CSEP544
Data Management

Introduction, SQL

Outline

- Introduction, class overview
- Database management systems (DBMS)
- The relational model
- SQL

Course Staff

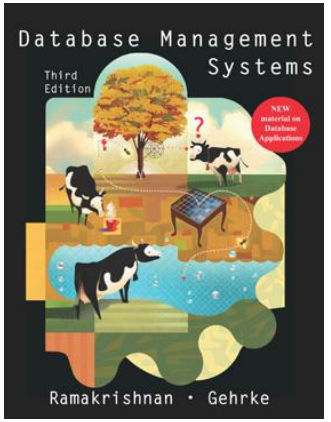
- Instructor: Dan Suciu
  - Office hours: Tuesdays, 5:30-6:20
- TAs (Office hours TBD)
  - Kyle Deeds

Goals of the Class

- Relational Data Model
  - Data models, data independence, declarative QL.
- Relational Database Systems
  - Storage, query execution and optimization
- Datalog
- Advanced topics (depending on time):
  - WCOJ, LSM trees, data sketches
- Transactions
  - Optimistic/pessimistic concurrency control

Readings

- Paper reviews
  - Mix of old seminal papers and new papers
  - Papers are available on class website
- Lecture notes (the slides)
  - Posted on class website after each lecture
- Background from:
  - Database Management Systems. **Third Ed**. Ramakrishnan and Gehrke. McGraw-Hill.



Other Readings

- Database course at CMU
  - Great for low-level details (storage, TXNs)
- Database theory course at Berkeley
  - Finite model theory, complexity

Class Resources

- Website: lectures, assignments
  - http://www.cs.washington.edu/csep544
- Canvas: zoom, videos
  - https://canvas.uw.edu/courses/1785992
- Ed: discussion board
  - https://edstem.org/us/courses/72202/discussion
- Gitlab: https://gitlab.cs.washington.edu/

Evaluation

- Assignments 50%
- Reviews 25%
- Mini-Project 25%

Assignments – 50%

- **HW1**: Data analysis in postgres
  - Posted, and due on January 26
- **HW2**: Advanced SQL
- **HW3**: SimpleDB
- **HW4**: Datalog
- **HW5**: Query optimization

Paper reviews – 25%

- Recommended length: ½ page – 1 page
  - Summary of main points
  - Critical discussion
- Grading: 0 or 1 or 2 points
- Submit by 6pm. *Before* the lecture

- First review due next Wednesday

MiniProject – 25%

Topic of your own choosing, open ended. E.g.

- Replicate experiments from a paper
- Compare 2-3 systems
- Add functionality to some DBMS
- Evaluate a technology that you need at work

How to Turn In

- Homeworks: gitlab
- Project: gitlab
- Reviews: google forms

See instructions posted with HW1

Now onward to the world of databases!

Database

Database = collection of files storing inter-related data.

Consists of

- **Entities**: e.g. products, suppliers, customers, employees, warehouses
- **Relationships**: e.g. suppliers-products, customer-products, employee-manager

Database Management System

DBMS: a software system designed to provide data management services

Examples

- Oracle, DB2 (IBM), SQL Server (Microsoft)
- Snowflake, Redshift
- PostgreSQL, Duckdb, MySQL, Sqlite

Database Example

A database of products and suppliers:

- **Product:** has a name, a price, a color
- **Supplier:** has a name, the products it supplies, city

Flat File Strawman

- Store data in csv files
- Manage your data in python

Flat File Strawman

- Store data in csv files
- Manage your data in python

Product(name, price, color)

```
iPhone, 599, gray
iPhone, 999, black
Gizmo, 399, blue
Pizza, 29, red
```

Flat File Strawman

- Store data in csv files
- Manage your data in python

Product(name, price, color)

iPhone, 599, gray
iPhone, 999, black
Gizmo, 399, blue
Pizza, 29, red

Supplier(name, product, city)

ACME, iPhone, Seattle
Walmart, iPhone, Renton
Costco, Pizza, Seattle

Flat File Strawman

- Store data in csv files
- Manage your data in python

Product(name, price, color)

```
iPhone, 599, gray
iPhone, 999, black
Gizmo, 399, blue
Pizza, 29, red
```

Flat File Strawman

- Store data in csv files
- Manage your data in python

Product(name, price, color)

iPhone, 599, gray
iPhone, 999, black
Gizmo, 399, blue
Pizza, 29, red

Find the price of Gizmo:

Flat File Strawman

- Store data in csv files
- Manage your data in python

Product(name, price, color)

```
iPhone, 599, gray
iPhone, 999, black
Gizmo, 399, blue
Pizza, 29, red
```

Find the price of Gizmo:

```
with open('product.csv') as f:
    r = csv.reader(f, delimiter=',')
    for t in r:
        if t[0] == "Gizmo":
            print(t[1])
```

Issues with Flat Files

Need to implement many generic tasks:

- Data integrity
- Efficient implementation
- Concurrency, durability
- Etc.

Flat Files: Data Integrity

- Price should be a number
- Supplier names should be unique
- Suppliers may supply >1 products
- Each supplier in a single city

Flat Files: Implementation

- How do we update/insert/delete?
- Data may be larger than main memory
- A query may traverse multiple files
- Data may be distributed

Flat Files: Concurrency, Durability

- What if multiple applications touch the data?
- What if the system crashes in the middle of an update?

