CSE544
Data Management

Lecture 3: Data Models

Announcements

- Review of “What goes around…” today
- HW1 is due tonight
- Project teams due Monday; see Ed

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

- Next lectures: query optimization, execution

References

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

Data Model Motivation

- Applications need to model real-world data

- User somehow needs to define data to be stored in DBMS

- Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

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**

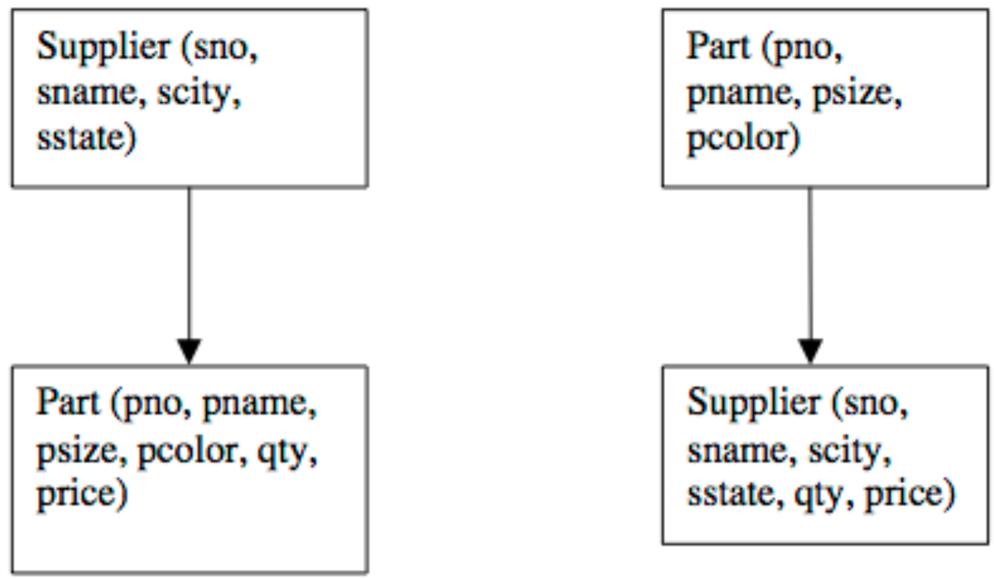
- **IMS database** is collection of instances of record types organized in a tree

* IBM Information Management System

IMS Example

- Figure 2 from “What goes around comes around”

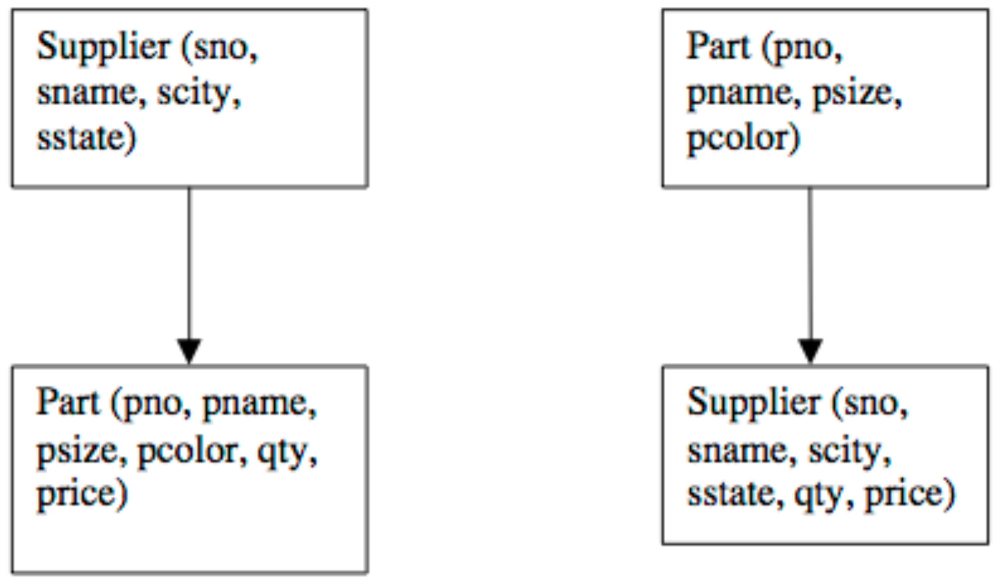
What does
this mean?



IMS Example

- Figure 2 from “What goes around comes around”

What does this mean?

