Hands-on SQL

LBSC 690: Jordan Boyd-Graber

University of Maryland

October 22, 2012

COLLEGE OF INFORMATION STUDIES

Goals

- Practical Database Manipulation
- Overview of Available Database Systems
- Introducing Assignment 3

Outline

1. DBMS
2. Connecting to a Remote Computer
3. Queries on an Existing Database
4. Creating Your Own Database
5. Assignment 3

Database Management Software

Software that

- MySQL - Free, used by Facebook and Wikipedia, support by Oracle (via Sun)
- Microsoft Access - Comes with Office
- Google App Engine - Free, online
- Microsoft SQL
- Oracle - Expensive, very full-featured
- DB2 - By IBM, common for legacy systems
- SQLite - Very lightweight, free
- PostgreSQL - Free, open source
- FoxPro / dBASE - Based on IBM databases

Database Management Software

Software that

- MySQL - Free, used by Facebook and Wikipedia, support by Oracle (via Sun)
- Microsoft Access - Comes with Office
- Google App Engine - Free, online
- Microsoft SQL
- Oracle - Expensive, very full-featured
- DB2 - By IBM, common for legacy systems
- SQLite - Very lightweight, free
- PostgreSQL - Free, open source
- FoxPro / dBASE - Based on IBM databases

Variable Types

- INTEGER - Counting numbers (and negatives)
- FLOAT - Rational numbers (e.g. 3.1415)
- VARCHAR - Strings
- BLOB - Arbitrary data (we won't use this)

Why is this important?

As you design your database, you need to choose which datatype to use! Use integers whenever you can.

Outline

1. DBMS

2. Connecting to a Remote Computer

3. Queries on an Existing Database

4. Creating Your Own Database

5. Assignment 3

Working on Another Computer

- The database runs on OIT's server
- You have your own directory (where you put your webpages)
- Exposure to command line interface

OS X

- Already installed
- Go to “Applications”
- Then “Utilities”
- Look for Terminal

What to type

```
ssh USERNAME@terpconnect.umd.edu
```

Windows: PuTTY

http://www.chiark.greenend.org.uk/ sgtatham/putty/download.html

- Might have come with WinSCP
- Similar interface

Outline

1 DBMS

2 Connecting to a Remote Computer

3 Queries on an Existing Database

4 Creating Your Own Database

5 Assignment 3

- I grabbed the index of Project Gutenberg
- Dumped first 500k lines of Dublin Core XML into SQLite database
- No cleanup; ignored all errors (just like real world!)

Source Data

```
<pgterms:etext rdf:ID="etext8476">
  <dc:publisher>&pg;</dc:publisher>
  <dc:title rdf:parseType="Literal">Life on the Mississippi,
    Part 6.</dc:title>
  <dc:creator rdf:parseType="Literal">Twain, Mark, 1835-1910</dc:creator>
  <pgterms:friendlytitle rdf:parseType="Literal">Life on the Mississippi,
    Part 6. by Mark Twain</pgterms:friendlytitle>
  <dc:language><dcterms:ISO639-2><rdf:value>en</rdf:value></dcterms:ISO639-2></dc:language>
  <dc:subject>
    <rdf:Bag>
      <rdf:li><dcterms:LCSH><rdf:value>Authors, American -- 19th century -- Biography
       </rdf:value></dcterms:LCSH></rdf:li>
      <rdf:li><dcterms:LCSH><rdf:value>Mississippi River -- Description and travel
       </rdf:value></dcterms:LCSH></rdf:li>
      <rdf:li><dcterms:LCSH><rdf:value>Mississippi River Valley -- Social life and customs -
       </rdf:value></dcterms:LCSH></rdf:li>
      <rdf:li><dcterms:LCSH><rdf:value>Pilots and pilotage -- Mississippi River</rdf:value><
      <rdf:li><dcterms:LCSH><rdf:value>Twain, Mark, 1835-1910 -- Travel -- Mississippi River
       </rdf:value></dcterms:LCSH></rdf:li>
    </rdf:Bag>
  </dc:subject>
  <dc:subject><dcterms:LCC><rdf:value>PS</rdf:value></dcterms:LCC></dc:subject>
  <dc:created><dcterms:W3CDTF><rdf:value>2004-07-09
   </rdf:value></dcterms:W3CDTF></dc:created>
  <dc:rights rdf:resource="&lic;" />
</pgterms:etext>
```

Getting Data into DBMS

- Download the database for Assignment 3

```
wget http://umiacs.umd.edu/ jbg/teaching/LBSC_690_2010/books.db
```

  - wget is a program to get files from the web
  - CAUTION: wget will not overwrite existing files
  - If you want to delete the database, do "rm books.db"

- Start SQLite

```
sqlite3 books.db
```

Optional: Make output look pretty

```
.header on
.mode column
```

- By default, SQLite assumes you know the data you're looking at
- Uses space in most efficient way possible
- Column format better for us

What tables are there?