Database Management System

A software system designed to provide data management services:

- Create/update/query a database
- Enforce integrity constraints
- Evaluate queries efficiently
- Handle concurrency, recovery
- …

Important Terminology

- Architecture
- Workloads

Architecture: Single Client

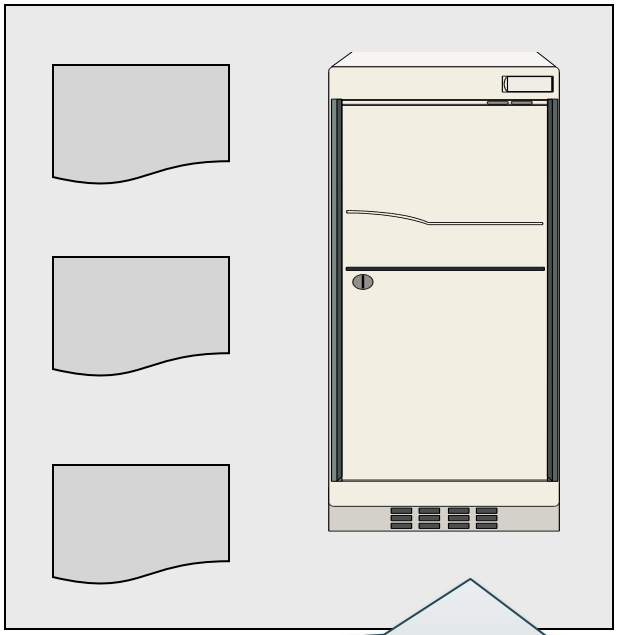
E.g. data analytics

Application and database
on the same computer

E.g. sqlite, postgres

Two-tier Architecture Client-Server

E.g. accounting, banking, …



Connection:

ODBC, JDBC

Database server

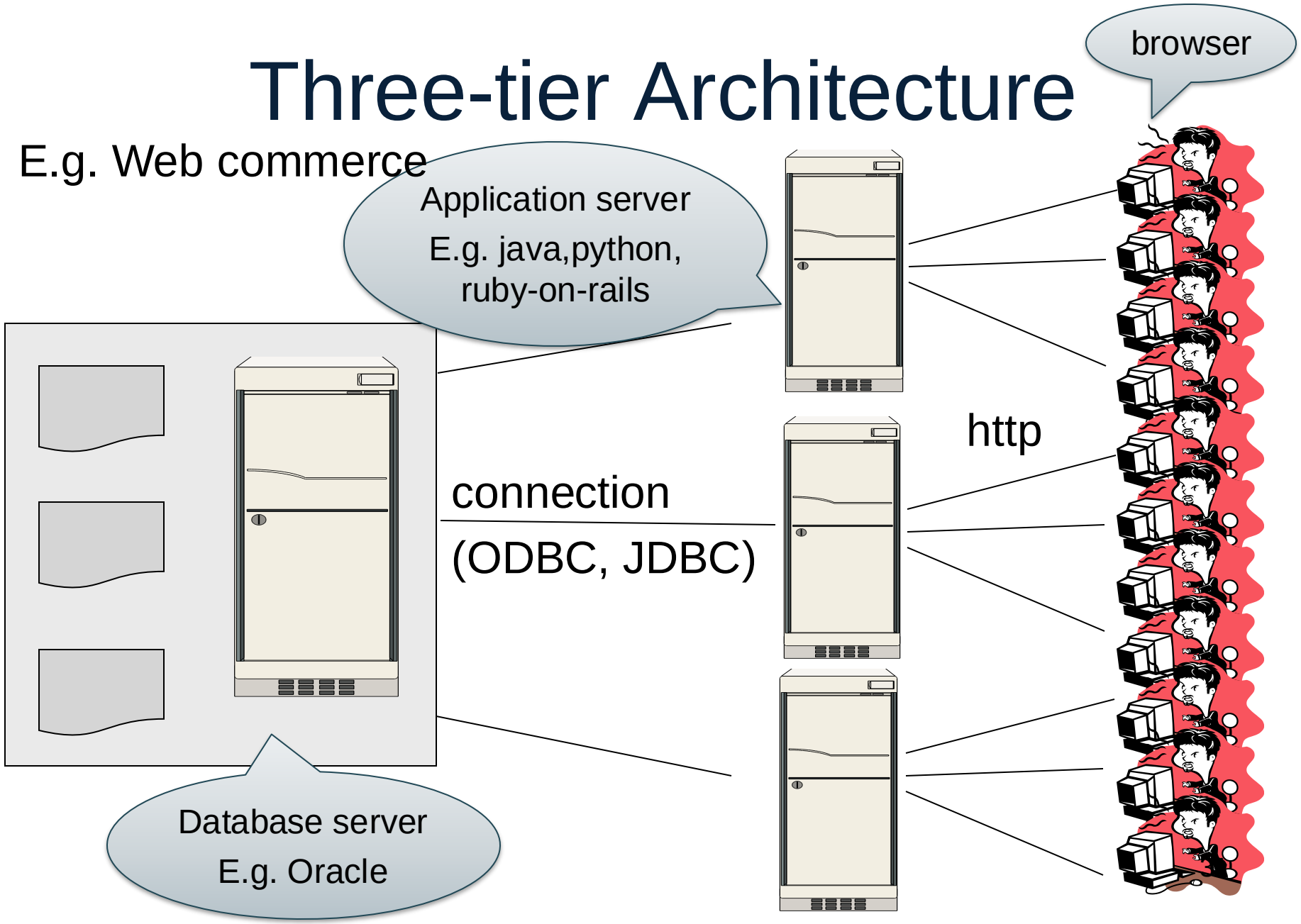
E.g. Oracle, DB2,…

Applications:

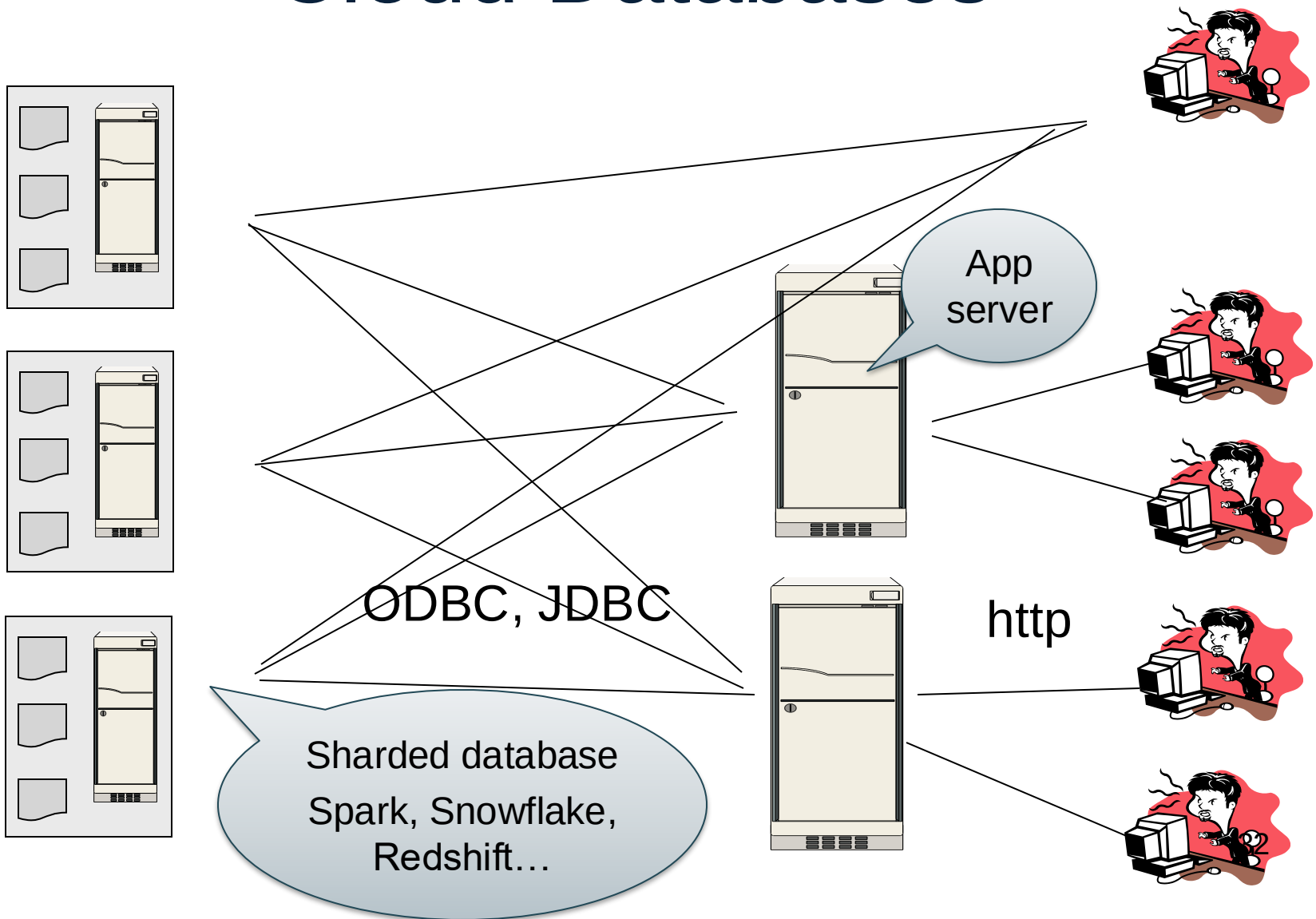
Java

Three-tier Architecture

E.g. Web commerce



Cloud Databases



Types of Workloads

- **OLTP** – online transaction processing
  - Single item read/update
  - Concurrency, transactions

- **OLAP** – online analytics processing
  - Complex queries w/ aggregages
  - Updates are rare

Summary

- DBMS: store and manage your data
- Everywhere:
  - On your phone: sqlite
  - On your laptop: sqlite, postgres, duckdb,…
  - In the cloud: snowflake/redshift/bigquery
- All use the Relational Data Model

Relational Data Model

Data Model

Mathematical formalism that models data.

Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…

Data Model

Mathematical formalism that models data.

Several exist:

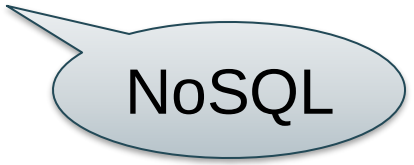
- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…

Data Model

Mathematical formalism that models data.

Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…



Data Model

Mathematical formalism that models data.

Several exist:

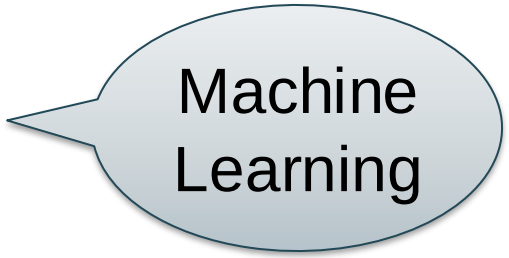
- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…

Data Model

Mathematical formalism that models data.

Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…



Data Model

Mathematical formalism that models data.

Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…

Data Model

Mathematical formalism that models data.

Several exist:

- Relational
- Key/value / Document / XML / Json
- Graph
- Arrays / matrices / tensors
- Hierarchical, network…

Relational Data Model

Database: Collection of relations

Relation (aka Table): Set of tuples

Tuple (row, record): $t \in Dom_1 \times \cdots \times Dom_n$

Relational Data Model

Product(name, price, color)

name	**price**	**color**
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Relational Data Model

Product(name, price, color)

name	**price**	**color**
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Supplier(name, product, city)

name	**product**	**city**
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Schema

Relational Data Model

Product(name, price, color)

name	price	color
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Supplier(name, product, city)

name	product	city
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Relational Data Model

Schema

Product(name, price, color)

name	price	color
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Instance

Supplier(name, product, city)

name	product	city
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Discussion

- Rows in a relation:
  - Ordering immaterial
  - All rows are distinct – sets
  - Query answers may have duplicates – bags

Discussion

- Rows in a relation:
  - Ordering immaterial
  - All rows are distinct – sets
  - Query answers may have duplicates – bags

- Attributes of a record:
  - Ordering is immaterial (mostly…)
  - Applications refer to columns by their names

Discussion

- Rows in a relation:
  - Ordering immaterial
  - All rows are distinct – sets
  - Query answers may have duplicates – bags
- Attributes of a record:
  - Ordering is immaterial (mostly…)
  - Applications refer to columns by their names
- Domain of each column is a primitive type
  - Relations are flat: First Normal Form, 1NF

Primary Key

- A key is an attribute, or a set of attributes whose value uniquely identify the record

- There may be more than one: we choose one that we call primary key

Primary Key

Product(name, price, color)

name	**price**	**color**
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Supplier(name, product, city)

name	**product**	**city**
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Primary key = name

Primary Key

Product(name, price, color)

name	**price**	**color**
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Primary key = ???