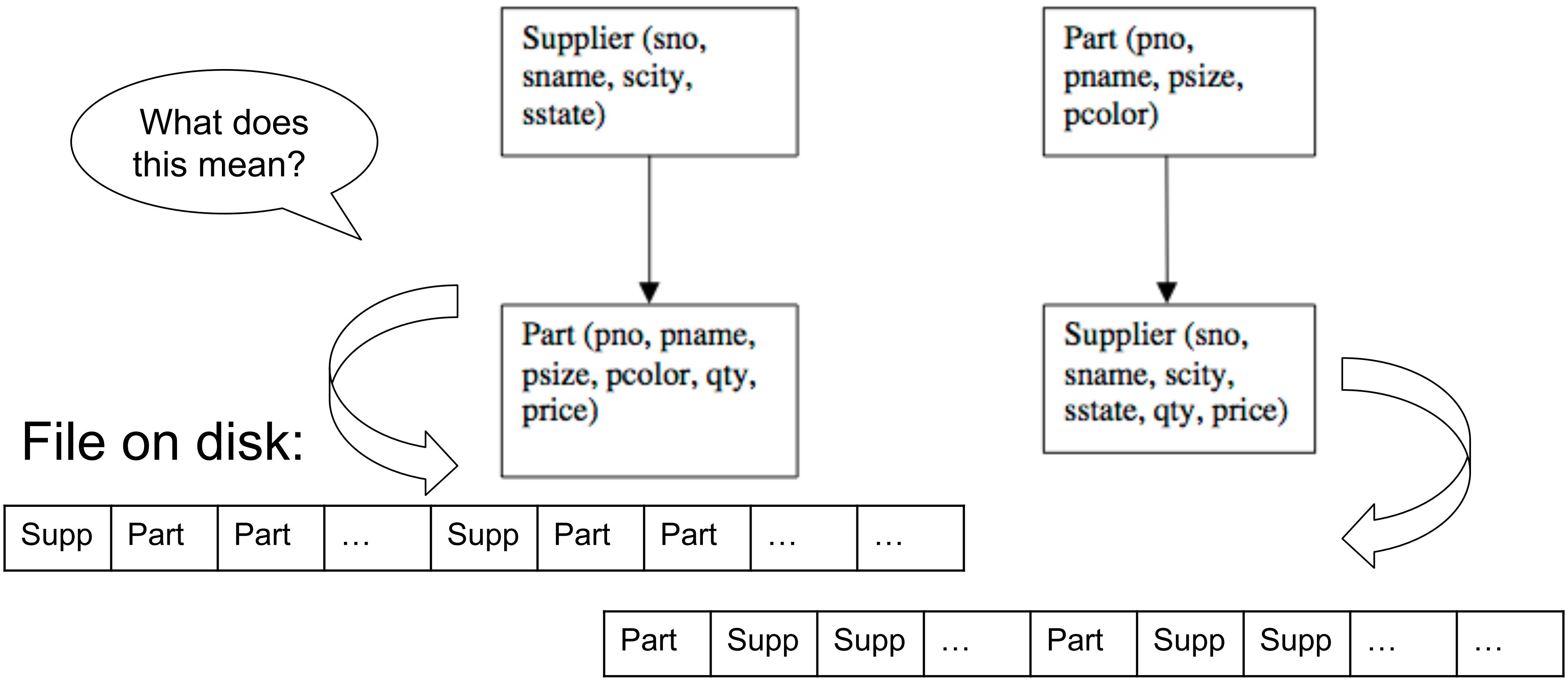


File on disk:

Supp	Part	Part	…	Supp	Part	Part	…	…

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?

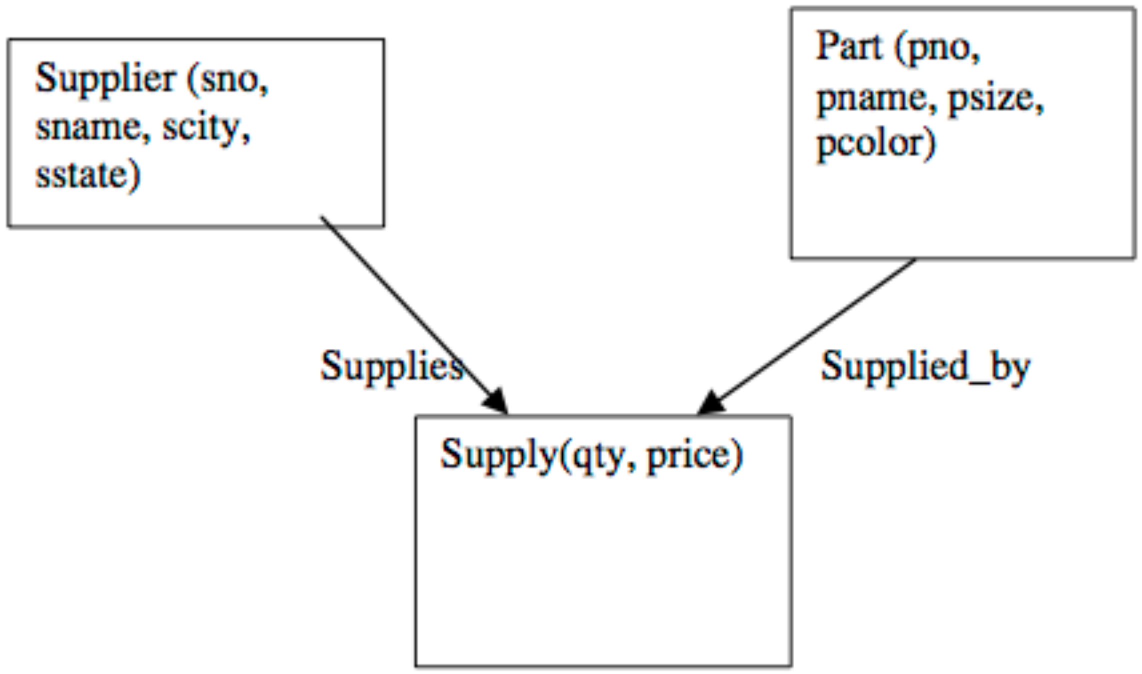
- **Networked data model**

- 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

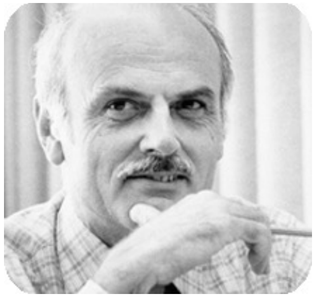
Outline

- 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?



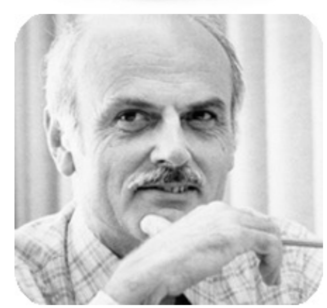
Relational Model Overview

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

Key Elements of the Relational Model

- Declarative query language
  - First Order Logic (FO)
  - Later: SQL
- Physical data independence
  - From FO/SQL to Relational Algebra
  - Optimization
- Design principles:
  - Normalization, to remove anomalies

First Order Logic

A formula consists of

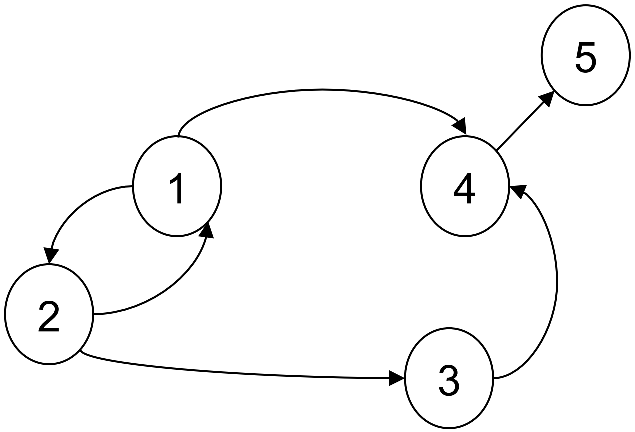
- Variables: x,y,z, …
- Relation names: R, S, …
- Relational atoms: R(x,y,z), S(x,w), …
- Connectives: $\vee, \wedge, \neg, \Rightarrow, \forall, \exists$

A sentence is a formula w/o free variables

A model = instance for all relation names

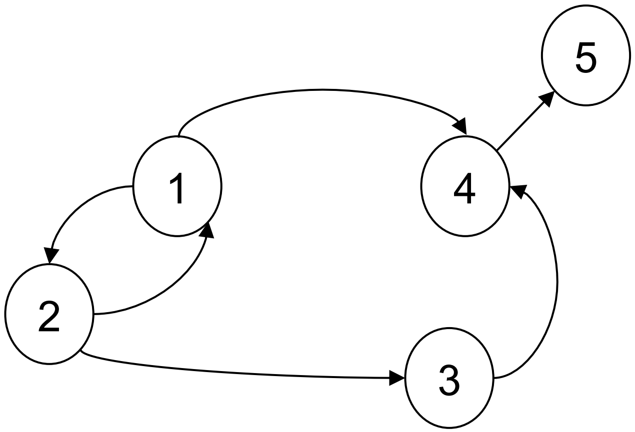
Example: Sentences

A graph:

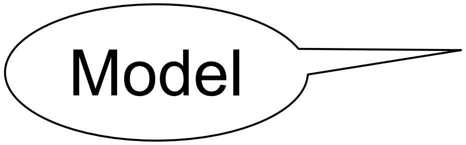


Example: Sentences

A graph:



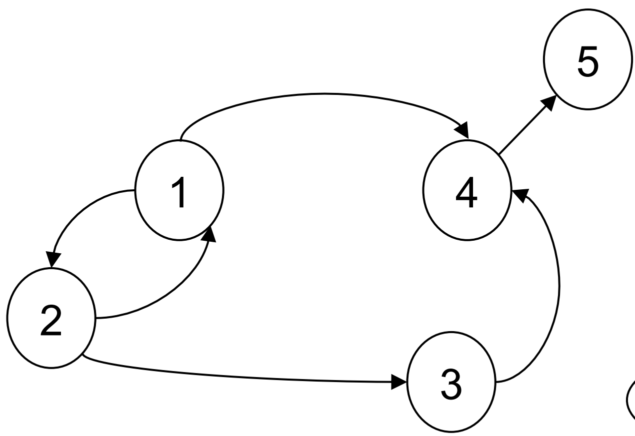
Edge



src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Example: Sentences

A graph:



Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

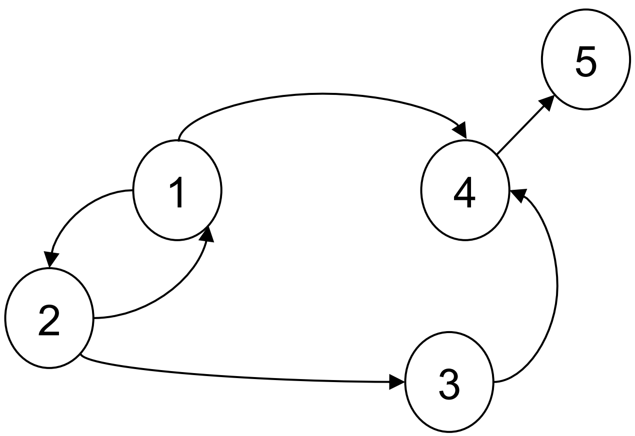
Model

Sentence

$$\exists x \exists y\ (Edge(x, y) \land Edge(y, x))$$

Example: Sentences

A graph:



Edge

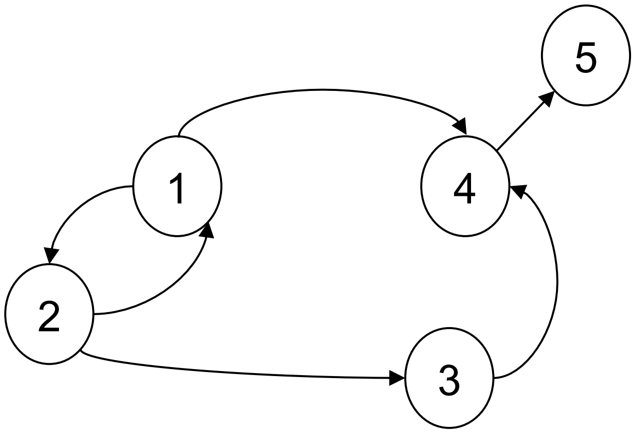
src	dst
1	2
2	1
2	3
1	4
3	4
4	5

True or false?

$$\exists x \exists y \ (Edge(x, y) \land Edge(y, x))$$

Example: Sentences

A graph:



Edge

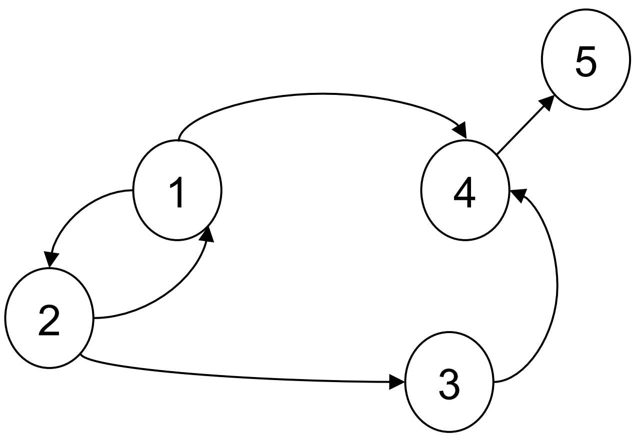
src	dst
1	2
2	1
2	3
1	4
3	4
4	5

True or false?

$\exists x \exists y\ (Edge(x, y) \wedge Edge(y, x))$ True

Example: Sentences

A graph:



Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

True or false?

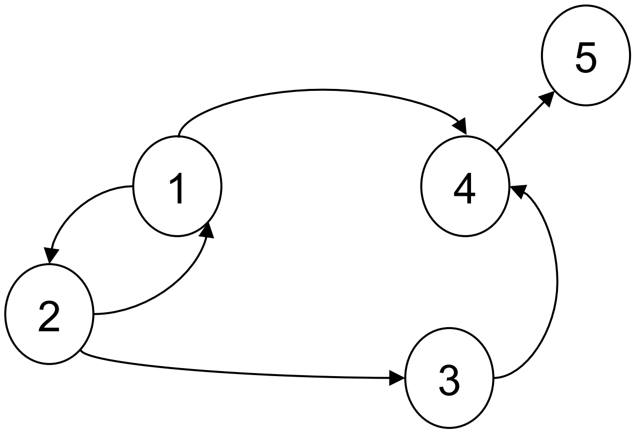
$\exists x \exists y \, (Edge(x, y) \wedge Edge(y, x))$ True

$\exists x \exists y \exists z \, (Edge(x, y) \wedge Edge(y, z) \wedge Edge(z, x))$

$\forall x \forall y (Edge(x, y) \Rightarrow \exists z \, Edge(z, y))$

Example: Sentences

A graph:



Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

True or false?

