L02-L06: SQL

CS3200 Database design (sp18 s2)

1/11/2018

L01: SQL introduction

SQL overview

SQL Introduction

- SQL is a standard language for querying and manipulating data

- SQL is a very high-level programming language
  - This works because it is optimized well!

SQL stands for
Structured Query Language

- Many standards out there:
  - ANSI SQL, SQL92 (a.k.a. SQL2), SQL99 (a.k.a. SQL3), ….
  - Vendors support various subsets

NB: Probably the world's most successful **parallel** programming language (multicore?)

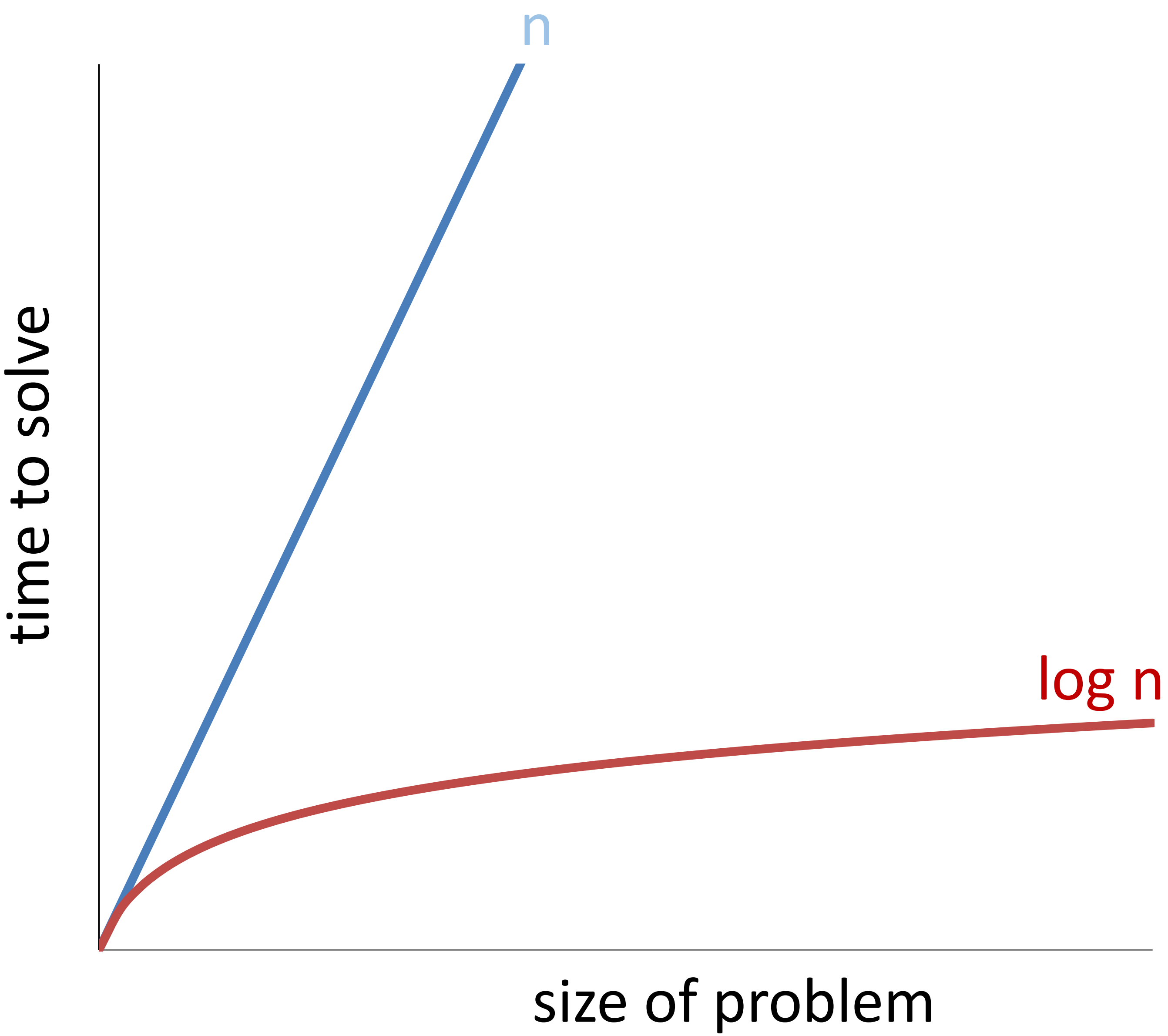
SQL Has Three Major Sub-Languages

- Data Definition Language (DDL)
  - Define a relational schema (create, alter, and drop tables; establish constraints
  - Create/alter/drop tables and their attributes
- Data Manipulation Language (DML)
  - Insert/delete/modify tuples in tables
  - Commands that maintain and query a database (our main focus!)
- Data Control Language (DCL)
  - Commands that control a database, including administering privileges and committing data

An Algorithm

- Stand up and think of the number 1
- Pair off with someone standing, add your numbers together, and adopt the sum as your new number
- One of you should sit down; the other should go back to step 2

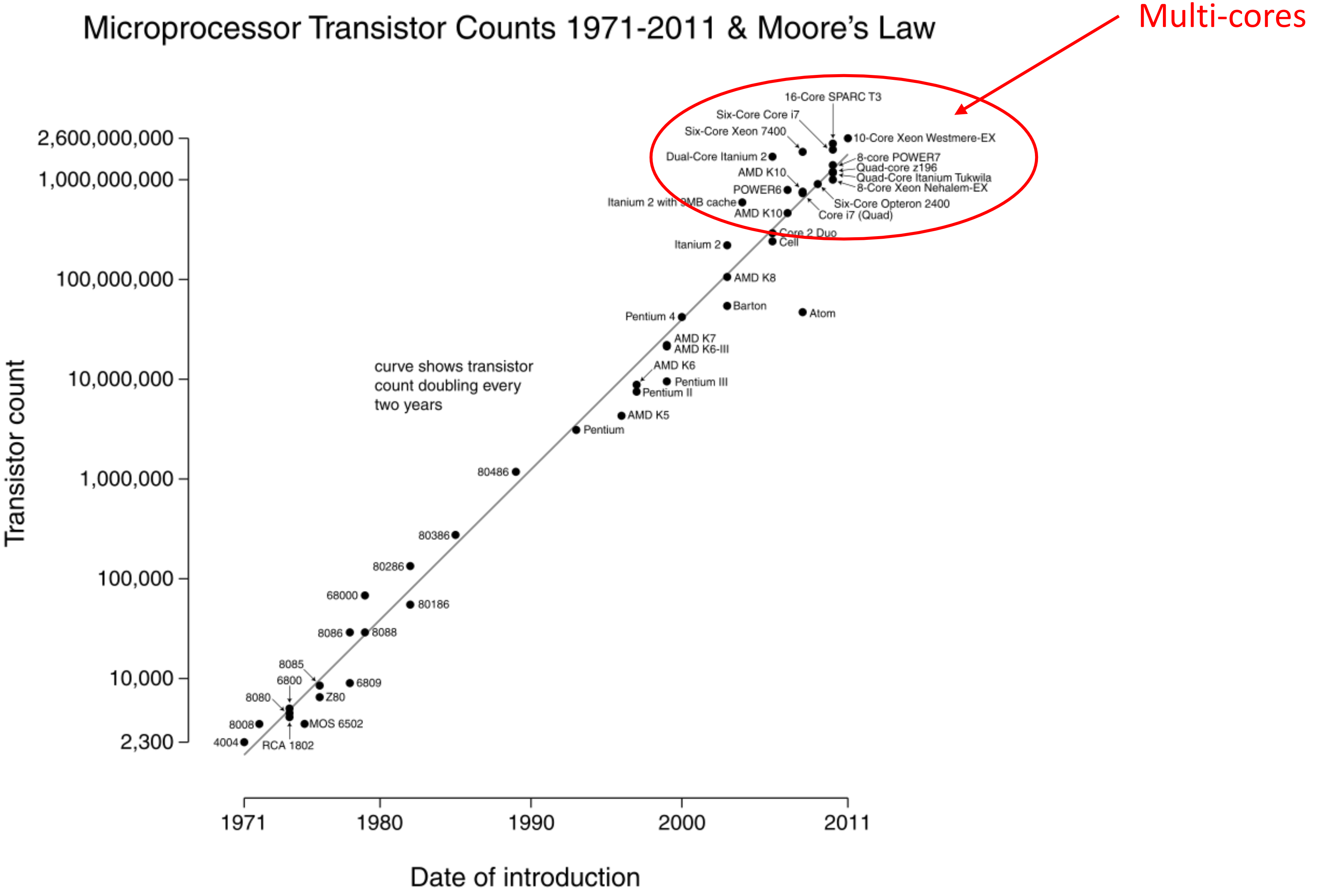
Scalability



Most spectacular these days: theoretic potential for perfect scaling!

- perfect scaling
  - given sufficient resources, performance does not degrade as the database becomes larger
- key: parallel processing
- cost: number of processors polynomial in the size of the DB
  - remember our in-class counting exercise
- all (most) relational operators highly parallelisable

Moore's law



Source: http://en.wikipedia.org/wiki/Moore's_law

What is SQL?

The Positives

- It's a language (like English, Spanish, German, …)
- There are only a few key words that you have to learn – it’s fairly simple
- It‘s major purpose is to communicate with a database and ask a database for data
- It‘s a declarative language (you define what to do)

The Challenges

- Simplicity has it‘s cost – it gets complex quickly
  - Imagine only having 2 verbs (go, put, wait) to express all you do in a lifetime
  - It's either infeasible or you have to combine a lot basic actions to construct a more complex action
    (e.g. skydiving = put parachute into backpack, put the backpack on your back, go airplane, wait until airplane is at 14k feet, go to open door, go outside airplane, …)
- Declarative programming is perceived as non-intuitive (well, decide for yourself ☺)

Different symantics between Excel and Database tables

Excel

	A	B	C	D
1	**PName**	**Price**	**Category**	**Manufacturer**
2	Gizmo	19.99	Gadgets	GizmoWorks
3	PowerGizmo	29.99	Gadgets	GizmoWorks
4	SingleTouch	149.99	Photography	Canon
5	MultiTouch	203.99	Household	Hitachi

table heading

row

column

Database[1]

Table name

TABLE Product | Search | Show All

rowid	PName	Price	Category	Manufacturer
1	Gizmo	19.99	Gadgets	GizmoWorks
2	PowerGizmo	29.99	Gadgets	GizmoWorks
3	SingleTouch	149.99	Photography	Canon
4	MultiTouch	203.99	Household	Hitachi

attribute name

tuple/ entity/ record/ row

attribute/ field/ column

[1] A Database (DB) is simply a system that holds multiple tables (like Excel has multiple sheets)

Tables in SQL

Attribute names

Table name

Key

Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Tuple / row
(Entity)

Attribute

Data Types in SQL

- Atomic types
  - Character strings: CHAR(20), VARCHAR(50)
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, …
- Record (aka tuple)
  - Every attribute must have an atomic type
- Table (aka relation)
  - A set of tuples (hence tables are flat!)

Table Schemas

- The schema of a table is the table name, its attributes, and their types:

```
Product(Pname: string, Price: float,
Category: string, Manufacturer: string)
```

- A key is an attribute whose values are unique; we underline a key

```
Product(Pname: string, Price: float,
Category: string, Manufacturer: string)
```

Basic SQL

SQL Query

- Basic form (there are many many more bells and whistles)

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Call this a **<u>SFW</u>** query.

Simple SQL Query

Our friend here shows that you can follow along in SQLite. Just install the database from the text file "300 - ..." available in our sql folder

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```

Simple SQL Query

Our friend here shows that you can follow along in SQLite. Just install the database from the text file "300 - ..." available in our sql folder

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

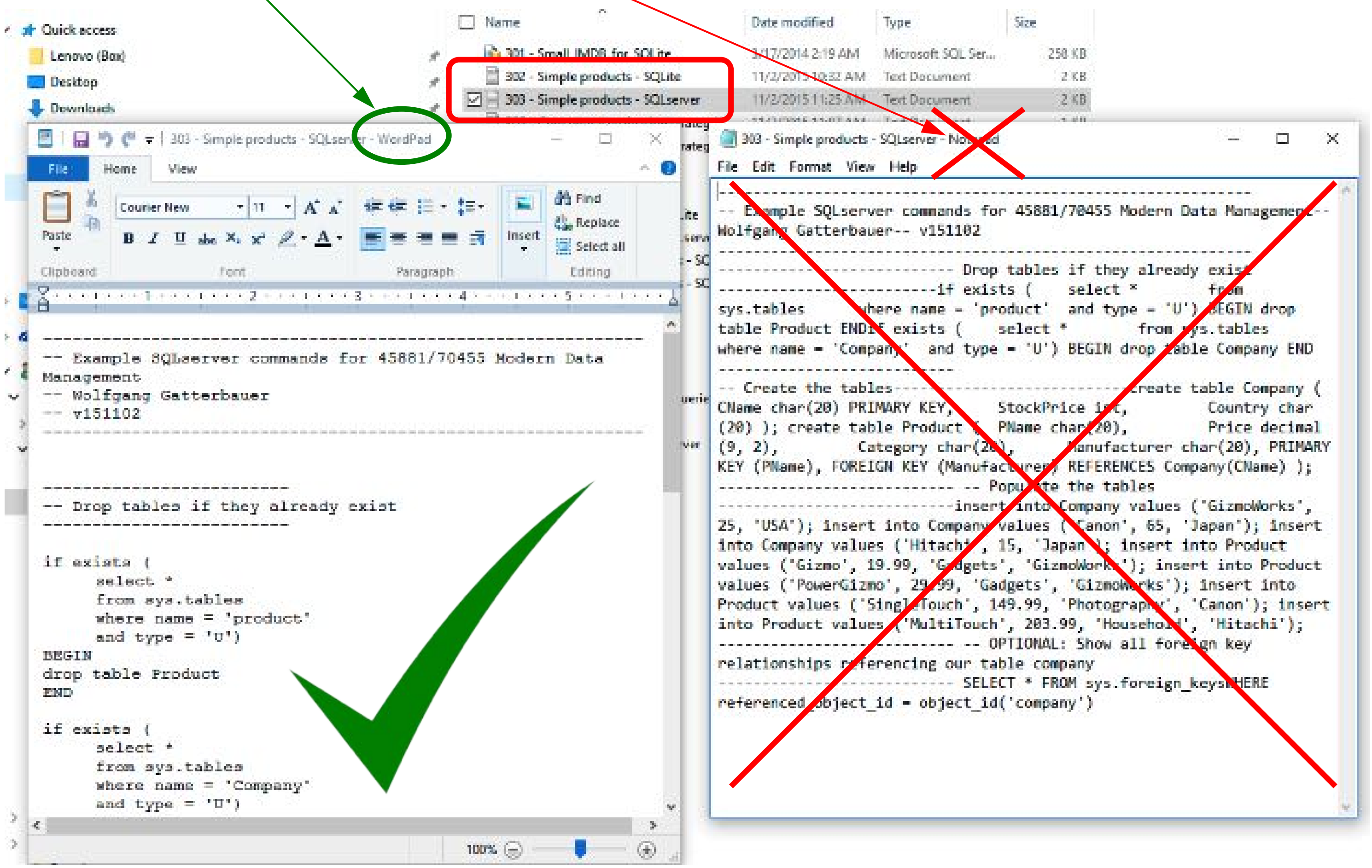
Selection

Practice with your own local databases

302

If you are using Windows:

1. *Download the appropriate text files from our repository*
2. *Open them with "**Wordpad**" (not "**Notepad**" which messes up the text!)*
3. *Paste the SQL commands into your SQLite version, and execute*



Simple SQL Query

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  pName, price, manufacturer
FROM    Product
WHERE   price > 100
```

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Selection
& Projection

Selection vs. Projection

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

One **projects** onto some attributes (columns)
-> happens in the **SELECT** clause

```
SELECT  pName, price
FROM    Product
WHERE   price > 100
```

One **selects** certain entires=tuples (rows)
-> happens in the **WHERE** clause
-> acts like a **filter**

PName	Price
SingleTouch	$149.99
MultiTouch	$203.99

A Few Details

- SQL commands are case insensitive:
  - SELECT = Select = select
  - Product = product
- But values are not:
  - Different: 'Boston', 'boston'
  - (Notice: in general, but default settings will vary from DBMS to DBMS. Just to be safe, always assume values to be case sensitive!)
- Use single quotes for constants:
  - 'abc' - yes
  - "abc" - no

Eliminating Duplicates

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
PowerGizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Set vs. Bag semantics

```
SELECT  category
FROM    Product
```

Category
Gadgets
Gadgets
Photography
Household

```
SELECT  DISTINCT category
FROM    Product
```

Category
Gadgets
Photography
Household

Ordering the Results

302

```sql
SELECT   pName, price, manufacturer
FROM     Product
WHERE    category='Gadgets'
     and price > 10
ORDER BY price, pName
```

- Ties in attribute *price* broken by attribute *pname*
- Ordering is ascending by default. Descending:

```sql
... ORDER BY price ASC, pname DESC
```

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

⇨ ?

```
SELECT   category
FROM     Product
ORDER BY pName
```

⇨ ?