Supplier(name, product, city)

name	**product**	**city**
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Primary key = name

Primary Key

Product(name, price, color)

name	**price**	**color**
iPhone	599	Gray
iPhone	999	Black
Gizmo	399	Blue
Pizza	29	red

Primary key = name, price color

Supplier(name, product, city)

name	**product**	**city**
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Primary key = name

Primary Key

Product(pid, name, price, color)

pid	**name**	**price**	**color**
p001	iPhone	599	Gray
p002	iPhone	999	Black
p003	Gizmo	399	Blue
p004	Pizza	29	red

Primary key = pid

Supplier(name, product, city)

name	**product**	**city**
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

Primary key = name

Good practice to have a single attribute as primary key

Foreign Key

- An attribute whose values are keys of another relation is called a foreign key

- Also called “semantic pointer”

Foreign Key

Product(pid, name, price, color)

pid	**name**	**price**	**color**
p001	iPhone	599	Gray
p002	iPhone	999	Black
p003	Gizmo	399	Blue
p004	Pizza	29	red

Supplier(name, product, city)

name	**product**	**city**
ACME	iPhone	Seattle
Walmart	iPhone	Renton
Costco	Pizza	Seattle

product is ambiguous. Why?

Foreign Key

Product(pid, name, price, color)

pid	**name**	**price**	**color**
p001	iPhone	599	Gray
p002	iPhone	999	Black
p003	Gizmo	399	Blue
p004	Pizza	29	red

Supplier(name, pid, city)

name	**pid**	**city**
ACME	p002	Seattle
Walmart	p002	Renton
Costco	p004	Seattle

pid is a foreign key

Foreign Key

Product(pid, name, price, color)

pid	name	price	color
p001	iPhone	599	Gray
p002	iPhone	999	Black
p003	Gizmo	399	Blue
p004	Pizza	29	red

Supplier(name, pid, city)

name	pid	city
ACME	p002	Seattle
Walmart	p002	Renton
Costco	p004	Seattle

pid is a foreign key

What if Walmart sells both iPhones?

Relationships

Product(pid, name, price, color)

pid	name	price	color
p001	iPhone	599	Gray
p002	iPhone	999	Black
p003	Gizmo	399	Blue
p004	Pizza	29	red

Supplier(name, city)

name	city
ACME	Seattle
Walmart	Renton
Costco	Seattle

pid	name
p002	ACME
p001	Walmart
p002	Walmart
p004	Costco

Relationships

Product(pid, name, price, color)

pid	name	price	color
p001	iPhone	599	Gray
p002	iPhone	999	Black
p003	Gizmo	399	Blue
p004	Pizza	29	red

Supplier(name, city)

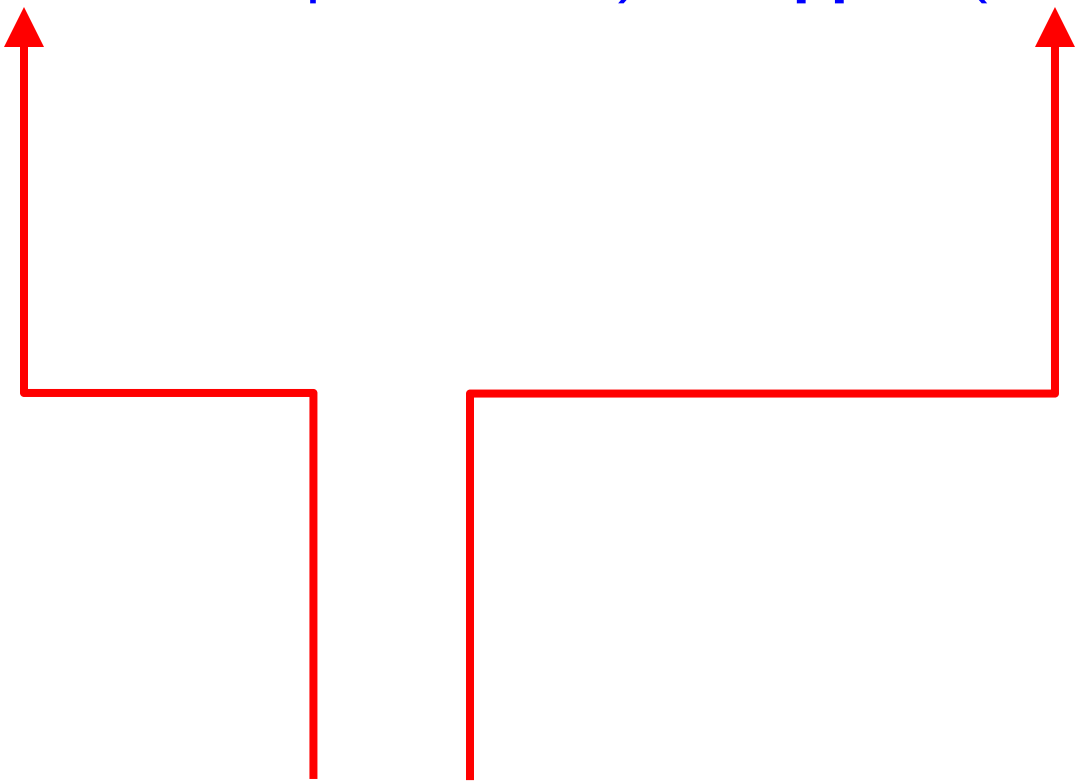
name	city
ACME	Seattle
Walmart	Renton
Costco	Seattle

Supply(pid, name)

pid	name
p002	ACME
p001	Walmart
p002	Walmart
p004	Costco

Relationships

Product(pid, name, price, color) **Supplier**(name, city)