.tables

```
sqlite> .tables
authors         books           categories      category_map
```

What columns does a table have?

```
Pragma table_info(authors);
```

Special SQLite Command

```
sqlite> Pragma table_info(authors);
cid         name        type        notnull     dflt_value  pk
----------  ----------  ----------  ----------  ----------  ----------
0           authorID    integer     0                       1
1           year_born   integer     0                       0
2           year_died   integer     0                       0
3           name        text        0                       0
```

What columns does a table have?

Pragma table_info(authors);

Table name

```
sqlite> Pragma table_info(authors);
cid         name        type        notnull     dflt_value  pk
__________  __________  __________  __________  __________  __________
0           authorID    integer     0                       1
1           year_born   integer     0                       0
2           year_died   integer     0                       0
3           name        text        0                       0
```

Viewing contents of a table

```
select * from authors limit 8;
```

- Asterisk means "all columns"
- "from" is followed by a table name
- "limit 8" (optional) means we only see eight results

```
sqlite> select * from authors limit 8;
authorID    year_born   year_died   name
----------  ----------  ----------  ---------------
0           1862        1932        Parker, Gilbert
1                                   Unknown
2           1887        1969        Dell, Floyd
3
4           1852        1907        Skinner, Charle
5           1855        1919        Wilcox, Ella Wh
6           1817        1888        Storm, Theodor
```

Filtering and Sorting

select * from authors where year_born is not null order by name limit 8;

- Asterisk means "all columns"
- year_born must be **something**
- sorts by name
- "from" is followed by a table name
- "limit 8" (optional) means we only see eight results

```
sqlite> select * from authors where year_born is not null order by name limit 8;
authorID    year_born   year_died   name
----------  ----------  ----------  ----------
8883        1821        1893        A. L. O. E.
7919        1877        1970        Aaberg, J.
4171        1866        1930        Aakjr, Je
3129        1863        1953        Aanrud, Han
8738        1888        1948        Aaronsohn,
7307        1838        1910        Abba, Giuse
2320        1840        1905        Abbe, Ernst
1957        1842        1911        Abbey, Henr
```

Filtering and Sorting

```
select * from authors where year_born is not null order by name limit 8;
```

- Asterisk means “all columns”
- year_born must be **something**
- sorts by name
- “from” is followed by a table name
- “limit 8” (optional) means we only see eight results

```
sqlite> select * from authors where year_born is not null order by name limit 8;
authorID    year_born   year_died   name
----------  ----------  ----------  ----------
8883        1821        1893        A. L. O. E.
7919        1877        1970        Aaberg, J.
4171        1866        1930        Aakjr, Je
3129        1863        1953        Aanrud, Han
8738        1888        1948        Aaronsohn,
7307        1838        1910        Abba, Giuse
2320        1840        1905        Abbe, Ernst
1957        1842        1911        Abbey, Henr
```

Counting and Arithmetic

```
select count(*) from authors;
```

- Available functions: SUM, MIN, MAX, COUNT, AVG
- "from" is followed by a table name
- "limit 8" (optional) means we only see eight results

```
sqlite> select count(*) from authors;
count(*)
----------
10691
```

How long does the average author live?

select count(), avg(year_died) - avg(year_born) from authors where year_born is not null;

- Expression of functions - this is okay!
- Filtering is fine

```
sqlite> select count(*), avg(year_died) - avg(year_born) from authors where year\_born is no
count(*)      avg(year_died) - avg(year_born)
------------  ---------------------------------
6356          69.753775959723
```

- Why is the count different?
- Does this number seem plausible?
- Why are data missing?
  - Unknown author
  - Unknown birthdate
  - Transcription errors
- Would missing data make this lower or higher?

