

L02: SQL Basics

CS3200 Database design (fa18 s2)

<https://northeastern-datalab.github.io/cs3200/>

Version 9/10/2018

Announcements!

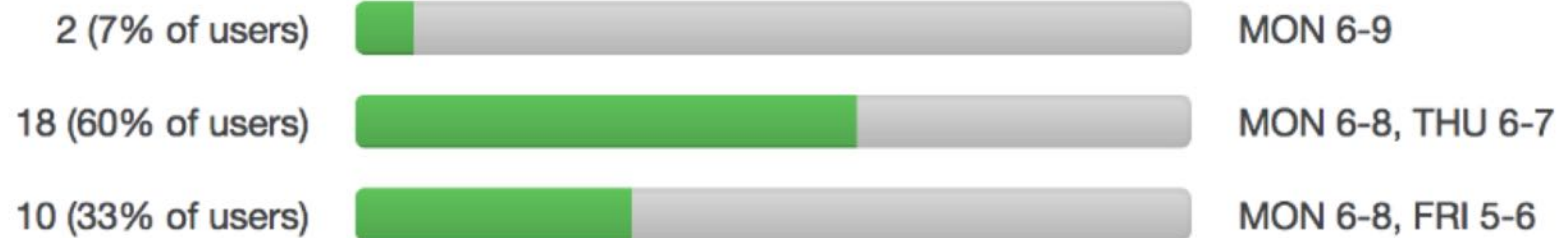
- Please pick up your name plates. Recall you find a template on our website
 - I tend to cold call on those in the back of the class and/or without name plate
- SQLite is optional for this class. PostgreSQL mandatory.
 - If you still have troubles, please ask Niklas for help during lecture
- Piazza: please be specific with your problem, so we can help you remotely.
 - Compare: "I can't install Jupyter Python. What should I do?" vs. "I get error message XYZ when I do ZYX following the tutorial slide 12. Here is a screenshot. What should I do?"
- Piazza: please feel free to post your lessons learned
 - e.g., "When I tried installing XYZ, I got error message YZX. Then I did ZXY and it worked"
- Poll results

Polls

Poll: Office Hours

We have three options for regular office hours with Niklas throughout the semester. We can revisit the times during the semester. Notice that most homeworks are due on Mondays. OHs are at 462 WVH.

A total of **30** vote(s) in **84** hours



Poll: 5min break during class

Which one do you prefer? Have a 5min break during the class, or having no break and finishing the class 5 min early (thus 11:20 instead of 11:25)

A total of **38** vote(s) in **84** hours



SQL: A Few Details on Syntax

- SQL commands are case insensitive:
 - SELECT = Select = select
 - Product = product, Category = category
- But values are not:
 - Different: 'Gadgets', 'gadgets'
 - (Notice: in general, but default settings will vary from DBMS to DBMS. E.g. MySQL is case insensitive. Just to be safe, always assume values to be case sensitive!)

```
WHERE LOWER(Category)='gadgets'
```

- Use single quotes for constants:
 - 'abc' - yes
 - "abc" - no (except MySQL and SQLite)

Eliminating Duplicates

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

```
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:

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

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
```



?

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 more "principled" 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."

Some history

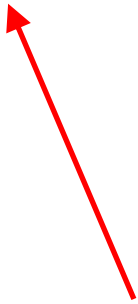
Some "birth-years"

