ECS 189
WEB PROGRAMMING

5/17

Photobooth

- A photo storage site that can see the photos.
- This week: upload photos, put into database, edit tags interactively.
- Work in groups of up to three; due Mon 22.

SQL

- Monday we looked at the CREATE TABLE command.

```
CREATE TABLE PhotoLabels (
fileName TEXT UNIQUE NOT NULL PRIMARY KEY,
labels     TEXT,
favorite   INTEGER)
```

fileName	labels	favorite

- Now let's put something into it.

Inserting new rows

fileName	labels	favorite
hula.jpg	""	0

```
INSERT INTO photoLabels VALUES ("hula.jpg", "", 0)
```

- Table and field names use lower case
- Strings are in double quotes
- Values is a list, containing the values of each column
- Error if row with that fileName already exists

Replacing a row

- Alternative:

```
INSERT OR REPLACE INTO photoLabels VALUES
  ("hula.jpg", "", 0)
```

- No error, just replaces row if already there
- So we could always replace an entire row to change any data we have to…but a better way…

Updating one item

```
UPDATE photoLabels
SET labels = "Dance, Performing Arts, Sports,
  Entertainment, Quinceañera, Event, Hula, Folk Dance"
WHERE fileName = "hula.jpg"
```

- The WHERE clause selects the row…or rows! Always safe to choose by filename since that is the unique primary key.
- OMITTING WHERE CHANGES ALL ROWS!!!
- Use = not ==

Fill in the blanks

```
db.run(
  'UPDATE photoLabels SET labels = ?
  WHERE fileName = ? ',
  ['Bird, Beak, Bird Of Prey, Eagle, Vertebrate, Bald
  Eagle, Fauna, Accipitriformes, Wing',
  'eagle.jpg'],
  errorCallback);
```

- Question marks get replaced with values in following array.

Getting output

```
SELECT labels
FROM photoLabels
WHERE fileName = "hula.jpg"
```

- Returns the labels string for hula.jpg

Select statement

SELECT columns FROM table WHERE Boolean

- Handy example:

```
SELECT * FROM photoLabels
```

- Dumps the whole table. The * means all columns, and omitting the WHERE gets all rows.

WHERE expressions

- A few interesting Booelan operators

WHERE fileName IN ("hula.jpg", "eagle.jpg")
- Matches one in the list

WHERE labels LIKE "%Bird%"
- String in labels column contains substring "Bird"
- A bit like a lame regular expression

Using SQL through Node

- Usual structure: issue SQL command, provide callback for when it completes.

```
db.run( 'INSERT OR REPLACE INTO photoLabels
  VALUES ("hula.jpg", "", 0)',   errorCallback);

function errorCallback(err) { if (err)
  { console.log("error: ",err,"\n"); }}
```

- Could also use anonymous callback….

Getting data into Node

```
db.get( ' SELECT labels FROM photoLabels WHERE
  fileName = "hula.jpg" ', dataCallback);

function dataCallback(err, rowData) {
    if (err) {console.log("error: ",err,"\n");
    } else {console.log("got: ",rowData,"\n");   }}
```

- rowData is an object containing the selected data.

Getting an array

```
db.all('SELECT * FROM photoLabels',arrayCallback);

function arrayCallback(err, arrayData) {
  if (err) { console.log("error: ",err,"\n");
  } else { console.log("array: ",arrayData,"\n");  }}
```

- arrayData contains an array of objects, each object contains one row.

Closing the database

- Running in memory most of the time
- Nothing gets written to disk until you issue

```
db.close();
```

- Always remember to do at the end, otherwise work will be lost.

Asynchronous execution

```
db.run( 'INSERT OR REPLACE INTO photoLabels
  VALUES ("hula.jpg", "", 0)',   errorCallback);
db.run('UPDATE photoLabels SET labels = "Dance,
  Performing Arts, Sports, Entertainment,
  Quinceañera, Event, Hula, Folk Dance" WHERE
  fileName = "hula.jpg" ',    errorCallback);
db.get( ' SELECT labels FROM photoLabels WHERE
  fileName = "hula.jpg" ', dataCallback);
```

- Produces…. got: { labels: '' }
- WHY?

Even worse

```
amenta@cs189h:~$ node Dbops
  got:  { labels: '' }
amenta@cs189h:~$ node Dbops
  got:  { labels: 'Dance, Performing Arts, Sports,
  Entertainment, Quinceañera, Event, Hula, Folk
  Dance' }
amenta@cs189h:~$ node Dbops
  got:  { labels: '' }
```

Race condition

- The UPDATE command takes a while. So possibly:
  - UPDATE requested
  - SELECT requested
  - SELECT completes, calls callback
  - UPDATE completes, calls callback
- Or possibly:
  - UPDATE requested
  - SELECT requested
  - UPDATE completes, calls callback
  - SELECT completes, calls callback

Forcing SQL to run in order

```
db.serialize( function () {
  db.run( 'INSERT …
  db.run('UPDATE…
  db.get( ' SELECT…
  db.close();
}
```

- Does not start one command until previous one completes and returns

Forcing SQL to run in order

```
db.serialize( function () {
   db.run( 'INSERT …
   db.run('UPDATE…
   db.get( ' SELECT…
   db.close();
}
```

- Produces reliably as expected

```
got:  { labels: 'Dance, Performing Arts, Sports,
   Entertainment, Quinceañera, Event, Hula, Folk
   Dance' }
```