```
SELECT   DISTINCT category
FROM     Product
ORDER BY pName
```

⇨ ?

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

Category
Gadgets
Household
Photography

```
SELECT   category
FROM     Product
ORDER BY pName
```

Category
Gadgets
Household
Gadgets
Photography

```
SELECT   DISTINCT category
FROM     Product
ORDER BY pName
```

Syntax error on large DBMSs (Oracle, PostgreSQL, SQL server) / unpredictable results on others(MySQL, SQLite)

"ORDER BY items must appear in the select list if SELECT DISTINCT is specified."

L02: SQL Basics

Announcements!

- Microphone
- If you still have SQLite trouble, please ask Disha or Priyal for help during lecture!
- Piazza: please be specific on Piazza with your problem, so we can help you remotely.
  - Compare: "I can't install SQLite. What should I do?" (-> come to office hours) vs. "I get error message XYZ when I do ZYX. Here is a screenshot. What should I do?"
- Piazza: please also post your lessons learned (e.g., John's comment on FF v56)
- Textbooks
- Homework #1 will be released by tonight together with PostgreSQL installation guide (you have 2 weeks)
- Python, Jupyter

Some history

Some "birth-years"

- 2004: Facebook

- 1998: Google
- 1995: Java, Ruby
- 1993: World Wide Web
- 1991: Python

- 1985: Windows

- 1974: SQL

SQL: Declarative Programming

SQL

```
select (e.salary / (e.age - 18)) as comp
from employee as e
where e.name = "Jones"
```

Declarative Language: you say what you want without having to say how to do it.

Procedural Language: you have to specify exact steps to get the result.

SQL: was not the only Attempt

SQL

```
select (e.salary / (e.age - 18)) as comp
from employee as e
where e.name = "Jones"
```

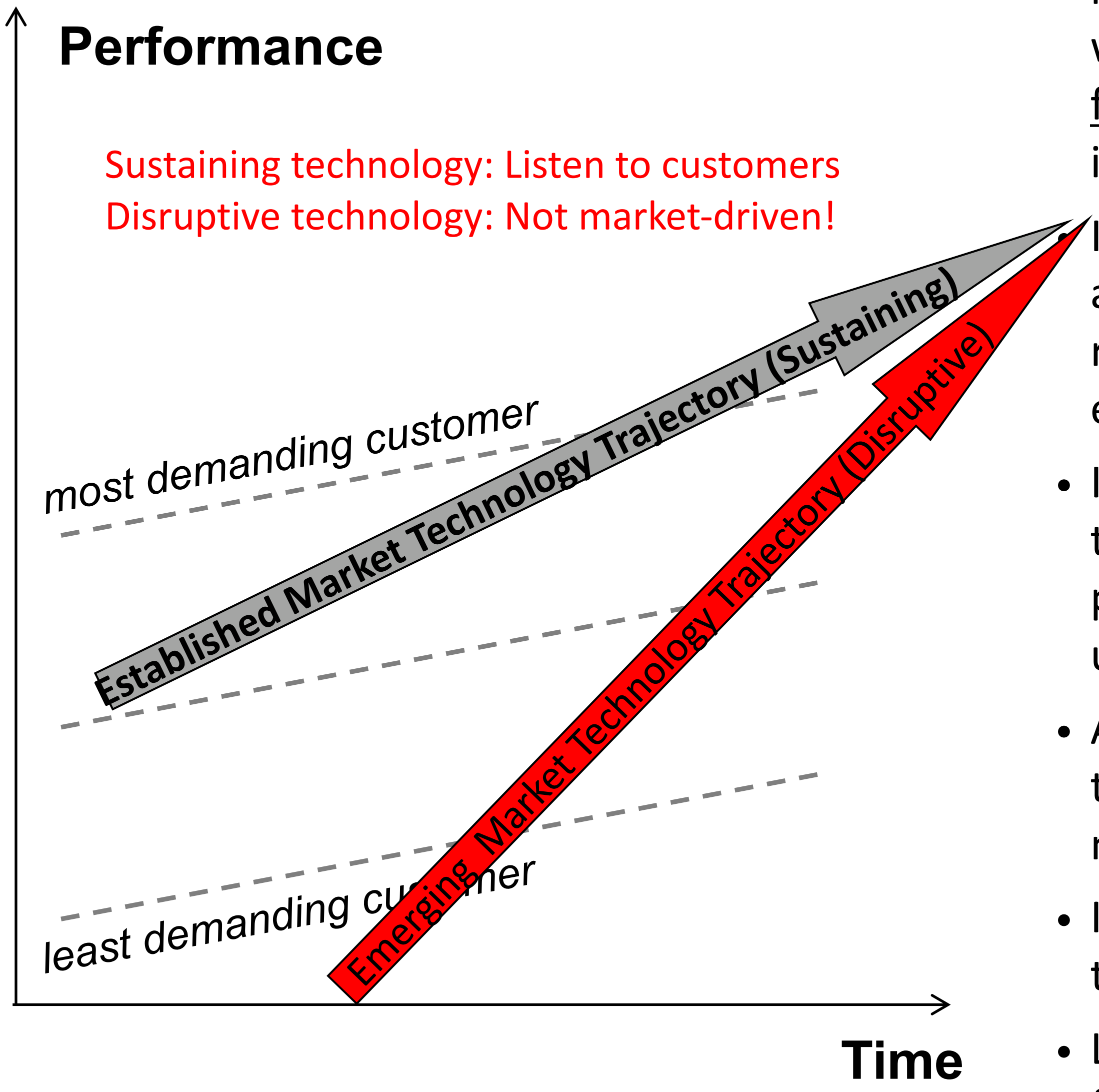
QUEL

```
range of e is employee
retrieve (comp = e.salary / (e.age - 18))
where e.name = "Jones"
```

Commercially not used anymore since ~1980

Source: http://en.wikipedia.org/wiki/QUEL_query_languages

Disruptive Innovation



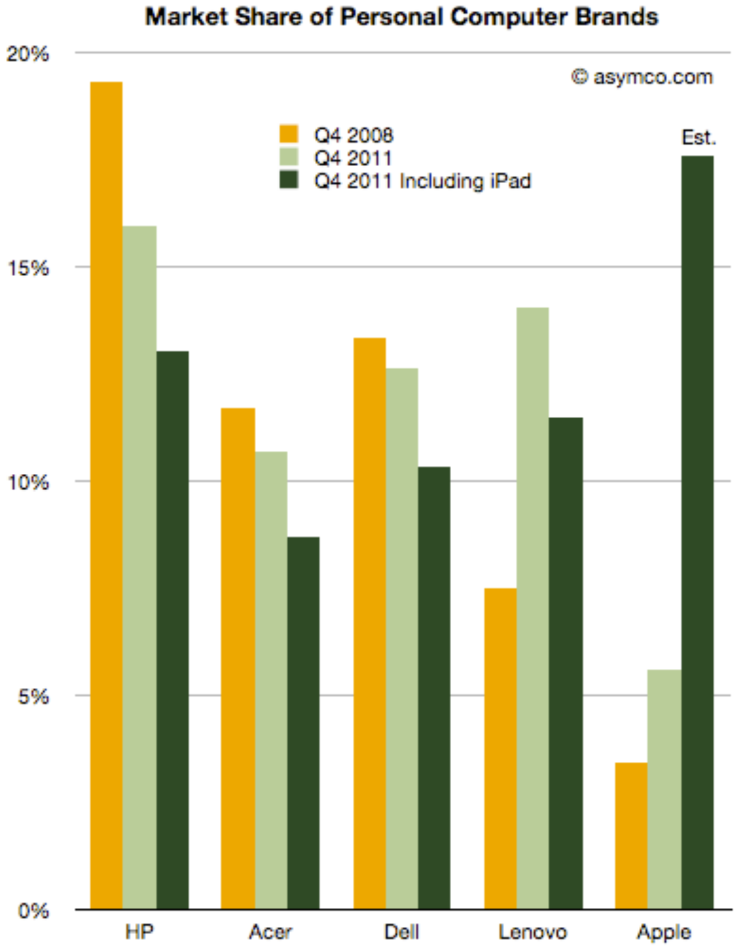
Source: After "The Innovator's Dilemma" by Clayton Christensen, 1997.

- Disruptive innovations are generally not acceptable for the mass market when they are introduced. Only the fringes of the market pick up the innovation in the first iteration
- It performs worse in one or more areas, but is typically simpler, more reliable, or more convenient than existing technologies.
- It is less profitable than existing technologies. Leading firms' most profitable customers generally can't use it and don't want it.
- As the innovator continues to refine their product the utility value to the market increases
- Its performance trajectory is steeper than that of existing technologies.
- Large organizations are fundamentally incapable of successfully bringing it to market.

iPhone: Disruptive Innovation or not?

1: "Business Phones"
Microsoft in 2007

2: Laptops



Source: http://www.youtube.com/watch?v=eywi0h_Y5_U , http://www.asymco.com/2012/01/16/apple-is-the-top-personal-computer-vendor/

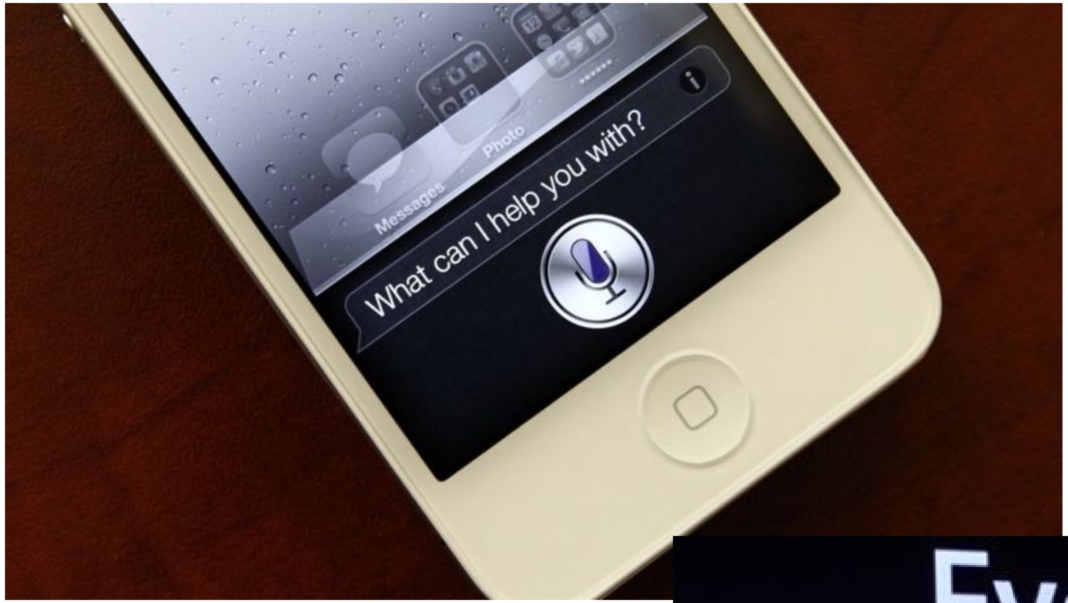
What keyboard without keys can do...

In Feb 2016, SwiftKey was purchased by Microsoft, for 250 M$

Sources: https://en.wikipedia.org/wiki/Swype, https://en.wikipedia.org/wiki/SwiftKey

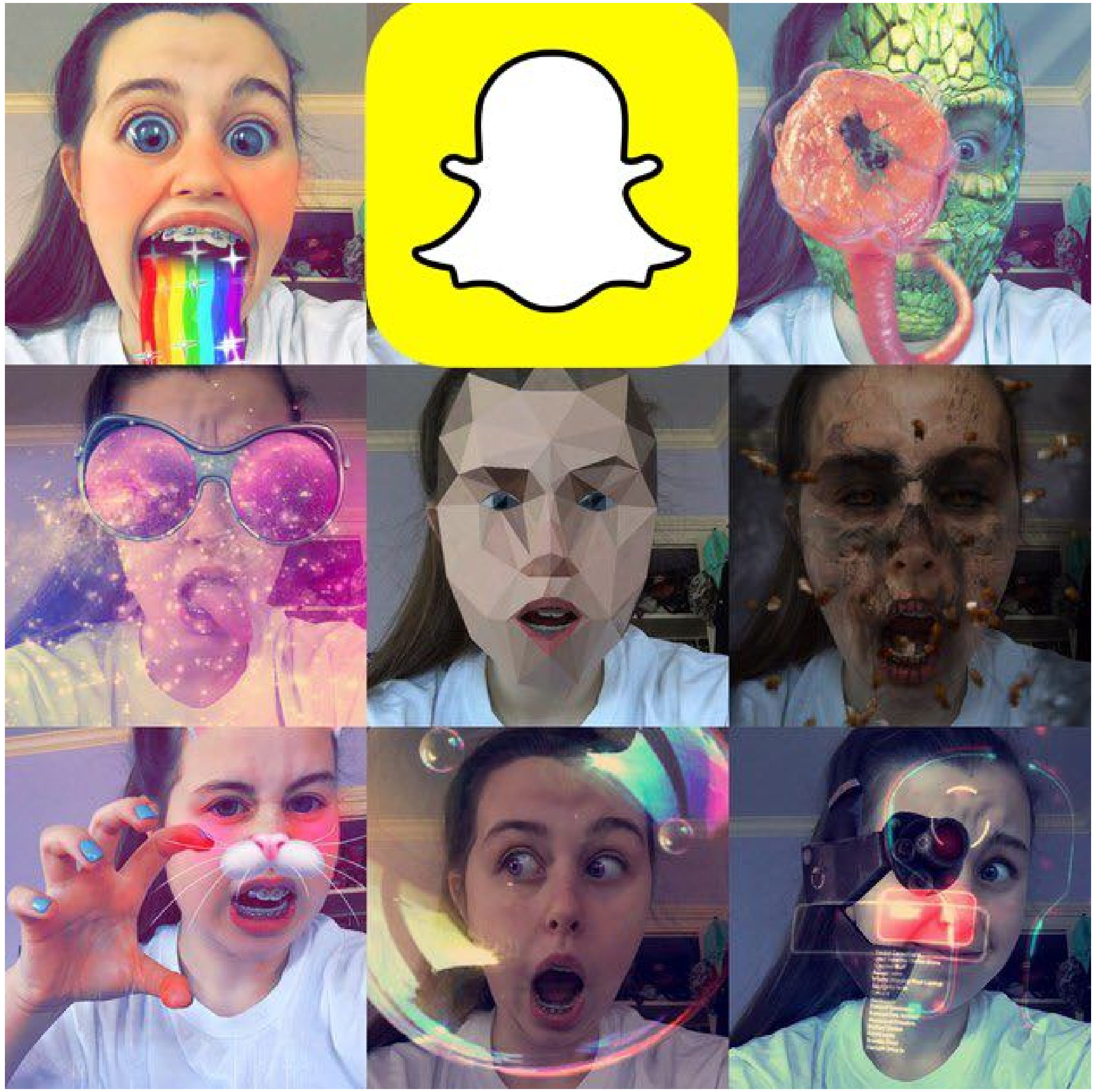
The keyboard of the future?

Sources: http://www.wired.com/2014/06/siri_ai/


What can I help you with?
Messages
Photo


Eyes Free
MODE
FM/AM
engadget

Keyboards and emails?

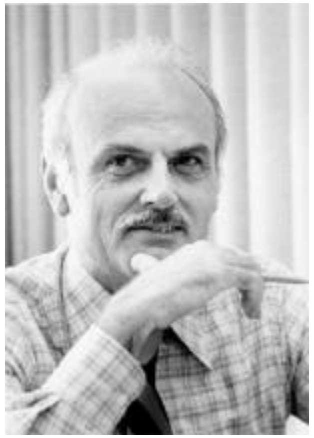


Sources: http://www.buzzfeed.com/benrosen/how-to-snapchat-like-the-teens

What is this? (1975)

Source: http://pluggedin.kodak.com/pluggedin/post/?id=687843

SQL: some history



- Dr. Edgar Codd (IBM)
  - CACM June 1970: "A Relational Model of Data for Large Shared Data Banks" http://seas.upenn.edu/~zives/03f/cis550/codd.pdf
- Standardized
  - 1986 by ANSI: SQL1
  - 1992: Revised: SQL2
    - Approx 580 page document describing syntax and semantics
  - Revised: 1999, 2003, 2008, ...
- Players
  - Microsoft, IBM, Relational Software (Oracle), ...

- Every vendor has a slightly different version of SQL
- But the main commands are standardized

Codd's (disruptive ?) innovation

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

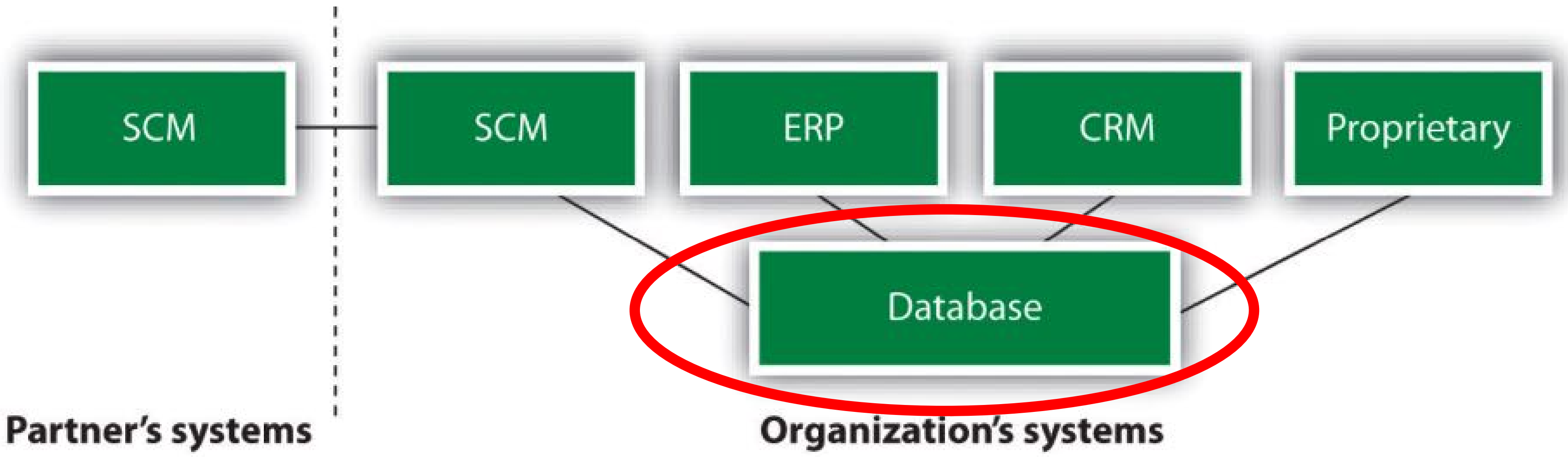
1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

Source: http://seas.upenn.edu/~zives/03f/cis550/codd.pdf

SQL and the relational model as standard



Source: "Information Systems: A Manager's Guide to Harnessing Technology (book v1.4)," p.185, Gallaugher, 2012.

Databases we are using

Client/Server Architecture

- There is a single server that stores the database (called DBMS or RDBMS):
  - Usually a beefy system, e.g. IISQLSRV
  - But can be your own desktop…
  - … or a huge cluster running a parallel dbms (later assign.)
- Many clients run apps and connect to DBMS
  - E.g. Microsoft’s Management Studio
  - More realistically some Java, Python, or C++ program
- Clients “talk” to server using some protocol

DBMSs we will work with

- SQLlite
  - most widely deployed database engine
  - in particular with embedded systems, browsers, etc., e.g., Microsoft's Windows Phone 8, Apple's iOS, Skype, Firefox
- PostgreSQL
  - popular and powerful open source database (Microsoft)

SQLite vs. PostgreSQL

SQLlite

- open source & cross-platform
- easy to install
- has no server ("embedded")
- ideal for single-user application; has limitations when it comes to concurrency / simultaneous transactions (one writer at a time)
- does not allow partitioning; everything is stored in one single file
- extra functions are written in C/C++

PostgreSQL

- commercial (Microsoft)
- takes a bit longer to install
- uses a server
- ideal for shared repository; allows concurrency (many simultaneous transactions), locking and fine-grained access control
- scales to >GB easily; allows partitioning (distributing) the data across several files / nodes
- supports user-defined functions

SQL overview

Key constraints

A key is a **minimal subset of attributes** that acts as a unique identifier for tuples in a relation

- A key is an implicit constraint on which tuples can be in the relation
  - i.e. if two tuples agree on the values of the key, then they must be the same tuple!