How long does the average author live?

select count(), avg(year_died) - avg(year_born) from authors where year_born is not null;

- Expression of functions - this is okay!
- Filtering is fine

```
sqlite> select count(*), avg(year_died) - avg(year_born) from authors where year\_born is no
count(*)    avg(year_died) - avg(year_born)
----------  -------------------------------
6356        69.753775959723
```

- Why is the count different?
- Does this number seem plausible?
- Why are data missing?
  - Unknown author
  - Unknown birthdate
  - Transcription errors
- Would missing data make this lower or higher?

How many authors lived to be an age that's a multiple of 7?

select count(*), year_died - year_born as age from authors where age % 7 == 0;

- Shorthand
- Filtering is fine

```
sqlite> select count(*), year_died - year_born as age from authors where age \% 7 == 0 limit
count(*)    age
----------  ----------
906         56
```

- Why is the count different?
- Does this number seem plausible?
- Why are data missing?
  - Unknown author
  - Unknown birthdate
  - Transcription errors
- Would missing data make this lower or higher?

How many authors lived to be an age that's a multiple of 7?

```
select count(*), year_died - year_born as age from authors where age % 7
== 0;
```

- Shorthand
- Filtering is fine

```
sqlite> select count(*), year_died − year_born as age from authors where age \% 7 == 0 limit
count(*)    age
----------  ----------
906         56
```

- Why is the count different?
- Does this number seem plausible?
- Why are data missing?
  - Unknown author
  - Unknown birthdate
  - Transcription errors
- Would missing data make this lower or higher?

Combining restrictions and bad data

```
sqlite> select * from authors where year_died - year_born > 100;
authorID    year_born   year_died   name
----------  ----------  ----------  ------------------------------------
5328        1868        1970        Enock, C. Reginald (Charles Reginald)
6843        1847        1948        Jennings, Frederick Charles
7706        1856        1963        Brown, Arthur Judson
7941        176         1849        Gallatin, Albert
9423        1895        1998        J nger, Ernst
```

- Probably a typo.
- Let's remove that record from our calculation (as an example of multiple constraints)

```
sqlite> select count(*), avg(year_died) - avg(year_born) from authors where year_born
   ...> is not null and not name = "Gallatin,␣Albert";
count(*)    avg(year_died) - avg(year_born)
----------  -------------------------------
6355        69.5014948859166
```

Who wrote the most books?

```
select count(*) as num_books, author from books group by author order
by num_books desc limit 8;
```

- Count the number of records and call it num_books so we can use it to sort later
- Shows the author id
- Do each count per-author
- Sort by the number of books each author wrote
- “limit 8” (optional) means we only see eight results

```
sqlite> select count(*) as num_books, author from books group by author order by num_books d
num_books    author
----------   ----------
2114         69
944          7
907          3
555          40
276          188
217          33
203          24
164          686
```

Who wrote the most books?

select count(*) as num_books, author from books group by author order by num_books desc limit 8;

- Count the number of records and call it num_books so we can use it to sort later
- Shows the author id
- Do each count per-author
- Sort by the number of books each author wrote
- “limit 8” (optional) means we only see eight results

```
sqlite> select count(*) as num_books, author from books group by author order by num_books d
num_books   author
----------  ----------
2114        69
944         7
907         3
555         40
276         188
217         33
203         24
164         686
```

Who wrote the most books?

select count(*) as num_books, author from books group by author order by num_books desc limit 8;

- Count the number of records and call it num_books so we can use it to sort later
- Shows the author id
- Do each count per-author
- Sort by the number of books each author wrote
- "limit 8" (optional) means we only see eight results

```
sqlite> select count(*) as num_books, author from books group by author order by num_books d
num_books   author
----------  ----------
2114        69
944         7
907         3
555         40
276         188
217         33
203         24
164         686
```

Who wrote the most books?

select count(*) as num_books, author from books group by author order by num_books desc limit 8;

- Count the number of records and call it num_books so we can use it to sort later
- Shows the author id
- Do each count per-author
- Sort by the number of books each author wrote
- “limit 8” (optional) means we only see eight results

```
sqlite> select count(*) as num_books, author from books group by author order by num_books d
num_books     author
----------    ----------
2114          69
944           7
907           3
555           40
276           188
217           33
203           24
164           686
```

Who wrote the most books?

select count(*) as num_books, author from books group by author order by num_books desc limit 8;

- Count the number of records and call it num_books so we can use it to sort later
- Shows the author id
- Do each count per-author
- Sort by the number of books each author wrote
- "limit 8" (optional) means we only see eight results

```
sqlite> select count(*) as num_books, author from books group by author order by num_books d
num_books    author
----------   ----------
2114         69
944          7
907          3
555          40
276          188
217          33
203          24
164          686
```

Author 69 wrote the most books . . . who is he / she?