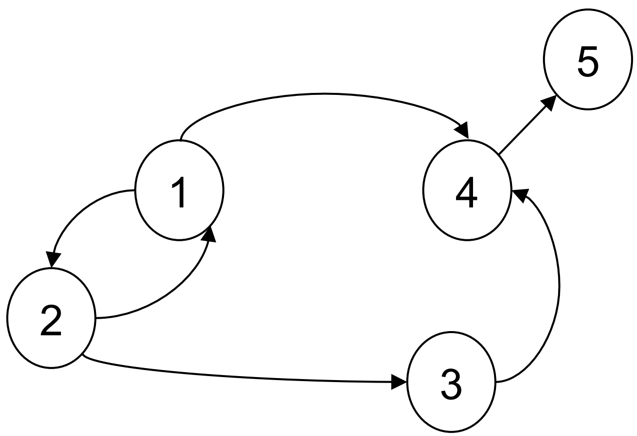
$\exists x \exists y\ (Edge(x,y) \wedge Edge(y,x))$ True

$\exists x \exists y \exists z\ (Edge(x,y) \wedge Edge(y,z) \wedge Edge(z,x))$ False

$\forall x \forall y (Edge(x,y) \Rightarrow \exists z\ Edge(z,y))$ True

Example: Formula

A graph:



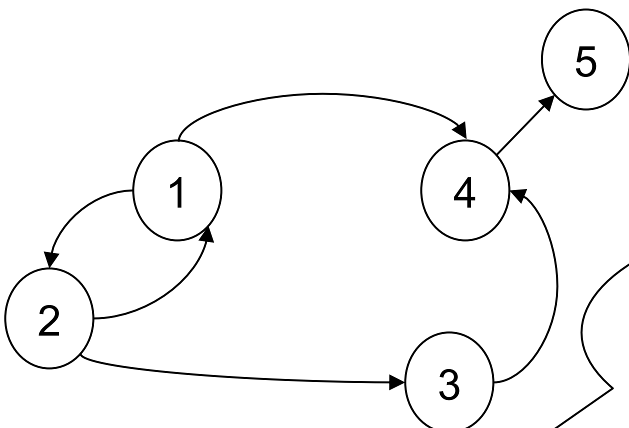
Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

$$\exists y\ (Edge(x, y) \wedge Edge(y, z))$$

Example: Formula

A graph:



Edge

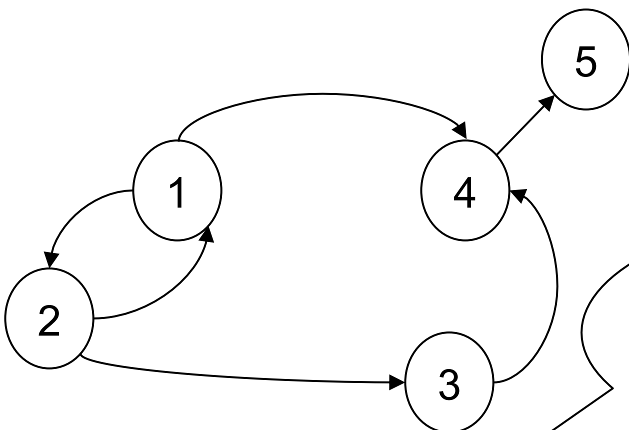
src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Formula
(x,z are free)

$$\exists y \; (Edge(x,y) \wedge Edge(y,z))$$

Example: Formula

A graph:



Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

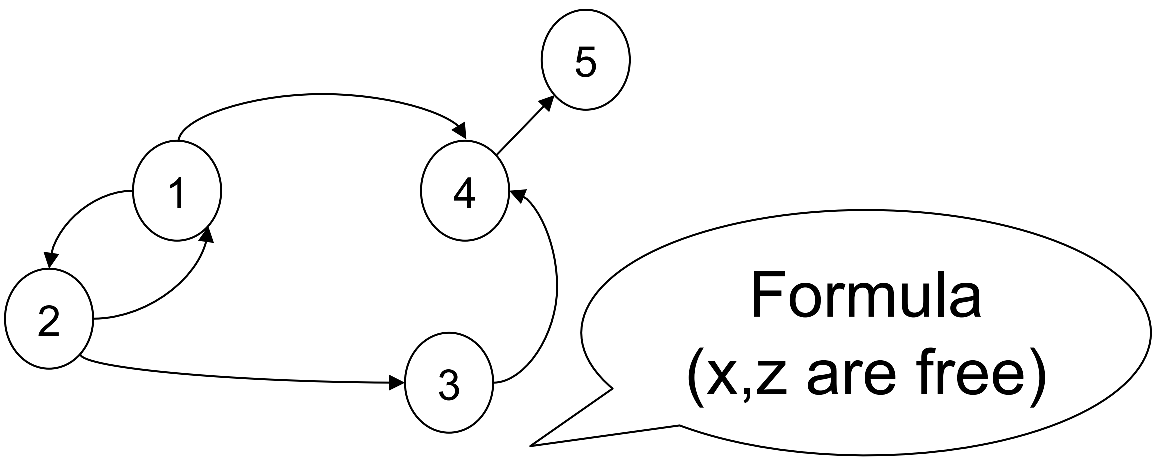
Formula
(x,z are free)

$$\exists y\ (Edge(x,y) \wedge Edge(y,z))$$

Neither true nor false.
A predicate on x,z. A query!

Example: Formula

A graph:



Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

$$\exists y\ (Edge(x,y) \wedge Edge(y,z))$$

Values x,z where true

x	z
1	1
1	5
1	3
…	…

Neither true nor false.
A predicate on x,z. A query!

Discussion

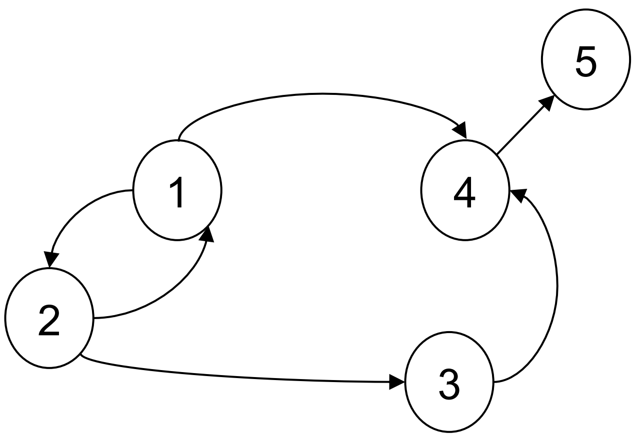
- Codd's proposal:
  - A database is a model
  - A query is a formula

- But FO is too abstract for programmers

- SQL was designed to be more user-friendly

FO v.s. SQL

A graph:



Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

$$Q(x,z) = \exists y\ (Edge(x,y) \wedge Edge(y,z))$$

```
SELECT DISTINCT e1.src as X, e2.dst as Z
FROM Edge e1, Edge e2
WHERE e1.dst = e2.src;
```