- 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"
```

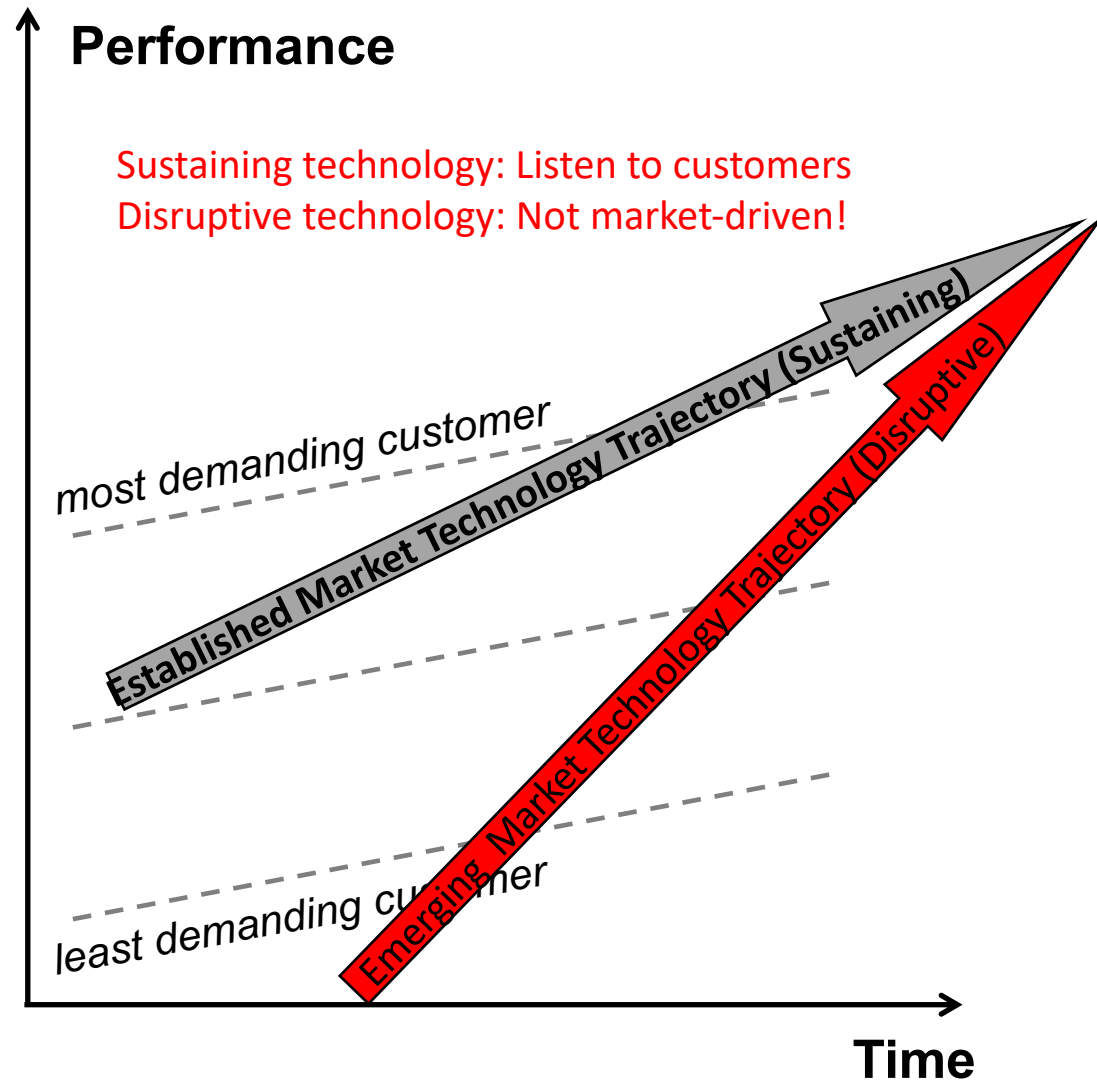
QUEL

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



Commercially not used anymore since ~1980

Disruptive Innovation



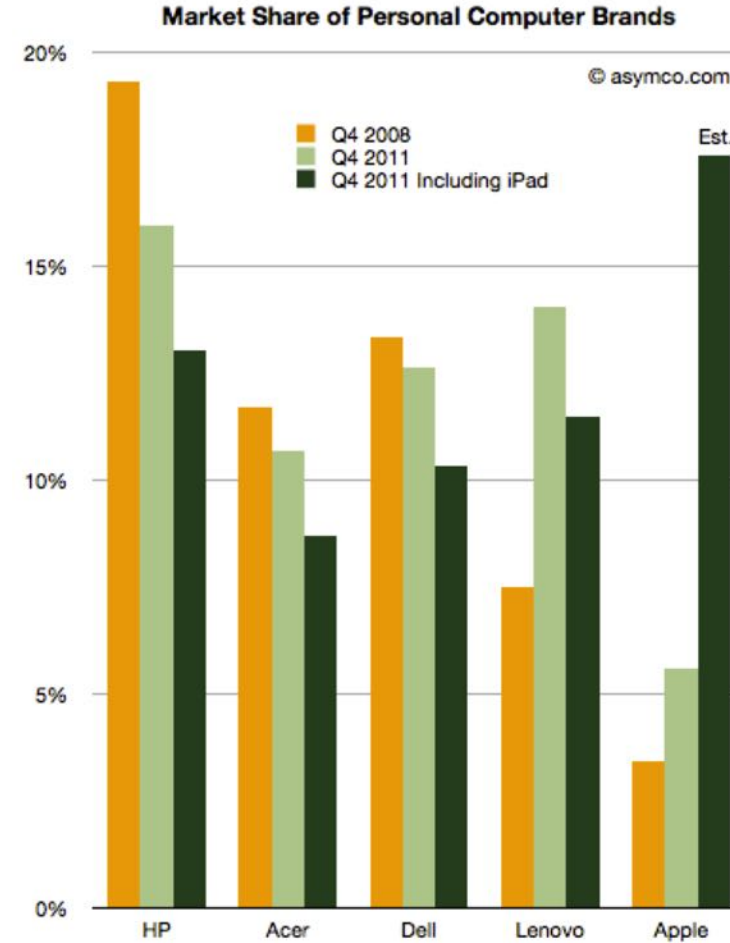
- Disruptive innovations are generally not acceptable for the mass market when they are introduced. Only the fringes of the market pick up the innovation in the first iteration
- It performs worse in one or more areas, but is typically simpler, more reliable, or more convenient than existing technologies.
- 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



What keyboards without keys can do...

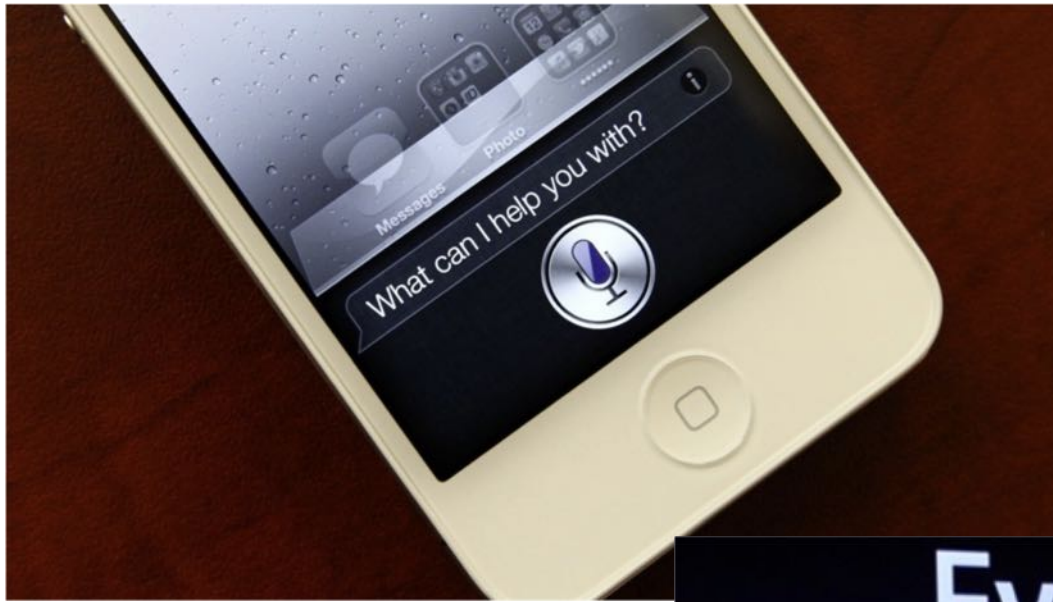


*In Feb 2016,
SwiftKey was
purchased by
Microsoft, for
250 M\$*

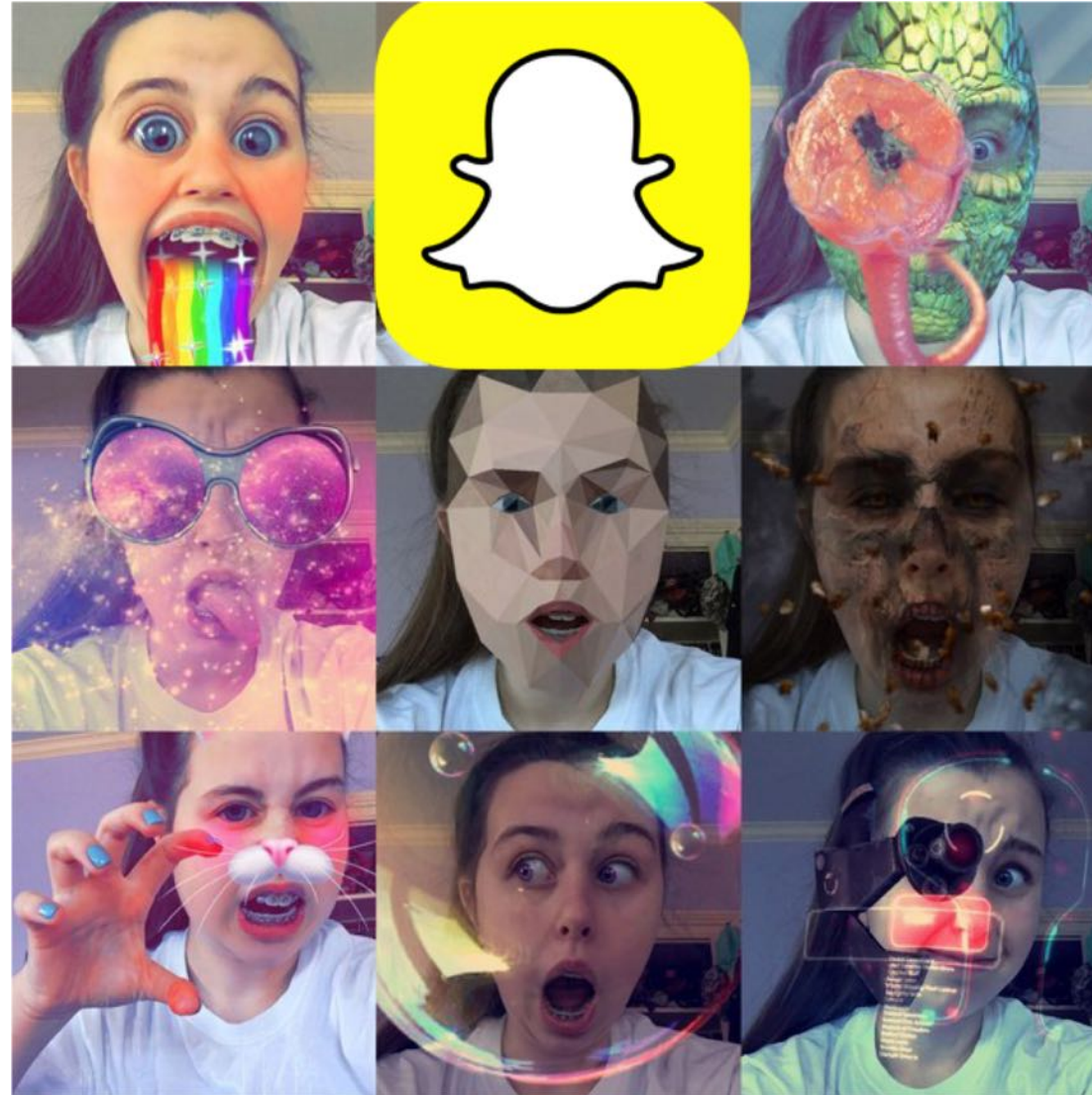


The keyboard of the future?





Keyboards? Do we need text to communicate?

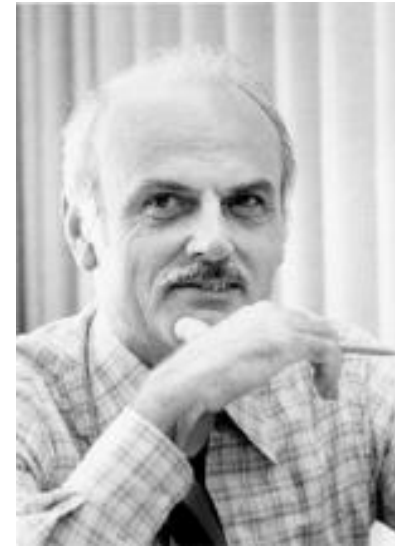


