SQL - Essentials

Starting points

- Relational Databases excels at storing and retrieving records that are related to each other
- `CRUD` : creating, reading, updating, and deleting `
- check the version of mysql db using `mysql --version`
- `quit` to exit out of mysql cli

Definitions

- A `database` is a collection of interrelated data. This data is stored in one or more tables that are related to one another.
- A `table` is composed of rows and columns. A column represents a field. A row represents a record.
- A `query` is a request for data from a database table or a combination of tables.
- start mysql `mysql -u root -p` hit enter enter

Create Database

- `CREATE DATABASE shinobi;` notice command end with semicolon, and name is case sensitive
- `USE shinobi` this is to state from now on any commands should work on this db
- `DROP DATABASE IF EXISTS shinobi`

Creating tables

- we will create customer table
- VARCHAR : variable character value inside `()` represents the length of the varchar
- BOOLEAN, INTEGER
- To verify that the table was created successfully, we can use the `describe TABLE_NAME`

```
CREATE TABLE customers (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(30) NOT NULL,
  last_name VARCHAR(30) NOT NULL,
  project_id INTEGER
);
```

Primary Key

- `RIMARY KEY` states that each value in this column must be unique for each record in the table.
- `AUTO_INCREMENT`: increments with each successive row and assigns that new value to the id
- `NOT NULL` keyword means that a column must contain a value.

Inserting Data into DB

- `DEFAULT` keyword is used to specify a default value for a column.
  `CREATE TABLE DEMOTABLE ( NAME demo_name default 'some name')`

```
INSERT INTO customers (first_name, last_name, project_id)
VALUES ('Shane', 'Abbas', 786);
```

Selecting data

```
SELECT * FROM customers;
```

- We could list individual column names separated by commas, but we use the wildcard *, which means "all the columns."

Where clause

- The WHERE clause is a powerful filtering tool that can be used with equality operators like less than (<) or not equal to (!=). We can also use the OR and AND logical operators to evaluate multiple conditions. If the expression evaluates to true, the row is returned.

```
SELECT first_name, project_id
FROM customers
WHERE project_id = 1;
```

Schema

- instead of setting up the db via cli, we make a file to setup for the database and tables
- `seed.sql` to conatain out data to instrt into DATABASE
- `db.sql` to have information on database and creation of
- `schema.sql` to contain database table information and creation.

Deleting and updating columns

```
DELETE FROM custmers,
WHERE first_name = "Montague";
```

```
UPDATE customers
SET project_id = 1
WHERE id = 3;
```

SQL Data Types

- `VARCHAR`: variable character representing a string of characters with a maximum length of 255 characters.
- `BOOLEAN`: a boolean value that can be either true or false.
- `TEXT`: a variable character string with a maximum length of 65,535 characters.
- `INTEGER`: a number that is stored without a decimal point.
- `DATE`: a date and time value.
- `DATETIME`: a date and time value.
- `TIMESTAMP`: a date and time value.
- `REAL`: a number that is stored with a decimal point.
- `FLOAT`: a number that is stored with a decimal point.

Building relationships

- `CONSTRAINT`: a constraint is a rule that must be satisfied before a database operation can be performed.
- `FOREIGN KEY`: a reference to another table, using the `REFERENCES` clause, here we reference the primary key of the other table.

```
create table customers (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(30) NOT NULL,
  last_name VARCHAR(30) NOT NULL,
  project_id INTEGER,
  CONSTRAINT fk_customers_project_id FOREIGN KEY (project_id) REFERENCES projects(id) ON DELETE SET NULL
);
```

ALTER TABLE

- `ALTER TABLE`: used to add, drop, or modify columns in a table.
- if we want to change a table to add FOREIGN key constraint, we use the `ALTER TABLE` command.
- `ON DELETE SET NULL`: when a row is deleted from the `projects` table, the foreign key value in the `customers` table is set to NULL.
- `ON DELETE CASCADE`: cascade refer to the action of deleting the parent row if the child row is deleted.

```
ALTER TABLE customers ADD CONSTRAINT fk_customers_project_id FOREIGN KEY (project_id) REFERENCES projects(id) ON DELETE SET NULL;

ALTER TABLE projects
MODIFY COLUMN description Text;
```

