}(S) =$

```
SELECT *
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Selection

$\sigma_{condition}(T)$

Returns tuples in T
that satisfy the condition

$\sigma_{C=20 \wedge D \geq 66}(S) =$

C	D
20	66
20	77

```
SELECT *
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Selection

$\sigma_{condition}(T)$

Returns tuples in T
that satisfy the condition

$\sigma_{D \geq 66}(S) =$

C	D
20	66
20	77
30	66

$\sigma_{C=20 \wedge D \geq 66}(S) =$

C	D
20	66
20	77

```
SELECT *
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Projection

$\Pi_{col1,col2,\ldots}(T)$

Return columns 1, 2, …
from T; all rows

```
SELECT T.col1, T.col2,…
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Projection

$\Pi_{col1,col2,\ldots}(T)$

Return columns 1, 2, …
from T; all rows

$\Pi_C(S) =$

C
10
10
20
20
20
30

```
SELECT T.col1, T.col2,…
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Bag semantics…

Projection

$\Pi_{col1,col2,\ldots}(T)$

Return columns 1, 2, …
from T; all rows

$\Pi_C(S) =$

C
10
10
20
20
20
30

```
SELECT T.col1, T.col2,…
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Projection

Bag semantics…

$\Pi_{col1,col2,\ldots}(T)$

Return columns 1, 2, …
from T; all rows

$\Pi_C(S) =$

C
10
10
20
20
20
30

or

C
10
20
30

…set semantics

```
SELECT T.col1, T.col2,…
FROM T
WHERE condition;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Join

$T_1 \bowtie_{cond} T_2$

Joins the two tables

```
SELECT *
FROM T1, T2
WHERE cond;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Join

$T_1 \bowtie_{cond} T_2$

Joins the two tables

```
SELECT *
FROM T1, T2
WHERE cond;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

$R \bowtie_{B=C} S =$

A	B	C	D
1	10	10	33
1	10	10	44
1	20	20	55
1	20	20	66
1	20	20	77
2	20	20	55
2	20	20	66
2	20	20	77

Join

$T_1 \bowtie_{cond} T_2$

Joins the two tables

$R\ x \bowtie_{x.B=y.B} R\ y =$

Disambiguate attributes: various conventions…

x.A	x.B	y.A	y.B
1	10	1	10
1	20	1	20
1	20	2	20
2	20	1	20
2	20	2	20

```
SELECT *
FROM T1, T2
WHERE cond;
```

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Many Variants of Joins

- Eq-join: $R \bowtie_{B=C} S$
- Theta-join: $R \bowtie_{B \leq C \wedge A*D<20} S$
- Cartesian product: $R \times S$
- Natural join: next!

Natural Join

- Joins on common column names
- Retains only one copy of the column

R=

A	B
1	10
1	20
2	20

S=

B	C
10	33
10	44
20	55
20	66
20	77
30	66

$R \bowtie S =$

Implicit: R.B=S.B

A	B	C
1	10	33
1	10	44
1	20	55
1	20	66
1	20	77
2	20	55
2	20	66
2	20	77

Quiz Time!

What do these natural joins compute?

- $R(A, B) \bowtie S(B, C)$
- $R(A, B) \bowtie S(C, D)$
- $R(A, B) \bowtie S(A, B)$

Quiz Time!

What do these natural joins compute?

- $R(A,B) \bowtie S(B,C)$

- $R(A,B) \bowtie S(C,D)$
- $R(A,B) \bowtie S(A,B)$

Quiz Time!

What do these natural joins compute?

- $R(A,B) \bowtie S(B,C)$
- $R(A,B) \bowtie S(C,D)$
- $R(A,B) \bowtie S(A,B)$

Quiz Time!

What do these natural joins compute?

- $R(A,B) \bowtie S(B,C)$
- $R(A,B) \bowtie S(C,D)$
- $R(A,B) \bowtie S(A,B)$

Joins on B=B

Cartesian product $R \times S$

Intersection $R \cap S$

Even More Joins

- Inner joins: ⋈ (all variations)
- Left outer join: ⟕
- Right-, full outer join: ⟖, ⟗
- Semi-join: ⋉
- Anti-semi-join: ▷

People use various symbols

Semi-Join

$R \ltimes S$

A	B
1	20
1	30

Tuples in R that join with S