Supply(pid, name)

Relational Data Model: Summary

- Data is stored in flat relations: 1NF
- No prescription of the physical storage
- Access to the data through high-level declarative language: SQL

SQL

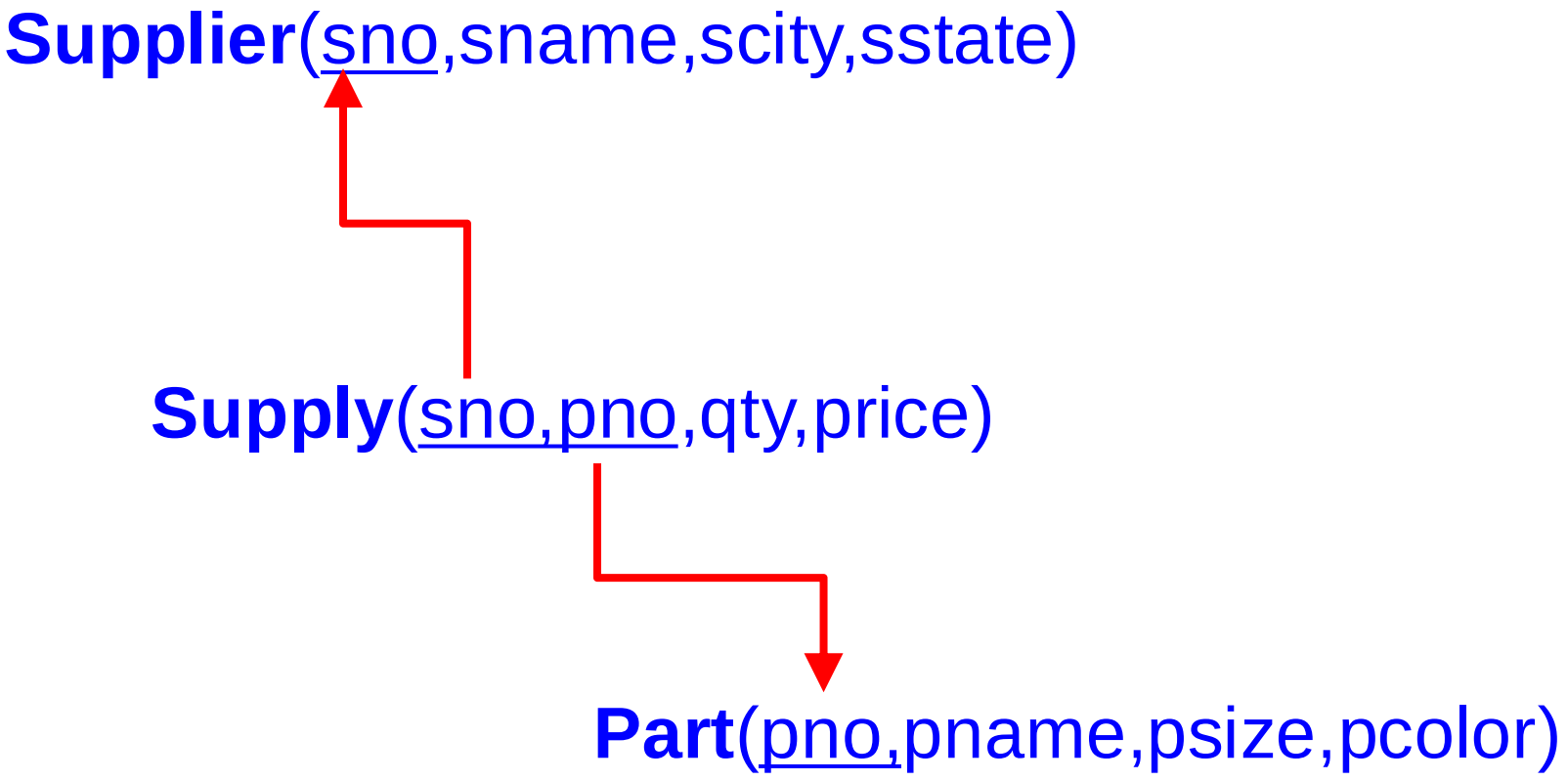
SQL

- Introduced in the late 70s
- Standard has been continuously evolving into a huge language
- DBMS support various subsets
- We will study mostly SQL’92

