Databases and Info Systems

Structured Query Language (SQL)

Dr. Seán Russell
sean.russell@ucd.ie,

School of Computer Science,
University College Dublin

February 23, 2020

About SQL

- The name means Structured Query Language
- SQL is an extremely powerful language for querying relational databases
- SQL was first proposed in 1974 by IBM and first implemented in 1981
- Most relational systems support the basic functionality of the standard and offer custom extensions

Pronouncing SQL

- The pronunciation used varies depending on the company
- S.Q.L (Saying individual letters) is used by
  - IBM
  - MariaDB
  - MySQL
  - PostgreSQL
- Sequel is used by
  - Microsoft
  - Oracle

Different SQL Version

- Each database management systems will implement different components of the language
- We will focus on the version of SQL supported by MySQL
  - MySQL manual: http://dev.mysql.com/doc/refman/8.0/en/
- However, we will also see some features from other RDBMS systems
  - Some things that MySQL does not support
  - Some things from SQLite

Installing MySQL

- Instructions for installing MySQL on Windows and Mac are available on Moodle
- You should install MySQL and be able to use it for week 3
- This will allow you to do some practice and try out solutions to the quizzes

Table of Contents

Creating, Using and Viewing Databases

- When you log in to MySQL, the first thing you must do is select a database to use.
- Each database consists of a group of tables (relations) that make up the database.
- You can find a list of databases using the following command:
  - `SHOW DATABASES;`
- To create a new database (e.g. named "my_db"):
  - `CREATE DATABASE my_db;`
- To select a database (e.g. named "my_db"):
  - `USE my_db;`

What tables are in my database?

- After selecting a database to use, you can see a list of tables (relations) in that database
  - `SHOW TABLES;`
  - For a new database, it will display the message Empty Set.

What is in my Table?

- If you want to see the schema of a table (the "schema" is the structure of a database), use the "describe" command (e.g. for a table called "my_table"):
  - `DESCRIBE my_table;`

Domains

- We discussed the concept of a **domain** in the last lecture:
  - The set of all possible values that an attribute may contain
- When using standard SQL we must understand the different domains and how they are defined
- Some implementations may use different domains, or have different names for similar domains
- There are two categories of domains
  - Elementary (predefined by the standard)
  - User-defined (not supported by MySQL)

Characters and Strings

- A **CHAR** data type is a fixed-length string (if the data is shorter than the length, MySQL will store space characters in the empty space)
- A **VARCHAR** data type is a variable-length string
- In both cases, we must state the maximum length when we create the attribute
  - `CHAR(30)`
  - `VARCHAR(30)`

Restricted Types

- The **ENUM** type allows us to restrict an attribute to a particular set of strings, e.g.:
  - `ENUM("small", "medium", "large")`
- This domain will only allow one of these three values to be inserted into this attribute

Large Text

- For very large amounts of data, you can use a **BLOB** (stands for Binary Large OBject) or **TEXT** type
- There are different sizes of these types (**TINYTEXT**, **MEDIUMTEXT**, **LONGTEXT**, etc.).
- https://dev.mysql.com/doc/refman/8.0/en/string-type-overview.html

Number Types

- Standard integer types are **INT** (or `INTEGER`) or **SMALLINT**
  - MySQL also supports other integer types: `TINYINT`, `MEDIUMINT`, `BIGINT`
- Floating point types such as FLOAT or DOUBLE.
  - These are similar to float and double datatypes in C or Java. They are not exact.
- Fixed point types (exact values up to a specific number of decimal places): **DECIMAL** (also called **NUMERIC**)
  - DECIMAL(P,S)
  - P is total number of digits
  - S is number of digits after decimal point
  - e.g. DECIMAL(5,2) stores values from -999.99 to 999.99

Numeric Types: Be Careful!

- A golden rule of choosing data types:
  Not everything that looks like a number is actually a number!
- General strategy:
  - Only choose numeric types if you need to **perform calculations** on the data
- Student numbers, phone numbers are NOT numeric data: they are character data
  - We don't need to add or subtract them
  - What happens if we store 06373313 as an INT type?

