# Introduction to Database Systems

Lecture 2: Data Models & SQL

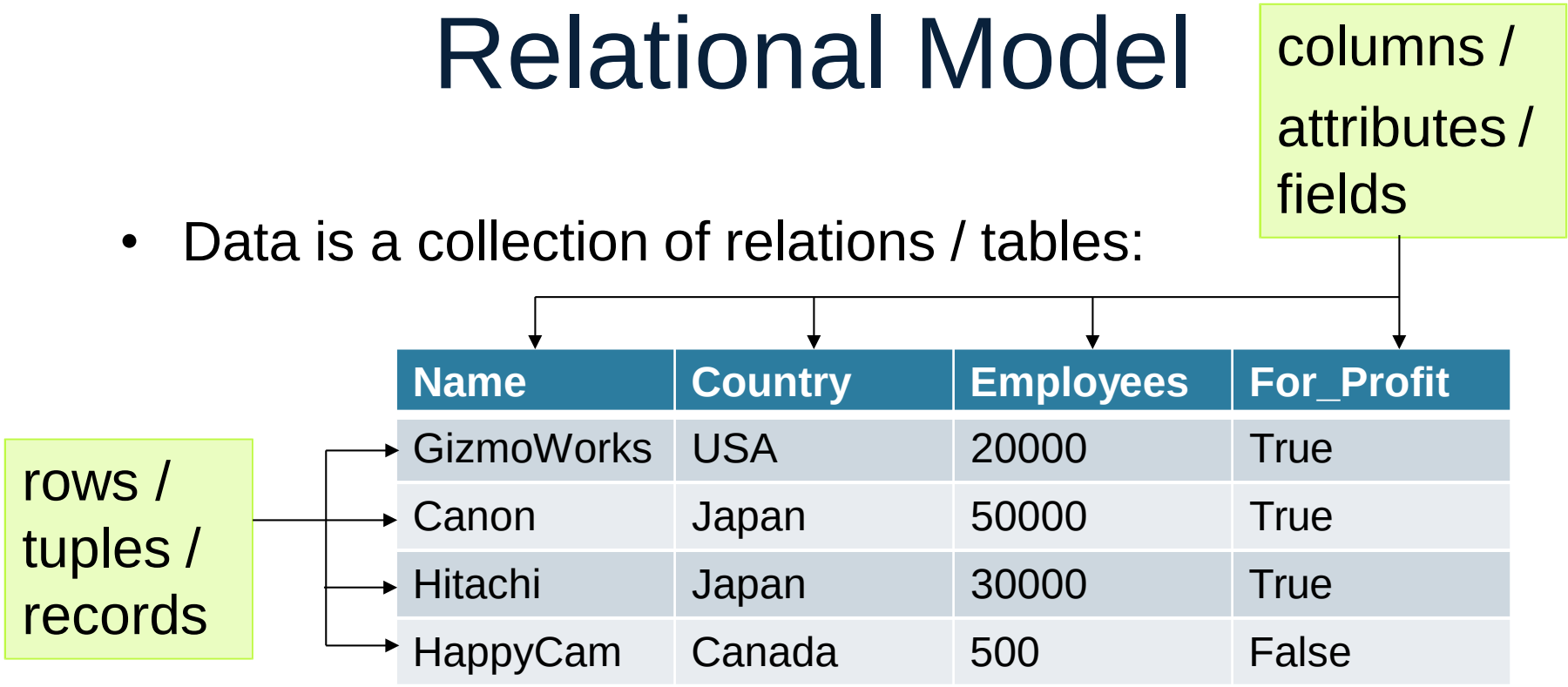
(Ch. 2.1-2.3)

# Data Models

- language / notation for talking about data
- models we will use:
  - relational: data is a collection of tables
  - semi-structured: data is a tree
- other models:
  - key-value pairs: used by NoSQL systems
  - graph data model: used by RDF (semi-structured can also do)
  - object oriented: often layered on relational, J2EE

# Relational Model

- Data is a collection of relations / tables:




| Name | Country | Employees | For_Profit |
|---|---|---|---|
| GizmoWorks | USA | 20000 | True |
| Canon | Japan | 50000 | True |
| Hitachi | Japan | 30000 | True |
| HappyCam | Canada | 500 | False |

- mathematically, relation is a set of tuples
  - each tuple appears 0 or 1 times in the table
  - order of the rows is unspecified

# Relational Schema

- Each column has a “domain” (or type)
  - SQL has Java-like types for numbers, strings, etc.
  - domain is a constraint on the data allowed in the table
- Names and types part of the “schema” of the table:

```
Company(Name: string, Country: string,
        Employees: int, For_Profit: boolean)
```

- Particular data is an “instance” of that relation
  - data changes over time
  - DBMS usually just stores the current instance