SQL

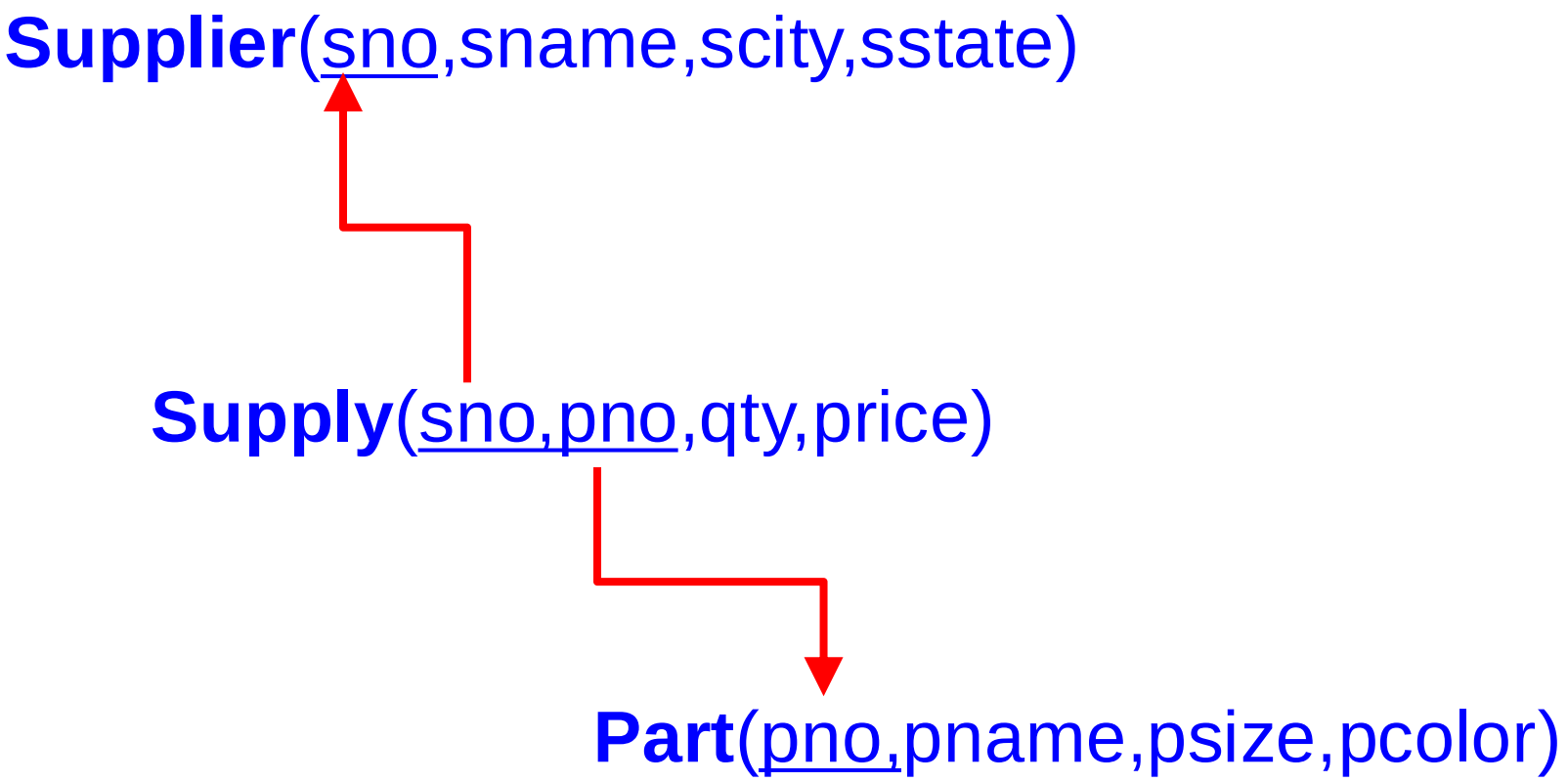
- Data Definition Language (DDL):
  - CREATE TABLE, …
  - You will mostly read on your own
- Data Manipulation Language (DML)
  - SELECT-FROM-WHERE
  - INSERT/DELETE/UPDATE

Relational Data Model



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

Relational Data Model



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

Relational Data Model

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

SQL

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int,
             sname text,
             scity text,
             sstate text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text,
             scity text,
             sstate text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text,
             scity text,
             sstate text);
```

```
CREATE TABLE
    Part(pno int primary key,
         pname text,
         psize int,
         pcolor text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text
```

```
CREATE TABLE
    Supply(sno int,
           pno int,
           qty int,
           price int);
```

```
ary key,
           pcolor text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text
```

```
CREATE TABLE
    Supply(sno int references Supplier,
           pno int references Part,
           qty int,
           price int);
```

```
ary key,
           pcolor text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text
```

```
CREATE TABLE
    Supply(sno int references Supplier,
           pno int references Part,
           qty int,
           price int,
           primary key (sno, pno));
```

```
ry key,
                pcolor text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text,
             scity text,
             sstate text);
```

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

SQL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text,
             scity text,
             sstate text);
```

sno	sname	scity	sstate

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

SQL

Supplier

sno	sname	scity	sstate

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

SQL

```
INSERT INTO Supplier VALUES
    (11,'ACME','Seattle','WA'),
    (12,'Walmart','Portland','OR'),
    (13,'Safeway',Seattle','WA'),
    (14,'Walmart','Seattle','WA');
```

Supplier

sno	sname	scity	sstate

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

SQL

```
INSERT INTO Supplier VALUES
    (11,'ACME','Seattle','WA'),
    (12,'Walmart','Portland','OR'),
    (13,'Safeway',Seattle','WA'),
    (14,'Walmart','Seattle','WA');
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

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

SQL

```
SELECT …columns…
FROM …tables…
WHERE …condition…
```

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

SQL

```
SELECT …columns…
FROM …tables…
WHERE …condition…
```

Original design philosophy: *walk up and read*

It did not age well: “Problems with SQL” paper

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

SQL

```
SELECT sname
FROM Supplier
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

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

SQL

```
SELECT sname
FROM Supplier
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

sname
ACME
Walmart
Costco
Walmart

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

SQL


```
SELECT sname
FROM Supplier
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

sname
ACME
Walmart
Costco
Walmart

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

SQL

Remove duplicates

```
SELECT DISTINCT sname
FROM Supplier
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

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

SQL

Remove duplicates

```
SELECT DISTINCT sname
FROM Supplier
```

Now it's a set

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

sname
ACME
Walmart
Costco

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

SQL

```
SELECT sname
FROM Supplier
```

```
SELECT sname, scity
FROM Supplier
```

What do these queries return?

```
SELECT *
FROM Supplier
```

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

SQL: WHERE

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Returns only suppliers in Washington State

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

SQL: WHERE

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Returns only suppliers in Washington State

WHERE condition can have complex predicates

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

Discussion

- Case insensitive: keywords, table/attribute names
  - SELECT, select, sElEcT, Supplier, SUPPLIER, …
- Case sensitive: strings
  - ‘WA’ different from ‘wa’
- WHERE conditions: complex predicates
  - WHERE psize>15 and pcolor=‘red’ or pcolor=‘blue’
- SQL has lots of built-in predicates; look them up!

Joins

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM
WHERE
```

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM    Supplier x, Supply y, Part z
WHERE
```

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
```

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = ‘WA’
        and z.pcolor = ‘red’;
```

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM      Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = ‘WA’
        and z.pcolor = ‘red’;
```

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

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

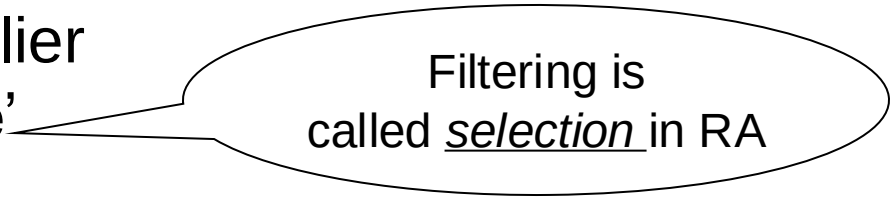
```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM      Supplier x, Supply y, Part z
WHERE     x.sno = y.sno
      and y.pno = z.pno
      and x.sstate = ‘WA’
      and z.pcolor = ‘red’;
```

What happens if we don’t include these conditions?