Date and Time

- Describe an instant in time.
- Important to use the right data types, because there are time and date functions that can operate on these
  - E.g. find all students who have registered within the last year
  - E.g. Find all appointments between 2pm and 5pm today
- `YEAR`
- `DATE`
  - `"YYYY-MM-DD"`
- `TIME`
  - `"HH:MM:SS"`

  `DATETIME`
  - `"YYYY-MM-DD HH:MM:SS"`

User Defined Domains

- User Defined Domains are useful for abstracting common attribute types between tables into a single location for maintenance
  - These are not supported by MySQL
- For example, an email address column may be used in several tables, all with the same properties
- Define a domain and use that rather than setting up each table's constraints individually

User Defined Domains

- A domain is characterised by
  - name
  - elementary domain
  - default value (optional)
  - set of constraints (optional)
- Syntax:
  - `CREATE DOMAIN DomainName as ElementaryDomain [ DefaultValue ] [ Constraints ]`
- Example:
  - `CREATE DOMAIN grade AS SMALLINT DEFAULT 0 CHECK(VALUE=> 0 AND VALUE =< 100)`

Default Values

- Define the value that the attribute must store when a value is not specified during row insertion
- Syntax:
  - `DEFAULT value`
- Value represents a value compatible with the domain, in the form of a constant or an expression
- Example:
  - `DEFAULT 2`
  - `DEFAULT "2016-01-01"`
  - `DEFAULT NOW()`

Tables

- An SQL table definition consists of
  - an ordered set of **attributes**
  - a (possibly empty) set of **constraints**
- Statement CREATE TABLE defines a relation schema, creating an empty instance
- Syntax:

```
1 CREATE TABLE table_name (
2       attr_name DOMAIN [DEFAULT_VALUE] [CONSTRAINTS]
3       [, attr_name DOMAIN [DEFAULT_VALUE] [CONSTRAINTS] ]
4       [, OTHER_CONSTRAINTS]
5 );
```

Table Example

- employee(emp_num, first_name, last_name, join_date, dept)

```
CREATE TABLE employees (
   emp_num INT,
   first_name VARCHAR(20),
   last_name VARCHAR(20),
   hire_date DATE,
   dept INT NOT_NULL
);
```

Constraints

- Constraints are rules that the data in a database must follow
- If we define the constraints when we create the database, the DBMS can enforce the constraints and make sure that the data doesnt break the rules
- Two Types:

  Intra-relational constraints are constraints that operate within one table (relation)

  Inter-relational constraints are constraints that operate between multiple tables

Intra-Relational Constraints

- Constraints are conditions that must be verified by every database instance
- Intra-relational constraints involve a single relation
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY

NOT NULL Constraint

- In SQL, the special value NULL is used to mean unknown
- A NOT NULL constraint states that an attribute can never be NULL
- An update that attempts to set a value to NULL will be rejected by this constraint

UNIQUE Constraint

- UNIQUE defines a key
  - i.e. the value stored in an attribute or a set of attributes cannot be repeated
- For single attributes: add UNIQUE, after the domain

- For multiple attributes (as other constraints):
  `UNIQUE( attribute_name , attribute_name )`

PRIMARY KEY Constraint

- Defines the primary key of a table
- There can only be one primary key for a table
- This implies the NOT NULL and UNIQUE constraints
- For single attributes: add PRIMARY KEY, after the domain
- For multiple attributes (as other constraints): `PRIMARY KEY( att_name , att_name )`

Example of Intra-Relational Constraints

- Consider a table with the attributes `given_name` and `family_name`
- If we want to have the combination of these attributes be unique we first declare them
- Later in the other constraints section we add the constraint to state they should be unique

```
1 CREATE TABLE names (
2     given_name  VARCHAR(20) NOT NULL,
3     family_name VARCHAR(20) NOT NULL,
4     UNIQUE(given_name,family_name) );
```

Example of Intra-Relational Constraints

```
1 CREATE TABLE names (
2     given_name  VARCHAR(20) NOT NULL,
3     family_name VARCHAR(20) NOT NULL,
4     UNIQUE(given_name,family_name) );
```

- Notice the difference between the above and the **stricter** definition below

```
1 CREATE TABLE names (
2     given_name  VARCHAR(20) UNIQUE,
3     family_name VARCHAR(20) UNIQUE );
```

AUTO_INCREMENT fields

- One other useful feature is to create AUTO_INCREMENT fields (attributes)
- Often, a table will have a primary key that is an INT type that has a unique identifier for each row
  - This is not always a good idea, but is often useful
- If this is defined as AUTO_INCREMENT, then MySQL will automatically give the next available value to each row that is inserted
- For this to happen automatically, we insert a new row with a NULL value for this field