```
SELECT DISTINCT R.*
FROM R, S
WHERE join-cond;
```

R=

A	B
1	20
1	30
2	40

S=

B	C
10	33
10	44
20	55
20	66
20	77
30	66

Anti Semi-Join

$$R \triangleright S = R - R \ltimes S$$

A	B
2	40

Tuples in R that do not join with S

R=

A	B
1	20
1	30
2	40

S=

B	C
10	33
10	44
20	55
20	66
20	77
30	66

Finally: Union and Difference

- Set operations:

$$R \cup S \ , \quad R - S$$

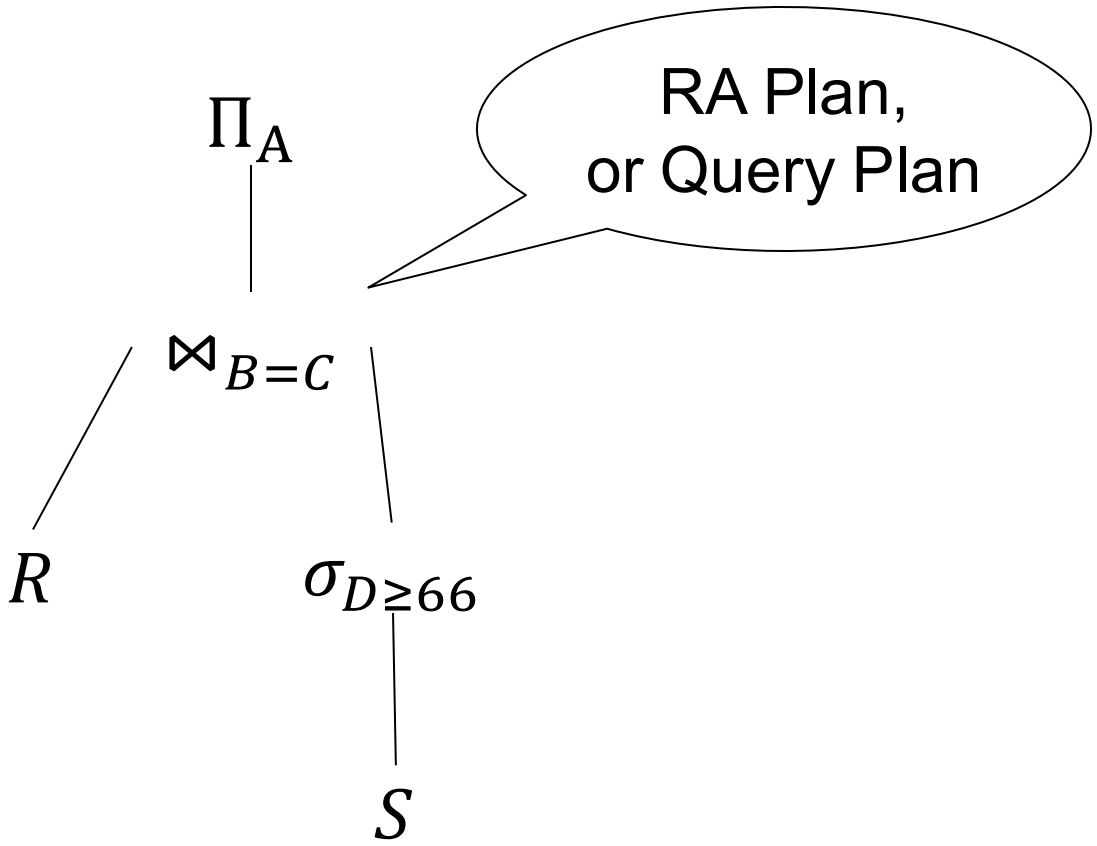
- R, S must have the same schema!

RA by Example

$$\Pi_{\mathrm{A}}(R \bowtie_{B=C} \sigma_{D \geq 66}(S))$$

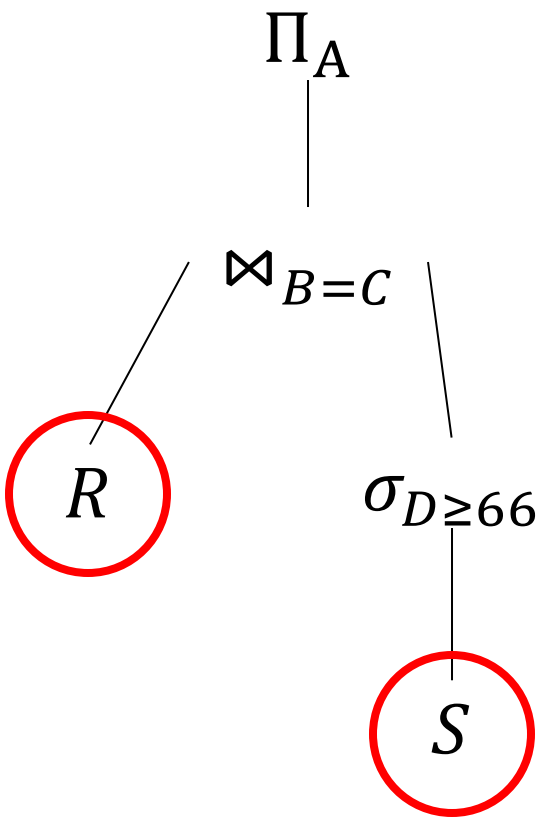
RA by Example

$$\Pi_A(R \bowtie_{B=C} \sigma_{D \geq 66}(S))$$



RA by Example

$$\Pi_A(R \bowtie_{B=C} \sigma_{D \geq 66}(S))$$



R=

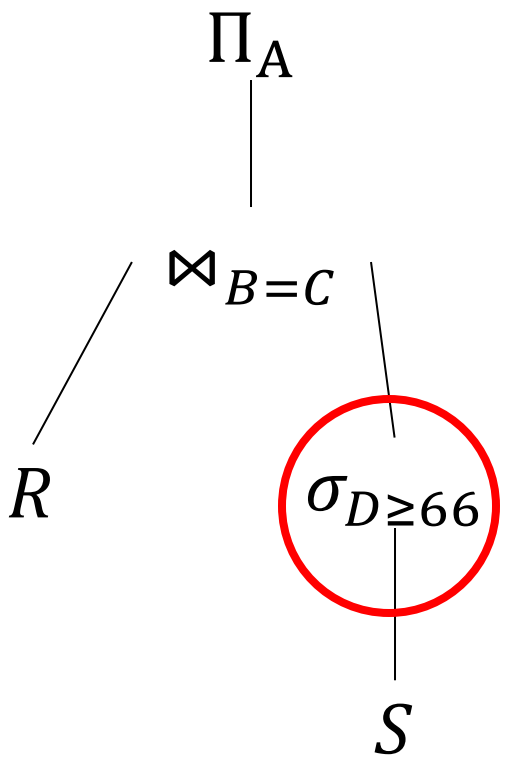
A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

RA by Example

$$\Pi_A(R \bowtie_{B=C} \sigma_{D \geq 66}(S))$$



R=

A	B
1	10
1	20
2	20

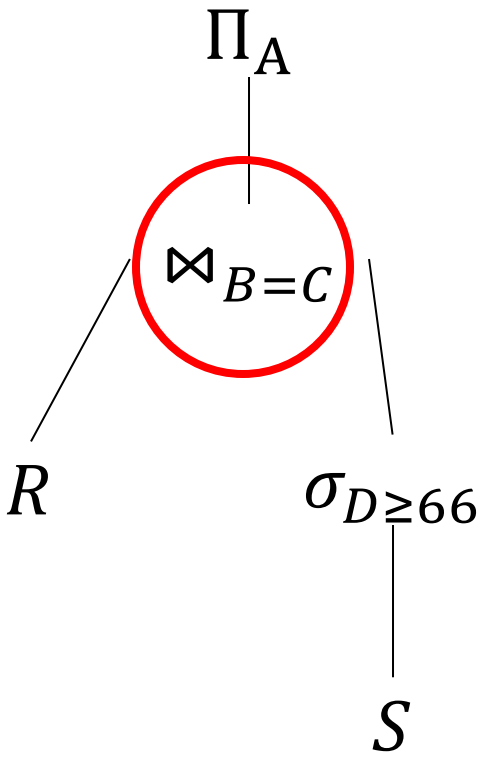
S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

C	D
20	66
20	77
30	66

RA by Example

$$\Pi_A(R \bowtie_{B=C} \sigma_{D \geq 66}(S))$$



A	B	C	D
1	20	20	66
1	20	20	77
2	20	20	66
2	20	20	77

C	D
20	66
20	77
30	66

R=

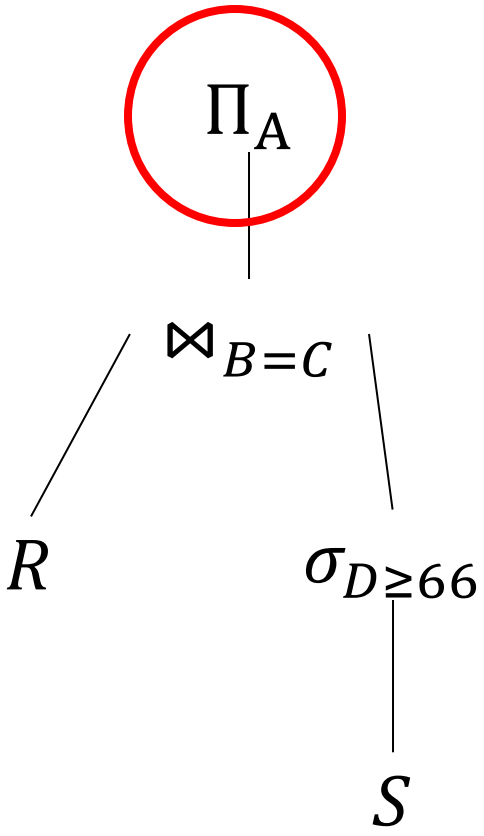
A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

RA by Example

$$\Pi_A(R \bowtie_{B=C} \sigma_{D \geq 66}(S))$$



A
1
2

Final output

A	B	C	D
1	20	20	66
1	20	20	77
2	20	20	66
2	20	20	77

C	D
20	66
20	77
30	66

R=

A	B
1	10
1	20
2	20

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Relational Algebra

Five operators:

- Selection σ
- Projection Π
- Join or cartesian product $\bowtie$, $\times$
- Union $\cup$
- Difference $-$

Relational Algebra

Which operations are monotone?

Five operators:

- Selection σ
- Projection Π
- Join or cartesian product $\bowtie$, $\times$
- Union $\cup$
- Difference $-$

Relational Algebra

Which operations are monotone?

Five operators:

- Selection σ
- Projection Π
- Join or cartesian product $\bowtie$, $\times$
- Union $\cup$

Monotone

- Difference $-$

Non-monotone

Extended Relational Algebra

- Group-by and aggregate: γ
- Duplicate elimination: δ
- Sorting: τ

Group-by and Aggregates

$\gamma_{col1,col2,\ldots,agg1,\ldots}(T)$

Standard group-by:

```
SELECT col1,…,agg1(..),agg2(..)
FROM T
GROUP-BY condition;
```

$\gamma_{C,sum(D)}(S) =$

C	D
10	77
20	196
30	66

S=

C	D
10	33
10	44
20	55
20	66
20	77
30	66

Translation

Every SQL query can be translated into an expression in the Extended RA

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

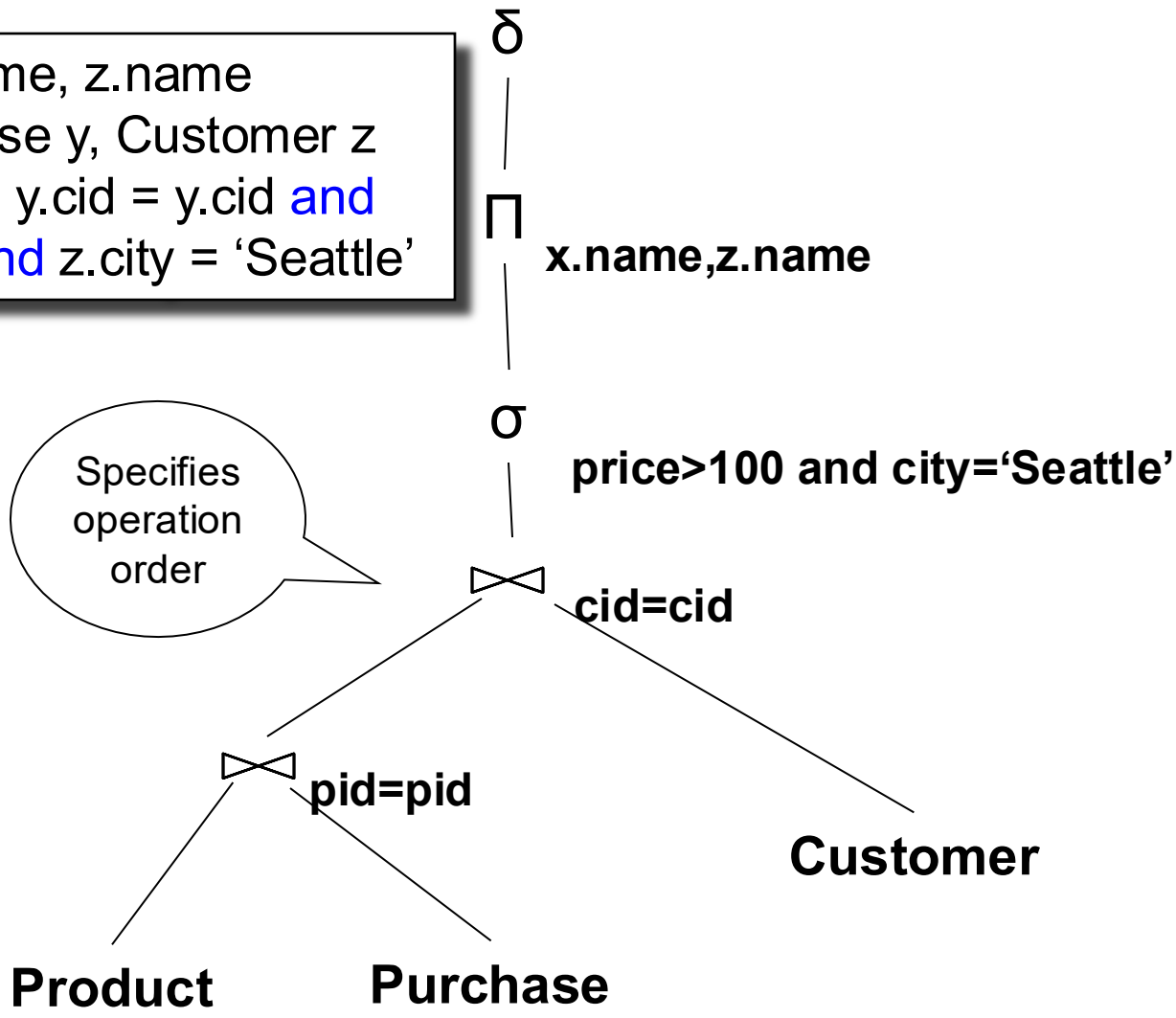
SQL…