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

Review: Operations

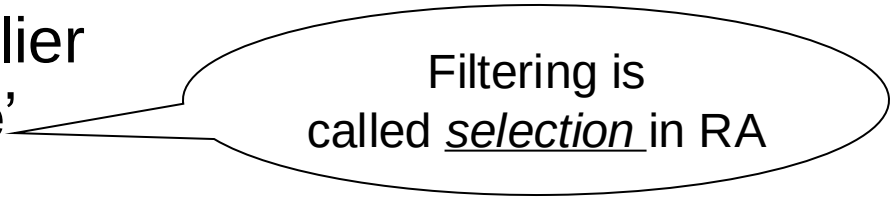
- Selection/filter: return a subset of the rows:
  - SELECT * FROM Supplier
    WHERE scity = 'Seattle'



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

Review: Operations

- Selection/filter: return a subset of the rows:
  - SELECT * FROM Supplier
    WHERE scity = 'Seattle'

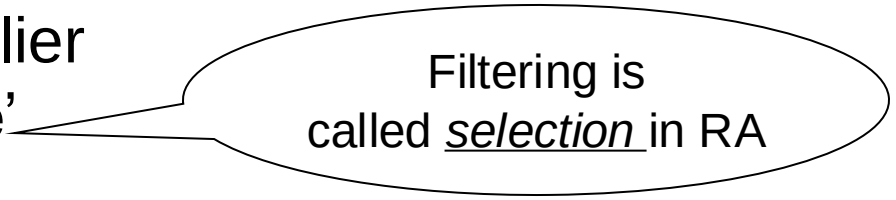


- Projection: return subset of the columns:
  - SELECT DISTINCT scity FROM Supplier;

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

Review: Operations

- Selection/filter: return a subset of the rows:
  - SELECT * FROM Supplier
    WHERE scity = 'Seattle'



- Projection: return subset of the columns:
  - SELECT DISTINCT scity FROM Supplier;

- Join: refers to combining two or more tables
  - SELECT * FROM Supplier, Supply, Part …

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

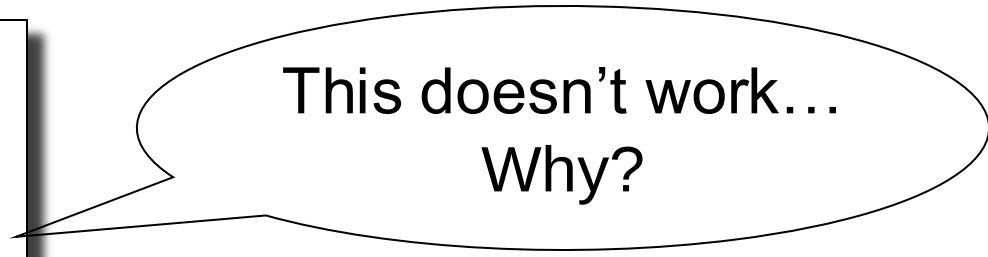
```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    x.scity = 'Seattle'
        and x.scity = 'Portland'
        and x.sno = y.sno
```

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    x.scity = 'Seattle'
         and x.scity = 'Portland'
         and x.sno = y.sno
```



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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    (x.scity = 'Seattle'
             or x.scity = 'Portland')
          and x.sno = y.sno
```

Does this work?

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    (x.scity = 'Seattle'
             or x.scity = 'Portland')
          and x.sno = y.sno
```

Does this work?

Nope!

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

Need TWO Suppliers and TWO Supplies

```
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE    x1.scity = 'Seattle'
         and x1.sno = y1.sno
         and x2.scity = 'Portland'
         and x2.sno = y2.sno
         and y1.pno = y2.pno
```

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM      Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE     x1.scity = 'Seattle'
          and x1.sno = y1.sno
          and x2.scity = 'Portland'
          and x2.sno = y2.sno
          and y1.pno = y2.pno
```

Need TWO Suppliers and TWO Supplies

one in Seattle the other in Portland

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

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE    x1.scity = 'Seattle'
         and x1.sno = y1.sno
         and x2.scity = 'Portland'
         and x2.sno = y2.sno
         and y1.pno = y2.pno
```

Need TWO Suppliers and TWO Supplies

one in Seattle the other in Portland

the SAME part

Discussion

SELECT-FROM-WHERE

- FROM clause: cartesian product
- WHERE clause: filter out
- SELECT clause: says what to return

The results can be described as a set of nested loops. Next.

Nested Loop Semantics

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
  for x_2 **in** R_2 **do**
    …..
      for x_n **in** R_n **do**
        if Conditions
          then Answer = Answer ∪ $\{(a_1,\ldots,a_k)\}$
return Answer

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

This SEMANTICS! It is NOT how the engine computes the query!

Answer = {}
for x_1 **in** R_1 **do**
  for x_2 **in** R_2 **do**
    …..
      for x_n **in** R_n **do**
        if Conditions
          then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Data Independence

- SQL engines do NOT compute the query using nested loop semantics
- Instead, they choose between a variety of execution plans

Data Independence

Precise semantics given by the nested-loop program:

```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM      Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

But the DBMS can evaluate it in many other ways!

How would you execute this query?

