LBSC 690: Information Technology
Lecture 06
RDBMS and SQL

William Webber
CIS, University of Maryland

Spring semester, 2012

The Relational Database Management System

The Relational Database Management System (RDBMS)

- A computer program, often running on a server
- Constructs and maintains relational databases and their data
- Provides an interface that allows users, or more generally programs, to create, retrieve, update, delete, and query data

Database vs. DBMS

Note possible confusion in terminology:

Database
- A schema of multiple related entities
- An instantiation of that schema, with its data

DBMS A (server) program that holds databases

A (R)DBMS has multiple (generally independent) databases. A database has multiple (related) tables. A table has multiple fields.

But people sometimes refer to the DBMS as the “database”.

Classes of DBMS

Three main classes of DBMS:

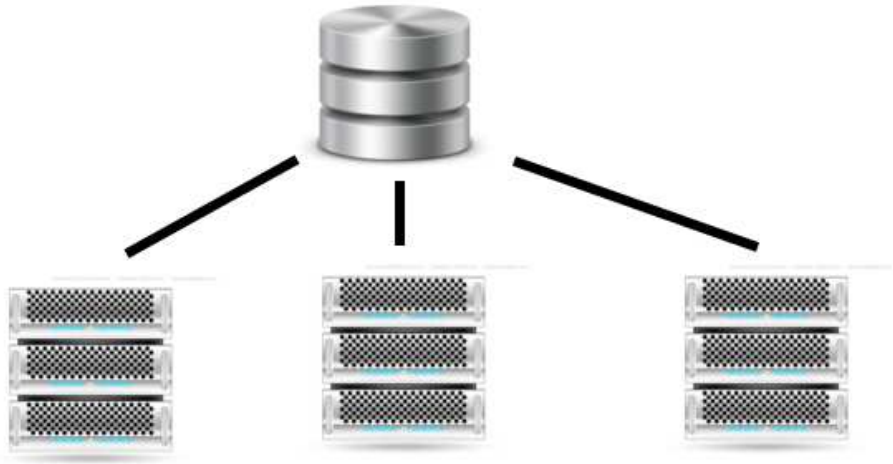
Embedded Database runs entirely inside a single program, as a “library” of that program. Example: SQLite.

Desktop Database runs as a client application, typically on a desktop machine, accessed directly by user (often with GUI interface). Example: MS Access.

Server Database runs as server in client-server architecture. Example: MySQL, Oracle.

We’ll look at server-style DBMS for the next few slides.

DBMS architecture



Client-server DBMS architecture

- A DBMS can run on a server by itself and be connected to over the network
- A DBMS can serve multiple programs
- The primary customer of DBMS are programs, not (directly) users

Services of a DBMS

As well as basic database management, a DBMS provides services such as:

Concurrency multiple users and operations can be processed in parallel

Transactions complex operation either fully completes or is rolled back

Replication database can be replicated over multiple nodes (servers), for performance and/or redundancy

Access control access rights of users and applications can be controlled (e.g. read-only permissions, deny access to certain tables or databases)

Consistency checking database can check that relations between entities are consistent (e.g. foreign keys point to something)

MySQL

We'll be using the MySQL database

- Open source and cross-platform
- Widely used in website development
- Increasingly acquiring enterprise features
  - so much so that Oracle bought them to stop them being a threat

PhpMyAdmin

As an interface to MySQL, we'll be using PhpMyAdmin

- Web-based semi-graphical interface to MySQL
- Allows (reasonably) user-friendly interface for:
  - creating tables
  - inserting data
  - performing simple queries
- Also allows us to drop down in SQL (the programming language for RDBMS – see later) if required

Setup

- I've installed MySQL and PhpMyAdmin on my server; PhpMyAdmin is accessible at: `https://codalism.com/phpmyadmin`
- Each student has had an account created for them, and a database created for them with the same name as their account (login details sent separately)
- Note that we have:
  - Multiple databases hosted in the one DBMS
  - Access control (one account can't see another account's databases)
  - PhpMyAdmin acting as a program that talks to the DBMS (and to us)

Demo: logging in, and basic view

- *Log in as admin user, view multiple databases*
- *Log in as normal user, view only single database*
- *Demonstrate separate, CLI interface to same RDBMS*

Creating a table in PhpMyAdmin

- First, select the database
- Then, click “create table” in left-hand
- Type in table name (avoid spaces and punctuation except for “_” (underscore))
- Select “InnoDB” as Storage Engine (for checking of foreign keys)

Demo: create table

- *Select personal database*
- *Hit “create table”*
- *Enter table name*
- *Select “InnoDB” as Storage Engine*

Table columns

Fields that you want to care about are:

Column the name of the column (avoid spaces and punctuation except for "_" (underscore))

Type type of column. Choose "VARCHAR" (variable-length character) for a brief character column, "TEXT" for a large amount of text (e.g. a paragraph of free text)

Length if you've chosen "VARCHAR", choose maximum length of the field. Better too big than too small!