Who wrote the most books?

select title, name from books INNER JOIN authors on authors.authorID = books.author limit 10;

- Combine the books table with the authors table
- Make sure the IDs match up
- “limit 10” (optional) means we only see ten results

```
sqlite> select title, name from books INNER JOIN authors on
   ...> authors.authorID = books.author limit 10;
title                                          name
--------------------------------------------   ----------------
When Valmond Came to Pontiac, Volume 1.        Parker, Gilbert
The Translation of a Savage, Complete          Parker, Gilbert
The Trespasser, Volume 2                       Parker, Gilbert
The Right of Way   Volume 02                Parker, Gilbert
The Log-Cabin Lady   An Anonymous Aut       Unknown
King Arthurs Socks and Other Village Pe        Dell, Floyd
Two Months in the Camp of Big Bear
Myths and Legends of Our Own Land    V      Skinner, Charle
Myths and Legends of Our Own Land    V      Skinner, Charle
Poems of Sentiment                             Wilcox, Ella Wh
```

Who wrote the most books?

select title, name from books INNER JOIN authors on authors.authorID = books.author limit 10;

- Combine the books table with the authors table
- Make sure the IDs match up
- “limit 10” (optional) means we only see ten results

```
sqlite> select title, name from books INNER JOIN authors on
   ...> authors.authorID = books.author limit 10;
title                                         name
--------------------------------------------  ------------------
When Valmond Came to Pontiac, Volume 1.       Parker, Gilbert
The Translation of a Savage, Complete         Parker, Gilbert
The Trespasser, Volume 2                      Parker, Gilbert
The Right of Way   Volume 02                Parker, Gilbert
The Log-Cabin Lady   An Anonymous Aut       Unknown
King Arthurs Socks and Other Village Pe     Dell, Floyd
Two Months in the Camp of Big Bear
Myths and Legends of Our Own Land   V       Skinner, Charle
Myths and Legends of Our Own Land   V       Skinner, Charle
Poems of Sentiment                            Wilcox, Ella Wh
```

Who wrote the most books?

select count(*) as num_books, author, name from books INNER JOIN authors on authors.authorID = books.author group by author order by num_books desc limit 20;

```
sqlite> select count(*) as num_books, author, name from books INNER JOIN
...> authors on authors.authorID = books.author group by author order
...>  by num_books desc limit 20;
num_books   author      name
----------  ----------  ----------
2114        69          Various
555         40          Anonymous
276         188         Shakespear
217         33          Lytton, Ed
203         24          Twain, Mar
164         686         Ebers, Geo
145         71          Dickens, C
132         0           Parker, Gi
130         1           Unknown
125         41          Verne, Jul
125         610         Balzac, Ho
113         265         Kingston,
111         86          Jacobs, W.
109         662         Meredith,
106         53          Doyle, Art
103         670         Motley, Jo
101         26          Howells, W
99          958         Ballantyne
```

Asking and Answering Questions

- How many books were written by authors born before 1800?
- What's the average age of authors who wrote more than 10 books?

Outline

1 DBMS

2 Connecting to a Remote Computer

3 Queries on an Existing Database

4 Creating Your Own Database

5 Assignment 3

Design

- Need to keep track of employees
    - Age
    - Degree
    - Driver's License Number
    - First Name
    - Last Name
    - Pay (determined exclusively by degree)
- How many tables do we need?
- What columns are in each table?

Design

- Two tables
- Degrees
  - degreeID
  - pay
  - degree_name
  - degree_abbrv
- Employees
  - First Name
  - Last Name
  - License
  - Age

Creating a Table

```
CREATE TABLE Degrees (degreeID integer, Abbrev varchar(5), pay integer, FullName varchar(30));
```

- We're creating a new table with this name
- The ID and the pay are both integers
- The full and short names are both strings
- Column format better for us

Creating a Table

```
CREATE TABLE Degrees (degreeID integer, Abbrev varchar(5), pay
integer, FullName varchar(30));
```

- We're creating a new table with this name
- The ID and the pay are both integers
- The full and short names are both strings
- Column format better for us

What's wrong with this table?

