# SQL Basics

## What is it?

- Stands for “Structured Query Language”
- Standard for storing and accessing information
- 3 Major “sub-languages” or components to SQL
  - DDL - Data Declaration Language
  - DML - Data Manipulation Language
  - DCL - Data Control Language

# DDL Commands

- Commands affect the structure and storage of data
- Create new tables:
  ```
  CREATE TABLE <table name> (<columns>);
  ```
- Delete tables that already exist
  ```
  DROP TABLE <table name>;
  ```
- Change the table definition:
  ```
  ALTER TABLE <table name> … ;
  ```

## DML - Data Manipulation Language

- Commands access and modify to the *contents* of database
- Read values out of the database:
  ```
  SELECT * FROM users;
  ```
- Put values into tables:
  ```
  INSERT INTO users …;
  ```
- Remove values from the database
  ```
  DELETE FROM users …;
  ```

## DCL - Data Control Language

- Commands operate on data permissions
- Allow users to read data from a table:
  ```
  GRANT SELECT ON <table> TO <user>;
  ```
- Prohibit users from modifying specific tables:
  ```
  REVOKE UPDATE ON <table> FROM <user>;
  ```
- Won’t go into more depth. Differs between database implementations significantly.

# SQL is **old**. Why are we still talking about it?

- Declarative language that describes “what you want” not “how to do it”
- Friendly format for both humans and machines
- Lack of better general-purpose alternatives
- Extremely flexible abstractions

# Basic Abstractions

- A SQL database stores data in one more more <u>tables</u>
- Tables are a collection of <u>columns</u> that have names and data types
- <u>Rows</u> are entries in tables that may or may not have values for each column
- <u>Primary Keys</u> uniquely identify rows within the table

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@harrys.com | James | Petty | TRUE |
| 2 | chris@harrys.com | Chris | Clouten | TRUE |
| 3 | bigfoot@gmail.com | NULL | NULL | FALSE |
| 4 | imissthe70s@aol.com | Cher | NULL | FALSE |

- Primary Key?
- Columns?
- Rows?
- What’s this NULL thing?

# DDL - Data Definition Language

```
CREATE TABLE users (
 id integer primary key,
 email_address varchar not null,
 first_name varchar,
 last_name varchar,
 is_admin boolean not null default false
);
```

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@harrys.com | James | Petty | TRUE |
| 2 | chris@harrys.com | Chris | Clouten | TRUE |
| 3 | bigfoot@gmail.com | NULL | NULL | FALSE |
| 4 | imissthe70s@aol.com | Cher | NULL | FALSE |

# Common SQL Data Types

| Name | Description | Values |
|---|---|---|
| boolean | Values of either “yes” or “no” | {true, false} |
| integer | Whole numbers (including 0) including negative values. | {…, -2, -1, 0, 1, 2, …} |
| decimal(p,s) | Decimal value with precision *p* (total number of digits) and scale (number of fractional digits) | eg: decimal(5,2): [-999.99, 999.99] |
| char(n) | Text value with exact length (n) | $\Sigma^n$ |
| varchar(n) | Text value with a variable length up to *n.* | $\Sigma^* = \bigcup \Sigma^n$ $n \in \mathbb{N} \cup \{0\}$ |

# More SQL Data Types

| Name | Description | Values |
|---|---|---|
| date | A date value including year, month, and day. | Varies by implementation |
| time | A time of day including hour, minute, second, and sometimes fractional seconds | 00:00:00 - 24:00:00 |
| timestamp | Combination of date with time, sometimes with timezone | Varies by implementation |

# DDL Column Definitions

- Columns *must* have a name and data type
- Columns *may* have constraints and a default value

| name | data type | constraints | default value |
|---|---|---|---|
| id | integer | primary key | (it’s complicated) |
| email_address | varchar | not null | None |
| first_name | varchar | None | NULL |
| last_name | varchar | None | NULL |
| is_admin | boolean | not null | FALSE |

## Interactive Time

- sqlite3 is a small, embeddable, SQL database engine that comes preinstalled with OS X
- make sure we can all run commands
- materials located here:

  https://pettyjamesm.github.io/mammoth-school-sql-intro/

# SQL Basics

Part II, The Sequel

# Recap

- SQL stands for “Structured Query Language.”
    - 3 major “sub-languages” or components to SQL
        - DDL - Data Declaration Language
        - DML - Data Manipulation Language
        - DCL - Data Control Language
    - DML is used to create, read, update, and delete information.

