You Made It!

Congratulations :)

Test Results

- Congratulations to one student, 109/100. An almost perfect score!
- Just make sure that whenever you use wildcards with '%' that you MUST use LIKE, not =
- About half of the class forgot this rule

SQL Unit 11
Altering Tables and Defining Indexes

Ryan Nixon

Another Compatibility Note

- Yet again, SQL's compatibility issues rear their ugly heads
- Many of the table altering statements I am about to discuss do not work in SQLite. This is partially due to SQLite's lightweight setup and because of how it stores its information on disk
- You are expected to know this syntax regardless of SQLite's limitations, but I will attempt to de-stress any questions on the test using these

Altering Tables

- There are many different commands that allow you to change a table after it has been created. Many use syntax that you have already seen
- Note -- It is in your best interest to design your database to handle all future uses at the beginning. Altering tables has side effects and should be avoided unless completely necessary

Altering Tables

- All altering commands begin with ALTER TABLE table_name
- Most of the parameters to this command will be extremely familiar, but be careful of the details

Altering Columns

- Say that your project requirements change after a database schema has been finalized. In this case you would have to change your columns in an existing table.

- For example, for person_student in the example database we might want to add a GPA column
- Afterwards you may add a grade column to the class_registration table. This could be used to calculate the GPA, so your new column would become obsolete

Altering Columns

- ```
  ALTER TABLE person_student
  ADD COLUMN gpa DECIMAL(2, 1) NOT NULL;
  ```

- ```
  ALTER TABLE person_student
  DROP COLUMN gpa;
  ```

- ```
  ALTER TABLE class_registration
  ADD COLUMN grade DECIMAL(2, 1);
  ```
  ```

Altering Columns

- ```
  ALTER TABLE person_student
  ADD COLUMN gpa DECIMAL(2, 1) NOT NULL;
  ```
- If you take another look at the first statement you will notice that after ADD COLUMN, the syntax is exactly the same that you would use in the CREATE TABLE statement
- This is a general rule, and an important one. The statement should be EXACTLY like the CREATE TABLE one
- If you for example forgot the NOT NULL, or the decimal's precision/scale, then you could potentially change an existing column's schema
  ```

Altering Columns

- `ALTER TABLE person_student DROP COLUMN gpa;`
- To drop a column from a table, specify DROP COLUMN, followed by the column's name
- Just like DELETE and DROP TABLE, this action is permanent. All data stored in that column will be gone forever

Altering Columns

- `ALTER TABLE class_registration ADD COLUMN grade DECIMAL(2, 1);`
- Lastly, this adds the new grade column to class_registration
- This is still decimal so that a grade can easily be averaged between all classes (3.0, 3.5, 4.0, etc.)

Altering Columns

- A couple more things to note here:
  - GPA is NOT NULL in person_student because all students will have a GPA if they are taking classes
  - Grade however allows nulls, since some students may audit/withdraw from a class and therefore not receive a valid grade
  - This is of course open to interpretation

Altering Columns

- If you don't like the gpa interpretation and think that some students might have NULL GPA columns (say, freshman who have never taken a class) you can execute this:
- ```
  ALTER TABLE person_student
  ADD COLUMN gpa DECIMAL(2, 1);
  ```
- Note that it excludes NOT NULL. This would either add a column by that name OR modify an existing column with the new data type, precision and constraints
- In this way ADD COLUMN also works for updates as well
  ```

Altering Constraints

- Table constraints can be managed exactly like columns, but using ADD CONSTRAINT
- In the previous slides I created a grade column on class_registration. Right now it is a DECIMAL, but what if I wanted to hide the numbers and use letters (A, B, C, D, F) instead?
- This is a perfect use for a code table

Altering Constraints

- ```
  CREATE TABLE ct_grade (
      id CHAR(1) PRIMARY KEY,
      value DECIMAL(2, 1) NOT NULL
  );
  ```
- ```
  ALTER TABLE class_registration
  DROP COLUMN grade;
  ```
- ```
  ALTER TABLE class_registration
  ADD COLUMN grade_id CHAR(1);
  ```
  ```

Altering Constraints

- To switch to using a code table we need to perform some boilerplate work, namely creating the table
- Next, we drop and recreate the grade column so that it has the correct name and data type to point to the table
- Lastly, let's add the foreign key constraint...