```
Students(sid:string, name:string, gpa: float)
```

1. Which would you select as a key?
2. Is a key always guaranteed to exist?
3. Can we have more than one key?

NULL and NOT NULL

- To say "don't know the value" we use NULL
  - NULL has (sometimes painful) semantics, more detail later

```
Students(sid:string, name:string, gpa: float)
```

sid	name	gpa
123	Bob	3.9
143	Jim	NULL

Say, Jim just enrolled in his first class.

In SQL, we may constrain a column to be NOT NULL, e.g., "name" in this table

General Constraints

- We can actually specify arbitrary assertions
  - E.g. “There cannot be 25 people in the DB class”

- In practice, we don’t specify many such constraints. Why?
  - Performance!

Whenever we do something ugly (or avoid doing something convenient) it’s for the sake of performance

Summary of Schema Information

- Schema and Constraints are how databases understand the semantics (meaning) of data

- They are also useful for optimization

- SQL supports general constraints:
  - Keys and foreign keys are most important
  - We'll give you a chance to write the others

Basic SQL

Simple SQL Query


302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  pName
FROM    Product
WHERE   manufacturer in ('Canon','Hitachi')
```

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  pName
FROM    Product
WHERE   manufacturer in ('Canon','Hitachi')
```

PName
SingleTouch
MultiTouch

Selection
& Projection

WHERE ... IN (...) cp. to Excel

JFYI

	A	B	C	D	E	F
1	*Source: Walkenbach, Excel 2010 Formulas, p396, 2010.*					
2						
3	Is this name contained in the range?					
4	Name:	Barbara	TRUE			
5		Betty	1			
6		Betty	TRUE			
7						
8	**NameRange**					
9						
10	Barbara	Karen	Nancy			
11	Betty	Kimberly	Patricia			
12	Carol	Laura	Ruth			
13	Deborah	Linda	Sandra			
14	Donna	Lisa	Sarah			
15	Dorothy	Margaret	Sharon			
16	Elizabeth	Maria	Susan			
17	Helen	Mary				
18	Jennifer	Michelle				
19						
20						

Assume that there is a range defined for A10:C18 called "NameRange"

WHERE ... IN (...) cp. to Excel

JFYI

	A	B	C	D	E	F
1	*Source: Walkenbach, Excel 2010 Formulas, p396, 2010.*					
2						
3	Is this name contained in the range?					
4	Name:	Barbara	TRUE	**{=OR(NameRange=B4)}**		
5		Betty	1	**=COUNTIF(NameRange,B5)**		
6		Betty	TRUE	**=COUNTIF(NameRange,B6)>0**		
7						
8	**NameRange**					
9						
10	Barbara	Karen	Nancy			
11	Betty	Kimberly	Patricia			
12	Carol	Laura	Ruth			
13	Deborah	Linda	Sandra			
14	Donna	Lisa	Sarah			
15	Dorothy	Margaret	Sharon			
16	Elizabeth	Maria	Susan			
17	Helen	Mary				
18	Jennifer	Michelle				
19						
20						

Assume that there is a range defined for A10:C18 called "NameRange"

LIKE

LIKE: Simple String Pattern Matching

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  pName
FROM    Product
WHERE   pname LIKE '%izmo'
```

PName
Gizmo
Powergizmo

% is a wildcard for any sequence of zero or more characters.

More details: http://www.techonthenet.com/sql_server/like.php

LIKE: Simple String Pattern Matching

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  pName
FROM    Product
WHERE   pname LIKE '_izmo'
```

PName
Gizmo

_ is a wildcard for exactly one character.

More details: http://www.techonthenet.com/sql_server/like.php

Table selection using comparison predicates

	Numbers		**Text / Strings**	
Simple comparators	=	equal to	=	equal to (exact string)
	<	smaler than		
	<=	smaller or equal to		
	>	greater than		
	>=	greater or equal to		
	<>	unequal to		
Complex comparators	BETWEEN value1 AND value2		LIKE	equal to (pattern)
	any values within the range		‘S%’	string starting with S
			‘%S’	string ending with S
			‘%S%’	string containing an S
			‘S_S’	string with S at both ends and any character in the middle
Comparators that **work across types**	IN (value1, value2, …)	any values within the given set		
	IS NULL	has no value		
	IS NOT NULL	has a value		

Note: Combinations of multiple predicates with **AND** & **OR** (use brackets)

Date functions

Arithmetic expressions

```
SELECT  3+2
```

(no column name)
5

```
SELECT  (2*3 + 4*5) as name
```

name
26

Date functions are database-specific

JFYI

Worker

Name	Birthdate
Max	1980-01-01
Fred	1979-02-01
Susan	1990-01-31
Tilda	1988-01-01

We can specify the output column names

```
SELECT  date('now')-date(birthdate) as age
FROM    Worker
```

age
33
34
23
25

This is here SQLite semantics
Date functions are different between different databases.
In real life, you may need to look up how your DB handles date functions:
http://www.sqlite.org/lang_datefunc.html

Joins

Keys and Foreign Keys


302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is a foreign key vs. a key here?

Keys and Foreign Keys

302

Key → **Product** (PName)

Foreign key → Manufacturer

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Key → **Company** (CName)

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is a foreign key vs. a key here?

Referential Integrity

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the fields of a relation is a unique identifier for a tuple.

Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');

violates Key constraint

Gizmo	$14.99	Gadgets	Hitachi

Foreign key: must match field in a relational table that matches a candidate key of another table

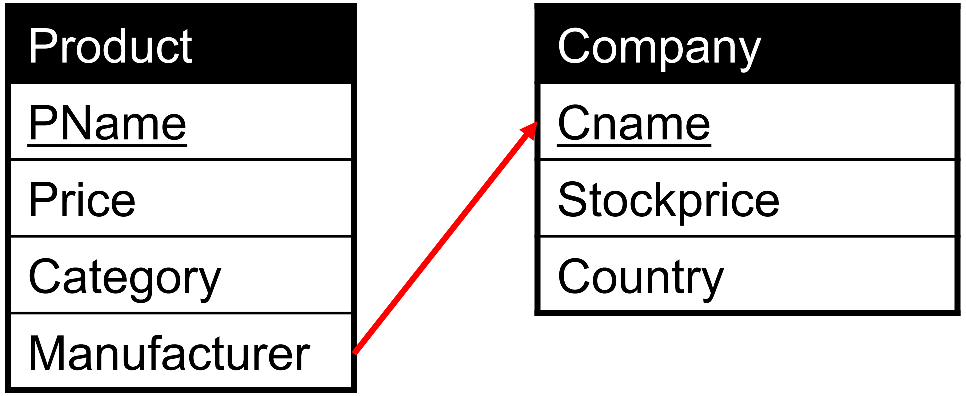
violate Foreign Key constraint

Insert into Product values ('SuperTouch', 249.99, 'Computer', 'NewCom');

SuperTouch	$249.99	Computer	NewCom

Delete from Company where CName = 'Canon';

(Relational Database) Schema



"Schema": describes the structure of data in terms of the relational data model.

A schema includes tables, columns, PKs, FKs, and other constraints

Product(pname, price, category, manufacturer)
Company(cname, stockprice, country)

Product.manufacturer is FK to Company

Joins

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all products under $200 manufactured in Japan; return their names and prices!

Joins

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all products under $200 manufactured in Japan; return their names and prices!

```
SELECT  pName, price
FROM    Product, Company
WHERE   manufacturer=cName
    and country='Japan'
    and price <= 200
```

Join b/w Product and Company

PName	Price
SingleTouch	$149.99

Quiz

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What does the query below return?

```
SELECT  pName, StockPrice
FROM    Product, Company
WHERE   manufacturer=cName
    and country = 'USA'
```

Quiz

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)


302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What does the query below return?

```
SELECT  pName, StockPrice
FROM    Product, Company
WHERE   manufacturer=cName
    and country = 'USA'
```

PName	StockPrice
Gizmo	25
Powergizmo	25

Table Alias (Tuple Variables)

312

Person (<u>pName</u>, address, works_for)
University (<u>uName</u>, address)

```
SELECT   DISTINCT pName, address
FROM     Person, University
WHERE    works_for = uName
```

Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

```
SELECT   DISTINCT pName, address
FROM     Person, University
WHERE    works_for = uName
```

which address?
Error!

```
SELECT   DISTINCT Person.pName, University.address
FROM     Person, University
WHERE    Person.works_for = University.uName
```

```
SELECT   DISTINCT X.pName, Y.address
FROM     Person as X, University Y
WHERE    X.works_for = Y.uName
```

Note that "as" is optional !!

L03: SQL

Why I don't post slides *before* lecture

From the Preamble of one of the best physics books there is: „How to read this book“

The best way to use this book is NOT to simply read it or study it, but to read a question and STOP. Even close the book. Even put it away and THINK about the question. Only after you have formed a reasoned opinion should you read the solution. Why torture yourself thinking? Why jog? Why do push-ups?

If you are given a hammer with which to drive nails at the age of three you may think to yourself, "OK, nice." But if you are given a hard rock with which to drive nails at the age of three, and at the age of four you are given a hammer, you think to yourself, "What a marvelous invention!" You see, you can't really appreciate the solution until you first appreciate the problem.

...

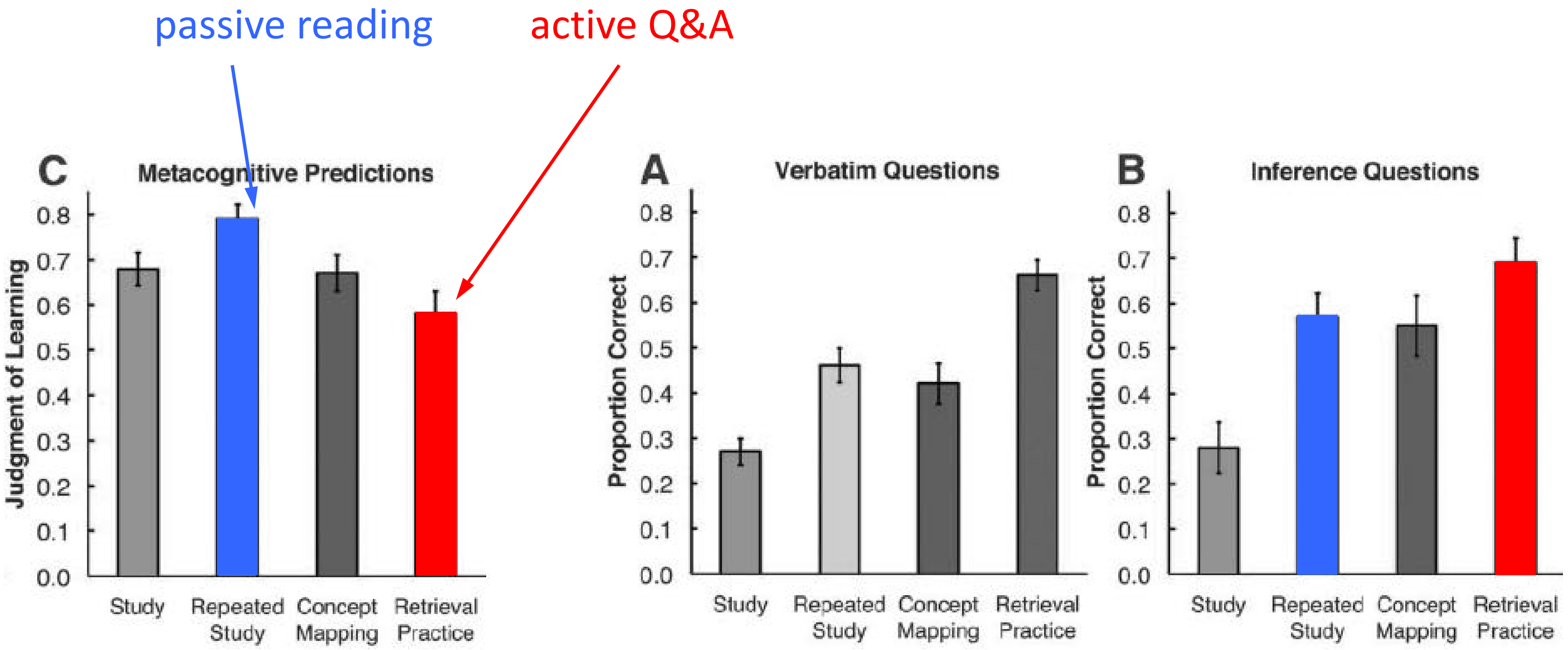
...

Let this book, then, be your guide to mental push-ups. Think carefully about the questions and their answers *before* you read the answers offered by the author. **You will find many answers don't turn out as you first expect. Does this mean you have no sense for physics? Not at all. Most questions were deliberately chosen to illustrate those aspects of physics which seem contrary to casual surmise. Revising ideas, even in the privacy of your own mind, is not painless work.** But in doing so you will revisit some of the problems that haunted the minds of Archimedes, Galileo, Newton, Maxwell, and Einstein.* The physics you cover here in hours took them centuries to master. Your hours of thinking will be a rewarding experience. Enjoy!

Lewis Epstein

Source: "Thinking Physics: Understanding Practical Reality", Lewis Carroll Epstein, 1979-2009.
http://www.amazon.com/Thinking-Physics-Understandable-Practical-Reality/dp/0935218084

Studying material: "Under which study condition do you learn better?"

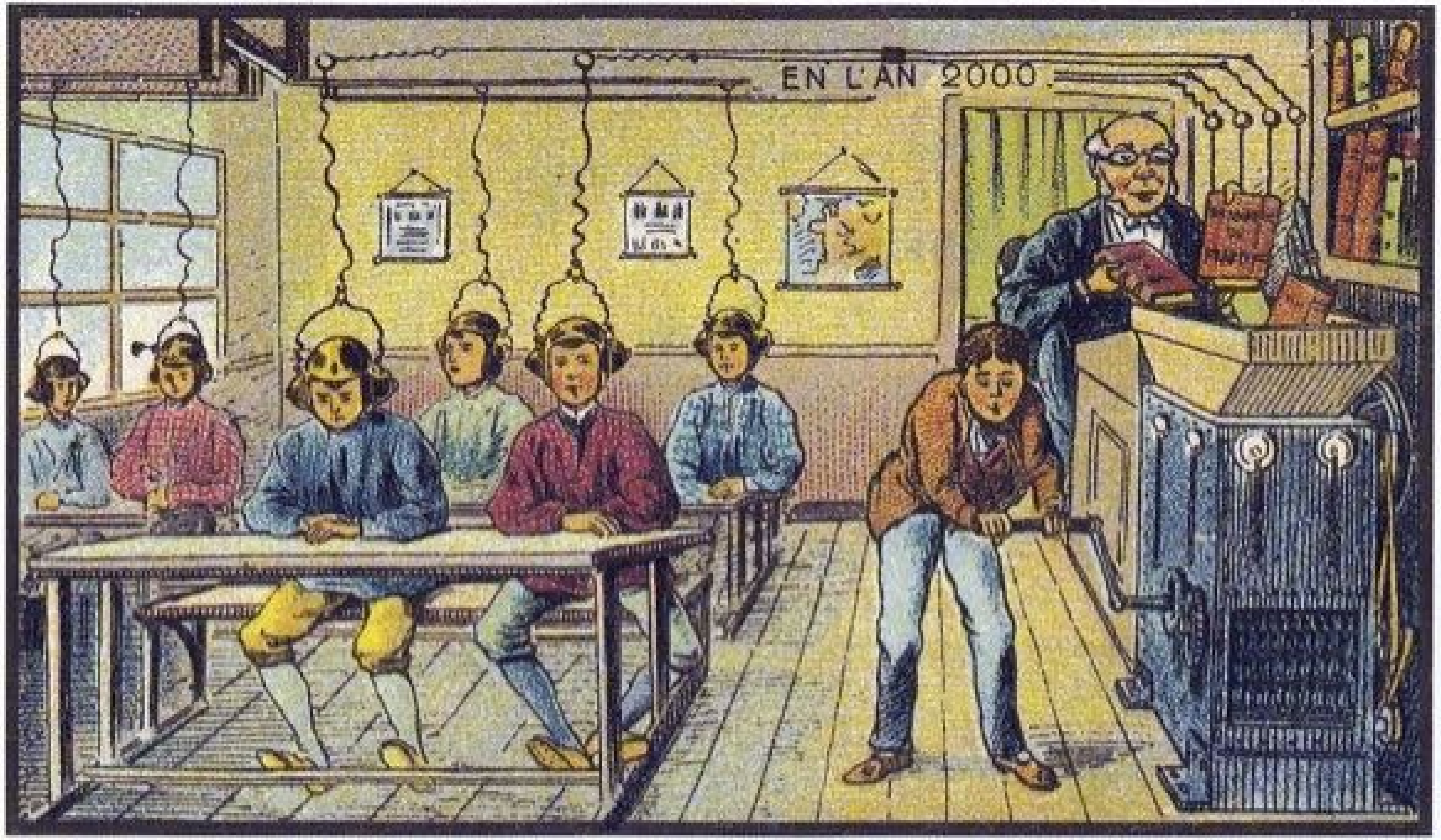


Judged performance
(=what people think)

Actual performance
(=what is actually working)

Source: Karpicke & Blunt, "Retrieval Practice Produces More Learning than Elaborative Studying with Concept Mapping," Science, 2011.

The year 2000 imagined in 1900



At School

Source: http://5.mshcdn.com/wp-content/gallery/the-year-2000-as-imagined-in-1900/future.jpg

Announcements!

- Textbooks (v2): link to Amazon international ed
- Python, Jupyter
- Keep up the great class interactions ☺
- Microphone
- Continue giving feedback
- Talk announcement today at 3pm

DISTINGUISHED SPEAKER: RETHINKING QUERY EXECUTION ON BIG DATA

JANUARY 18 3:00 PM - 4:30 PM EST

Title: Rethinking Query Execution on Big Data

Speaker: Dan Suciu, Professor of Computer Science at the University of Washington

Location: Northeastern University, 45 Forsyth St., Cargill Hall, Lower Level, Room #97, Boston, Massachusetts 02115

Abstract
Database engines today use the same approach to evaluate a query as they did forty years ago: convert the query into a query plan, then execute each operator individually, e.g. a join, followed by another join, followed by duplicate elimination. It turns out that converting a query into binary joins is theoretically suboptimal, and this can lead to poor performance over very large datasets. The theoretical database research community has studied a new query evaluation paradigm, which in some cases leads to provably optimal algorithms. In this talk I will give a brief survey of this new paradigm: I will review the AGM bound on the query size (Atserias, Grohe and Marx), the worst-case optimal "generic join" algorithm for full conjunctive queries (Ngo, Re, and Rudra), and our new algorithm for aggregate queries, called PANDA, which matches the best known running times for certain graph problems.

Table Alias (Tuple Variables)

312

Person (pName, address, works_for)
University (uName, address)

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

which address?
Error!