# Keys

- Key = subset of columns that uniquely identifies tuple
- Another constraint on the table
  - no two tuples can have the same values for those columns
- Examples:
  - Movie(title, year, length, genre): key is (title, year)
  - what is a good key for Company?
- Part of the schema (book notation is underline):

```
Company(Name: string, Country: string,
        Employees: int, For_Profit: boolean)
```

# Keys (cont.)

- Can have multiple keys for a table
- Only one of those keys may be “primary”
  - DBMS often makes searches by primary key fastest
  - other keys are called “secondary”
- “Foreign key” is a column (or columns) whose value is a key of another table
  - i.e., a reference to another row in another table

# SQL (“sequel”)

- Standard query language for relational data
  - used for databases in many different contexts
  - inspires query languages for non-relational (e.g. SQL++)
- Everything not in quotes (‘…’) is case insensitive
- Provides standard types. Examples:
  - numbers: INT, FLOAT, DECIMAL(p,s)
    - DECIMAL(p,s): Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
  - strings: CHAR(n), VARCHAR(n)
    - CHAR(n): Fixed-length n
    - VARCHAR(n): Variable length. Maximum length n

# SQL (“sequel”) – Cont.

- Provides standard types. Examples:
  - BOOLEAN
  - DATE, TIME, TIMESTAMP
    - DATE: Stores year, month, and day values
    - TIME: Stores hour, minute, and second values
    - TIMESTAMP: Stores year, month, day, hour, minute, and second values
- Additional types differ by vendor:
  - SQLite: http://www.sqlite.org/datatype3.html

# SQL statements

- create table …
- drop table ...
- alter table ... add/remove ...
- insert into ... values ...
- delete from ... where ...
- update ... set ... where ...
- select … from … where

# create table …

```
CREATE TABLE Company(
   name VARCHAR(20) PRIMARY KEY,
   country VARCHAR(20),
   employees INT,
   for_profit CHAR(1));
```

# Multi-column Keys

- This makes name a key:

```
CREATE TABLE Company(
   name VARCHAR(20) PRIMARY KEY,
   country VARCHAR(20),
   employees INT,
   for_profit BOOLEAN);
```

- How can we make a key on name & country?

# Multi-column Keys

- Syntax change if a primary key has multiple columns:

```
CREATE TABLE Company(
   name VARCHAR(20) PRIMARY KEY,
   country VARCHAR(20),
   employees INT,
   for_profit BOOLEAN,
   PRIMARY KEY (name, country));
```

goes away

added

# Multi-column Keys (2)

- Likewise for secondary keys:

```
CREATE TABLE Company(  name VARCHAR(20)
   UNIQUE,
   country VARCHAR(20),
   employees INT,
   for_profit BOOLEAN,
   UNIQUE (name, country));
```

goes away

added

# Multi-column Keys (3)

- This makes manufacturer a foreign key:

```
CREATE TABLE Product(
   name VARCHAR(20),
   price DECIMAL(10,2),
   manufacturer VARCHAR(20)
     REFERENCES Company(name));
```

good idea to include target column name

# Multi-column Keys (3)

- Similar syntax for foreign keys:

```
CREATE TABLE Product(
   name VARCHAR(20),
   price DECIMAL(10,2),
   manu_name VARCHAR(20),
   manu_co VARCHAR(20),
   FOREIGN KEY (manu_name, manu_co)
     REFERENCES Company(name, country));
```

now need both name & country

added

# UNIQUE

- PRIMARY KEY adds implicit “NOT NULL” constraint while UNIQUE does not
  - you would have to add this explicitly for UNIQUE:

```
CREATE TABLE Company(
   name VARCHAR(20) NOT NULL, …
   UNIQUE (name));
```

- You almost always want to do this (in real schemas)
  - SQL Server behaves strangely with NULL & UNIQUE
  - otherwise, think through NULL for every query
  - you can remove the NOT NULL constraint later

# drop table ...

```
DROP TABLE Company;
```

# alter table ... add/remove ...

```
ALTER TABLE Company
ADD CEO VARCHAR(20);
```

# insert into ... values ...

```
INSERT INTO Company VALUES
('GizmoWorks', 'USA',  20000, 'y');
```

# One Way to Input Data

- Write a program that outputs SQL statements:

```
for (int a = 1; a <= 50; a++)
  for (int b = 1; b <= 50; b++)
    System.out.format(
      "INSERT INTO T VALUES (%d,%d);\n",
      a, b);
```