Altering Constraints

- ```
  ALTER TABLE class_registration
  ADD CONSTRAINT fkey_grade FOREIGN KEY
  (grade_id) REFERENCES grade(id);
  ```
- This adds the foreign key constraint. Note that the syntax is the same as the CREATE TABLE statement has
- It also gives the foreign key a name. This was possible in the CREATE TABLE statement, just like UNIQUE constraints. If you didn't specify a name one was automatically added
- When working with ALTER TABLE names are important...
  ```

Altering Constraints

- `ALTER TABLE class_registration DROP CONSTRAINT fkey_grade;`
- This would remove the foreign key constraint previously set. Notice that it must be dropped by name, which shows that it is important to choose an easy to remember name for your constraints

Indexes

- Another type of "constraint" that hasn't been mentioned yet is an index
- Indices do not actually force any requirements on your data entry or have any benefit to referential integrity
- Instead, they're all about *speed*

Indexes

- Another type of "constraint" that hasn't been mentioned yet is an index
- Indices do not actually force any requirements on your data entry or have any benefit to referential integrity
- Instead, they're all about *speed*

Indexes

- An index will keep all the values in a specified column in a ‘cache’ (database-specific). In essence, it will keep the values as available as possible so reading them will be extremely fast
- In MySQL for example indexes are stored in memory rather than disk, making any query using them to be almost instant in data retrieval

Indexes

- Primary keys and Foreign keys, as they are often used in all databases, always have indices created for them
- In a larger database you will see this if you attempt to use Joins with or without primary keys, or use a non-indexed field in the WHERE clause

Indexes

- `CREATE INDEX idx_class_grade ON class_registration(grade);`
- `DROP INDEX idx_class_grade;`

Indexes

- `CREATE INDEX idx_class_grade ON class_registration(grade);`
- `DROP INDEX idx_class_grade;`
- This is very similar to previous syntaxes that you have seen. You call CREATE INDEX, give the index a name of your choosing, then specify what table and column(s) should be indexed
- Note -- The DROP INDEX does not need a table specified. This is because indices are global. This also means you really need to choose a strong name since only one per database is allowed

Indexes

- As indices are already automatically created for Primary & Foreign keys, you won't have to create any for those fields
- You *should* however create indices for any columns that you will often be using in a query, such as in the WHERE clause
- The only con to creating extra indices is that your database will take up more disk space to hold the index, and INSERT/UPDATE operations will take slightly longer as they need to update the index

A Short Database Example

Regular Expressions

- Wait, what's this?
- Due to popular demand (and extra time) I'd like to show you how I managed to add so much data to the example database without tearing my hair out writing INSERT statements
- I used a little programmer trick called 'Regular Expressions'

Regular Expressions

- Put simply, regular expressions introduces an unfortunately confusing syntax for processing patterns in text
- Emphasis on patterns; your data needs to already be decently structured
- And it processes it *well*. Like, really well. Like, 10,000 rows out of nowhere well.
- I'm quite enamored with their power, so excuse my giddiness

Regular Expressions

- **.** A single character
- **+** One or more characters
- ***** Zero or more characters
- **?** Zero or one characters
- **^** the beginning of the line
- **$** the end of the line
- **[...]** a collection of characters
- **[^...]** anything but a collection of characters (NOT)
- **(...)** a 'capturing group'
- **\d** a digit
- **\t** a tab/indent
- **A-Za-z** all alpha characters (a range)
- **0-9** all digits
- **A-Za-z0-9** all alphanumerics
- **** The 'escape' character

Regular Expressions

- I mentioned it was confusing. When first learning regular expressions you will be constantly using a reference. Wikipedia has an excellent article on them including a larger character reference than what I have provided
- I'm going to focus on what you will likely use: tab-delimited patterns with capturing groups. This is what you will encounter when working with Microsoft Excel data

Regular Expressions

- First, a simple pattern:
- **12515 Meadow Drive, Anchorage**
- \d+ .+, [A-Za-z]+
- 1+ Digits, space, 1+ characters, comma, space, 1+ alpha characters
- Regular expressions -- Easy to write, very hard to read

Regular Expressions

- **12515 Meadow Drive, Anchorage**
- \d+ .+, [A-Za-z]+
- What if some cities have single quotations in their names?
- \d+ .+, [A-Za-z’]+
- How about exclamation marks?
- \d+ .+, [A-Za-z’!]+

Regular Expressions

- Another pattern:
- **(907) 602-2349**
- \(\d+\) ?\d+-\d+
- (, 1+ digits,), space [optional], 1+ digits, dash, 1+ digits
- Note that I used \ to escape the parentheses. Parentheses are used for capturing groups so they must be ‘escaped’ (ignored)

Regular Expressions

- This pattern has capturing groups:
- **Sean Connery**
- ([A-Za-Z]+) ([A-Za-Z]+)
- 1+ alpha characters, space, 1+ alpha characters
- Capturing allows me to use the characters between the parentheses elsewhere

Regular Expressions

- **Sean Connery**
- ([A-Za-z]+) ([A-Za-z]+)
- Using the captured text:
  First: $1
  Last: $2
  - The first parentheses would be inserted in the $1 spot, and the second would be in $2
  - You can see how useful this becomes when working at a large scale

Let's Try It With Some Data

Regular Expressions

- Feel free to look further into regular expressions. They do not necessarily need to be used for SQL
- If you have ever ran into a website that requires an email address or phone number, there is a regular expression behind the scenes ensuring that you enter your information in the proper way
- Anyone having to clean up or process large amounts of structured text will also benefit from an understanding of regular expressions

Reminders

- Lab Wednesday in SSB 172 for your Individual SQL projects (Due 12/14)
- Final Test Monday in the classroom, 4pm-6:45pm
- The test will *not* be cumulative but be prepared for it to cover all of units 10 and 11 which will strongly pull from unit 9's content
- Expect a question about regular expressions, but you will not need to use them on the test
- A 3"x5" index card will be allowed. Extra study time will be on Sunday again from 2-4.

Reminders With Emphasis

- Study all of the two units! Without homework assignments you may discover you know less than you will need on the test
- Because the test has less content to cover, more points will be assigned to each question. This means you can lose many more points if you miss a topic!

Pizza?