```
SELECT  DISTINCT Person.pName, University.address
FROM    Person, University
WHERE   Person.works_for = University.uName
```

```
SELECT  DISTINCT X.pName, Y.address
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

Note that "as" is optional !!

Column Alias (rename attributes)

Person (pName, address, works_for)
University (uName, address)

```
SELECT   DISTINCT X.pName as name, Y.address adr
FROM     Person as X, University Y
WHERE    X.works_for = Y.uName
```

```
SELECT   DISTINCT X.pName, Y.address
FROM     Person as X, University Y
WHERE    X.works_for = Y.uName
```

Quiz 2

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all US companies that manufacture products in the 'Gadgets' category!

```
SELECT  cName
FROM
WHERE
```

Quiz 2

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all US companies that manufacture products in the 'Gadgets' category!

```
SELECT  cName
FROM    Product P, Company
WHERE   country = 'USA'
    and P.category = 'Gadgets'
    and P.manufacturer = cName
```

⇨

Cname
GizmoWorks
GizmoWorks

Quiz 2

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all US companies that manufacture products in the 'Gadgets' category!

```
SELECT  DISTINCT cName
FROM    Product P, Company
WHERE   country = 'USA'
    and P.category = 'Gadgets'
    and P.manufacturer = cName
```

Cname
GizmoWorks

Quiz 3

302

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM
WHERE
```

Quiz 3

302

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT   DISTINCT cName
FROM     Product as P, Company
WHERE    country = 'USA'
    and  P.price < 20
    and  P.price > 25
    and  P.manufacturer = cName
```

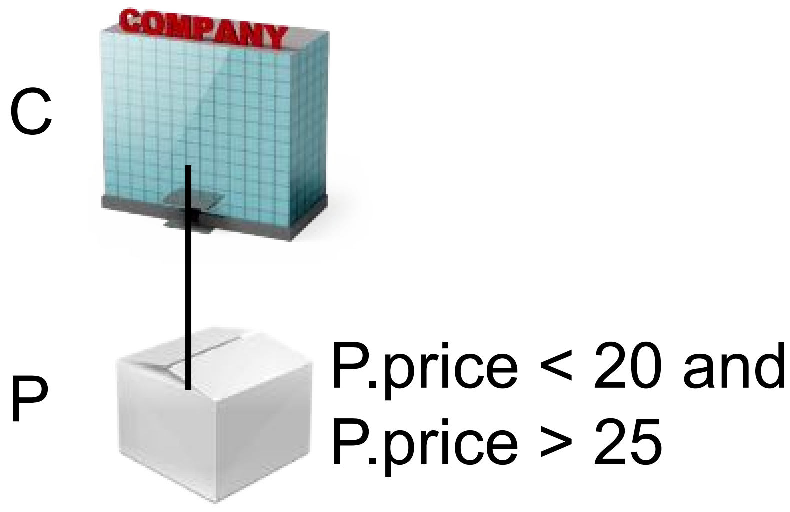
Wrong! Gives empty result: There is no product with price <20 and >25

Quiz 3

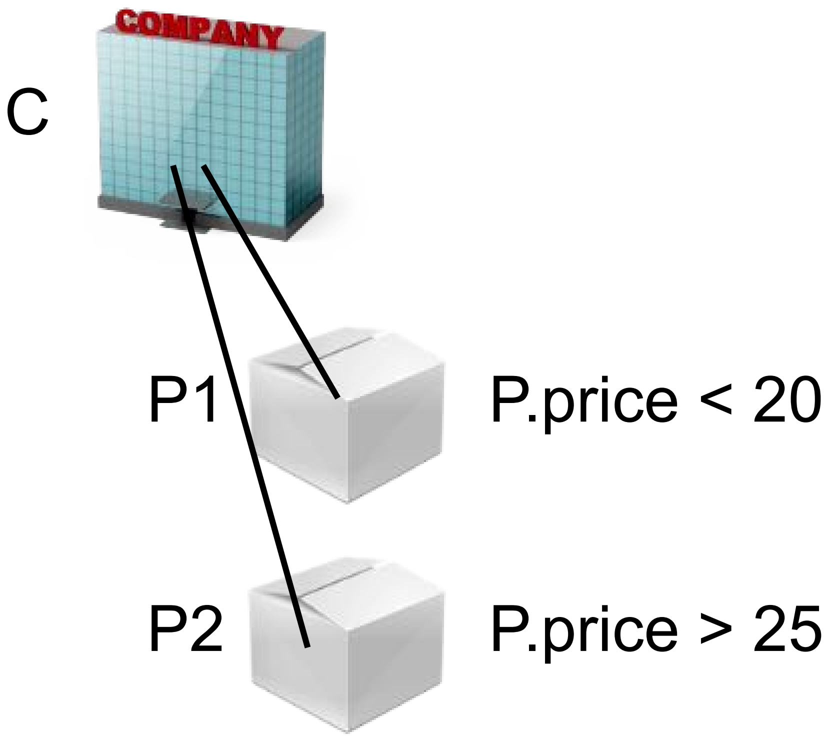
302

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.



not possible!
-> Empty result



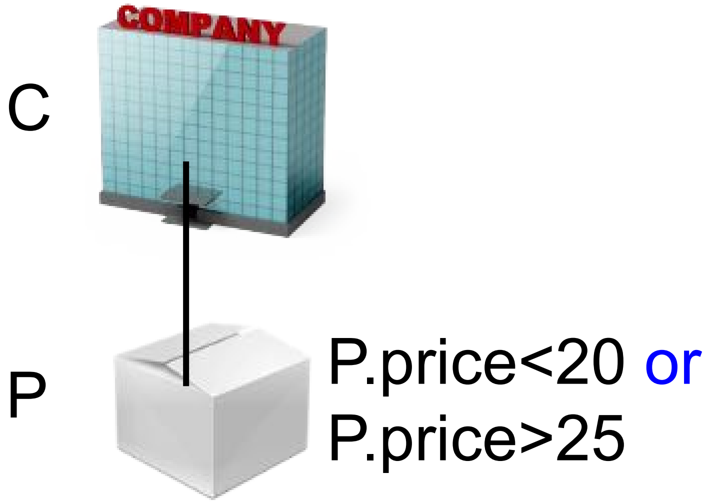
Quiz 3

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```sql
SELECT   DISTINCT cName
FROM     Product as P, Company
WHERE    country = 'USA'
    and  (P.price < 20
    or   P.price > 25)
    and  P.manufacturer = cName
```

Returns companies with single product w/price (<20 or >25)



Quiz 3


302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Quiz 3

302

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
…	…	…	…

P2

PName	Price	Category	Manufacturer
…	…	…	…
Powergizmo	$29.99	Gadgets	GizmoWorks

Company

CName	StockPrice	Country
GizmoWorks	25	USA
…	…	…

```
SELECT  DISTINCT cName
FROM    Product ~~as~~ P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Cname
GizmoWorks

Meaning (Semantics) of conjunctive SQL Queries

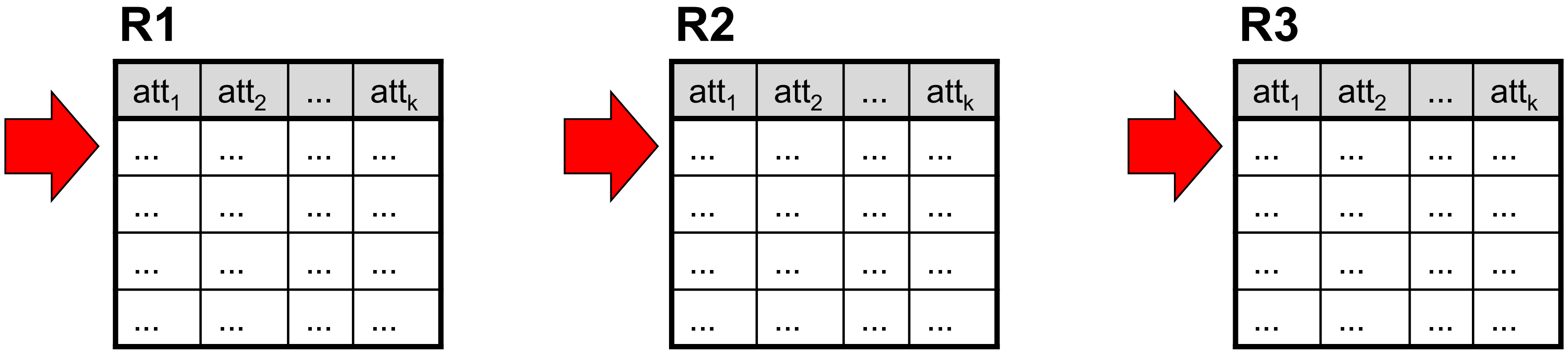
```
SELECT  a1, a2, …, ak
FROM    R1 as x1, R2 as x2, …, Rn as xn
WHERE   Conditions
```

$$
\begin{aligned}
&\texttt{SELECT}\quad a_1, a_2, \ldots, a_k\\
&\texttt{FROM}\quad R_1 \text{ as } x_1, R_2 \text{ as } x_2, \ldots, R_n \text{ as } x_n\\
&\texttt{WHERE}\quad \text{Conditions}
\end{aligned}
$$

Conceptual evaluation strategy (nested for loops):

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

Meaning (Semantics) of conjunctive SQL Queries



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

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - FROM: Compute the cross-product of relation-list.
  - WHERE: Discard resulting tuples if they fail qualifications.
  - SELECT: Delete attributes that are not in target-list.
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.

Inner Joins

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
WHERE   E.DepartmentID = D. DepartmentID
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Jones	33	33	Engineering
Smith	34	34	Clerical
Steinberg	33	33	Engineering
Rafferty	31	31	Sales

Source: http://en.wikipedia.org/wiki/Join_(SQL)#Cross_join

Cross Joins: usually not what you want ☹

344

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
~~WHERE   E.DepartmentID = D. DepartmentID~~
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Rafferty	31	31	Sales
Jones	33	31	Sales
Steinberg	33	31	Sales
Smith	34	31	Sales
Robinson	34	31	Sales
Rafferty	31	33	Engineering
Jones	33	33	Engineering
Steinberg	33	33	Engineering
Smith	34	33	Engineering
Robinson	34	33	Engineering
Rafferty	31	34	Clerical
Jones	33	34	Clerical
Steinberg	33	34	Clerical
Smith	34	34	Clerical
Robinson	34	34	Clerical
Rafferty	31	35	Marketing
Jones	33	35	Marketing
Steinberg	33	35	Marketing
Smith	34	35	Marketing
Robinson	34	35	Marketing

Definitions (for job interviews?)

- An <u>equi-join</u> is a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table

- A <u>natural join</u> is an equi-join in which one of the duplicate columns is eliminated in the result table

- A <u>cross join</u> returns the Cartesian product of rows from tables in the join
  - (i.e. it will produce rows which combine each row from the first table with each row from the second table, that's usually *not* what you want)

Definitions (for job interviews?)

Equi-join

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Jones	33	33	Engineering
Smith	34	34	Clerical
Steinberg	33	33	Engineering
Rafferty	31	31	Sales

Natural join

E.LastName	DepartmentID	D.DepartmentName
Robinson	34	Clerical
Jones	33	Engineering
Smith	34	Clerical
Steinberg	33	Engineering
Rafferty	31	Sales

Cross join

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Rafferty	31	31	Sales
Jones	33	31	Sales
Steinberg	33	31	Sales
Smith	34	31	Sales
Robinson	34	31	Sales
Rafferty	31	33	Engineering
...	...	...	...

Alternative JOIN Syntax

344

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT   *
FROM     Employee E, Department D
WHERE    E.DepartmentID = D. DepartmentID
AND      E.DepartmentID = 34
```

```
SELECT   *
FROM     Employee E JOIN Department D
      ON E.DepartmentID = D. DepartmentID
WHERE    E.DepartmentID = 34
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Smith	34	34	Clerical

NATURAL JOIN Syntax


344

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT   *
FROM     Employee E, Department D
WHERE    E.DepartmentID = D. DepartmentID
AND      E.DepartmentID = 34
```

```
SELECT   *
FROM     Employee E NATURAL JOIN Department D

WHERE    E.DepartmentID = 34
```

Syntax is not supported by all DBMS's

LastName	DepartmentID	DepartmentName
Robinson	34	Clerical
Smith	34	Clerical

Using the Formal Semantics

305

R(a), S(a), T(a)

What do these queries compute?

R

a
1
2

S

a
1

T

a
2

```
SELECT  DISTINCT R.a
FROM    R, S
WHERE   R.a=S.a
```

⇒

a
1

Returns R ∩ S
(intersection)

```
SELECT  DISTINCT R.a
FROM    R, S, T
WHERE   R.a=S.a
     or R.a=T.a
```

⇒

a
1
2

Returns R ∩ (S ∪ T)
if S ≠ ∅ and T ≠ ∅

Using the Formal Semantics

305

R(a), S(a), T2(a)

R

a
1
2

S

a
1

T2

a

What do these queries compute?

```
SELECT  DISTINCT R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

```
SELECT  DISTINCT R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
    or  R.a=T.a
```

a

Returns ∅ if S = ∅ or T = ∅

Can seem counterintuitive! But remember conceptual evaluation strategy: Nested loops. If one table is empty -> no looping

Illustration with Python

305

```python
'''
Created on 3/23/2015
Illustrates nested Loop Join in SQL
__author__ = 'gatt'
'''

print "--- 1st nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(2):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 2nd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(1):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 3rd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(0):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print
```

```
/Library/Frameworks/Python.framework/Versio
--- 1st nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=0, k=1:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=1, k=1:
i=0, j=2, k=0:  TRUE
i=0, j=2, k=1:
i=1, j=0, k=0:
i=1, j=0, k=1:  TRUE
i=1, j=1, k=0:  TRUE
i=1, j=1, k=1:  TRUE
i=1, j=2, k=0:
i=1, j=2, k=1:  TRUE

--- 2nd nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=2, k=0:  TRUE
i=1, j=0, k=0:
i=1, j=1, k=0:  TRUE
i=1, j=2, k=0:

--- 3rd nested loop ---

Process finished with exit code 0
```

The comparison gets never evaluated

1. Aggregates
2. Groupings
3. Having

Aggregation

Car (name, price, maker)


348

```
SELECT  avg(price)
FROM    Car
WHERE   maker='Toyota'
```

```
SELECT  count(*)
FROM    Car
WHERE   price > 100
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

Aggregation

348

```
SELECT   avg(price)
FROM     Car
WHERE    maker='Toyota'
```

Car

Name	Price	Maker
~~M3~~	~~120~~	~~BMW~~
~~M5~~	~~150~~	~~BMW~~
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

Database creates new attribute name (for SQLserver)

(No column name)
75

Aggregation with rename

"as" optional

```
SELECT   count(*) as n
FROM     Car
WHERE    price > 100
```

Database creates *our* new attribute name

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
~~Prius~~	~~50~~	~~Toyota~~
~~Lexus1~~	~~75~~	~~Toyota~~
~~Lexus2~~	~~100~~	~~Toyota~~

⇒

n
2

Aggregation: Count Distinct

348

```
SELECT  count(maker)
FROM    Car
WHERE   price > 100
```

Same as count(*)

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
~~Prius~~	~~50~~	~~Toyota~~
~~Lexus1~~	~~75~~	~~Toyota~~
~~Lexus2~~	~~100~~	~~Toyota~~

We probably want to ignore duplicates:

```
SELECT  count(DISTINCT maker)
FROM    Car
WHERE   price > 100
```

(No column name)
1

Simple Aggregation 1/3

308

Purchase (product, price, quantity)

```
SELECT  sum(price * quantity)
FROM    Purchase
```

```
SELECT  sum(price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

What do these queries mean?