AUTO_INCREMENT Example

```
1 CREATE TABLE animals (
2    id INT PRIMARY KEY AUTO_INCREMENT,
3    name VARCHAR(30)
4 );
```

- If a NULL value is inserted into the id attribute, the next integer value will be used
- It can be a little complicated if data has been deleted
- MySQL has a counter to remember the last automatic value it chose, and will increment this counter each time

Changing the Schema

- `ALTER TABLE` - Change attributes and/or constraints
  - https://dev.mysql.com/doc/refman/8.0/en/alter-table.html
- `DROP DATABASE databasename;` - delete an entire database
- `DROP TABLE tablename;` - delete an entire table

Alter Examples

```
1 ALTER TABLE employees ADD COLUMN birth_date DATE;
```

Add a new columns to a table

```
1 ALTER TABLE employee DROP COLUMN hire_date;
```

Remove a column from a table

```
1 ALTER TABLE employees MODIFY emp_no CHAR(10);
```

Change the domain of an attribute

```
1 ALTER TABLE employees CHANGE birth_date dob DATE;
```

Change the name of an attribute

Inserting Data

- When inserting data into a table, we have two options
- Insert values
- Insert using a query

Inserting Values

- Syntax:

```
1 INSERT INTO table_name [ (attribute_list) ] VALUES
      (list_of_values);
```

- Example (Only some attributes):

```
1 INSERT INTO departments (dept_num, name) VALUES (9,
      "Finance");
```

- Attribute list is not required if we are inserting all values

```
1 INSERT INTO departments VALUES (9, "Finance", 34);
```

Inserting From A Query

- Syntax:

```
1 INSERT INTO table_name SELECT ...;
```

- Example:

```
1 INSERT INTO names SELECT first_name, last_name from
      employees;
```

Inserting Data

- The ordering of the attributes (if present) and of values is meaningful (first value with the first attribute, and so on)
- If we don't include an attribute list, all the attributes in the table are considered, in the same order as in the table definition
- If the attribute list does not contain all the table's attributes, the remaining attributes are assigned the default value (if defined) or the null value

Inserting Multiple Rows of Data

- It is also possible to insert multiple rows in the same query
- This will be faster than using a separate INSERT statement for each row
- With attribute list:

```
1 INSERT INTO departments (dept_num, name) VALUES
2   (9, "Finance"),
3   (10, "Accounting");
```

- Without attribute list:

```
1 INSERT INTO departments VALUES
2   (9, "Finance", 34),
3   (12, "Agriculture", 8),
4   (45, "Physics", 89);
```

Deleting Data

- Syntax:

```
1 DELETE FROM table_name [ WHERE condition ];
```

- Example - Remove the finance department:

```
1 DELETE FROM departments WHERE name = "Finance";
```

- Example - Remove the departments with no employees:

```
1 DELETE FROM departments WHERE dept_num NOT IN (SELECT dept
      from employees);
```

  - Here the statement IN tests if a value is contained in a set of values

- The WHERE clause works the same as in SELECT, but all matching rows are deleted

Deleting Data - Effects

- The removal may produce deletions from other tables if a referential integrity constraint with cascade policy has been defined
- If there is no WHERE clause, delete removes all the rows in the table
- To remove all the rows from Departments

```
DELETE FROM department;
```

- To remove table departments completely

```
DROP TABLE department;
```

Deleting A Row

- If deleting one row from a table, we should use the primary key in the WHERE clause
- The whole reason we have primary keys is to uniquely identify a row of data
- Using other attributes for deleting things means there is a risk
- `DELETE FROM employees where first_name = "Sean";`
- What if two employees have the name Sean?
  - We should use `emp_num` instead

Updating Data

- The UPDATE statement is used to change the attributes of rows already in a table
- Syntax:

```
UPDATE table_name
  SET attribute = <expression|SELECT ...|NULL|DEFAULT>
  {, attribute = <expression|SELECT ...|NULL|DEFAULT>}
  [ WHERE condition ];
```

- Set the salary of employee 1555 to 54000:

```
UPDATE jobs SET salary=54000 WHERE emp_num = 1555;
```

- Give all Assistant professors a 5% increase in salary

```
UPDATE jobs SET salary=salary*1.05 WHERE title="Assistant
    Professor";
```