Discussion

- FO = very concise, but too abstract
- SQL
  - “Walk up and read”
  - Easy to express ∃
  - Harder to express ∀
  - Bag semantics
  - Aggregates
  - Etc, etc

Relational Algebra

- FO and SQL are declarative languages
  - Users say what they want

- System translates to Relational Algebra
  - RA specifies how to evaluate a query

Relational Algebra

Five operators:

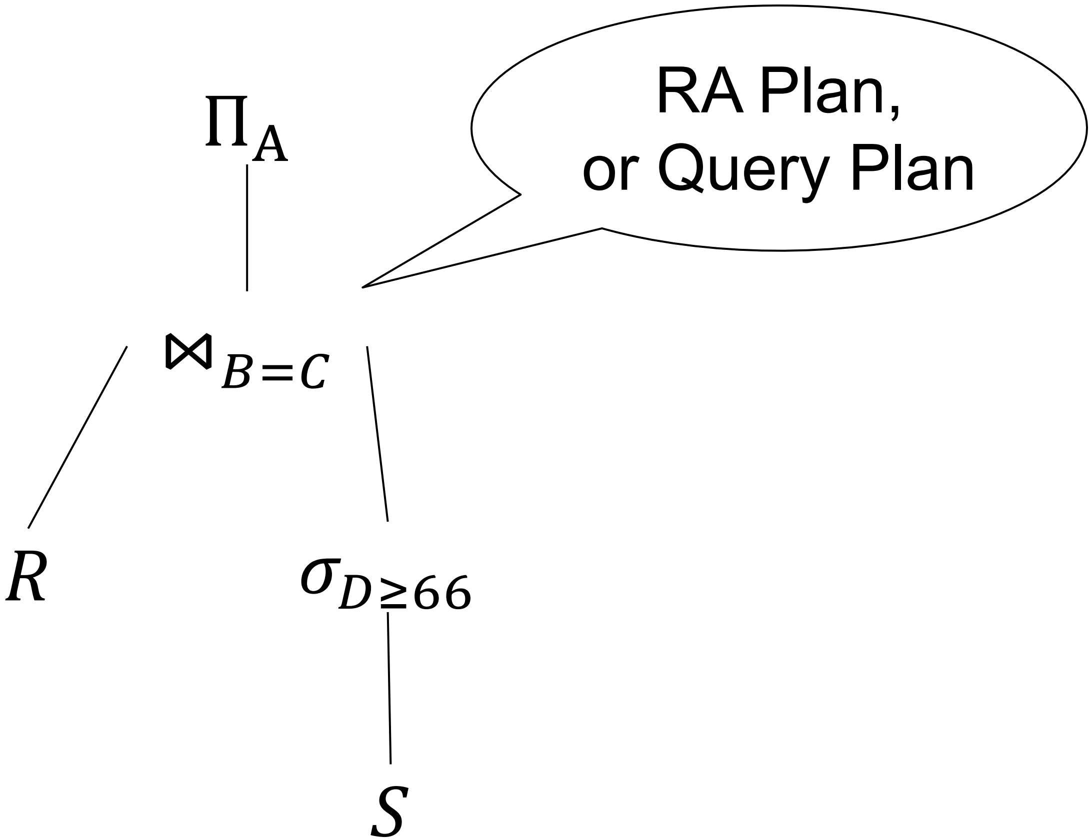
- Selection σ
- Projection Π
- Join or cartesian product $\bowtie$, $\times$
- Union $\cup$
- Difference $-$

