SQL Unit 9
Creating Tables, Adding Keys & Constraints

Ryan Nixon

No More Queries

- Using SELECT can retrieve existing information from databases
- From here on we will focus on how to add to and modify that information rather than retrieve it

CRUD

- **Cr**eate, **U**pdate & **D**elete
- This is a common term used to describe the minimum functionality to work with data
- Today we're focusing on the CR

Data Types Review

- Before jumping in to tables themselves we should discuss the data types being used
- There are many types, but we're going to focus on:
  - Strings
  - Numbers
  - Dates

“Strings”

- Strings consist of multiple characters. Think of paragraphs in a book or form fields on a website

- There are 3 major data types for strings:
  - CHAR - Fixed-width string
  - VARCHAR - Variable-width string
  - TEXT - Unlimited-width string

“Strings”

- CHAR types are best used when you need something to be a specific length. They will pad their contents with spaces to match the specified length
- VARCHAR types are similar, but allow for strings less than the specified length. These are the most common fields
- TEXT types allow unlimited length strings. They work best when you are holding strings that you can’t estimate the length of, such as bios or blog entries

“Strings”

- You can specify the length of CHARs and VARCHARS similar to a function, by placing the number of characters in parentheses after the type:
  - VARCHAR(10) - Holds up to 10 characters
  - CHAR(255) - Holds exactly 255 characters

“Strings”

- Examples: state abbreviation, last name, lorem ipsum
  - CHAR(2) - “AK”
    VARCHAR(2) - “AK”
    TEXT - “AK”
  - CHAR(10) - “Jones “
    VARCHAR(10) - “Jones”
    TEXT - “Jones”
  - CHAR(15) - “Lorem Ipsum Dol”
    VARCHAR(15) - “Lorem Ipsum Dol”
    TEXT - “Lorem Ipsum Dolor Sit Amet”

Numbers

- Numbers can hold only digits up to a certain size. Some support decimals and others do not. SQL implementations unfortunately vary wildly
- The major number data types are:
  - BOOLEAN - Supports only 0 or 1
  - INTEGER - No decimal point
  - FLOAT - Decimal point, but math operations do not work well
  - DECIMAL - Decimal point, good for math

Numbers

- BOOLEAN holds a single number that can be 0 or 1. It is perfect for dealing with checkboxes or permissions
- In some versions of SQL when working with BOOLEAN you may also use True or False in place of 0 and 1

Numbers

- INTEGERS are the most common data types for numbers. They have a few varieties
  - TINYINT: -128 to 127
  - SMALLINT: -32768 to 32767
  - INT: -2147483648 to 2147483647
  - BIGINT: -9223372036854775808 to 9223372036854775807
- These types also support “unsigned” variations which double the maximum size but disallow negative numbers

Numbers

- FLOAT data types are common for holding numbers with decimal points. They work extremely fast with math operations, but are not 100% accurate. Do not use them when accuracy is important
- Depending on the precision you need, DOUBLE may also be used which allows for larger numbers
- In the context of this class, feel free to use FLOAT rather than DOUBLE

Numbers

- DECIMAL is similar to FLOAT but is meant for math operations. It is safe to store precise numbers such as currency in a DECIMAL field
- DECIMAL is like CHAR in that it takes parameters in parentheses
- DECIMAL(precision, scale)
  - Precision - The number of significant digits
    5 - 123.4567890
  - Scale - The number of digits after the decimal point
    5 - 123.4567890

Dates

- Special strings that are meant to hold calendar information up to the subsecond. They can also hold timezone information if supported
- There are 3 date types that we will focus on:
  - DATE - Holds just the year, month & day
  - TIME - Holds only the hour, minute & second
  - DATETIME - Holds both the DATE & TIME's information

Dates

- Choose only what you need!
- Here are some examples:
  - DATE:“2007-08-19”
  - TIME:“14:30:00”
  - DATETIME:“2007-08-19 14:30:00”
- Remember to use the datetime functions if you need more specific date parts

A Note About Data Types

- Since implementations are very different between SQL versions, I can only cover the basics
- SQLite in particular uses "type affinity" versus "type enforcement". It will allow you to put practically any value in any field regardless of type, but if you use types it recognizes then it will optimize appropriately
- Because of this type affinity, SQLite is very allowing. *Be careful* when setting data types; it will let you set incorrect types!

CREATE TABLE

- As mentioned previously, tables store all of the data in a database and are the most important part of it
- Fortunately, and unlike SELECTs, table creation has a very simple syntax
- Unfortunately, creating tables is one-way as it modifies the database. Before you create tables make sure that you have completely thought through your data types, constraints and relationships!