What is this? (1975)



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
 - Oracle (Relational Software), Microsoft, IBM,
- 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. Levin 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 non-inferential 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”).

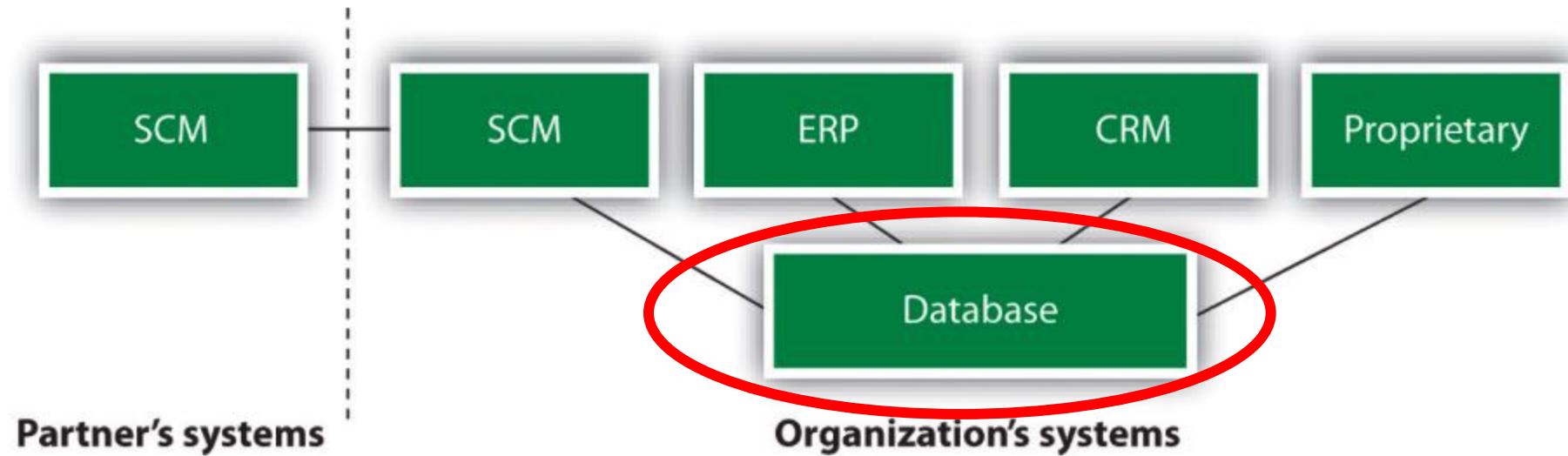
Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various 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

SQL and the relational model as standard



Databases we are using

Client/Server Architecture

- There is a single server that stores the database (called DBMS or RDBMS):
 - Could be a beefy system in the cloud (e.g. SQL Azure, or any DBMS on EC2)
 - But can be your own desktop...
 - ... or a huge cluster running a parallel DBMS
- Many clients run apps and connect to DBMS
 - E.g. PGAdmin
 - More realistically some Java, Python, or C++ program
- Clients “talk” to server using some protocol

DBMSs we discuss in this class

- SQLite (Optional)
 - 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 (Required)
 - popular and powerful open source database

Notice that SQLite setup is not required but recommended (adding to your maturity in setting up different databases).

We prefer PostgreSQL over MySQL because it has a more principled interpretation of SQL (and a powerful EXPLAIN command)

SQLite vs. PostgreSQL

SQLite

- 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

- open source & cross-platform
- 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 details later

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

sid	name	gpa
123	Alice	3.9
143	Bob	NULL

Say, Bob just enrolled in his first class.

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

Can a column with NULLs be a key?

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

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')
```

