CS 215
Web Oriented Programming

Databases & SQL

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs215/

Readings

- Chapter 8 & 9

- Assignment 4 is due in a 3 weeks (Nov 21)

Introduction to SQL

- Structured Query Language (SQL) is a standard language for specifying access to and modifications of relational databases
  - core SQL is supported by all major database vendors
    - create and delete tables
    - insert data
    - update data
    - delete data
    - query/filter data
  - some databases have extra language constructs that are specific to their databases; we'll stick to the core

Formatting Convention

- SQL reserved words are not case sensitive
- Whether tables and columns are case sensitive or not depends on the database vendor

- In order to avoid confusion and ensure that the SQL commands are clear, it is good to follow some kind of convention
  - all SQL language constructs in ALL CAPS
  - all table and column names in either lower case or capitalize the first letter

Create Table

- A table can be created in a database with the CREATE command in the following format:

```
CREATE TABLE table_name (
    column_name_1 data_type constraints,
    …
    column_name_n data_type constraints,
    key_constraints
);
```

- The common *data_type* options are:
  - integers: INTEGER or INT; SMALLINT
  - floats: FLOAT(n); REAL; DOUBLE
  - fixed length strings: CHARACTER(n) or CHAR(n)
  - variable length strings: VARCHAR(n)
  - Boolean values: BOOLEAN
  - Dates: DATE, DATETIME, TIMESTAMP

Constraints

- There are several *constraints* that can be applied to the columns in a table:
  - value must be present: NOT NULL
  - automatically updating integer values: AUTO_INCREMENT

- There are two common options for *key_constraints*:
  - unique identifier for row: PRIMARY KEY
  - link to unique identifier in another table: FOREIGN KEY

- These key constraints allow for fast access to the data

Examples

```
CREATE TABLE Users (
   user_id INT NOT NULL AUTO_INCREMENT,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   email VARCHAR(512) NOT NULL,
   PRIMARY KEY (user_id)
);

CREATE TABLE Orders (
   order_id INT NOT NULL AUTO_INCREMENT,
   uid INT NOT NULL,
   total FLOAT(2) NOT NULL,
   shipped BOOLEAN,
   complete BOOLEAN,
   PRIMARY KEY (order_id),
   FOREIGN KEY (uid) REFERENCES Users(user_id)
);
```

Indexes

- When searching for information stored in a table, the normal process is to search record by record
  - for a table with a small number of records, this is not a problem
  - for tables with many records, things will get slower and slower as more data is added
- Solution: add indexes to the fields against which searching/querying will be done
  - the PRIMARY KEY and FOREIGN KEY specifications are doing this for us
  - however, we may want to add an index to other fields as well

Index specification

- Indexing a field is as simple as specifying what is to be indexed, and for strings, how much of it

```
CREATE TABLE Users (
   user_id INT NOT NULL AUTO_INCREMENT,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   email VARCHAR(512) NOT NULL,
   PRIMARY KEY (user_id)
   INDEX (last_name (4)),
   INDEX (email(30))
);
```

- Specifying the index properly requires that you know something about the data and how it will be queried

Drop

- Deleting a table (or whole database) can be done with a simple and short statement in the following format:

```
DROP (TABLE | DATABASE) [IF EXISTS] name;
```

- Examples:

```
DROP TABLE Users;
DROP TABLE IF EXISTS Orders;
DROP DATABASE IF EXISTS OnlineStore;
```

- The IF EXISTS option avoids errors when the database or table is not present

Alter

- Rather than deleting the entire table every time you want to make a change, the alter command allows you to modify the table structure
- Examples:
  - ALTER TABLE books MODIFY year SMALLINT;
  - ALTER TABLE books ADD author VARCHAR (255);
  - ALTER TABLE books CHANGE author authors VARCHAR (512);
  - ALTER TABLE books DROP pages;

Insert

- The INSERT command can be used to add a row of data to a table, using the following format:

```
INSERT INTO table_name
(column_name_1, column_name_2, …, column_name_n)
VALUES
(value_1, value_2, …, value_n);
```

- The correspondence between column names and values is based on their order in the command
- If a NOT NULL constraint exists on a column, and that *column_name* is not included, an error will be reported
- If an AUTO_INCREMENT constraint exists on a column, it can be excluded and the next logical value will be provided