Relational data

- `inner join`: joins two tables on a common column.
- `left join`: joins two tables on a common column, and includes the rows from the left table in the result.
- `right join`: joins two tables on a common column, and includes the rows from the right table in the result.
- `full join`: joins two tables on a common column, and includes the rows from both tables in the
- `INNER JOIN`: joins two tables on a common column.

```
SELECT * FROM customers
Left JOIN projects ON customers.project_id = projects.id;
```

loading sql files

- start mysql `mysql -u root -p` hit enter enter
- `source db.sql` to load db.sql file

adding mysql to node project

- `npm install mysql2`
- `mysql2` enables us to connect to mysql db

Connecting to db from node

```
const mysql = require('mysql2');
  const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'shinobi'
  });
```

Reading data from db

```
connection.query('SELECT * FROM customers', function(err, results, fields) {
  if (err) throw err;
  console.log(results);
});
```

Inserting data into db

```
connection.query('INSERT INTO customers (first_name, last_name, project_id) VALUES (?, ?, ?)', ['John', 'Doe', 786], function(err, results, fields) {
  if (err) throw err;
  console.log(results);
});
```

Updating data in db

```
connection.query('UPDATE customers SET project_id = ? WHERE id = ?', [1, 3], function(err, results, fields) {
  if (err) throw err;
  console.log(results);
});
```

- the `?` is a placeholder for the value we want to insert into the query.
- `[1, 3]` is the value we want to insert into the query when

Deleting data from db

```
connection.query('DELETE FROM customers WHERE id = ?', [3], function(err, results, fields) {
  if (err) throw err;
  console.log(results);
});
```

query call back function objects

- A query callback function is a function that is called when a query is completed. it returns with it
  - `err`: an error object if the query failed.
  - `results`: an array of rows returned by the query. some properties in results
    - `affectedRows`: the number of rows affected by the query.
    - `insertId`: the id of the last inserted row.
    - `warningStatus`: the warning status returned by the server.
  - `fields`: an array of metadata about the columns in the result set.

additional fun sql commands

-- get all voters who do not have a last name of Cooper or Jarman

```
SELECT * FROM voters WHERE last_name != 'Cooper' AND last_name != 'Jarman';
```

-- get all voters who have a .qUni email address

```
SELECT * FROM voters WHERE email LIKE '%.qUni';
```

-- get only the last created voter

```
SELECT * FROM voters ORDER BY created_at DESC LIMIT 1;
```

-- get all voters who have a .qUni email address and have a last name of Cooper or Jarman

```
SELECT * FROM voters WHERE email LIKE '%.qUni' AND (last_name = 'Cooper' OR last_name = 'Jarman');
```

-- get all voters who have a .qUni email address and have a last name of Cooper or Jarman and have a first name of John

```
SELECT * FROM voters WHERE email LIKE '%.qUni' AND (last_name = 'Cooper' OR last_name = 'Jarman') AND first_name = 'John';
```

-- get all voters who have a .qUni email address and have a last name of Cooper or Jarman and have a first name of John and have a phone number of 123-456-7890

```
SELECT * FROM voters WHERE email LIKE '%.qUni' AND (last_name = 'Cooper' OR last_name = 'Jarman') AND first_name = 'John' AND phone_number = '123-456-7890';
```

Other useful aggregate functions in SQL include:

- `AVG()` to return the average value within a group
- `COUNT()` to return the number of rows within a group
- `SUM()` to add up all of the values in a group
- `MIN()` to return the smallest value within a group
- `MIN()` to return the minimum value of a group

```
SELECT AVG(age) FROM voters;
```

OUTPUT FILTERS

- `GROUP BY` : used to group the results of a query by a column.
- `HAVING` : used to filter the results of a query by a condition.
- `ORDER BY` : used to order the results of a query by a column.
- `LIMIT` : used to limit the number of results returned by a query.

```
SELECT COUNT(age) FROM voters GROUP BY age HAVING age < 100;
```