Simple Aggregation 2/3

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
~~Banana~~	~~2~~	~~10~~
~~Banana~~	~~4~~	~~10~~

3 * 20 = 60
2 * 20 = 40
sum: 100

Database creates new attribute name

```
SELECT  sum(price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

(No column name)
100

Simple Aggregation 3/3

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
~~Banana~~	~~2~~	~~10~~
~~Banana~~	~~4~~	~~10~~

3	20
2	20
sum: 5	sum: 40

sum: 5 * sum: 40 = 200

```
SELECT   sum(price) * sum(quantity)
FROM     Purchase
WHERE    product = 'Bagel'
```

(No column name)
200

Grouping and Aggregation

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

Product	TotalSales
Bagel	40
Banana	20

Notice: we use "sales" for total number of products sold

Find total quantities for all purchases with price over $1 grouped by product.

From → Where → Group By → Select

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

Product	TotalSales
Bagel	40
Banana	20

Select contains
- grouped attributes
- and aggregates

```
4  SELECT     product, sum(quantity) as TotalSales
1  FROM       Purchase
2  WHERE      price > 1
3  GROUP BY   product
```

Let's confuse the database engine

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	Quantity
Bagel	?
Banana	?

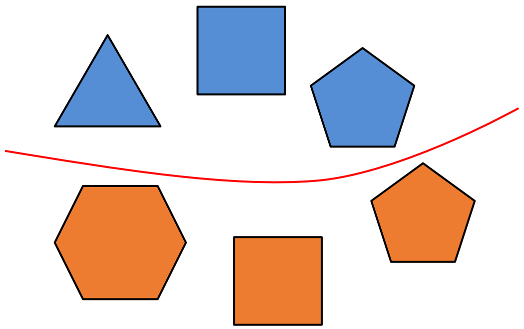
What quantity should the DB return for Banana?

```
SELECT      product, quantity
FROM        Purchase
GROUP BY    product
```

The DB engine is confused, there is no single quantity for banana (it's an ill-defined query). It should thus return an error (only SQLite misbehaves and returns something, but which makes no sense). Please think this through carefully!

Groupings illustrated with colored shapes

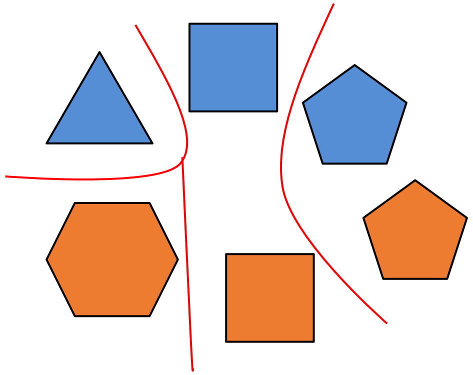
group by color



```
SELECT   color,
         avg(numc) anc
FROM     Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT   numc
FROM     Shapes
GROUP BY numc
```

numc
3
4
5
6

Another Example

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇒

Product	SumQ	MaxP
Bagel	40	3
Banana	70	4

```
SELECT   product,
         sum(quantity) as SumQ,
         max(price) as MaxP
FROM     Purchase
GROUP BY product
```

Next, focus only on products with at least 50 sales

Having Clause

Q: Similar to before, but only products with at least 50 sales.

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	SumQ	MaxP
Banana	70	4

```
SELECT   product,
         sum(quantity) as SumQ,
         max(price) as MaxP
FROM     Purchase
GROUP BY product
HAVING   sum(quantity) > 50
```

Quizz

What does this query return over the given database?

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
~~Banana~~	~~2~~	~~10~~
~~Banana~~	~~4~~	~~10~~

Product	SumQ
~~Bagel~~	~~40~~
Banana	50

```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   sum(quantity) > 40
```

General form of Grouping and Aggregation

5	SELECT	S
1	FROM	$R_1,\ldots,R_n$
2	WHERE	C1
3	GROUP BY	$a_1,\ldots,a_k$
4	HAVING	C2

S: may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but no other attributes

C1: is any condition on the attributes in $R_1,\ldots,R_n$

C2: is any condition on aggregates and on attributes $a_1,\ldots,a_k$

Evaluation

1. Evaluate FROM
2. WHERE, apply condition C1
3. GROUP BY the attributes $a_1,\ldots,a_k$
4. Apply condition C2 to each group (may have aggregates)
5. Compute aggregates in S and return the result

General form of SQL Query

5	**SELECT**	S
1	**FROM**	$R_1,\ldots,R_n$
2	**WHERE**	C1
3	**GROUP BY**	$a_1,\ldots,a_k$
4	**HAVING**	C2
6	**ORDER BY**	S2

S: may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but no other attributes

C1: is any condition on the attributes in $R_1,\ldots,R_n$

C2: is any condition on aggregates and on attributes $a_1,\ldots,a_k$

The logical order is useful for under-standing, but not always correct. The ANSI SQL standard does not require a specific processing order and leaves that to the implementation. Recall our intro example with SELECT DISTINCT and order by! Notice that that example can't be explained with the order shown here

Evaluation

1. Evaluate FROM
2. WHERE, apply condition C1
3. GROUP BY the attributes $a_1,\ldots,a_k$
4. Apply condition C2 to each group (may have aggregates)
5. Compute aggregates in S and return the result
6. Sort rows by ORDER BY clause

Conceptual Evaluation Strategy

- The cross-product of relation-list is computed (FROM), tuples that fail qualification are discarded (WHERE), then:
- GROUP BY: the remaining tuples are partitioned into groups by the value of attributes in grouping-list.
- HAVING: The group-qualification is then applied to eliminate some groups. Expressions in group-qualification must have a single value per group!
  - In effect, an attribute in group-qualification that is not an argument of an aggregate op must also appear in grouping-list. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Don't use new Alias in HAVING clause

308

What does this query return over the given database?

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇒

Product	SumQ
Bagel	40
Banana	50

Error in SQL server!
Reason: HAVING is evaluated before SELECT!
(However, SQLite works: different implementation)

```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   SumQ > 35
```

Source: http://stackoverflow.com/questions/2068682/why-cant-i-use-alias-in-a-count-column-and-reference-it-in-a-having-clause

Don't use new Alias in HAVING clause

What does this query return over the given database?

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨

Product	SumQ
Banana	50
Bagel	40

Works! Notice that new sorting

```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   sum(quantity)  > 35
ORDER BY sumQ desc
```

L04: SQL

Announcements!

- Polls on Piazza. Open for 2 days
- Outline today:
  - practicing more joins and specifying key and FK constraints
  - nested queries
- Next time: "witnesses" (traditionally students find this topic the most difficult)

Queries via SQL have multiple words: If you master this structure you know 50% about SQL Queries

- **SELECT …** — List of attributes to be included in final result (also called projection! ("*" selects all attributes)
- **FROM …** — Indicates the table(s) from which data is to be retrieved
- **WHERE …** — Lists a comparison predicate, which restricts the rows returned by the query, e.g. "price < 20" or different join conditions
- **GROUP BY …** — Groups rows that have one more common values together into a smaller set of rows
- **HAVING …** — A comparison predicate used to restrict the rows resulting from the GROUP BY clause
- **ORDER BY …** — Identifies which columns are used to sort the resulting data, plus the direction each column is sorted by (ascending or descending)

Note1 : SQL is generally case insensitive, e.g. SELECT = Select = select

Note2 : The words always appear in this order – you CANNOT reorder them

How to specify Foreign Key constraints

- Suppose we have the following schema:

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

- And we want to impose the following constraint:
  - 'Only bona fide students may enroll in courses' i.e. a student must appear in the Students table to enroll in a class

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8

Enrolled

student_id	cid	grade
123	564	A
123	537	A+

student_id alone is not a key- what is?

We say that student_id is a <u>**foreign key**</u> that refers to Students

Declaring Primary Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

```
CREATE TABLE Students(
     sid  CHAR(20) PRIMARY KEY,
     name CHAR(20),
     gpa  REAL


)
```

Declaring Primary Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

```
CREATE TABLE Students(
     sid  CHAR(20),
     name CHAR(20),
     gpa  REAL,
     PRIMARY KEY (sid)

)
```

Declaring Foreign Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

```
CREATE TABLE Enrolled(
    student_id CHAR(20),
    cid        CHAR(20),
    grade      CHAR(10),
    PRIMARY KEY (student_id, cid),
    FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

An example of SQL semantics

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

R

A
1
3

S

B	C
2	3
3	4
3	5

Cross Product

A	B	C
1	2	3
1	3	4
1	3	5
3	2	3
3	3	4
3	3	5

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

Apply Projection

Output

A
3
3

Note the semantics of a join

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

2. Apply **selections / conditions**:

$$Y = \{(r,s) \in X \mid r.A = r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,)\ for\ y \in Y$$

= Returning only *some* attributes

Remembering this order is critical to understanding the output of certain queries (see later on…)

Note: we say “semantics” not “execution order”

- The preceding slides show what a join means
- Not actually how the DBMS executes it under the covers

Practicing more Joins

Quiz 4

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture at least two different products.

```
SELECT  DISTINCT cName
FROM
WHERE
```

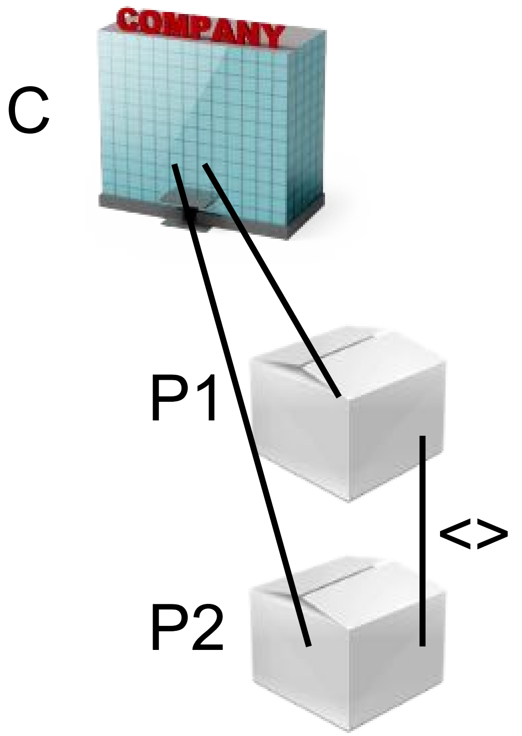
Quiz 4

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture at least two different products.

```
SELECT  DISTINCT cName
FROM    Product P1, Product P2, Company
WHERE   country = 'USA'
   and  P1.manufacturer = cName
   and  P2.manufacturer = cName
   and  P1.pName <> P2.pName
```



Quiz 4

302

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
…	…	…	…

<>

P2

PName	Price	Category	Manufacturer
…	…	…	…
Powergizmo	$29.99	Gadgets	GizmoWorks

Company

CName	StockPrice	Country
GizmoWorks	25	USA
…	…	…

```
SELECT  DISTINCT cName
FROM    Product P1, Product P2, Company
WHERE   country = 'USA'
    and P1.manufacturer = cName
    and P2.manufacturer = cName
    and P1.pName <> P2.pName
```

⇒

Cname
GizmoWorks

Quiz 5

302

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture a product below $20 and a product above $15.

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
   and  P1.price < 20
   and  P2.price > 15
   and  P1.manufacturer = cName
   and  P2.manufacturer = cName
```

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product

PName	Price	Category	Manufacturer
Gizmo	19.99	Gadgets	GizmoWorks
Powergizmo	29.99	Gadgets	GizmoWorks
SingleTouch	149.99	Photography	Canon
MultiTouch	203.99	Household	Hitachi

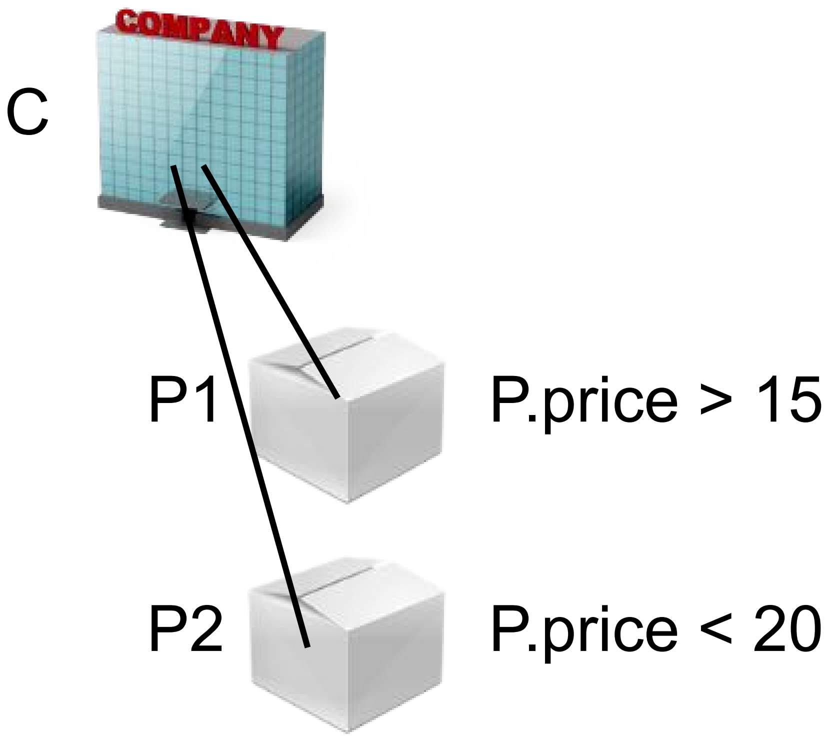
Quiz 5

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture a product below $20 and a product above $15.

Note that we did not specify any condition that P1 and P2 need to be distinct. An alternative interpretation is "...and another product above..."



Quiz 5

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
...	...	...	...

P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
...	...	...	...

Company

CName	StockPrice	Country
GizmoWorks	25	USA
...	...	...

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
   and  P1.price < 20
   and  P2.price > 15
   and  P1.manufacturer = cName
   and  P2.manufacturer = cName
```

Cname
GizmoWorks

Quiz 6

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all countries that have companies that manufacture some product in the 'Gadgets' category!

```
SELECT  country
FROM    Product, Company
WHERE   manufacturer = cName
    and category = 'Gadgets'
```

⇨ ?

Quiz 6

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all countries that have companies that manufacture some product in the 'Gadgets' category!

```
SELECT  country
FROM    Product, Company
WHERE   manufacturer = cName
    and category = 'Gadgets'
```

Country
USA
USA

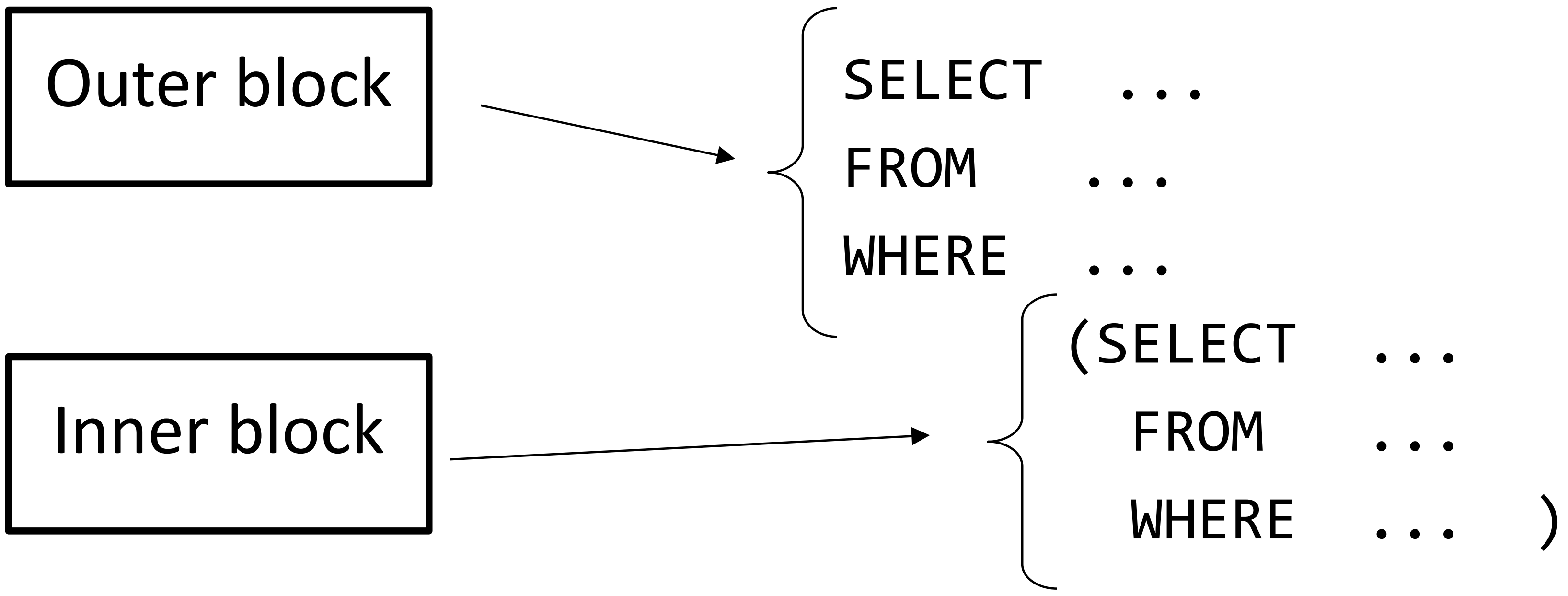
Joins can introduce duplicates -> remember to use DISTINCT!

Nested queries (Subqueries)

High-level note on nested queries

- We can do nested queries because SQL is compositional:
  - Everything (inputs / outputs) is represented as multisets- the output of one query can thus be used as the input to another (nesting)!
- This is extremely powerful!
- High-level idea: subqueries return relations (yet sometimes just values)

Subqueries = Nested queries



Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in a:
  - SELECT clause
  - FROM clause
  - WHERE clause
  - HAVING clause

important!

- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it’s impossible

1. Subqueries in SELECT

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each product return the city where it is manufactured!