Creating a Table

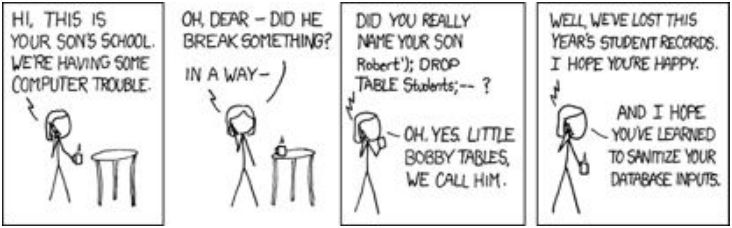
```
CREATE TABLE Degrees (degreeID integer, Abbrev varchar(5), pay
integer, FullName varchar(30));
```

- We're creating a new table with this name
- The ID and the pay are both integers
- The full and short names are both strings
- Column format better for us

What's wrong with this table?

Deleting a table and injection

```
drop table degrees;
```


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

- Make sure you check input
- Most packages have methods to “sanitize” inputs

Creating a Table (With keys and defaults)

```
CREATE TABLE Degrees (degreeID integer PRIMARY KEY, Abbrev
varchar(5), pay integer DEFAULT 30000, degree_name varchar(30));
```

- The degreeID is the primary key (must be unique, and lookup is fast)
- If we don't tell the database the pay, it assumes 30000

Creating a Table (With keys and defaults)

```
CREATE TABLE Degrees (degreeID integer PRIMARY KEY, Abbrev
varchar(5), pay integer DEFAULT 30000, degree_name varchar(30));
```

- The degreeID is the primary key (must be unique, and lookup is fast)
- If we don't tell the database the pay, it assumes 30000

Adding Data

```
insert into Degrees (degreeID, Abbrev, FullName) values (0, ”BA”,
”Bachelor of Arts”);
```

Adding Data

```
insert into Degrees (degreeID, Abbrev, FullName) values (0, ”BA”,
”Bachelor of Arts”);
```

Adding Data

```
insert into Degrees (degreeID, Abbrev, FullName) values (0, ”BA”,
”Bachelor of Arts”);
```

Adding Data

```
insert into Degrees (degreeId, Abbrev, degree_name) values (0, "BA", "Bachelor␣of␣Arts");
insert into Degrees (degreeId, Abbrev, degree_name) values (0, "MA", "Master␣of␣Arts");
```

Adding Data

```
insert into Degrees (degreeId, Abbrev, degree_name) values (0, "BA", "Bachelor␣of␣Arts");
insert into Degrees (degreeId, Abbrev, degree_name) values (0, "MA", "Master␣of␣Arts");
```

```
Error: PRIMARY KEY must be unique
```

Adding Data

```
insert into Degrees (degreeId, Abbrev, degree_name) values (0, "BA", "Bachelor␣of␣Arts");
insert into Degrees (degreeId, Abbrev, degree_name) values (0, "MA", "Master␣of␣Arts");

Error: PRIMARY KEY must be unique

insert into Degrees (degreeId, Abbrev, degree_name) values (1, "HS", "High␣School");
insert into Degrees (degreeId, Abbrev, degree_name) values (2, "MLS", "Master␣of␣Library␣Sci
```

Modifying Data

update degrees set pay=60000 where degreeID=2;

- Which table are we updating?
- The column to update
- The new cell contents
- Which rows to update (could apply to more than one)

```
sqlite> select * from degrees;degreeID     Abbrev       pay          degree_name
----------   ----------   ----------   ----------------
0            BA           30000        Bachelor of Arts
1            HS           30000        High School
2            MLS          30000        Master of Librar
sqlite> update degrees set pay=60000 where degreeID=2;
sqlite> update degrees set pay=20000 where degreeID=1;
sqlite> select * from degrees;
degreeID     Abbrev       pay          degree_name
----------   ----------   ----------   ----------------
0            BA           30000        Bachelor of Arts
1            HS           20000        High School
2            MLS          60000        Master of Librar
```

Modifying Data

```
update degrees set pay=60000 where degreeID=2;
```

- Which table are we updating?
- The column to update
- The new cell contents
- Which rows to update (could apply to more than one)

```
sqlite> select * from degrees;degreeID     Abbrev       pay          degree_name
----------  ----------  ----------  ----------------
0           BA          30000       Bachelor of Arts
1           HS          30000       High School
2           MLS         30000       Master of Librar
sqlite> update degrees set pay=60000 where degreeID=2;
sqlite> update degrees set pay=20000 where degreeID=1;
sqlite> select * from degrees;
degreeID    Abbrev      pay         degree_name
----------  ----------  ----------  ----------------
0           BA          30000       Bachelor of Arts
1           HS          20000       High School
2           MLS         60000       Master of Librar
```

