

Introduction to Data Management

CSE 414

Lecture 3: More SQL

(including most of Ch. 6.1-6.2)

Overload: <https://goo.gl/forms/2pFBteeXg5L7wdC12>

Announcements

- WQ2 will be posted tomorrow and due on Oct. 17, 11pm
- HW2 will be posted tomorrow and due on Oct. 16, 11pm

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

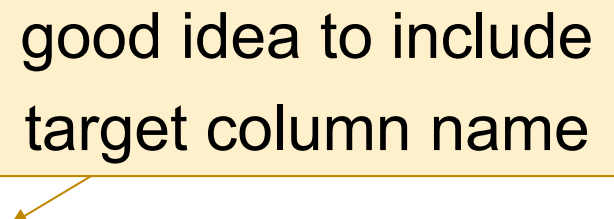
added

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

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

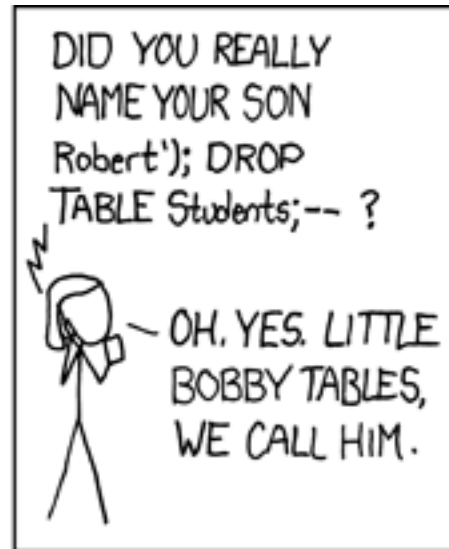
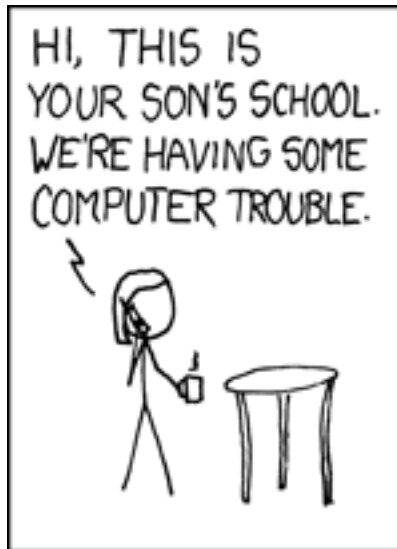
- Be very careful when doing this with strings:

```
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`)

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

Demo: websql.html in **Chrome**

(Note: this HTML/JS code is out of class scope)

Also selection & projection examples
(see `lec03-sql-basics.sql`)

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...

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`

Joins

- Can use data from multiple tables:

```
SELECT pname, price  
FROM Product, Company  
WHERE manufacturer = cname AND  
        country = 'Japan' AND  
        price < 150;
```

- This is a selection and projection of the “join” of the Product and Company relations.

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

Cname	Country	Pname	Price	Manufacturer
Canon	Japan	SingleTouch	149.99	Canon
GizmoWorks	USA	Gizmo	19.99	GizmoWorks
		PowerGizmo	29.99	GizmoWorks

('Canon', 'Japan', 'SingleTouch', 149.99, 'Canon')

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

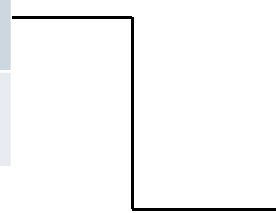
Cname	Country	Pname	Price	Manufacturer
Canon	Japan	SingleTouch	149.99	Canon
GizmoWorks	USA	Gizmo	19.99	GizmoWorks
		PowerGizmo	29.99	GizmoWorks

(‘Canon’, ‘Japan’, ‘Gizmo’, 19.99, ‘GizmoWorks’)

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

Cname	Country	Pname	Price	Manufacturer
Canon	Japan	SingleTouch	149.99	Canon
GizmoWorks	USA	Gizmo	19.99	GizmoWorks
		PowerGizmo	29.99	GizmoWorks

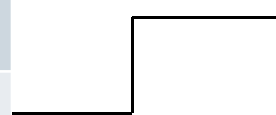


('Canon', 'Japan', 'PowerGizmo', 29.99, 'GizmoWorks')

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

Cname	Country	Pname	Price	Manufacturer
Canon	Japan	SingleTouch	149.99	Canon
GizmoWorks	USA	Gizmo	19.99	GizmoWorks
		PowerGizmo	29.99	GizmoWorks



('GizmoWorks', 'USA', 'SingleTouch', 149.99, 'Canon')

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

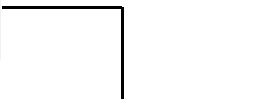
Cname	Country	Pname	Price	Manufacturer
Canon	Japan	SingleTouch	149.99	Canon
GizmoWorks	USA	Gizmo	19.99	GizmoWorks
		PowerGizmo	29.99	GizmoWorks

('GizmoWorks', 'USA', 'Gizmo', 19.99, 'GizmoWorks')

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

Cname	Country	Pname	Price	Manufacturer
Canon	Japan	SingleTouch	149.99	Canon
GizmoWorks	USA	Gizmo	19.99	GizmoWorks
		PowerGizmo	29.99	GizmoWorks



('GizmoWorks', 'USA', 'PowerGizmo', 29.99, 'GizmoWorks')

Interpreting Joins

- A JOIN B produces one row for every pair of rows
 - one row from A and one row from B

Cname	Country
Canon	Japan
GizmoWorks	USA

JOIN

Pname	Price	Manufacturer
SingleTouch	149.99	Canon
Gizmo	19.99	GizmoWorks
PowerGizmo	29.99	GizmoWorks

- This join produces 6 different rows
 - in general, # rows in join is (# rows in A) * (# rows in B)
 - number of rows often **much smaller** after selection...
 - DBMS will do everything in its power to not compute A JOIN B

Interpreting Joins (2)

- Can think of a join in terms of code:

```
for every row C in Company {  
  for every row P in Product {  
    if (P.manufacturer = C.cname and  
        C.country = 'Japan' and  
        P.price < 150.00)  
      output (C.cname, C.country,  
              P.pname, P.price, P.category,  
              P.manufacturer);  
  }  
}
```

Types of Joins

- We usually think of the selection as part of the join
 - e.g., manufacturer = cname and country = 'Japan' and ...
 - called the “join predicate”
- Join without a predicate is cross product / cross join
- Special names depending on predicate
 - natural join if “=” between pairs of columns with same name
 - with well chosen col names, many joins become natural
- These are “inner” joins. We will discuss outer later...

Join Examples

- See `lec03-sql-basics.sql...`