NULLs

NULLs in SQL

- A NULL value means missing, or unknown, or undefined, or inapplicable

Part(pno,pname,price,psize,pcolor)

NULLs in WHERE Clause

Boolean predicate:

- Atomic: Expr1 op Expr2
- Complex: AND / OR / NOT

price < 100 and (pcolor='red' or psize=2)

Part(pno,pname,price,psize,pcolor)

NULLs in WHERE Clause

Boolean predicate:

- Atomic: Expr1 op Expr2
- Complex: AND / OR / NOT

price < 100 and (pcolor='red' or psize=2)

How do we compute predicates when values are NULL?

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL

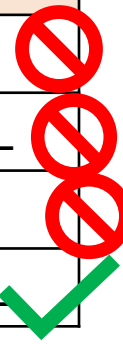
Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL



Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
-- problem: (A or not(A)) ≠ true
-- does NOT return all Products
select *
from Product
where (price <= 100) or (price > 100)
```

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A op B is
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
-- problem: (A or not(A)) ≠ true
-- does NOT return all Products
select *
from Product
where (price <= 100) or (price > 100)
```

```
-- returns ALL Products
select *
from Product
where (price <= 100) or (price > 100)
        or isNull(price)
```

Discussion

- NULLs: weird behavior
- ”A Case Against SQL”
- Try to avoid having NULLs:

```
CREATE TABLE Product
                    (… pcolor int NOT NULL…)
```

- But very useful for OUTER JOINS. Next

OUTER JOIN

Outer Joins

- A join returns only those outputs that have a tuple from each of the input tables
- Sometimes we want to include tuples from one table without a match from the other table
- Missing attributes are NULL

```
Product(name, category)
Purchase(prodName, store)
```

Outer joins

```
Product(name, category)
Purchase(prodName, store)
```

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
Product(name, category)
Purchase(prodName, store)
```

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product(name, category)
Purchase(prodName, store)

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

missing

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz
OneClick	Photo	NULL

Now it's present

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

1. Compute cross product R×S
2. Filter on C1
3. Add all R records without a match
4. Filter on C2

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
```

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
for x in R do
    if not (x in Tmp) then Tmp = Tmp ∪ {(x,NULL)}
```

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
for x in R do
    if not (x in Tmp) then Tmp = Tmp ∪ {(x,NULL)}

Answer = {}                            // apply condition C2
for (x,y) in Tmp if C2 then Answer = Answer ∪ {(x,y)}
return Answer
```

```
Product(name, category)
Purchase(prodName, store, price)
```

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause

```
Product(name, category)
Purchase(prodName, store, price)
```

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price <10

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price <10

Same as inner join!

Joins

Inner join = include just matching tuples

Left outer join = include everything from the left

Right outer join = include everything from the right

Full outer join = include everything

Aggregates

Aggregate Operator

Aggregate op: set of values to single value

Aggregates in SQL:

- sum(1, 4, 3, 4) = 1+4+3+4 = 12
- max(1, 4, 3, 4) = 4
- min(1, 4, 3, 4) = 1
- count(1, 4, 3, 4) = 4
- avg(1, 4, 3, 4) = 3

Aggregate Operator

Aggregate op: set of values to single value

Aggregates in SQL:

- sum(1, 4, 3, 4) = 1+4+3+4 = 12
- max(1, 4, 3, 4) = 4
- min(1, 4, 3, 4) = 1
- count(1, **4**, 3, **4**) = 4
- avg(1, **4**, 3, **4**) = 3

May have duplicates

Count

```
SELECT  count(*)
FROM    Part
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

Count

```
SELECT  count(*)
FROM    Part
```

4

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

Count

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

```
SELECT  count(*)
FROM    Part
```

4

```
SELECT  count(sstate)
FROM    Part
```

Count

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

```
SELECT  count(*)
FROM    Part
```

4

```
SELECT  count(sstate)
FROM    Part
```

4

Count

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Safeway	Seattle	WA
13	Walmart	Seattle	WA

```
SELECT  count(*)
FROM    Part
```

4

```
SELECT  count(sstate)
FROM    Part
```

4

```
SELECT  count(DISTINCT sstate)
FROM    Part
```

2

Semantics

```
select AGG1(A), AGG2(B), …
from…
where…
```

Semantics

```
select AGG1(A), AGG2(B), …
from…
where…
```

First compute the query w/o aggregates:

```
select A, B, …
from…
where…
```

Semantics

```
select AGG1(A), AGG2(B), …
from…
where…
```

First compute the query w/o aggregates:

```
select A, B, …
from…
where…
```

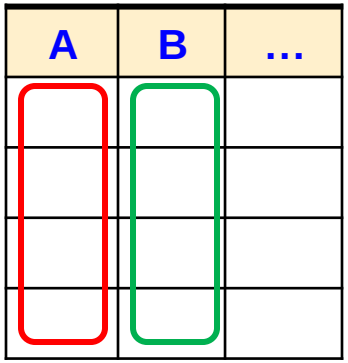
A	B	…

Semantics

```
select AGG1(A), AGG2(B), …
from…
where…
```

First compute the query w/o aggregates:

```
select A, B, …
from…
where…
```



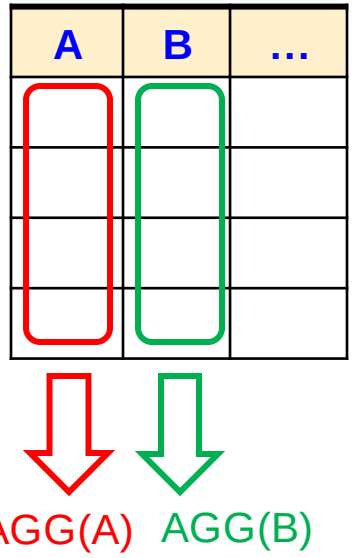
Then compute the aggregates

Semantics

```
select AGG1(A), AGG2(B), …
from…
where…
```

First compute the query w/o aggregates:

```
select A, B, …
from…
where…
```



Then compute the aggregates

Summary so Far

- SELECT-FROM-WHERE
- Joins, self-joins, outerjoins
- NULLs
- GROUP-BY, HAVING
- Combine them...

Next lecture: continue SQL