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company2 C
                  WHERE   C.cid = P.cid)
FROM    Product2 P
```

What happens if the subquery returns more than one city ?

Runtime error

→ "Scalar subqueries"

1. Subqueries in SELECT

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each product return the city where it is manufactured!

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company2 C
                  WHERE   C.cid = P.cid)
FROM    Product2 P
```

"unnesting the query"

```
SELECT  P.pname, C.city
FROM    Product2 P, Company2 C
WHERE   C.cid = P.cid
```

Whenever possible, don't use nested queries

1. Subqueries in SELECT

315

```
Product2 (pname, price, cid)
Company2 (cid, cname, city)
```

Q: Compute the number of products made by each company!

```
SELECT  C.cname, ( SELECT count (*)
                   FROM   Product2 P
                   WHERE  P.cid = C.cid)
FROM    Company2 C
```

Better: we can unnest by using a GROUP BY:

```
SELECT   C.cname, count(*)
FROM     Company2 C, Product2 P
WHERE    C.cid=P.cid
GROUP BY C.cname
```

2. Subqueries in FROM clause

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: Find all products whose prices are > 20 and < 30!

```
SELECT  X.pname
FROM    ( SELECT *
          FROM    Product2 as P
          WHERE   price >20 ) as X
WHERE   X.price < 30
```

unnesting

```
SELECT  pname
FROM    Product2
WHERE   price > 20 and price < 30
```

X

PName	Price	cid
Powergizmo	$29.99	1
~~MultiTouch~~	~~$203.99~~	~~3~~

Subqueries in WHERE clause

IN, ANY, ALL

3. Subqueries in WHERE

R
a
1
2

U
a
2
3
4

305

What do these queries compute?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT * from U)
```

→ ?

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT * from U)
```

→ ?

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT * from U)
```

→ ?

3. Subqueries in WHERE

305

R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT * from U)
```

a
2

Since 2 is in the set (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT * from U)
```

a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT * from U)
```

a
1

Since 1 is < than each ("all") of 2, 3, and 4

Something tricky about Nested Queries

Are these queries equivalent?

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
   AND p.buyer = ‘Joe B‘)
```

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
   AND pr.name = p.product
   AND p.buyer = ‘Joe B’
```

Beware of duplicates!

Something tricky about Nested Queries

Are these queries equivalent?

```
SELECT DISTINCT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
   AND p.buyer = ‘Joe B‘)
```

```
SELECT DISTINCT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
   AND pr.name = p.product
   AND p.buyer = ‘Joe B’
```

Now they are equivalent (both use set semantics)

Correlated subqueries

- In all previous cases, the nested subquery in the inner select block could be entirely evaluated before processing the outer select block.
- This is no longer the case for <u>correlated nested queries</u>.
- Whenever a condition in the WHERE clause of a nested query references some column of a table declared in the outer query, the two queries are said to be correlated.
- The nested query is then evaluated once for each tuple (or combination of tuples) in the outer query.

Correlated Queries (Using External Vars in Internal Subquery)

```
Movie(title, year, director, length)
```

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
           SELECT  year
           FROM    Movie
           WHERE   title =  m.title)
```

Find movies whose title appears in more than one year.

Note the scoping of the variables!

Note also: this can still be expressed as single SFW query…

Complex Correlated Query

```
Product(name, price, category, maker, year)
```

```
SELECT DISTINCT  x.name, x.maker
FROM   Product AS x
WHERE  x.price > ALL(
          SELECT y.price
          FROM   Product AS y
          WHERE  x.maker = y.maker
             AND y.year < 1972)
```

Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

Can be very powerful (also much harder to optimize)

3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using IN:

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    C.cid IN ( 1, 2 )
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

3. Subqueries in WHERE (existential)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using IN:

"Set membership"

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product2 P
                   WHERE   P.price < 25)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using EXISTS:

"Test for empty relations"

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    EXISTS ( SELECT  *
                  FROM    Product2 P
                  WHERE   C.cid = P.cid
                    and   P.price < 25)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

3. Subqueries in WHERE (existential)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Using ANY (also some):

"Set comparison"

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    25 > ANY ( SELECT  price
                    FROM    Product2 P
                    WHERE   P.cid = C.cid)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

SQLlite does not support "ANY" ☹

3. Subqueries in WHERE (existential)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Now, let's unnest:

```
SELECT  DISTINCT C.cname
FROM    Company2 C, Product2 P
WHERE   C.cid = P.cid
   and  P.price < 25
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Existential quantifiers are easy ! ☺

3. Subqueries in WHERE (universal)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make <u>only</u> products with price < 25!

same as:

Q: Find all companies for which <u>all</u> products have price < 25!

Universal quantifiers are more complicated ! ☹
(Think about the companies that should not be returned)

3. Subqueries in WHERE (exist not -> universal)

315

Q: Find all companies that make only products with price < 25!

1. Find the other companies: i.e. they have some product $\geq$ 25!

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN (  SELECT  P.cid
                    FROM    Product2 P
                    WHERE   P.price >= 25)
```

2. Find all companies s.t. all their products have price < 25!

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid NOT IN ( SELECT  P.cid
                       FROM    Product2 P
                       WHERE   P.price >= 25)
```

3. Subqueries in WHERE (exist not -> universal)

315

```
Product2 (pname, price, cid)
Company2 (cid, cname, city)
```

Universal quantifiers ∀

Q: Find all companies that make only products with price < 25!

Using NOT EXISTS:

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   NOT EXISTS ( SELECT  *
                     FROM    Product2 P
                     WHERE   C.cid = P.cid
                         and P.price >= 25)
```

3. Subqueries in WHERE (exist not -> universal)

315

Product2 (pname, price, cid) Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make <u>only</u> products with price < 25!

Using ALL:

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    25 > ALL (  SELECT   price
                     FROM     Product2 P
                     WHERE    P.cid = C.cid)
```

SQLlite does not support "ALL" ☹

Question for Database Fans & Friends

This topic goes beyond the course objectives; only for those who are really interested

- How can we unnest the universal quantifier query ?

Queries that must be nested

This topic goes beyond the course objectives; only for those who are really interested

- Definition: A query Q is monotone if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples
- Fact: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics
- Fact: Query with universal quantifier is not monotone
  - Add one tuple violating the condition. Then "all" returns fewer tuples
- Consequence: we cannot unnest a query with a universal quantifier

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

331

Challenge: write these in SQL.
Solutions: http://queryviz.com/online/

Find drinkers that frequent some bar that serves some beer they like.

$$x: \quad \exists y.\ \exists z.\ \text{Frequents}(x, y) \land \text{Serves}(y,z) \land \text{Likes}(x,z)$$

Find drinkers that frequent only bars that serve some beer they like.

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \land \text{Likes}(x,z))$$

Find drinkers that frequent some bar that serves only beers they like.

$$x: \quad \exists y.\ \text{Frequents}(x, y) \land \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find drinkers that frequent only bars that serve only beer they like.

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Basic SQL Summary

- SQL provides a high-level declarative language for manipulating data (DML)
- The workhorse is the SFW block
- Set operators are powerful but have some subtleties
- Powerful, nested queries also allowed.

WITH clause

1

WITH clause: temporary relations


315

```
SELECT pname, price
FROM   Product2
WHERE  price =
       (SELECT max(price)
        FROM   Product2)
```

↓

```
WITH   max_price(value) as
       (SELECT max(price)
        FROM   Product2)
SELECT pname, price
FROM   Product2, max_price
WHERE  price = value
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

WITH clause: temporary relations

315

```
SELECT  pname, price
FROM    Product2
WHERE   price =
        (SELECT  max(price)
         FROM    Product2)
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

```
WITH    max_price as
        (SELECT  max(price) as value
         FROM    Product2)
SELECT  pname, price
FROM    Product2, max_price
WHERE   price = value
```

Witnesses

Motivation: What are these queries supposed to return?

Product2

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company2

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products ⇨ ?

Motivation: What are these queries supposed to return?

Product2

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company2

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products ⇨

cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products ⇨ ?

Motivation: What are these queries supposed to return?

Product2

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company2

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products

cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products (Recall that "group by cid" can just give us one single tuple per cid)

cid	mp	pname
1	20	SuperGizmo
2	300	iTouch1
2	300	iTouch2

Witnesses: simple (1/4)

Product2 (pname, price, cid)


315

Q: Find the most expensive product + its price

(Finding the maximum price alone would be easy)

Witnesses: simple (2/4)

Product2 (pname, price, cid)


315

Q: Find the most expensive product + its price

(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery

1.
```
SELECT   max(P1.price)
FROM     Product2 P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (3/4)

Product2 (pname, price, cid)


315

Q: Find the most expensive product + its price

(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...

2.
```
SELECT  P2.pname, P2.price
FROM    Product2 P2
```

1.
```
SELECT  max(P1.price)
FROM    Product2 P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (4/4)

Product2 (pname, price, cid)


315

Q: Find the most expensive product + its price

(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...
  and compare the price with the max price

```
SELECT  P2.pname, P2.price
FROM    Product2 P2
WHERE   P2.price =
        (SELECT  max(P1.price)
         FROM    Product2 P1)
```

L05: SQL

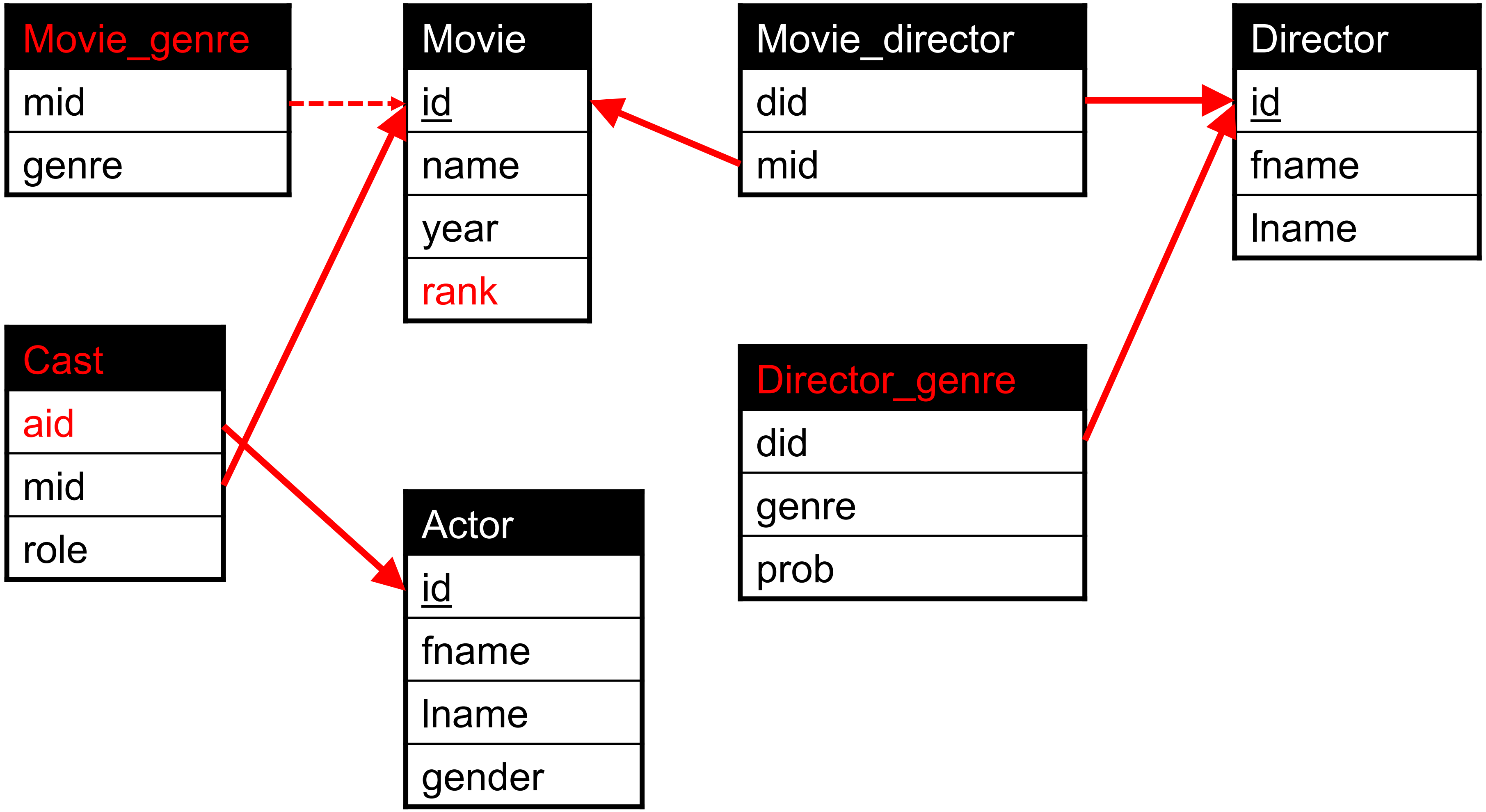
Announcements!

- HW1 is due tonight
- HW2 groups are assigned

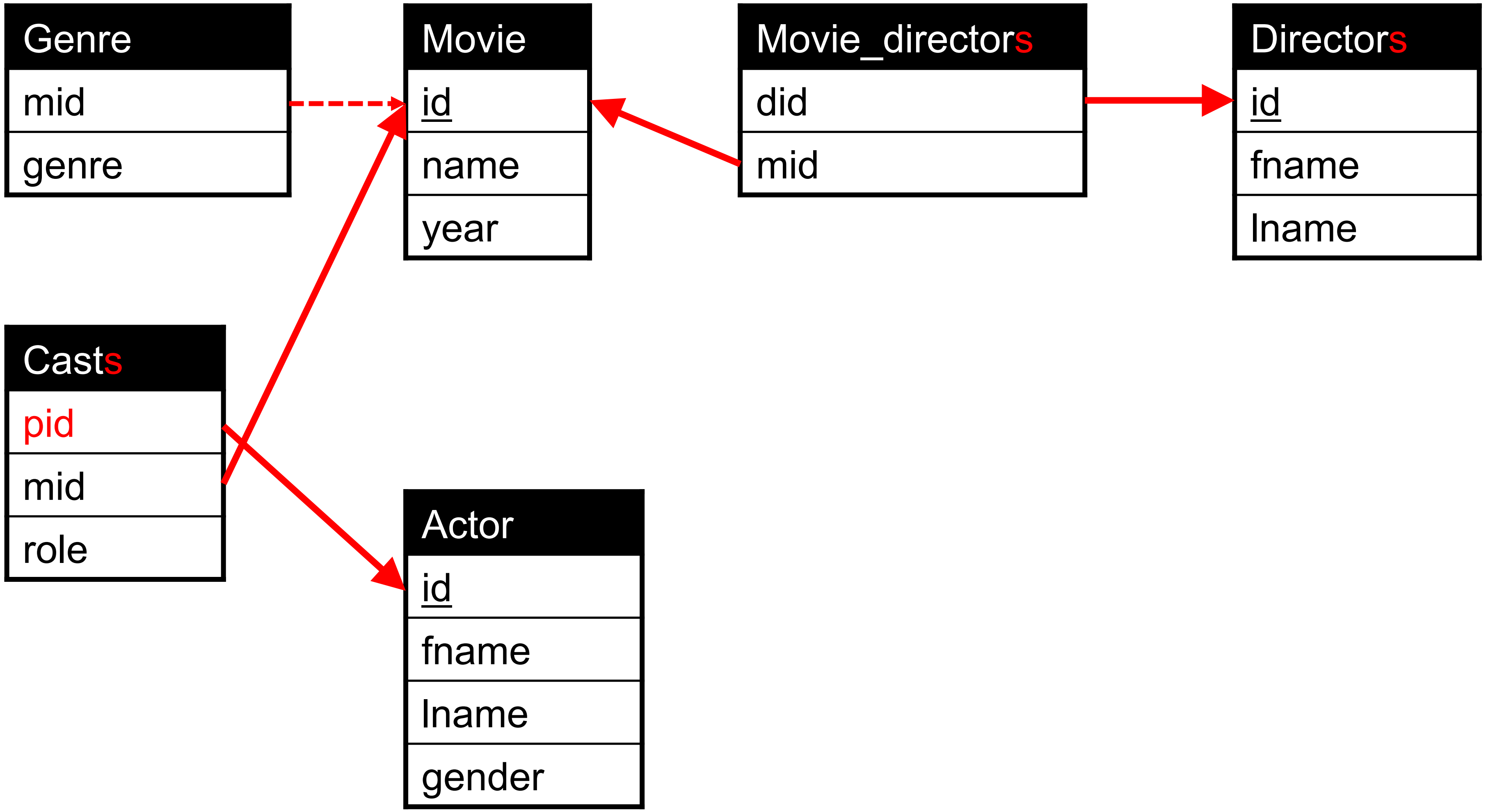
- Outline today:
  - nested queries and witnesses
  - We start with a detailed example!
  - outer joins, nulls?

Small IMDB schema (SQLite)

300



Big IMDB schema (Postgres)



Theta joins

305

What do these queries compute?

R

a
1
2

U

a
2
3
4

```
SELECT   R.a, U.a
FROM     R, U
WHERE    R.a < U.a
```

⇨ ?

```
SELECT   R.a, U.a
FROM     R, U
WHERE    R.a <= U.a
```

⇨ ?