# Tables / Relations

Columns

Rows

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | TRUE |
| 2 | chris@example.com | Chris | Clouten | TRUE |
| 3 | bigfoot@gmail.com | NULL | NULL | FALSE |
| 4 | imissthe70s@aol.com | Cher | NULL | FALSE |

- Types (varchar, integer, boolean, …)
- “No Value” marker NULL

# DML — INSERT

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| | | | | |
| Keys | | | | |
| | | | | |
| | | | | |

```
INSERT INTO users
  (id, email_address, first_name, last_name, is_admin)
VALUES
  (1, 'jpetty@example.com', 'James', 'Petty', 1),
  (2, 'chris@example.com', 'Chris', 'Clouten', 1);
```

# Result

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 (true) |
| 2 | chris@example.com | Chris | Clouten | 1 (true) |
| | | | | |
| | | | | |

Remember the **DDL**…

```
CREATE TABLE users (
    id integer primary key,
    email_address varchar not null,
    first_name varchar,
    last_name varchar,
    is_admin boolean not null default false
);
```

# Constraints

```
INSERT INTO users (is_admin) VALUES (0);
```

*Error: NOT NULL constraint failed: users.email_address*

```
INSERT INTO users (email_address, is_admin)
VALUES ('bigfoot@gmail.com', 0);
```

```
INSERT INTO users (id, email_address, is_admin)
VALUES (3, 'imissthe70s@aol.com', 0);
```

*Error: UNIQUE constraint failed: users.id*

# Constraints

- Database is checking consistency
  - Value in the correct domain?
  - Value specified?
  - Value unique?
  - Default?
- And more.

# DML — SELECT

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 |
| 2 | chris@example.com | Chris | Clouten | 1 |
| 3 | bigfoot@gmail.com | NULL | NULL | 0 |
| | | | | |

```
SELECT id, email_address
FROM users;
```

```
1 | jpetty@example.com
2 | chris@example.com
3 | bigfoot@gmail.com
```

# SELECT … WHERE

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 |
| 2 | chris@example.com | Chris | Clouten | 1 |
| 3 | bigfoot@gmail.com | NULL | NULL | 0 |
| | | | | |

```
SELECT id, email_address
FROM users
WHERE last_name IS NULL;
```

```
3 | bigfoot@gmail.com
```

## count(), max(), etc.

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 (true) |
| 2 | chris@example.com | Chris | Clouten | 1 (true) |
| 3 | bigfoot@gmail.com | NULL | NULL | 0 (false) |
| | | | | |

```
SELECT count(*)
FROM users
WHERE is_admin;

2
```

```
SELECT max(id)
FROM users;

3
```

# SELECT * … LIMIT

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 |
| 2 | chris@example.com | Chris | Clouten | 1 |
| 3 | bigfoot@gmail.com | NULL | NULL | 0 |
| | | | | |

```
SELECT *
FROM users
LIMIT 1;
```

```
2 | chris@example.com | Chris | Clouten | 1
```

# SELECT ... ORDER BY

products

| id | name | price |
|---|---|---|
| 1 | Daily Face Wash | 7.00 |
| 2 | Truman Set | 15.00 |
| 3 | Razor Blades | 16.00 |
| 4 | Foaming Shave Gel | 6.00 |

```
SELECT name, price FROM products
ORDER BY price;
```

```
Foaming Shave Gel | 6
Daily Face Wash | 7
Truman Set | 15
Razor Blades | 16
```

# JOINS

- Tables may be “joined” conceptually combining every row from one table with every row from the other table.