```
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'
```

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

…to RA

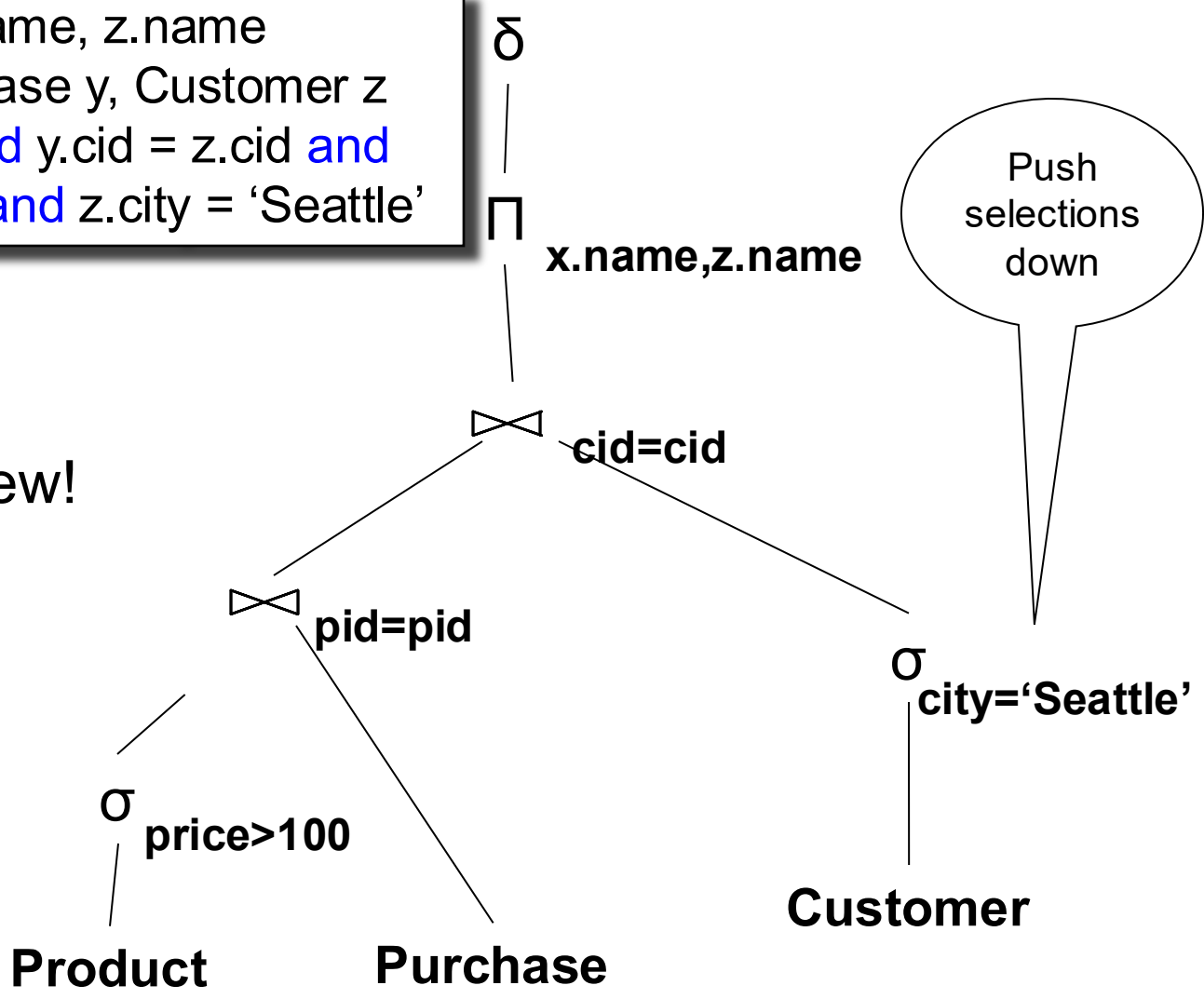
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
```