Examples

```
INSERT INTO Users
(first_name, last_name, email)
VALUES
("Orland", "Hoeber", "orland.hoeber@uregina.ca");

INSERT INTO Orders
(uid, total)
VALUES
(1, 24.38);
```

Update

- The UPDATE command can be used to change one or more values for a row (or set of rows) of a table:

```
UPDATE table_name
SET
column_name_1 = value_1,
column_name_2 = value_2,
…
column_name_n = value_n
WHERE condition
```

- The *condition* can be a comparison on a key, or a more complex comparison that results in multiple rows being updated

Examples

```
UPDATE Users
SET last_name = “Hoeber”
WHERE user_id = 1;

UPDATE Orders
SET complete = true
WHERE shipped = true;
```

Delete

- The DELETE command removes one or more rows from a table, depending on the condition

```
DELETE FROM table_name
WHERE condition
```

- The *condition* can be a comparison on a key, or a more complex comparison that results in multiple rows being deleted
- In most cases, deletes and updates cannot be undone, so be careful

Examples

```
DELETE FROM Users
WHERE user_id = 1;

DELETE FROM Orders
WHERE shipped = true;
```

Select

- The Select command is used to choose a subset of the table (both over the columns and rows)

```
SELECT column_names
FROM table_name
[WHERE condition]
[ORDER BY order_condition];
```

- The *column_names* specifies a subset of the columns over the *table_name*
- The *condition* specifies a subset of rows
- The WHERE and ORDER BY clauses are optional

Examples

```
SELECT user_id, email
FROM Users
WHERE user_id = 1;

SELECT *
FROM Orders
WHERE shipped = true
ORDER BY total DESC;

SELECT DISTINCT email
FROM Users;
```

Where Clause

- The WHERE clause in a SELECT statement limits the number of rows that will be returned to those that match the criteria
  - Full matches use comparison operators (=, <, >, etc.)

```
SELECT email FROM Users
WHERE login_count > 1;
```

  - Text can be matched using the LIKE operator

```
SELECT email FROM Users
WHERE email LIKE “%uregina.ca”;
```

Order By & Limit

- The ORDER BY clause will sort the results

```
SELECT email FROM Users
WHERE login_count > 1
ORDER BY email;
```

- The LIMIT clause will limit the number of results returned

```
SELECT email FROM Users
ORDER BY login_count DESC
LIMIT 20;
```

Join

- It is possible to automatically join the data from multiple tables into the results from one query
  - you can think of this as building a temporary table from which the data is accessed
  - some extra constructs are needed:
    - identifying multiple tables in the FROM clause
    - specifying how the tables are linked in the WHERE clause
    - explicitly identifying the table names in the SELECT clause

Example

```
SELECT Users.user_id, Users.email, Orders.total
FROM Users, Orders
WHERE Users.user_id = Orders.uid AND Orders.complete = true
ORDER BY Users.user_id ASC;
```

Joins using the JOIN clause

- The method for joining tables above is not as explicit as we would like
  - any users that do not have orders will be excluded

- The JOIN clause allows us to have a bit more control

```
SELECT column_names
FROM table_name_1
[INNER | LEFT | RIGHT | FULL] JOIN table_name_2
ON table_name_1.column_name = table_name_2.column_name
[WHERE conditions]
[ORDER BY order_conditions]
```

Examples

```
SELECT Users.user_id, Users.email, Orders.total
FROM Users
LEFT JOIN Orders
ON User.user_id = Orders.user_id
WHERE Orders.complete = true
ORDER BY Users.user_id ASC;
```

- LEFT JOIN will choose all rows from the first table, even if there isn't a matching row in the second table
- RIGHT JOIN will choose all rows from the second table, even if there isn't a matching row in the first table
- INNER JOIN will choose only rows where there is a match (same as specifying the join in the WHERE clause)
- FULL JOIN will choose all rows in both tables, even if there isn't a match

These are Just the Basics

- This small set of SQL commands are just the basics
  - this should be all that is needed for your assignments

- There is a wide range of more complex commands that can be used
- For more information:
  - Chapter 8 of the textbook
  - http://www.w3schools.com/sql/default.asp

Homework

- Read Chapter 10
- Next topic: MySQL & PHP
- Upcoming deadlines:
  - Assignment 4: Nov 21@ 11:55PM