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')
```

**Selection
& Projection**

PName
SingleTouch
MultiTouch

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

	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"

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

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

	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"

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

LIKE

LIKE: Simple String Pattern Matching

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'
```

LIKE: Simple String Pattern Matching

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.

LIKE: Simple String Pattern Matching

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'
```

LIKE: Simple String Pattern Matching

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.

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 any values within the range	LIKE equal to (pattern)
		'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)

Joins

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

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is a
foreign key vs.
a key here?

Keys and Foreign Keys

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

**Foreign
key**

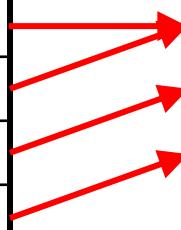
Key **Company**

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

What is a
foreign key vs.
a key here?

Referential Integrity

Product				Company		
<u>PName</u>	Price	Category	Manufacturer	<u>CName</u>	StockPrice	Country
Gizmo	\$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	\$29.99	Gadgets	GizmoWorks	Canon	65	Japan
SingleTouch	\$149.99	Photography	Canon	Hitachi	15	Japan
MultiTouch	\$203.99	Household	Hitachi			



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

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

Referential Integrity

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

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```

graph LR
    subgraph Product
        direction TB
        P1[Gizmo]
        P2[Powergizmo]
        P3[SingleTouch]
        P4[MultiTouch]
    end
    subgraph Company
        direction TB
        C1[GizmoWorks]
        C2[Canon]
        C3[Hitachi]
    end
    P1 --> C1
    P2 --> C1
    P3 --> C2
    P4 --> C3
  