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

Optimization

```
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'
```



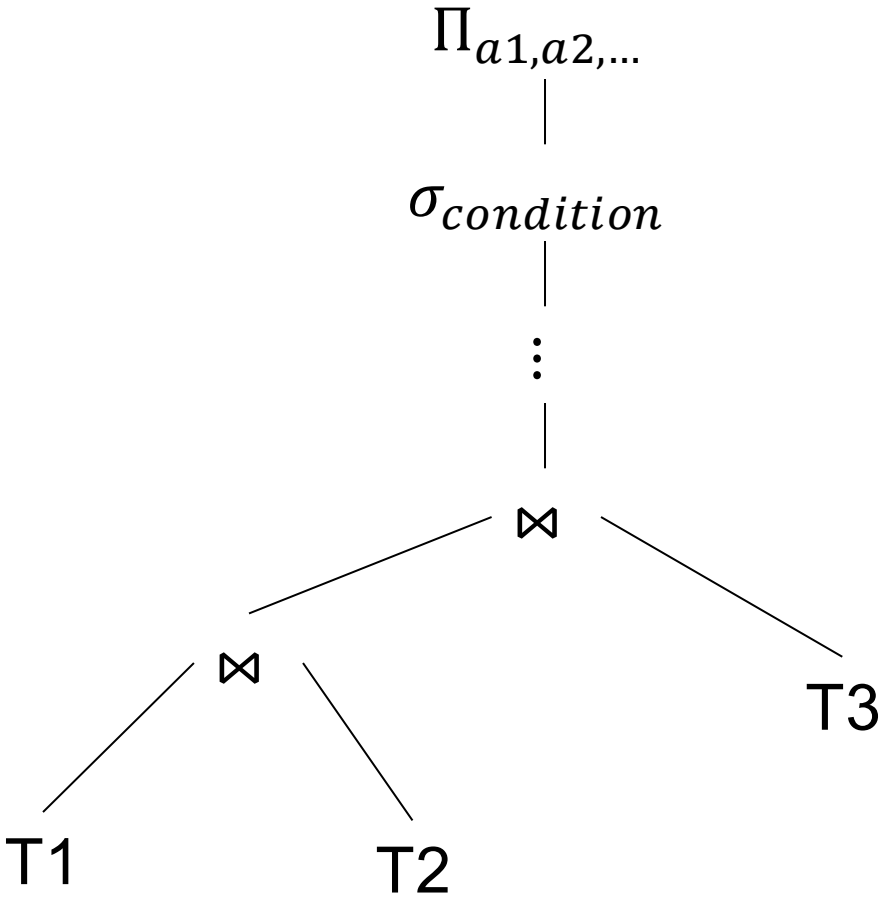
This is a quick preview!

More about this next lectures

SQL to RA

Simple SFW

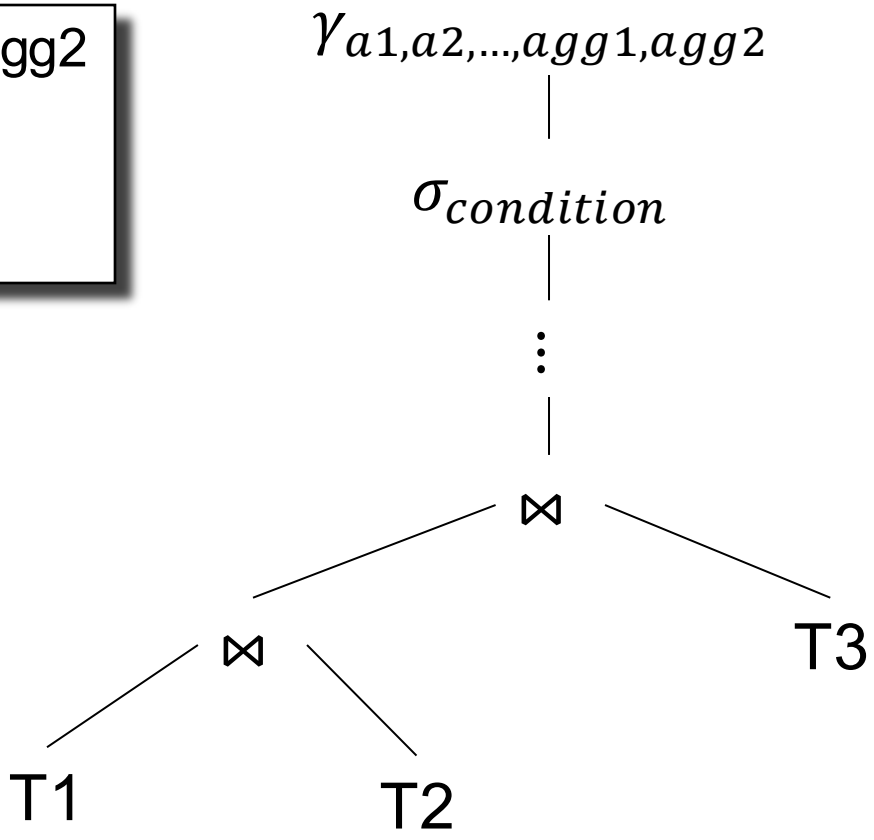
```
SELECT a1,a2,…
FROM T1, T2, …
WHERE condition
```