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Theta joins

305

R.a
1
2

U.a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

a	b
1	2
1	3
1	4
2	3
2	4

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

a	b
2	2

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Witnesses: with joins (1/6)

315

Product2 (pname, price, cid)
Company2 (<u>cid</u>, cname, city)

Q: <u>For each company</u>, find the most expensive product + its price

Witnesses: with joins (2/6)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company

S max(price)
F P
W cid = 1

Witnesses: with joins (2/6)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company

1.
```
SELECT  max(P1.price)
FROM    Product2 P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

315

Product2 (pname, price, cid)
Company2 (<u>cid</u>, cname, city)

Q: <u>For each company</u>, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery <u>for a given company</u>
- 2. Compute each product and its price, <u>per company</u>

S *
F P, C
W P.cid = C.cid

1.
```
SELECT  max(P1.price)
FROM    Product2 P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company

2.
```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company2 C2, Product2 P2
WHERE   C2.cid = P2.cid
```

1.
```
SELECT  max(P1.price)
FROM    Product2 P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
- 3. Compare the price with the max price

2.
```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company2 C2, Product2 P2
WHERE   C2.cid = P2.cid
```

1.
```
SELECT  max(P1.price)
FROM    Product2 P1
WHERE   P1.cid = 1
```

Witnesses: with joins (4/6)


315

Product2 (pname, price, cid)
Company2 (<u>cid</u>, cname, city)

Q: <u>For each company</u>, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery <u>for a given company</u>
- 2. Compute each product and its price, <u>per company</u>
- 3. Compare the price with the max price

```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company2 C2, Product2 P2
WHERE   C2.cid = P2.cid
   and  P2.price =
        (SELECT max(P1.price)
         FROM   Product2 P1
         WHERE  P1.cid = C2.cid)
```

How many aliases do we actually need?

Witnesses: with joins (5/6)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company and compare the price with the max price

```
SELECT  cname, pname, price
FROM    Company2, Product2
WHERE   Company2.cid = Product2.cid
    and price =
        (SELECT max(price)
         FROM   Product2
         WHERE  cid = Company2.cid)
```

We need no single alias here.

Next: can we eliminate the max operator in the inner query?

Witnesses: with joins (6/6)


315

Product2 (pname, price, cid)
Company2 (<u>cid</u>, cname, city)

Q: <u>For each company</u>, find the most expensive product + its price

Our Plan:

- 1. Compute all prices in a subquery, <u>for a given company</u>
- 2. Compute each product and its price, <u>per company</u> and compare the price with the all prices

```
SELECT  cname, pname, price
FROM    Company2, Product2
WHERE   Company2.cid = Product2.cid
    and price >= ALL
        (SELECT price
         FROM   Product2
         WHERE  cid = Company2.cid)
```

But: "ALL" does not work in SQLite ☹

Witnesses: with FROM (1/3)


315

Product2 (pname, price, cid)
Company2 (<u>cid</u>, cname, city)

Q: <u>For each company</u>, find the most expensive product + its price

Another Plan:

- 1. Create a table that lists the max price for each <u>company id</u>
- 2. Join this table with the remaining tables

F (

1.
```
SELECT   cid, max(price) as MP
FROM     Product2
GROUP    BY cid
```
) x, P, C

Finding the maximum price for each company was easy.
But we want the “witnesses”, i.e. the products with max price.

Witnesses: with FROM (2/3)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Another Plan:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

2.
```
SELECT  C2.cname, P2.pname, X.MP
FROM    Company2 C2, Product2 P2,
        (SELECT  cid, max(price) as MP
         FROM    Product2
         GROUP   BY cid) as X
WHERE   C2.cid = P2.cid
   and  C2.cid = X.cid
   and  P2.price = X.MP
```

Let's write the same query with a "WITH" clause

Witnesses: with FROM (3/3)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each company, find the most expensive product + its price

Another Plan with WITH:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

```
WITH     X(cid, MP) as
         (SELECT  cid, max(price)
          FROM     Product2
          GROUP    BY cid)
SELECT   C2.cname, P2.pname, X.MP
FROM     Company2 C2, Product2 P2, X
WHERE    C2.cid = P2.cid
    and  C2.cid = X.cid
    and  P2.price = X.MP
```

Witnesses: with aggregates per group (1/8)

308

First: How to get the product that is sold with maximum price?

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	mp
Banana	4

```
SELECT   product, max(price) as mp
FROM
WHERE
GROUP BY
HAVING
```

???

Witnesses: with aggregates per group (2/8)

308

First: How to get the product that is sold with maximum price?

Purchase *1) Find the maximum price*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

(no name)
4

```
SELECT   max(price)
FROM     Purchase
```

Witnesses: with aggregates per group (3/8)

First: How to get the product that is sold with maximum price?

Purchase *2) Now you need to find product with price = maximum price*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇒

Product	mp
Banana	4

```
SELECT   P2.product, P2.price as mp
FROM     Purchase P2
WHERE    P2.price = (
            SELECT  max(price)
            FROM    Purchase
)
```

Witnesses: with aggregates per group (4/8)


308

First: How to get the product that is sold with maximum price?

Purchase *Another way to formulate this query*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	mp
Banana	4

```
SELECT   P2.product, P2.price as mp
FROM     Purchase P2
WHERE    P2.price >= ALL (
           SELECT   price
           FROM     Purchase
)
```

Witnesses: with aggregates per group (5/8)

308

Second: How to get the product that is sold with max sales (=quanitities sold)?

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

→

Product	sales
Banana	70

```
SELECT
FROM
WHERE
GROUP BY
HAVING
```

???

Witnesses: with aggregates per group (6/8)

308

Second: How to get the product that is sold with max sales (=quanitities sold)?

Purchase *1: find the total sales (sum of quantity) for each product*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	sales
Bagel	40
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY product
```

Witnesses: with aggregates per group (7/8)

308

Second: How to get the product that is sold with max sales?

2: we can use the same query as nested query

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨

(no name)
40
70

```
SELECT      sum(quantity)
FROM        Purchase
GROUP BY product
```

Witnesses: with aggregates per group (8/8)

308

Second: How to get the product that is sold with max sales?

Purchase *3: putting things together*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	sales
Banana	70

```
SELECT      product, sum(quantity) as sales
FROM        Purchase
GROUP BY product
HAVING      sum(quantity) >= ALL (
                SELECT      sum(quantity)
                FROM        Purchase
                GROUP BY product
            )
```

Next: Can you write the query without the "ALL" quanitfier?

Witnesses: with aggregates per group (8/8)

Second: How to get the product that is sold with max sales?

Purchase *Another way to formulate this query without "ALL"*

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

Product	sales
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY  product
HAVING    sum(quantity) =
          (SELECT max (Q)
          FROM  (SELECT    sum(quantity) Q
                 FROM      Purchase
                 GROUP BY  product) X
          )
```

Understanding nested queries

More SQL Queries

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance *S*3 of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance *R*2 of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance *B*1 of Boats

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 1

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
        ( SELECT R.sid
        FROM Reserves R
        WHERE R.bid IN
                ( SELECT B.bid
                FROM Boats B
                WHERE B.color = 'red'))
```

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 2

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid not IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```

They must have reserved at least one boat in another color

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 3

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have not reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
        ( SELECT R.sid
          FROM Reserves R
          WHERE R.bid IN
                ( SELECT B.bid
                  FROM Boats B
                  WHERE B.color = 'red'))
```

They can have reserved 0 or more boats in another color, but must not have reserved any red boat

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 4

Sailors (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boats (<u>bid</u>, bname, color)

340

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
        ( SELECT R.sid
          FROM Reserves R
          WHERE R.bid not IN
                ( SELECT B.bid
                  FROM Boats B
                  WHERE B.color = 'red'))
```

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 5

Sailors (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boats (<u>bid</u>, bname, color)

340

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by him.

```
SELECT S.sname
FROM Sailors S
WHERE not exists
          ( SELECT B.bid
          FROM Boats B
          WHERE B.color = 'red'
          AND not exists
                    ( SELECT R.bid
                    FROM Reserves R
                    WHERE R.bid = B.bid
                    AND R.sid = S.sid))
```

To understand semantics of nested queries, think of a nested loops evaluation: For each Sailors tuple, check the qualification by computing the subquery

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

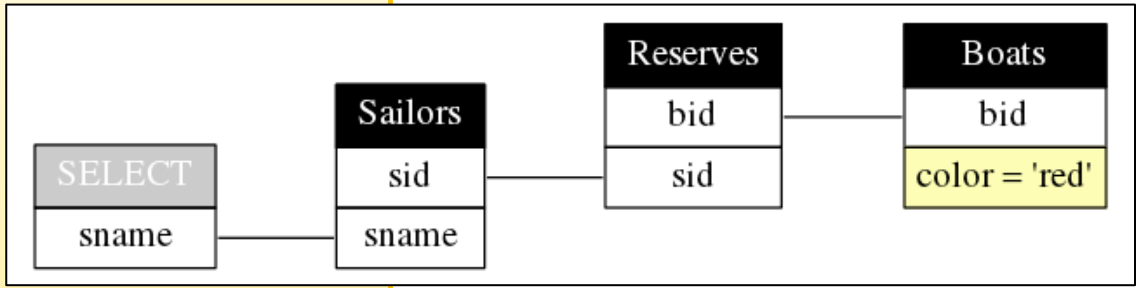
Once more: 1

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
        ( SELECT R.sid
          FROM Reserves R
          WHERE R.bid IN
                ( SELECT B.bid
                  FROM Boats B
                  WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

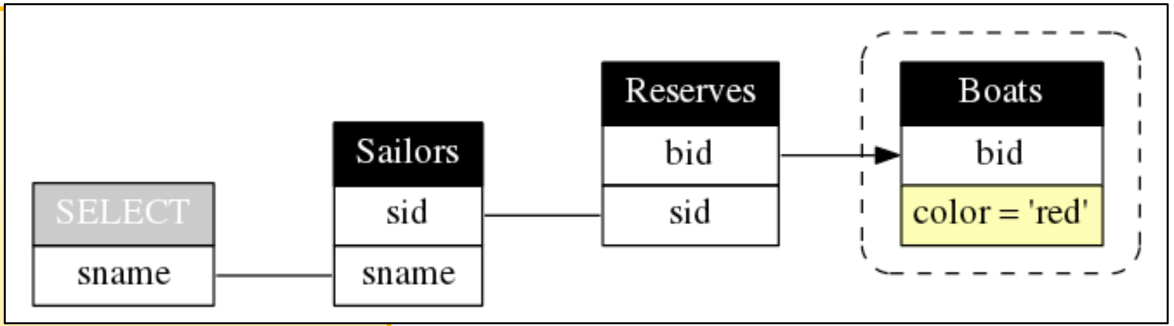
Once more: 2

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
        ( SELECT R.sid
          FROM Reserves R
          WHERE R.bid not IN
                ( SELECT B.bid
                  FROM Boats B
                  WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

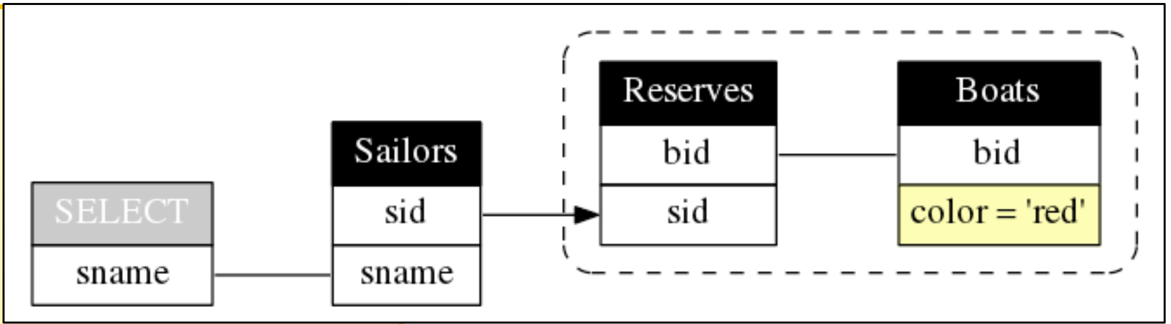
Once more: 3

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have not reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
        ( SELECT R.sid
          FROM Reserves R
          WHERE R.bid IN
                ( SELECT B.bid
                  FROM Boats B
                  WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

Once more: 4

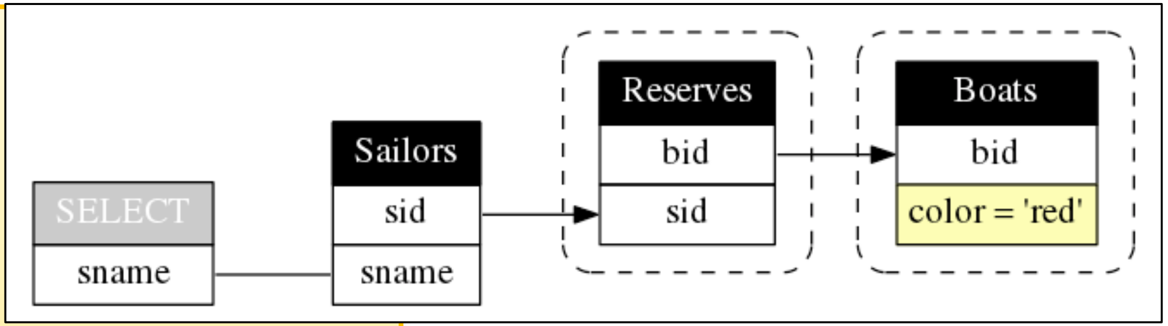
Sailors (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boats (<u>bid</u>, bname, color)

340

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
          ( SELECT R.sid
          FROM Reserves R
          WHERE R.bid not IN
                    ( SELECT B.bid
                    FROM Boats B
                    WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

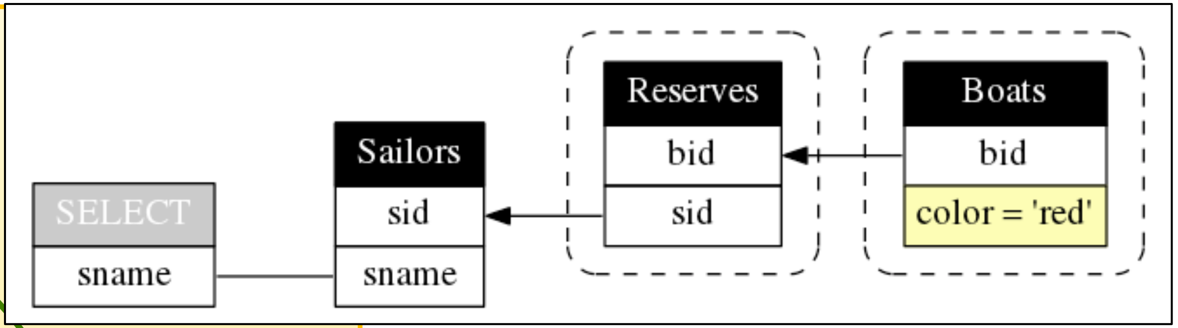
Once more: 5

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by him.

```
SELECT S.sname
FROM Sailors S
WHERE not exists
        ( SELECT B.bid
          FROM Boats B
          WHERE B.color = 'red'
          AND not exists
                ( SELECT R.bid
                  FROM Reserves R
                  WHERE R.bid = B.bid
                  AND R.sid = S.sid))
```



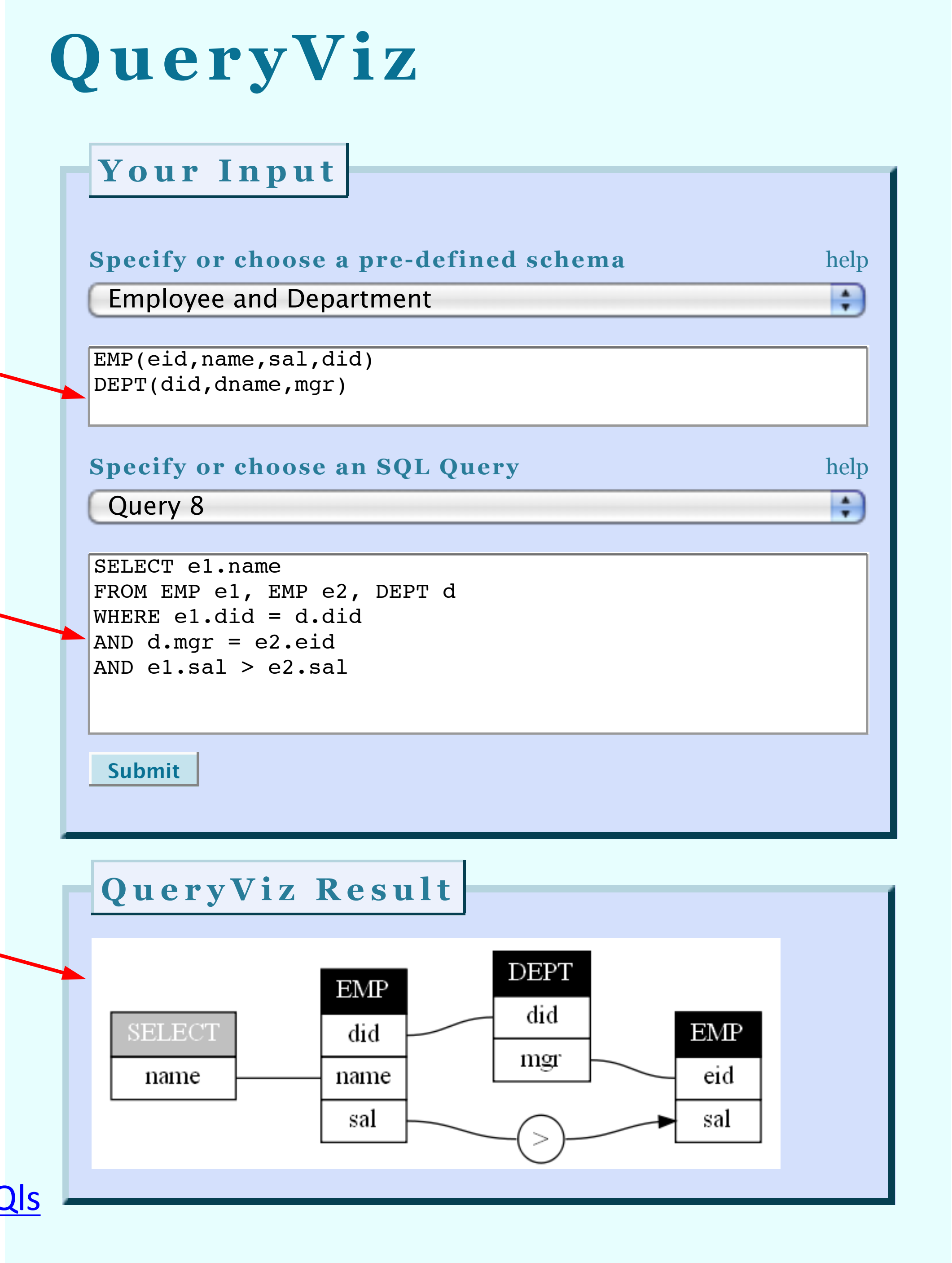
Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

http://queryviz.com

Input: Schema

Input Query

Output: Visualization



http://queryviz.com/online

http://www.youtube.com/watch?v=kVFnQRGAQls

Multiset operations (Intersect, Except)

Recall Multisets (Bags)

$\boldsymbol{\lambda(X)}$= "Count of tuple in X"
(Items not listed have implicit count 0)

Multiset X

Tuple
(1, a)
(1, a)
(1, b)
(2, c)
(2, c)
(2, c)
(1, d)
(1, d)

Equivalent Representations of a **<u>Multiset</u>**

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	1
(2, c)	3
(1, d)	2

Note: In a set all counts are {0,1}.