RA by Example

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

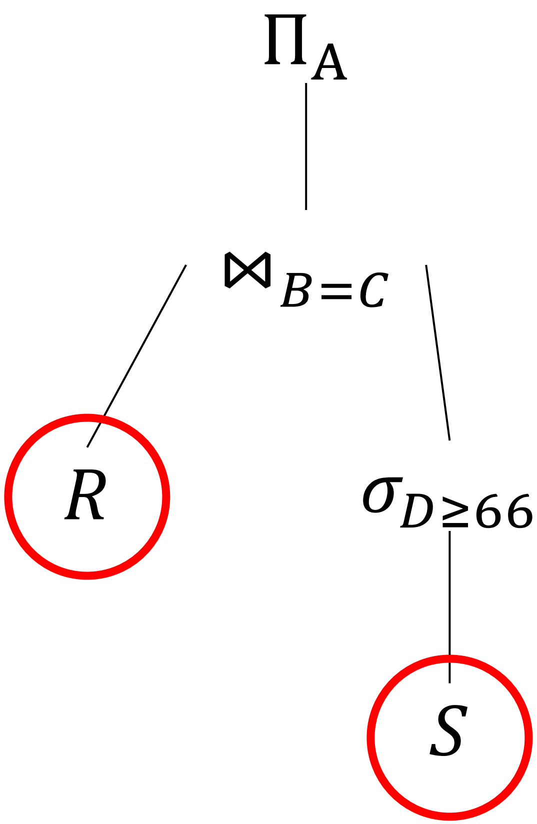
RA by Example

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



RA by Example

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



R=

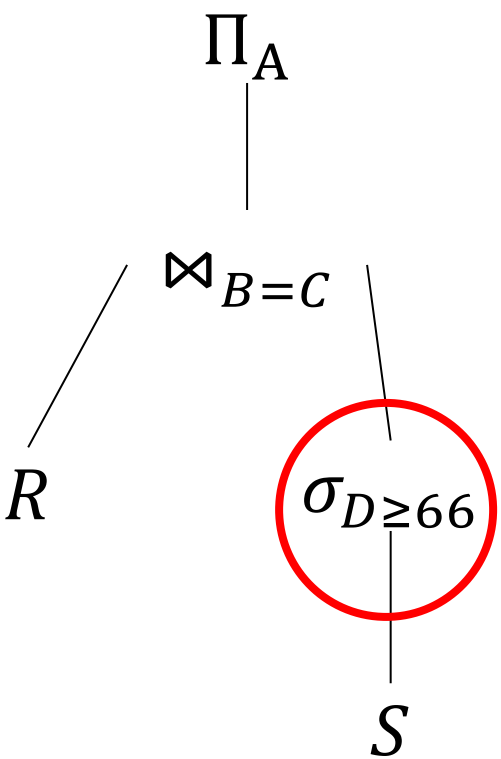
A	B
1	10
1	20
2	20

S=

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

RA by Example

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



C	D
20	66
20	77
30	66

R=

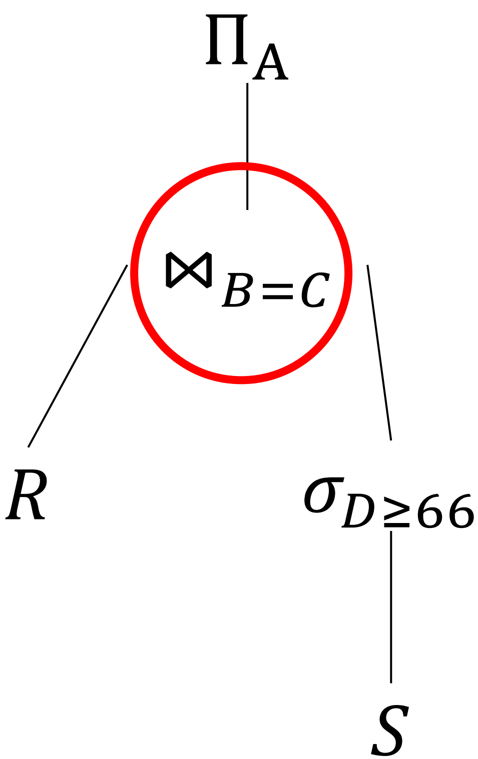
A	B
1	10
1	20
2	20

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	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=

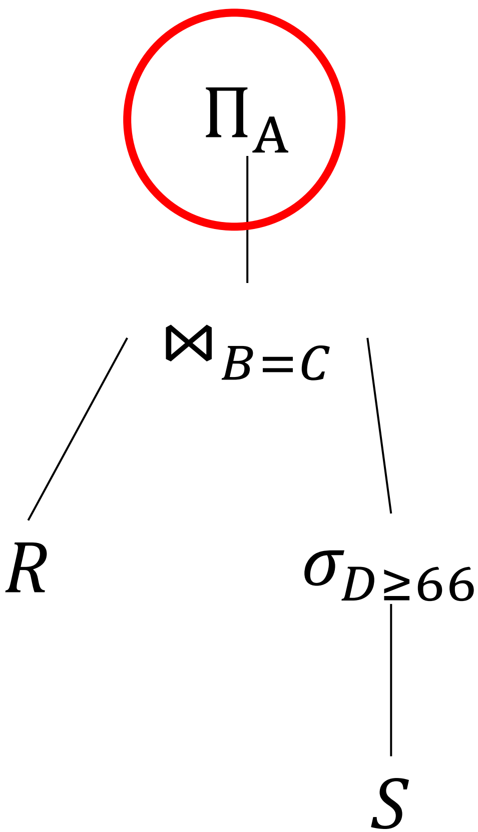
A	B
1	10
1	20
2	20

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

Translation

- Every FO formula can be translated into an equivalent RA expression

Translation

- Every FO formula can be translated into an equivalent RA expression
- Every SQL query can be translated into an expression in Extended RA:
  - Bag semantics
  - Aggregates
  - Duplicate Elimination
  - Etc

Query Plans

- Logical query plan:
  - An RA expression

- Physical query plan:
  - Refine logical operators to a physical ones
  - In other words, choose algorithms

Query Engine

- Convert SQL to RA called Logical Plan
- Optimize Logical Plan
- Convert Logical Plan to Physical Plan
- Execute Physical Plan

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'
```

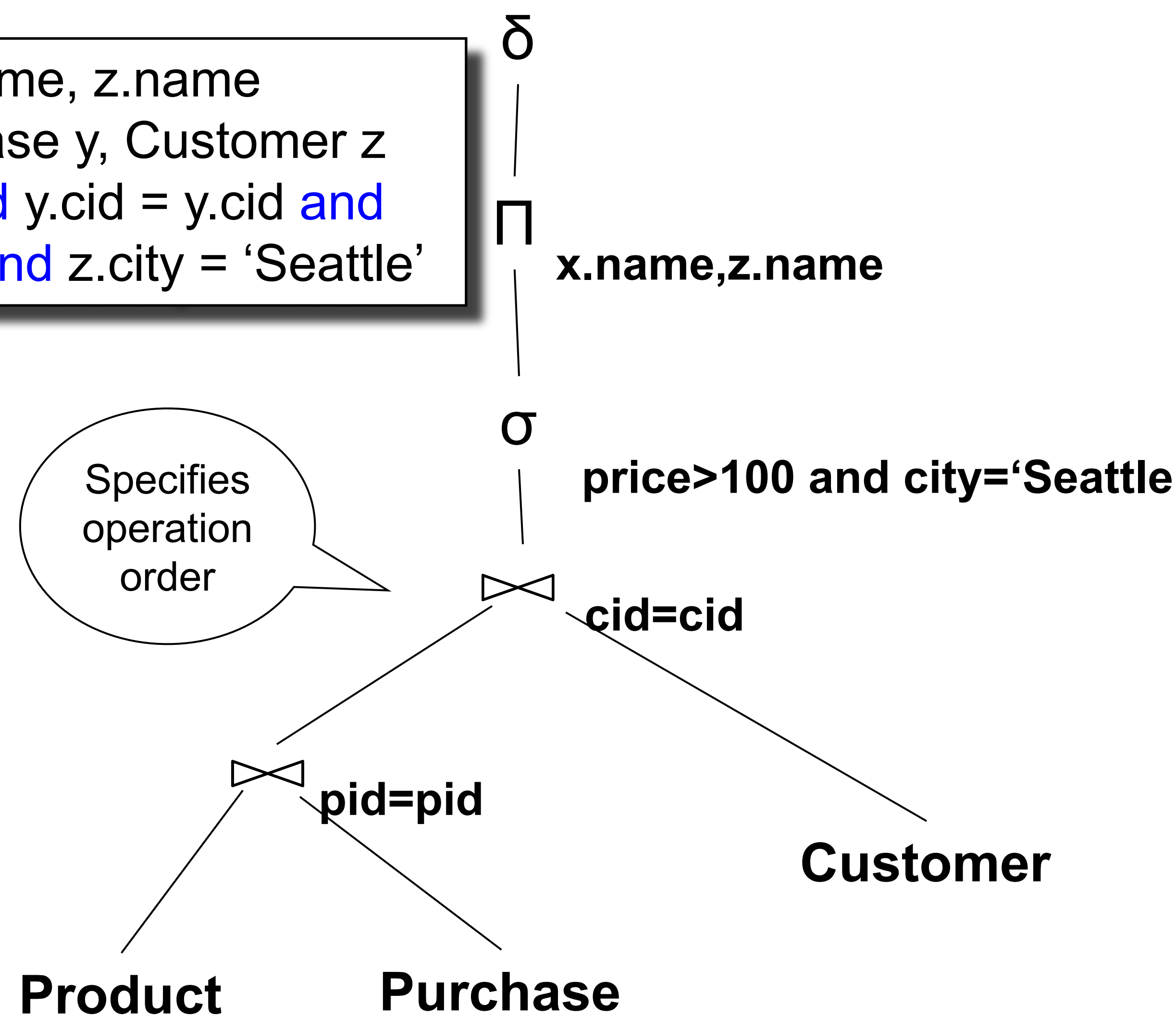
We say *What* we want

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

…to Logical Plan…