Modifying Data

```
update degrees set pay=60000 where degreeID=2;
```

- Which table are we updating?
- The column to update
- The new cell contents
- Which rows to update (could apply to more than one)

```
sqlite> select * from degrees;degreeID    Abbrev      pay         degree_name
----------  ----------  ----------  ---------------
0           BA          30000       Bachelor of Arts
1           HS          30000       High School
2           MLS         30000       Master of Librar
sqlite> update degrees set pay=60000 where degreeID=2;
sqlite> update degrees set pay=20000 where degreeID=1;
sqlite> select * from degrees;
degreeID    Abbrev      pay         degree_name
----------  ----------  ----------  ---------------
0           BA          30000       Bachelor of Arts
1           HS          20000       High School
2           MLS         60000       Master of Librar
```

Modifying Data

```
update degrees set pay=60000 where degreeID=2;
```

- Which table are we updating?
- The column to update
- The new cell contents
- Which rows to update (could apply to more than one)

```
sqlite> select * from degrees;degreeID     Abbrev       pay          degree_name
----------  ----------  ----------  ---------------
0           BA          30000       Bachelor of Arts
1           HS          30000       High School
2           MLS         30000       Master of Librar
sqlite> update degrees set pay=60000 where degreeID=2;
sqlite> update degrees set pay=20000 where degreeID=1;
sqlite> select * from degrees;
degreeID    Abbrev      pay         degree_name
----------  ----------  ----------  ---------------
0           BA          30000       Bachelor of Arts
1           HS          20000       High School
2           MLS         60000       Master of Librar
```

Creating Tables (Foreign Key & Constraints)

```
CREATE TABLE Employees (empID integer PRIMARY KEY, age integer,
degree integer, DriverLic varchar (10), FirstName varchar (30), LastName
varchar(30), CHECK (age > 15), FOREIGN KEY (degree) REFERENCES
Degrees(degreeID));
```

- Make sure the employee is legal; every change to database must preserve this
- Make sure this points to valid degree
- How could this solve our age problem?

Creating Tables (Foreign Key & Constraints)

```
CREATE TABLE Employees (empID integer PRIMARY KEY, age integer,
degree integer, DriverLic varchar (10), FirstName varchar (30), LastName
varchar(30), CHECK (age > 15), FOREIGN KEY (degree) REFERENCES
Degrees(degreeID));
```

- Make sure the employee is legal; every change to database must preserve this
- Make sure this points to valid degree
- How could this solve our age problem?

Constraints in Action

```
sqlite> insert into Employees (empId, age, degree, DriverLic, FirstName, LastName) values
 ...> (0, 10, 0, "NZ01234", "Bart", "Simpson");
 Error: constraint failed
```

Constraints in Action

```
sqlite> insert into Employees (empId, age, degree, DriverLic, FirstName, LastName) values
 ...> (0, 10, 0, "NZ01234", "Bart", "Simpson");
 Error: constraint failed
```

```
sqlite> insert into Employees (empId, age, degree, DriverLic, FirstName, LastName) values
 ...> (0, 30, 0, "NJ01234", "Homer", "Simpson");
```

What about foreign key?

- Can't add an employee with a undefined degree
- Can't delete degrees if employees associated with that degree

Constraints in Action

```
sqlite> insert into Employees (empId, age, degree, DriverLic, FirstName, LastName) values
 ...> (0, 10, 0, "NZ01234", "Bart", "Simpson");
 Error: constraint failed


sqlite> insert into Employees (empId, age, degree, DriverLic, FirstName, LastName) values
 ...> (0, 30, 0, "NJ01234", "Homer", "Simpson");
```

What about foreign key?

- Can't add an employee with a undefined degree
- Can't delete degrees if employees associated with that degree
- But won't work on terpconnect (for backwards compatibility)

Deleting Data

```
DELETE from employees where empID = 0;
```

- Can delete multiple things at once
- Won't complain when it deletes nothing

Outline

1 DBMS

2 Connecting to a Remote Computer

3 Queries on an Existing Database

4 Creating Your Own Database

5 Assignment 3

- You should now know everything you need to know to do Assignment 3
- Two questions
    - Limiting queries
    - Inner Joins
- Due November 12 (but you should do it sooner)