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cap$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	2
(1, b)	0
(2, c)	2
(1, d)	0

$$\lambda(Z) = min(\lambda(X), \lambda(Y))$$

For sets, this is **intersection**

Generalizing Set Operations to Multiset Operations

Multiset X

Tuple	$\lambda(X)$
(1, a)	2
(1, b)	0
(2, c)	3
(1, d)	0

$\cup$

Multiset Y

Tuple	$\lambda(Y)$
(1, a)	5
(1, b)	1
(2, c)	2
(1, d)	2

$=$

Multiset Z

Tuple	$\lambda(Z)$
(1, a)	7
(1, b)	1
(2, c)	5
(1, d)	2

$$\lambda(Z) = \lambda(X) + \lambda(Y)$$

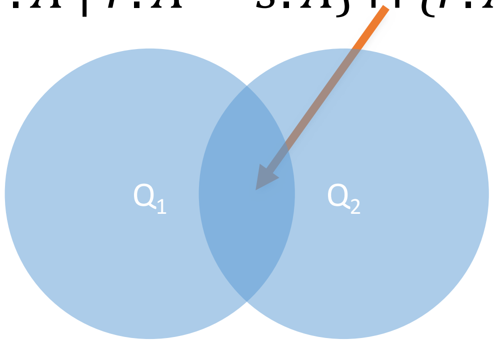
For sets,
this is **union**

Multiset Operations in SQL

Explicit Set Operators: INTERSECT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
INTERSECT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

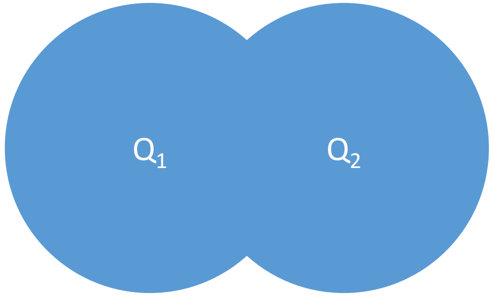
$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$



UNION

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION
SELECT  R.A
FROM    R, T
WHERE   R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$



Why aren't there duplicates?

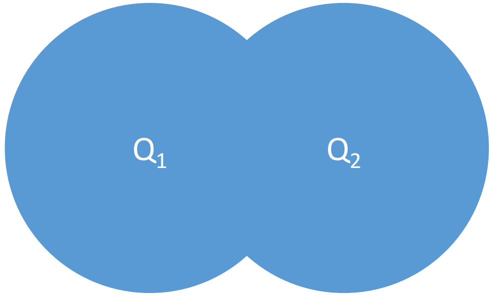
By default: SQL uses set semantics for INTERSECT and UNION!

What if we want duplicates?

UNION ALL

```
SELECT  R.A
FROM    R, S
WHERE   R.A=S.A
UNION ALL
SELECT R.A
FROM    R, T
WHERE   R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$$

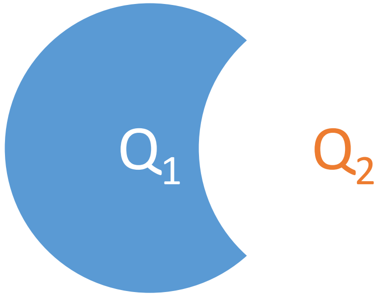


ALL indicates Multiset operations

EXCEPT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A
EXCEPT
SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \setminus \{r.A \mid r.A = t.A\}$$



What is the multiset version?

INTERSECT and EXCEPT*

R(a,b)
S(a,b)

```
(SELECT R.a, R.b
 FROM   R)

INTERSECT

(SELECT S.a, S.b
 FROM   S)
```

```
SELECT  R.a, R.b
FROM    R
WHERE
EXISTS  (SELECT  *
         FROM    S
         WHERE   R.a=S.a
         and     R.b=S.b)
```

If R, S have no duplicates, then can write without sub-queries (HOW?)

```
(SELECT R.A, R.B
 FROM   R)

EXCEPT

(SELECT S.A, S.B
 FROM   S)
```

```
SELECT R.A, R.B
FROM   R
WHERE
NOT  EXISTS  (SELECT  *
              FROM    S
              WHERE   R.A=S.A
              and     R.B=S.B)
```

*Not in all DBMSs. (SQLlite does not like the parentheses, Oracle uses "MINUS" instead of "EXCEPT")

L06: SQL

Announcements!

- Please pick up your name card
  - always come with your name card
  - If nobody answers my question, I will likely pick on those without a namecard or in the last row
- Polls on speed: we slow down and have another SQL lecture (likely no NoSQL)
- Use the anonymous feedback form
- HW3 and later: in teams

- Outline today:
  - HW1 together
  - outer joins, nulls

A word on capitalization

302

Product (pname, price, category, manufacturer)
Company (cname, stockprice, country)

Q: Find all US companies that manufacture products in the 'Gadgets' category!

```
SELECT  cname
FROM    Product P, Company
WHERE   country = 'USA'
AND     P.category = 'Gadgets'
AND     P.manufacturer = cname
```

My recommendation for capitalization

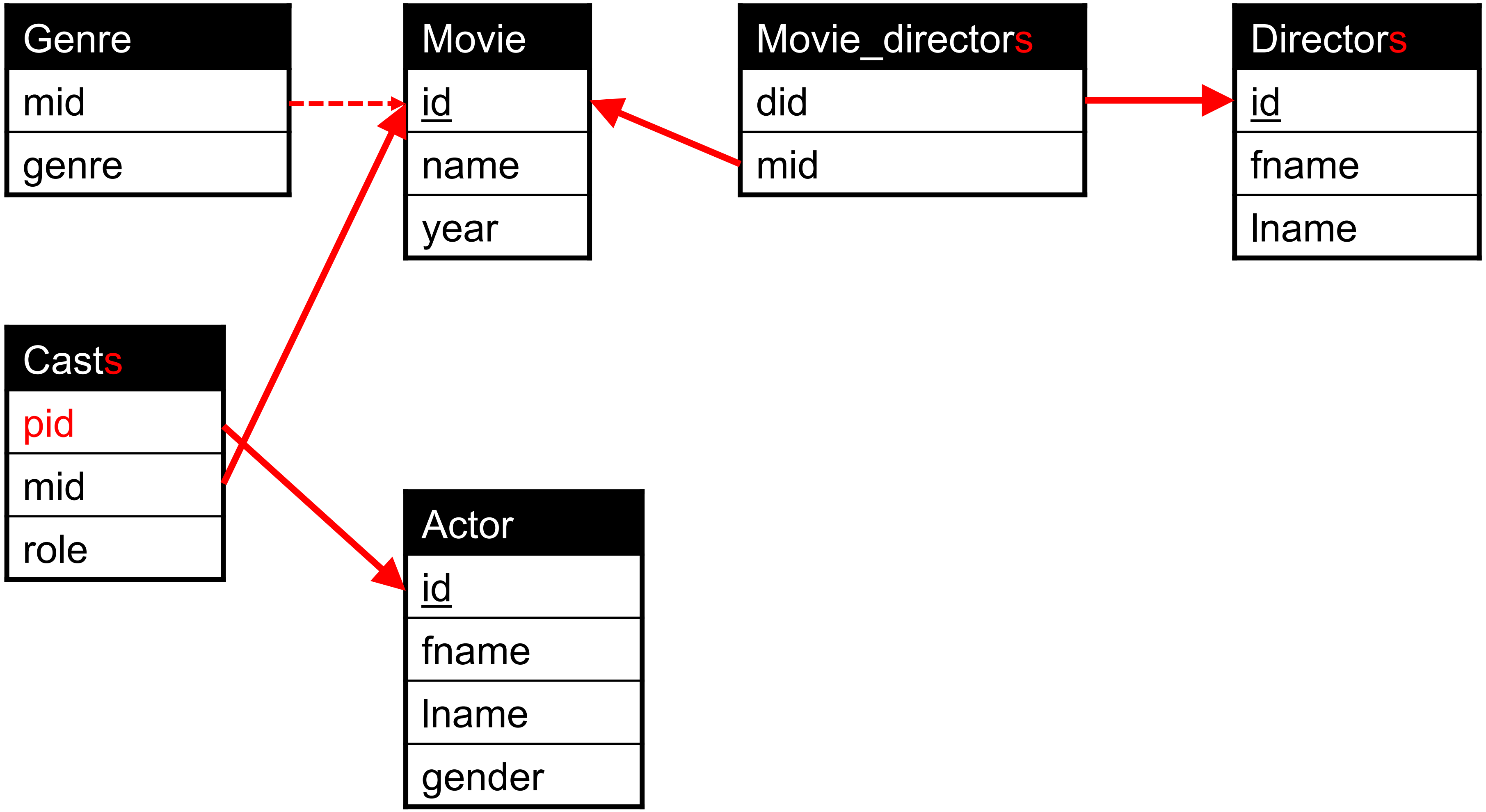
1. *SQL keywords in ALL CAPS,*
2. *Table names with Initial Caps*
3. *Column names all in lowercase.*

PostgreSQL treats all in lowercase.
Except if you write:
create table "Product" (…)
This will preserve capitalization of table name
But … you need to always use quotations

More information: http://blog.lerner.co.il/quoting-postgresql/ , https://stackoverflow.com/questions/6331504/omitting-the-double-quote-to-do-query-on-postgresql

HW1

Big IMDB schema (Postgres)



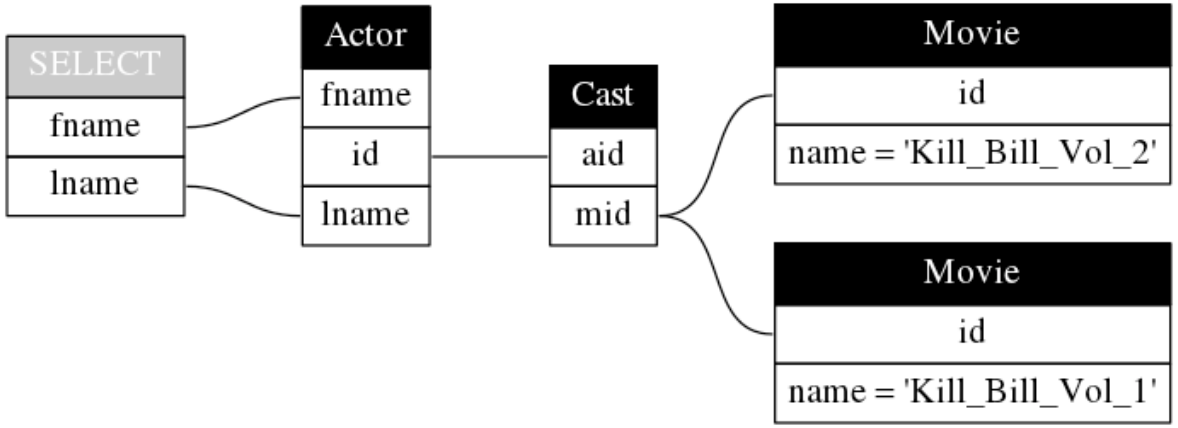
Quiz

IMDB (postgres)

Find the first/last names of all actors who appeared in both of the following movies: Kill Bill: Vol. 1 and Kill Bill: Vol. 2.

```
SELECT DISTINCT A.fname, A.lname
FROM     Actor A, Casts C, Movie M1, Movie M2
WHERE    M1.name = 'Kill Bill: Vol. 1'
   and   M2.name = 'Kill Bill: Vol. 2'
   and   M1.id = C.mid
   and   M2.id = C.mid
   and   C.pid = A.id
```

☹



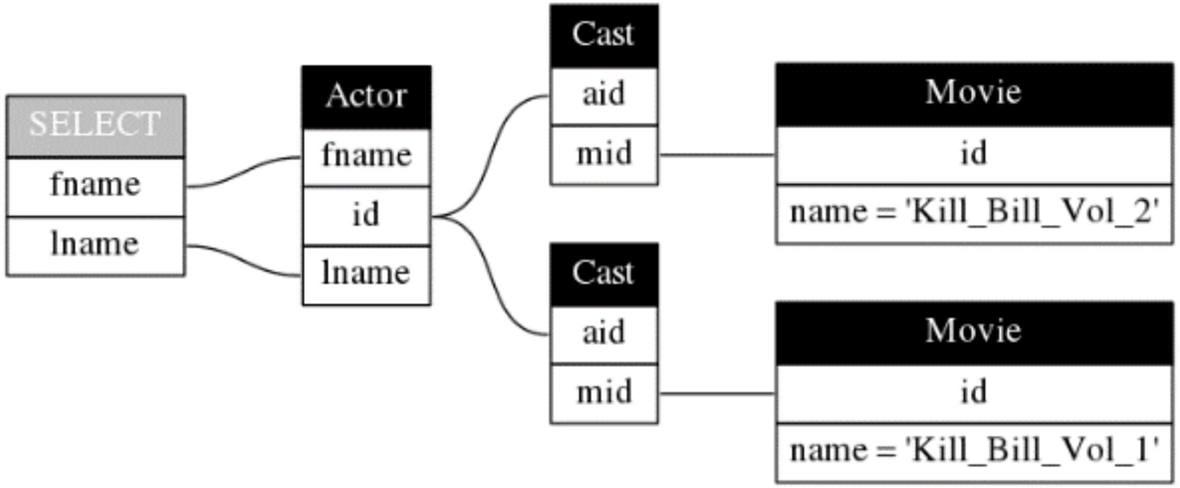
Picture Source: http://queryviz.com/online

Quiz

IMDB (postgres)

Find the first/last names of all actors who appeared in both of the following movies: Kill Bill: Vol. 1 and Kill Bill: Vol. 2.

```
SELECT DISTINCT A.fname, A.lname
FROM    Actor A, Casts C, Movie M1, Movie M2, Casts C2
WHERE   M1.name = 'Kill Bill: Vol. 1'
    and M2.name = 'Kill Bill: Vol. 2'
    and M1.id = C.mid
    and M2.id = C2.mid
    and C.pid = A.id
    and C2.pid = A.id
```



Picture Source: http://queryviz.com/online

Quiz

IMDB (postgres)

Find the first/last names of all actors who appeared in both of the following movies: Kill Bill: Vol. 1 and Kill Bill: Vol. 2.

```
SELECT   A.id, A.lname, A.fname,
FROM     actor A, cast C, movie M
WHERE    M.id = C.mid
   AND   A.id = C.pid
   AND   (M.name = 'Kill Bill: Vol. 1'
         OR M.name = 'Kill Bill: Vol. 2')
GROUP BY A.id, A.lname, A.fname
HAVING   count(M.id) > 1
```

What if an actor played two roles in Kill Bill 1?

Null Values

3-valued logic example

- Three logicians walk into a bar. The bartender asks: "Do all of you want a drink?"
- The 1st logician says: "I don't know."
- The 2nd logician says: "I don't know."
- The 3rd logician says: "Yes!"

Nulls in SQL

- Whenever we don't have a value, we can put a NULL

- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.

- The schema specifies for each attribute if it can be NULL (nullable attribute) or not

- How does SQL cope with tables that have NULLs ?

243

Null Values

- In SQL there are three Boolean values:
  - FALSE, TRUE, UNKNOWN

- If x= NULL then
  - Arithmetic operations produce NULL. E.g: 4*(3-x)/7
  - Boolean conditions are also NULL. E.g: x='Joe'
  - aggregates ignore NULL values

- Logical reasoning:
  - FALSE = 0
  - TRUE = 1
  - UNKNOWN = 0.5

x AND y = min(x,y)

x OR y = max(x,y)

NOT x = (1 – x)

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
NULL	6.5	170

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

Null Values: example

```
SELECT  *
FROM    Person
WHERE  (age < 25)
   and  (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

```
SELECT   *
FROM     Person
WHERE   age < 25  or age >= 25
```

Rule in SQL:
include only tuples that
yield TRUE

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25
```

← Unexpected behavior

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25 or age IS NULL
```

Test NULL
explicitly

Null Values and Aggregates

373

T

gid	val
1	NULL
1	NULL
2	a
2	a
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT gid,
       MAX(val) maxv,
       MIN(val) minv,
       COUNT(*) ctr,
       COUNT(val) ctv,
       COUNT(DISTINCT val) ctdv
FROM   T
GROUP BY gid
ORDER BY gid
```

→ ?

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT gid,
       MAX(val) maxv,
       MIN(val) minv,
       COUNT(*) ctr,
       COUNT(val) ctv,
       COUNT(DISTINCT val) ctdv
FROM   T
GROUP BY gid
ORDER BY gid
```

gid	maxv	minv	ctr	ctv	ctdv
1	NULL	NULL	2	0	0
2	z	B	5	4	3
3	Z	A	3	3	2

NULL is ignored by aggregate functions if you reference the column specifically. Exception: COUNT !

Inner Joins vs. Outer Joins

Alternaive Join Syntax

Item(name, category) 334
Purchase2(iName, store, month)

An "inner join":

```
SELECT  Item.name, Purchase2.store
FROM    Item, Purchase2
WHERE   Item.name = Purchase2.iName
```

Same as:

```
SELECT  Item.name, Purchase2.store
FROM    Item JOIN Purchase2 ON
        Item.name = Purchase2.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase2

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Illustration

361

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

An "inner join":

```
SELECT  *
FROM    English, French
WHERE   eid = fid
```

Same as:

```
SELECT  *
FROM    English JOIN French
ON      eid = fid
```

etext	eid	fid	ftext
One	1	1	Un
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz

"JOIN"
same as
"INNER JOIN"

Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

"FULL JOIN"
same as
"FULL OUTER JOIN"

```
SELECT  *
FROM    English FULL JOIN French
ON      English.eid = French.fid
```

~~SELECT *~~
~~FROM English JOIN French~~
~~ON eid = fid~~

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

SQLite does not support "FULL OUTER JOIN"s ☹ (but "LEFT JOIN")