```
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'
```

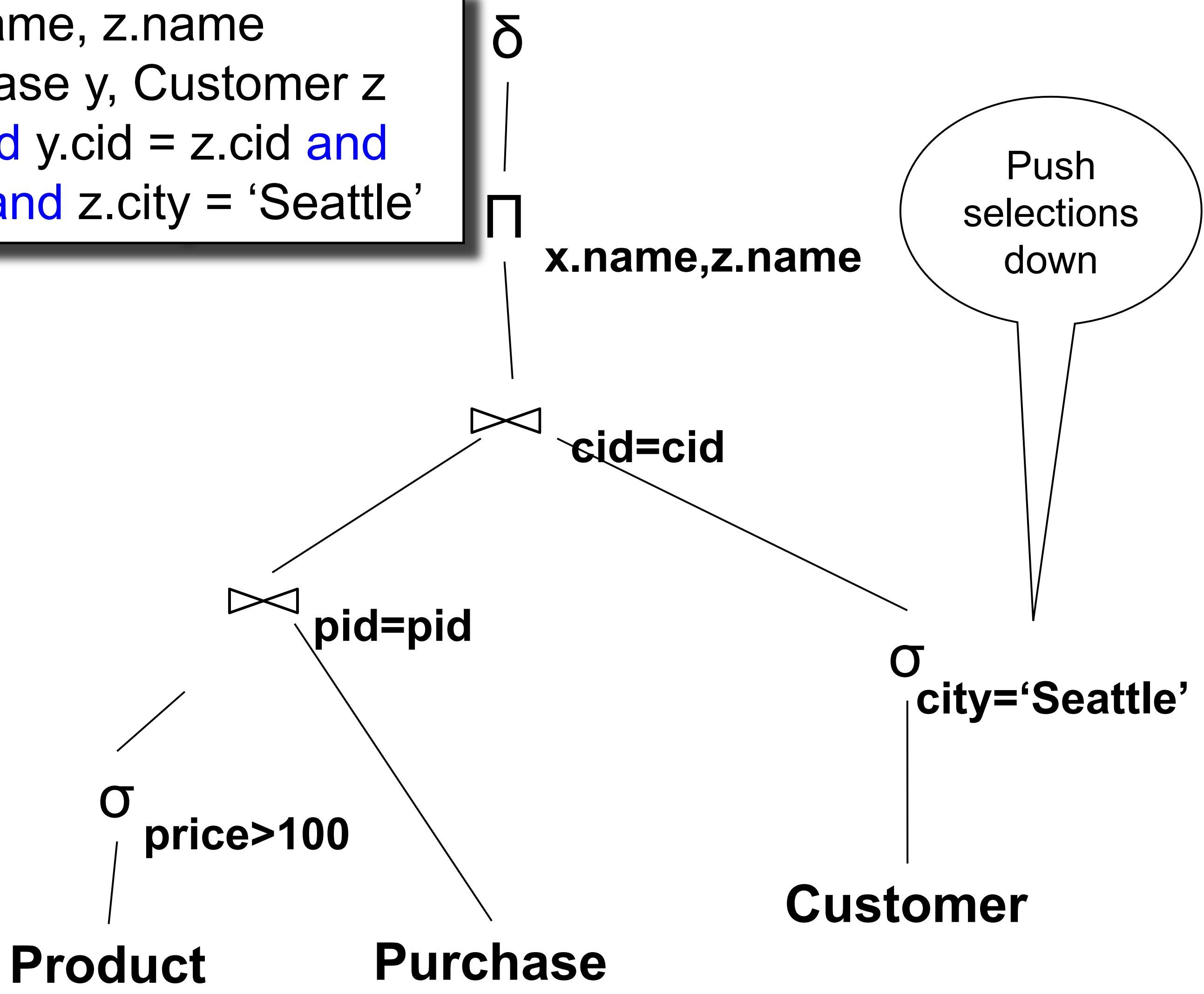
We say *What* we want



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'

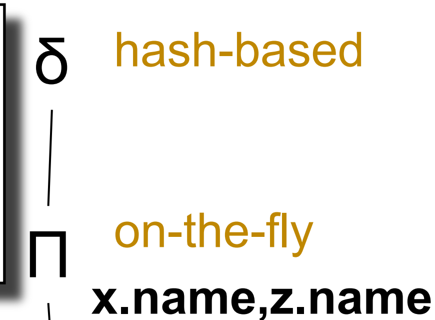


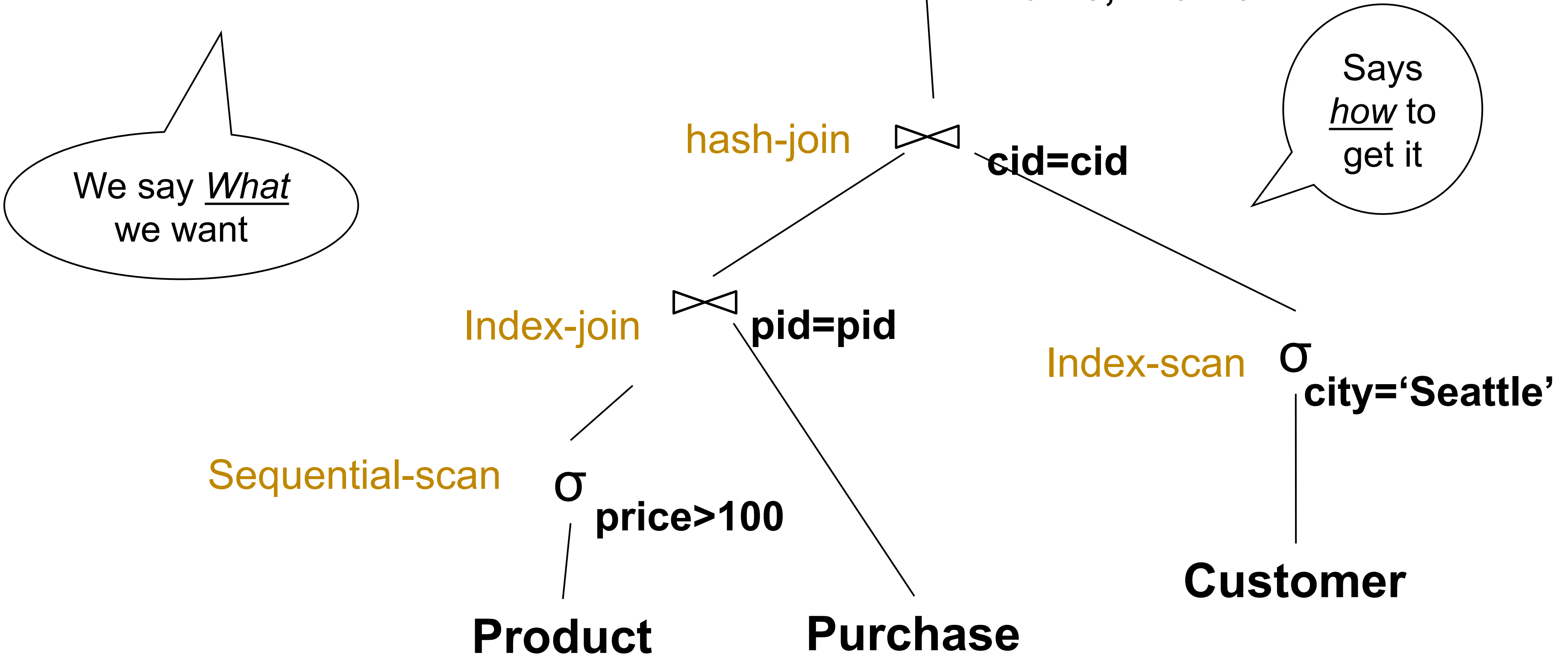
More about this next lectures

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

..Physical 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'
```