- Usually, you want a subset of the resulting rows…

- Each table stores data to model part of your problem — pulling the data together allows queries to look across tables for the answers.

# Orders ☞ Users

orders

| id | user_id | placed_at |
|---|---|---|
| 1 | 2 | 2015-12-04 06:02:30 |
| 2 | 4 | 2016-02-25 04:45:14 |
| 3 | 3 | 2016-02-24 04:12:05 |
|  | . |  |

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 |
| 2 | chris@example.com | Chris | Clouten | 1 |
| 3 | bigfoot@gmail.com | NULL | NULL | 0 |
| 4 | imissthe70s@aol.com | Cher | NULL | 0 |

# Orders ☞ Users

users

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 1 | jpetty@example.com | James | Petty | 1 |
| 2 | chris@example.com | Chris | Clouten | 1 |
| 3 | bigfoot@gmail.com | NULL | NULL | 0 |
| 4 | imissthe70s@aol.com | Cher | NULL | 0 |

orders

| id | user_id | placed_at |
|---|---|---|
| 1 | 2 | 2015-12-04 06:02:30 |
| 2 | 4 | 2016-02-25 04:45:14 |
| 3 | 3 | 2016-02-24 04:12:05 |

| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 4 | imissthe70s@aol.com | Cher | NULL | 0 |

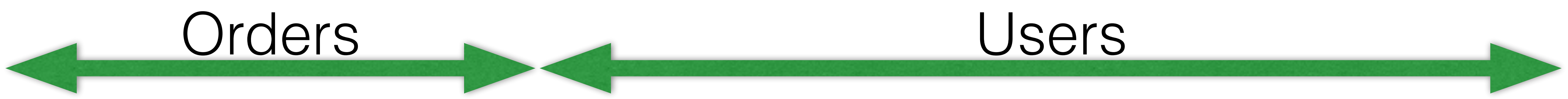
| id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|
| 3 | bigfoot@gmail.com | NULL | NULL | 0 |

| Orders | | | Users | | | | |
|---|---|---|---|---|---|---|---|
| **id** | **user_id** | **placed_at** | **id** | **email_address** | **first_name** | **last_name** | **is_admin** |
| 1 | 2 | 2015-12-04 06:02:30 | 2 | chris@example.com | Chris | Clouten | 1 |
| 2 | 4 | 2016-02-25 04:45:14 | 4 | imissthe70s@aol.com | Cher | NULL | 0 |
| 3 | 3 | 2016-02-24 04:12:05 | 3 | bigfoot@gmail.com | NULL | NULL | 0 |

```sql
SELECT orders.placed_at, users.email_address, users.first_name
FROM orders, users
WHERE orders.user_id = users.id;
```

```
2015-12-04 06:02:30 | chris@example.com | Chris
2016-02-25 04:45:14 | imissthe70s@aol.com | Cher
2016-02-24 04:12:05 | bigfoot@gmail.com | NULL
```

# FROM… JOIN … ON



| id | user_id | placed_at | id | email_address | first_name | last_name | is_admin |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 2015-12-04 06:02:30 | 2 | chris@example.com | Chris | Clouten | 1 |
| 2 | 4 | 2016-02-25 04:45:14 | 4 | imissthe70s@aol.com | Cher | NULL | 0 |
| 3 | 3 | 2016-02-24 04:12:05 | 3 | bigfoot@gmail.com | NULL | NULL | 0 |

```
SELECT orders.placed_at, users.email_address, users.first_name
FROM orders
JOIN users ON orders.user_id = users.id;
```

```
2015-12-04 06:02:30 | chris@example.com | Chris
2016-02-25 04:45:14 | imissthe70s@aol.com | Cher
2016-02-24 04:12:05 | bigfoot@gmail.com | NULL
```

## Interactive Time

- We will use sqlite3 again.
- Start terminal (using Spotlight), then type: **sqlite3**
- Online materials located here:

  https://pettyjamesm.github.io/mammoth-school-sql-intro/