SQL to RA

…add GROUP-BY

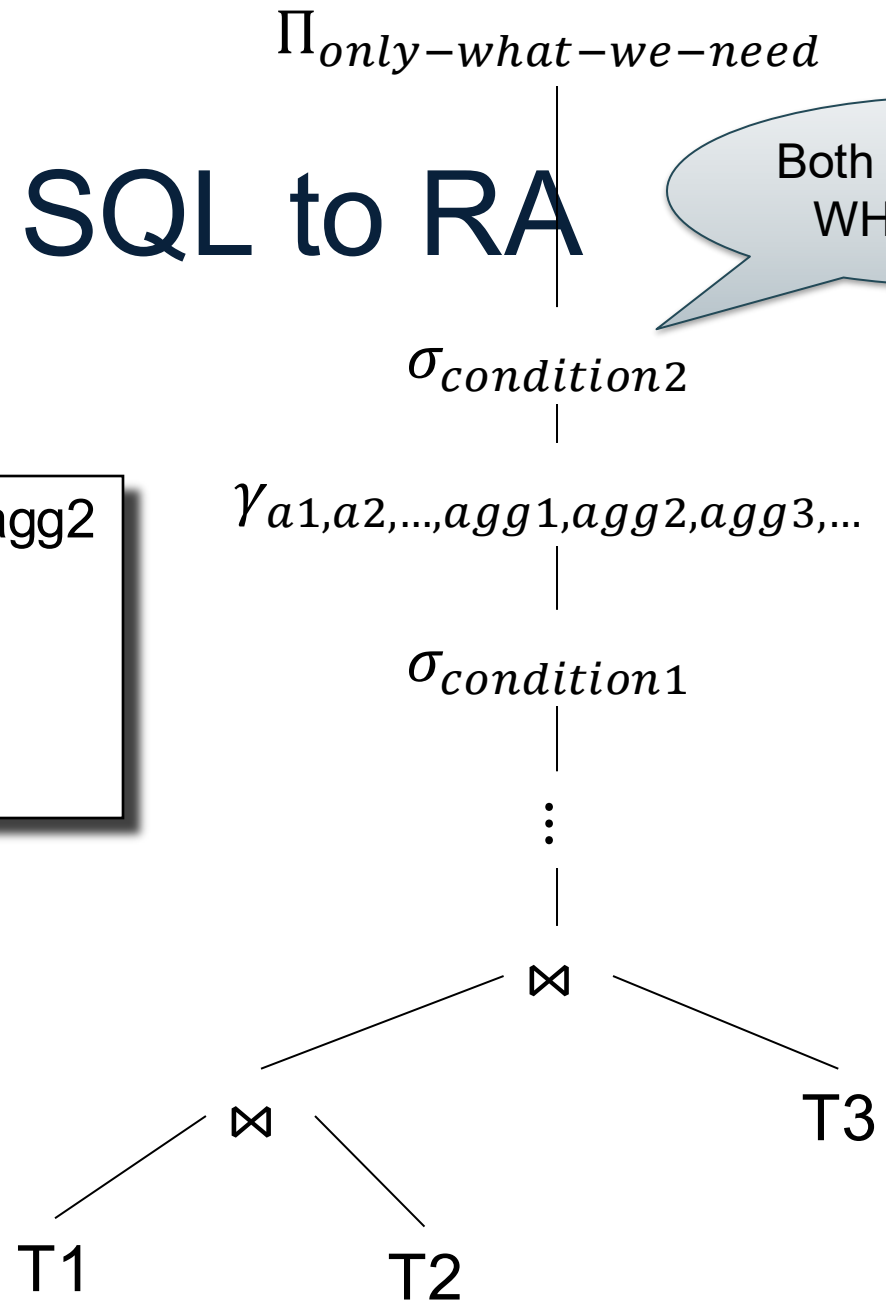
```
SELECT a1,a2,…,agg1,agg2
FROM T1, T2, …
WHERE condition
GROUP BY a1,a2,…
```



SQL to RA

…add HAVING

```
SELECT a1,a2,…,agg1,agg2
FROM T1, T2, …
WHERE condition1
GROUP BY a1,a2,…
HAVING condition2
```