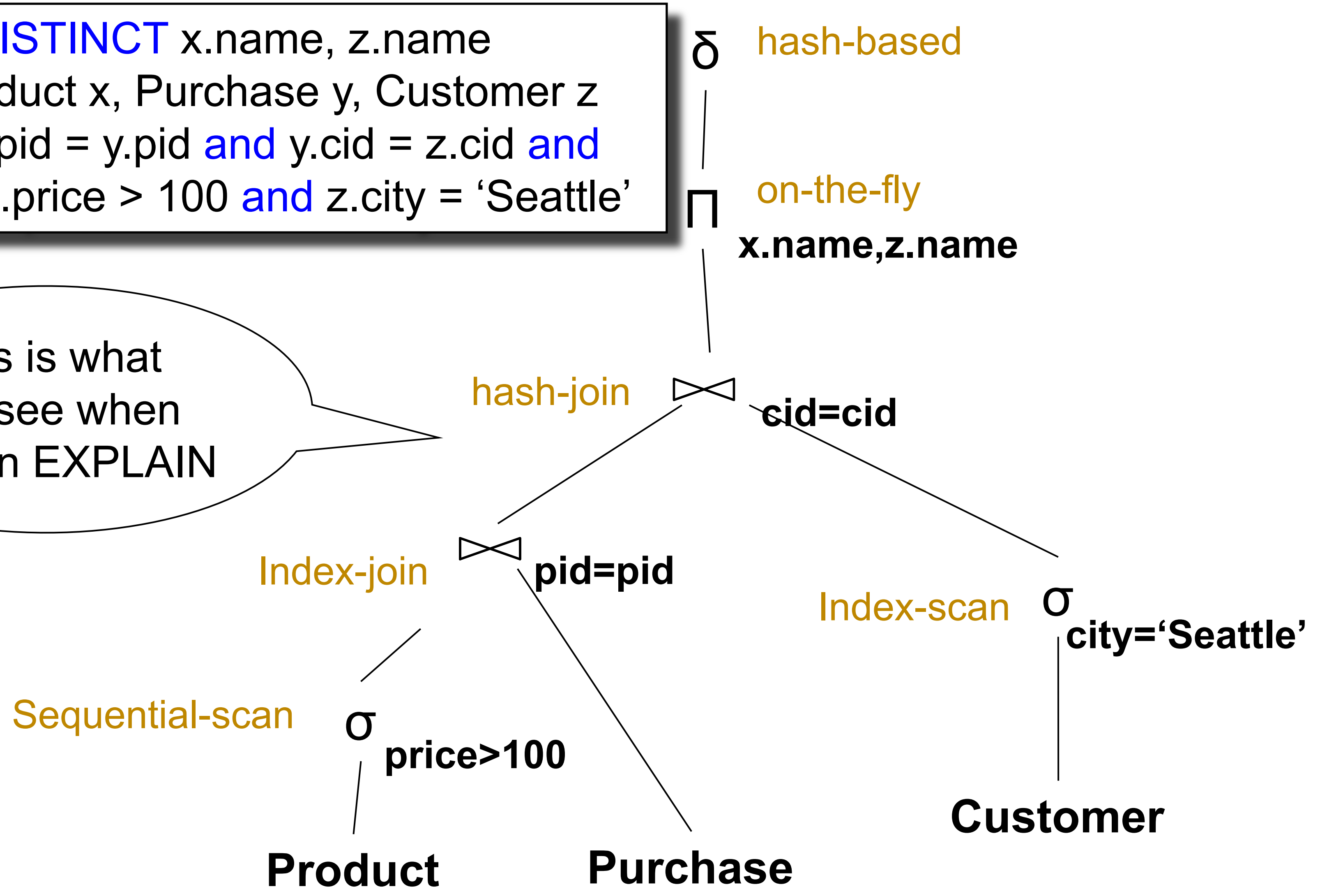


Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

..Physical 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'
```

This is what you see when you run EXPLAIN



Physical Data Independence

Separate the logical description of the data and queries from the concrete physical layout of the data and algorithms for running the query

Logical Data Independence

- Separates the logical schema of the database from that of the application

- Allows database logical schema to change without affecting applications

- Supported in SQL through views

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: Big_Parts(pno,pname,psize,pcolor)

Part(pno,pname,psize,pcolor)

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: Big_Parts(pno,pname,psize,pcolor)

Querying the view:

```
SELECT *
FROM Big_Parts
WHERE pcolor='blue';
```

Two Types of Views

- Virtual views:
  - Default in SQL
  - CREATE VIEW xyz AS …
  - Computed at query time

What are the pros and cons?

- Materialized views:
  - Some SQL engines support them
  - CREATE MATERIALIZED VIEW xyz AS
  - Computed at definition time

Relational Model Takeaways

- Simple relations, declarative language
- Optimizer plays key role
- Please read on your own:
  - E/R diagrams (needed for hw1)
  - Schema normalization (BCNF, 3NF)

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