```

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');

Gizmo	\$14.99	Gadgets	Hitachi
-------	---------	---------	---------

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

Referential Integrity

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

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```

graph LR
    subgraph Product
        direction TB
        P1[Gizmo]
        P2[Powergizmo]
        P3[SingleTouch]
        P4[MultiTouch]
    end
    subgraph Company
        direction TB
        C1[GizmoWorks]
        C2[Canon]
        C3[Hitachi]
    end
    P1 --> C1
    P2 --> C1
    P3 --> C2
    P4 --> C3
  
```

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

Referential Integrity

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

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```

graph LR
    subgraph Product
        direction TB
        P1[Gizmo]
        P2[Powergizmo]
        P3[SingleTouch]
        P4[MultiTouch]
    end
    subgraph Company
        direction TB
        C1[GizmoWorks]
        C2[Canon]
        C3[Hitachi]
    end
    P1 --> C1
    P2 --> C1
    P3 --> C2
    P4 --> C3
  
```

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

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

SuperTouch	\$249.99	Computer	NewCom
------------	----------	----------	--------

Referential Integrity

Product				Company		
<u>PName</u>	Price	Category	Manufacturer	<u>CName</u>	StockPrice	Country
Gizmo	\$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	\$29.99	Gadgets	GizmoWorks	Canon	65	Japan
SingleTouch	\$149.99	Photography	Canon	Hitachi	15	Japan
MultiTouch	\$203.99	Household	Hitachi			

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

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

violate Foreign Key constraint

SuperTouch	\$249.99	Computer	NewCom
------------	----------	----------	--------

Referential Integrity

Product				Company		
<u>PName</u>	Price	Category	Manufacturer	<u>CName</u>	StockPrice	Country
Gizmo	\$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	\$29.99	Gadgets	GizmoWorks	Canon	65	Japan
SingleTouch	\$149.99	Photography	Canon	Hitachi	15	Japan
MultiTouch	\$203.99	Household	Hitachi			

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');

Gizmo	\$14.99	Gadgets	Hitachi
-------	---------	---------	---------

violates Key constraint

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

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

SuperTouch	\$249.99	Computer	NewCom
------------	----------	----------	--------

violate Foreign Key constraint

Delete from Company
where CName = 'Canon';

Referential Integrity

Product				Company		
<u>PName</u>	Price	Category	Manufacturer	<u>CName</u>	StockPrice	Country
Gizmo	\$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	\$29.99	Gadgets	GizmoWorks	Canon	65	Japan
SingleTouch	\$149.99	Photography	Canon	Hitachi	15	Japan
MultiTouch	\$203.99	Household	Hitachi			

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');

Gizmo	\$14.99	Gadgets	Hitachi
-------	---------	---------	---------

violates Key constraint

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

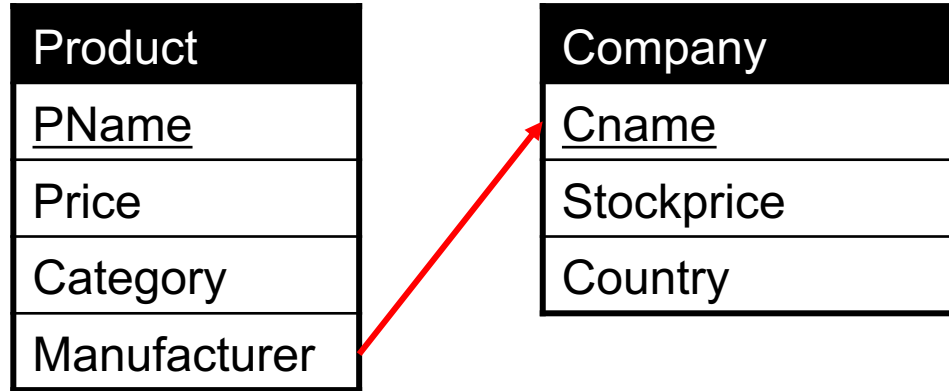
Insert into Product values ('SuperTouch', 249.99, 'Computer', 'NewCom');

SuperTouch	\$249.99	Computer	NewCom
------------	----------	----------	--------

violate Foreign Key constraint

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

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

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!*

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

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

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

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'
```

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

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