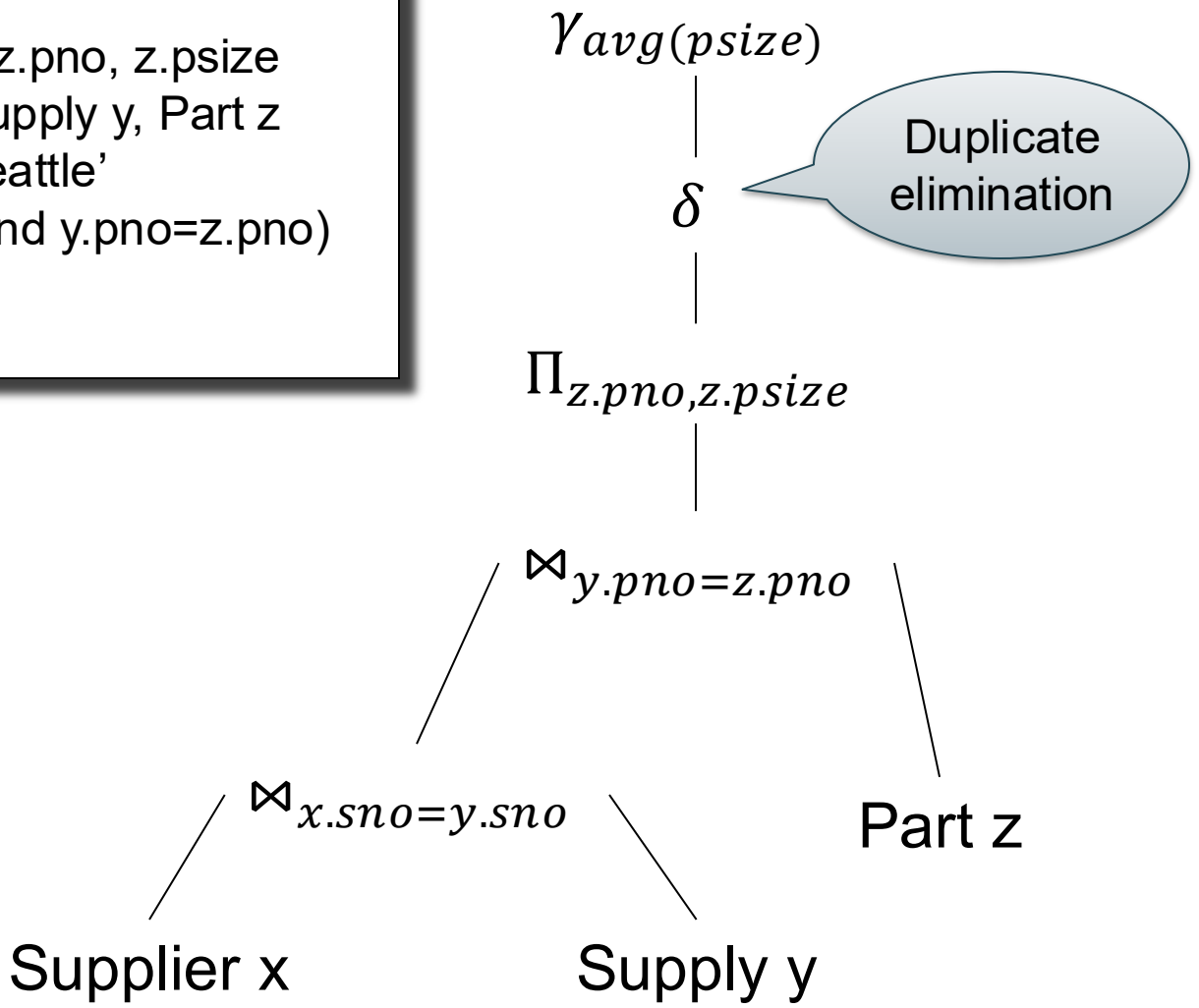
SQL to RA

- SQL queries without subqueries are straightforward to translate
- Subqueries may need to be flattened, then translated to SQL -- next

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

Subqueries

```
WITH Tmp AS (
        SELECT DISTINCT z.pno, z.psize
        FROM Supplier x, Supply y, Part z
        WHERE x.scity = 'Seattle'
           and x.sno=y.sno and y.pno=z.pno)
SELECT avg(psize)
FROM Tmp;
```