CREATE TABLE

- ```
  CREATE TABLE table_name (
      column_name data_type constraints,
      column_name data_type constraints,
      column_name data_type constraints,
      additional constraints,
      ...
  );
  ```
  ```

CREATE TABLE

- A couple of examples from the class database:
  - ```
    CREATE TABLE person (
        id INTEGER PRIMARY KEY,
        first VARCHAR(100) NOT NULL,
        middle VARCHAR(100),
        last VARCHAR(100) NOT NULL,
        gender_id CHAR(1) NOT NULL,
        dob DATE,
        address_id INT,
        FOREIGN KEY (gender_id) REFERENCES ct_gender(id),
        FOREIGN KEY (address_id) REFERENCES address(id)
    );
    ```
  - ```
    CREATE TABLE class (
        id INTEGER PRIMARY KEY,
        major_id INT NOT NULL,
        name VARCHAR(100) NOT NULL,
        course VARCHAR(10) NOT NULL,
        credits DECIMAL(3,1) NOT NULL,
        section INT,
        FOREIGN KEY (major_id) REFERENCES major(id)
    );
    ```

CREATE TABLE

- As you can see, while the syntax is easy to understand the number of parts working together can be complex
- Let's take the more confusing parts one at a time

Column Name

- `CREATE TABLE table_name (`
  `column_name data_type constraints,`
- The column name should be a string with the following rules:
  - No spaces allowed. You may use underscores to fake spaces
  - All lowercase letters
  - Cannot start with a number

Column Name

- ```
  CREATE TABLE table_name (
      column_name data_type constraints,
  ```
- There are also some reserved names you should avoid, such as the data types ('int', 'varchar'). I hope that none of you try to create a column called 'select'...
- You can use a reserved name if you must, just surround it with `backticks`. You will have to continue using those whenever you write queries to retrieve that information
  ```

Constraints

- `CREATE TABLE table_name (`
  `column_name data_type` **`constraints,`**
- PRIMARY KEY, NOT NULL, and DEFAULT
- NOT NULL enforces that the field must not contain NULLs in it
- DEFAULT often pairs with NOT NULL so that if a value is not provided a replacement value is set

Constraints

- Out of state in the example database is an example of `NOT NULL DEFAULT 0`
  - The student defaults to in-state if that information is not provided
- Class_registration also has `NOT NULL DEFAULT 'S'` for role_id
  - When a new registration is created, it is assumed that the registration is for a student unless otherwise stated

The Primary Key

- As taught, the Primary Key is by far the most important field in a table. It identifies each individual row
- This field also unfortunately has some variations between languages
- The following slide has examples for each implementation. Note that the data type can be anything you would like, but in order for it to automatically increase as new data is entered it will need to be an INTEGER

The Primary Key

- SQLite - `INTEGER PRIMARY KEY`
- MySQL - `INT PRIMARY KEY AUTO_INCREMENT`
- MS SQL - `INT PRIMARY KEY IDENTITY`
- `AUTO_INCREMENT`/`IDENTITY` allow the database to automatically add 1 to the row's Id when a new one is added. SQLite does this automatically if the data type is an integer

Additional Constraints

- ```
  CREATE TABLE table_name (
      column_name data_type constraints,
      additional constraints,
  ```

- These constraints are listed after each column and set more limitations on the table such as the UNIQUE constraint and the FOREIGN KEYs
  ```

FOREIGN KEY

- ```
  CREATE TABLE table_name (
        column_name data_type constraints,
        FOREIGN KEY (column_name) REFERENCES
  other_table(pkey)
  ```
- In the above syntax, the FOREIGN KEY is set on the column that was declared earlier in the statement, and points that key to the other table's primary key (often 'id')
- This is the glue for tables as mentioned in almost all prior units. It sets up the table relationships that you will need for your queries
  ```

FOREIGN KEY

- As an example, here is the M:M relationship between class & person
- ```
  CREATE TABLE person (
       id INT PRIMARY KEY
  );
  ```
- ```
  CREATE TABLE class (
       id INT PRIMARY KEY
  );
  ```
- ```
  CREATE TABLE class_registration (
       id INT PRIMARY KEY IDENTITY,
       person_id INT NOT NULL,
       class_id INT NOT NULL,
       FOREIGN KEY (person_id) REFERENCES person(id),
       FOREIGN KEY (class_id) REFERENCES class(id)
  );
  ```
  ```

UNIQUE

- ```
  CREATE TABLE table_name (
        column1 data_type constraints,
        column2 data_type constraints,
        CONSTRAINT key_name UNIQUE (column1, column2)
  ```

- There is also a UNIQUE constraint on the class_registration table. UNIQUE constraints ensure that no duplicates are created within the columns that you set
- PRIMARY KEY automatically sets a UNIQUE constraint on its column, so no two Ids can ever be the same
- The name you set for this key is up to you, but it should follow the same rules that column names adhere to
  ```

UNIQUE

- Here is the major table as an example of a single column UNIQUE constraint:

```
CREATE TABLE major (
      id INT PRIMARY KEY IDENTITY,
      abbr VARCHAR(10) NOT NULL,
      CONSTRAINT unq_abbr UNIQUE (abbr)
);
```

- And the class_registration table showing a multiple column constraint:

```
CREATE TABLE class_registration (
      id INT PRIMARY KEY IDENTITY,
      person_id INT NOT NULL,
      term_id INT NOT NULL,
      class_id INT NOT NULL,
      CONSTRAINT unq_reg UNIQUE(person_id, class_id,
term_id)
);
```

Dropping Tables

- If you make a mistake with a table, simply drop it before recreating it!
- DROP TABLE table_name;
- Exactly like if you were dropping a temporary table

Examples?

- Instead of using the example database, lets create a few tables from your projects
- The example database is up on the website for your reference. I would suggest you look at it and try a few tables of your own before the test!

Reminders

- Assignment 9 up tonight. Due Monday
- Lab Wednesday in SSB 172
  - If you would like to get started on creating your own database, I can help you in the lab
- Test Monday; 3”x5” index card allowed
  - Be sure to study table creation and the set operations. They will *both* be on the test