- Feed those into SQLite:

```
sqlite3 foo.db < inputs.sql
```

# Demo: MakeTriples.java

# Warning

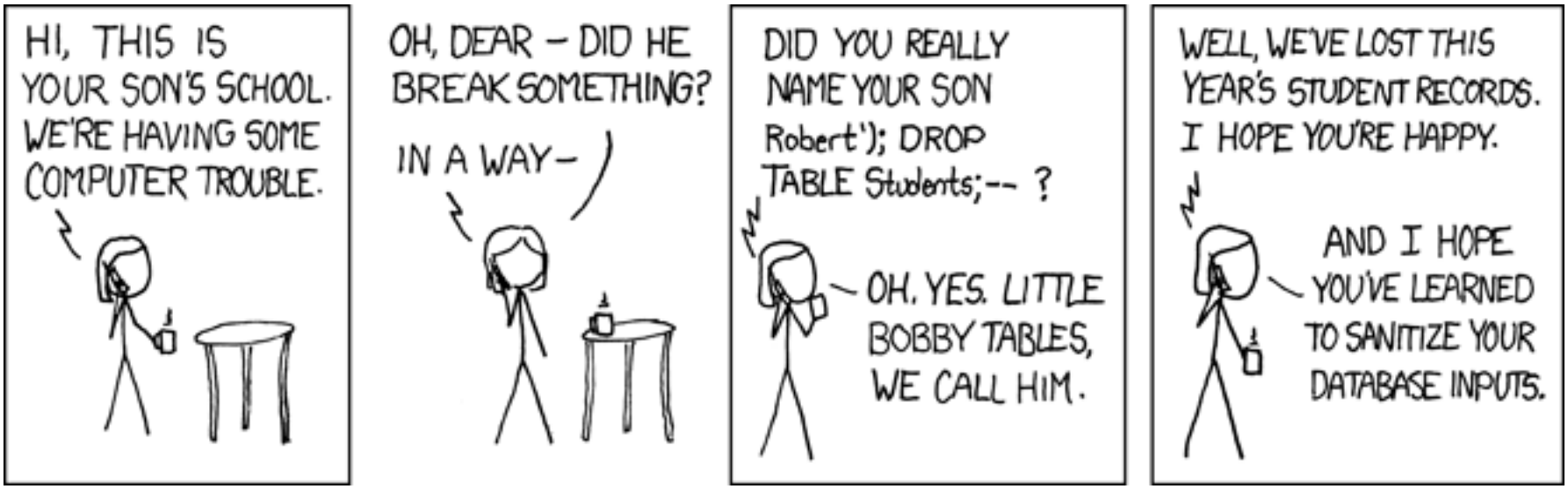
- Be very careful when doing this with strings:

```
System.out.format(
  ”INSERT INTO T2 VALUES (%d, ‘%s’);”,
  3, ”O’Shaughnessy”);
```

Becomes:

```
  INSERT INTO T2 VALUES (3, ‘O’Shaughnessy’);
```

which is a syntax error in this case




https://xkcd.com/327/

# Warning (cont)

- Be very careful when doing this with strings:

```
System.out.format(
  ”INSERT INTO T VALUES (%d, ‘%s’);”,
  3, ”O’Shaughnessy”);
```

- This allows a SQL injection attack!
  - Must check for quotes and escape (or disallow) them.
  - We’ll see safer ways to do this using JDBC
- DBMSs usually have faster ways to input data
  - SQLite has .import (try with .mode csv)

# delete from ... where ...

```
DELETE FROM Company
where name = 'GizmoWorks';
```

# update ... set ... where ...

```
UPDATE Company
SET employees = employees + 120
where name = 'GizmoWorks';
```

# select ... from ... where ...

```
SELECT * FROM Company
where name = 'GizmoWorks';
```

# DISTINCT and ORDER BY

- Query results do not have to be relations
  - i.e., they can have duplicate rows
  - remove them using DISTINCT
- Result order is normally unspecified
  - choose an order using ORDER BY
  - e.g., ORDER BY country, cname
  - e.g., ORDER BY price ASC, pname DESC
- Examples in lec03-sql-basics.sql

# Demo on Sqlite

- E.g., type sqlite3 in Cygwin
- .exit - exit from sqlite3

# SQLite Uses

- SQLite is just a library
- Can be used as part of any C/C++/Java program
  - ex: could be used in an iPhone app
- Can be used in Chrome & Safari
  - no support in Firefox or IE

# Physical Data Independence

- SQL doesn't specify how data is stored on disk
- No need to think about encodings of data types
  - ex: DECIMAL(10,2)
  - ex: VARCHAR(255)
    - does this need to use 255 bytes to store 'hello'?
- No need to think about how tuples are arranged
  - ex: could be row- or column-major ordered
  - (Most DBMSs are row-ordered, but Google's BigQuery is column-oriented.)

# SQLite Gotchas

- Allows NULL keys
  - At most one tuple can have NULL in the key
  - According to the SQL standard, PRIMARY KEY should always imply NOT NULL, but this is not the case in SQLite
- Does not support boolean or date/time columns
- Doesn’t always enforce domain constraints!
  - will let you insert a string where an INT is expected
- Doesn’t enforce foreign key constraints by default
- Etc…