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

Subqueries

Find all suppliers in 'WA' that supply *only* parts under $100

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

Subqueries

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

Find all suppliers in 'WA' that supply *only* parts under $100

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

Subqueries

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

Find all suppliers in 'WA' that supply *only* parts under $100

Translate to RA

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

Subqueries

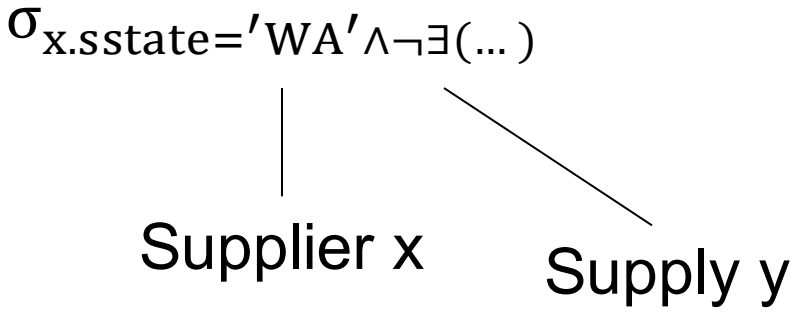
```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```


Supplier x

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

Subqueries

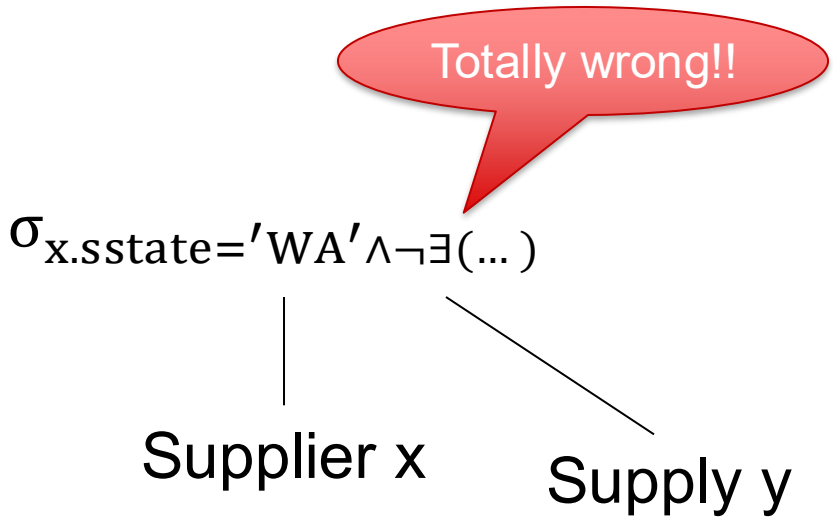
```
SELECT  x.sno
FROM Supplier x
WHERE  x.sstate = 'WA'
    and not exists
       (SELECT *
        FROM Supply y
        WHERE x.sno = y.sno
              and y.price > 100)
```



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

Subqueries

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



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

Subqueries

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

Need to unnest

Totally wrong!!

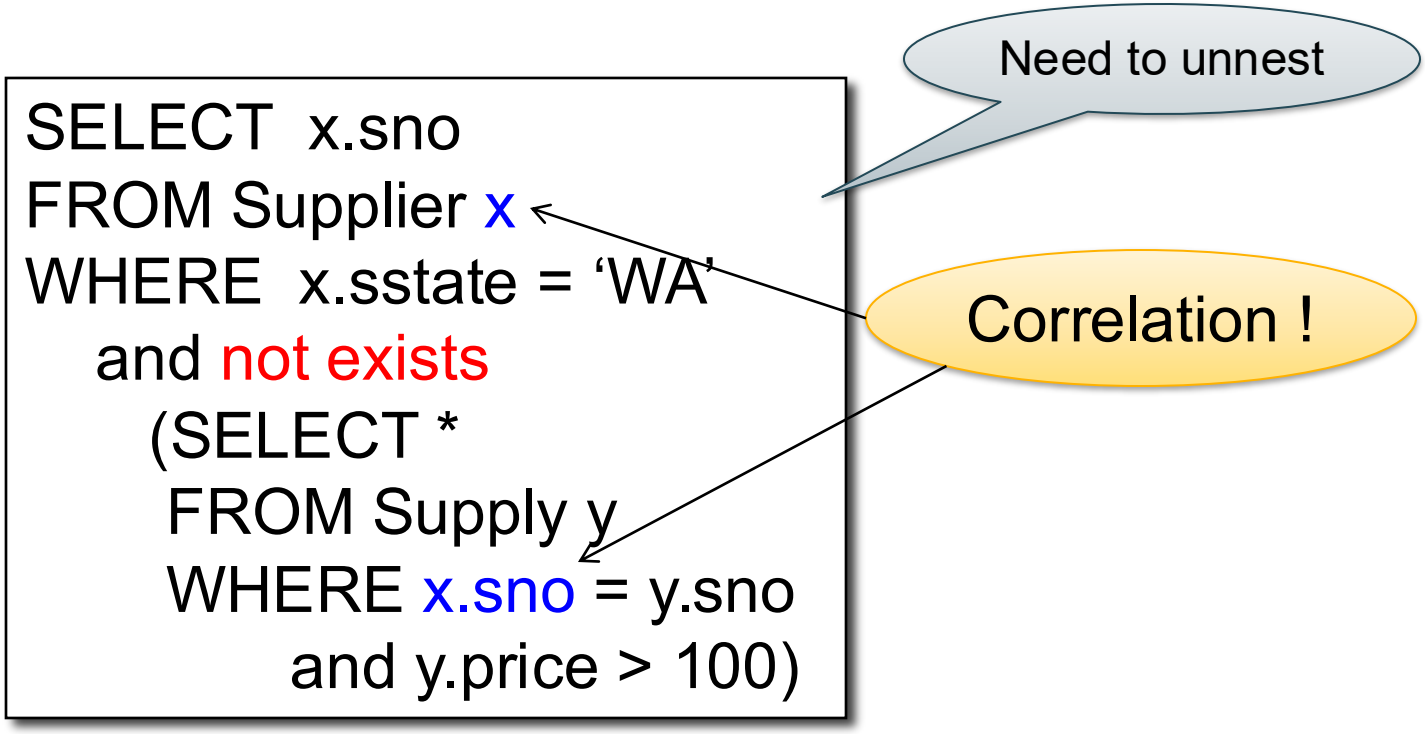
$\sigma_{x.sstate='WA' \wedge \neg\exists(\ldots)}$

Supplier x

Supply y

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

Subqueries



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

Subqueries

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

De-Correlation

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

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

Subqueries

Un-nest

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

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

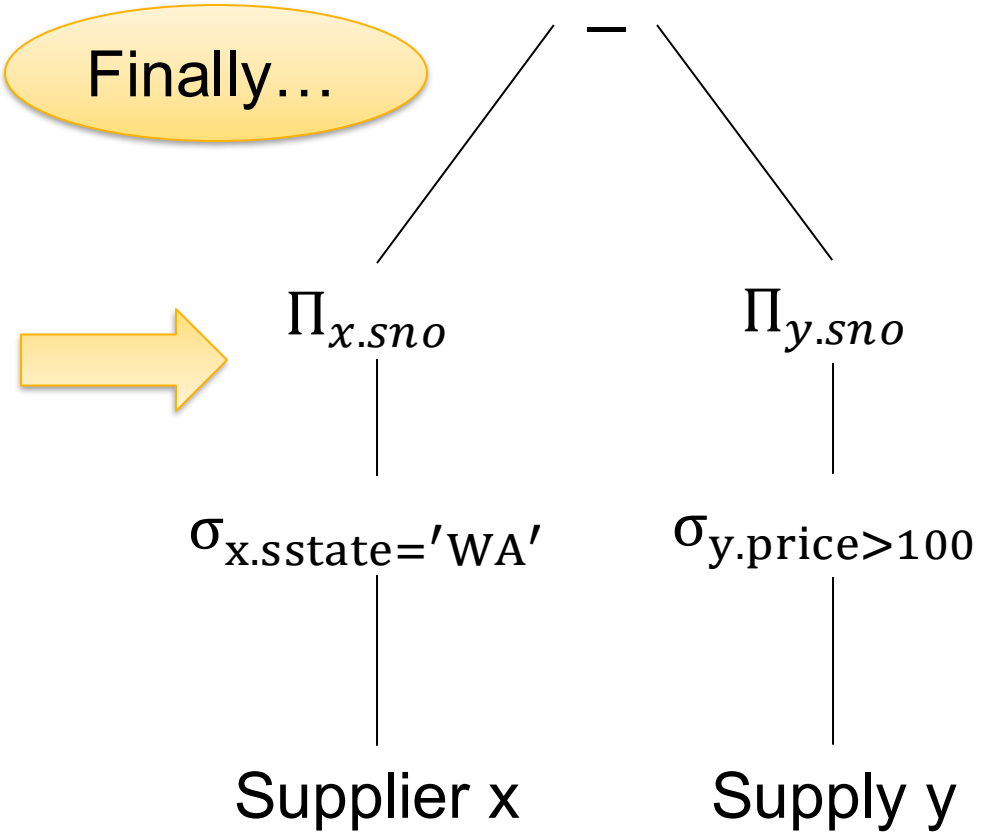
EXCEPT = set difference

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

Subqueries

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

Finally…



Summary

- User writes in SQL
  - Declarative language
  - Users say WHAT they want
- System: SQL → Relational Algebra
  - Explicit operation order: HOW to compute
  - RA expression a.k.a. Query Plan
- Query Plan: is optimized, then executed

Next lectures