Defining primary keys

For primary key columns:

- Make the type “INT”
- Set index to “PRIMARY”
- Select “AUTO_INCREMENT”

This way, every time we add a new record, the database will create a new primary key for it, a number one bigger than the last key

Demo: defining student table

- *Create primary key field “ID”*
- *Create additional fields “GIVEN” and “FAMILY”*

Defining relations

For foreign key columns:

- Create the table with the primary key you're linking to first!
- Make the foreign key type "INT"
- Set the index to "INDEX"
- Once created, go to "Structure" tab and select "Relation view"
- Under the foreign key, select the primary key it links to

NOTE: you'll get different options depending upon the Storage Engine type

Demo: defining drawing table

- *Create table “Drawing”*
- *Add foreign key back to “Student”*
- *Enforce foreign key relationship under “Relations view”*

Adding data

To add data, go to the “Insert” tab

- Select the table you want to add to, in the left-hand menu
- Go to the “Insert” tab.
- Insert the desired data (you can ignore the “Function” column).

Note that you have to manually select the correct foreign keys!

Demo: adding data

- *Add two students*
- *Add three drawings*

Browsing data

- The “Browse” tab gives you a paginated view of the data in a table
- Only becomes highlighted when some data has actually been added

Demo: browse tab

- *Go to browse tab for “Drawing”*

Searching

- The “Search” tab allows for simple searches
- Each field of the table can be tested
- For more complex, in particular multi-table, queries, we need to use SQL (see below)

Demo: search

- *Go to search tab for “Student”*
- *Search for students with given name “Jane”*

The RDBMS interface

- To use an RDBMS, we need a way to interact with it
- Method needs to be (reasonably) standard (not a different interface for each DBMS).
- Method also needs to be programmatic (i.e. a GUI alone is not adequate)

The Structured Query Language

SQL (Structured Query Language): the standard RDBMS interface

- A language, in the sense that HTML is a language.
- With a defined syntax.
- Supporting:
  - Data definition (creation of tables)
  - Data manipulation (creating, updating, and deleting records in tables)
  - Data queries (retrieving records by id or by query expressions)

Note that PhpMyAdmin communicates with MySQL using SQL.

Data definition language

```
CREATE TABLE Student (
    id int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    given varchar(30) NOT NULL,
    family varchar(30) NOT NULL,
    year_lvl int
);
```

- Basic statement is CREATE TABLE
- Contains list of field with name, type, and constraints
- Here, for instance, given is a character field of maximum length 30 which cannot be empty (NULL)

PhpMyAdmin will give you SQL statements to create a table if with the “Export” tab.

Demo: DDL from PhpMyAdmin

- *Select “Student” table*
- *Select “Export” tab*
- *Select “SQL” as format*
- *View SQL for creating Student table*

Data insert

```
INSERT INTO Student (given, family, year_lvl) VALUES
('Jane', 'Smith', 6),
('Anne', 'Black', 4);
```

- Insertion is performed with the INSERT INTO statement.
- Note that character values must be quoted, e.g. 'Jane', but integer values are not

PhpMyAdmin echoes back the SQL statement when you do an insert.

Demo: data insert

- *Select the “Student” table*
- *Select the “Insert” tab*
- *Add a new student, and hit “Go”*
- *Observe the SQL statement (note: it is more verbose than one might write by hand; e.g., it includes the database name, and quotes field names)*

Data query

```
SELECT family, year_level FROM Student
    WHERE given='Jane' AND year_level > 5;
```

- Queries are implemented using the
  SELECT fields FROM table WHERE condition statement
- Multiple fields can be extracted; $*$ will extract all fields

PhpMyAdmin shows query sql under “Query” tab.

Demo: data query

- *Select the “Student” table*
- *Select the “Search” tab*
- *Search for students whose given name is “Jane” and whose year level is greater than 5*
- *Observe the SQL statement*

Multiple-table queries and joins

- In modelling, we broke composite information down into separate, linked tables
  - e.g. separate the “Student” from the “Drawing” table
- In querying, we often want to re-unite these linked records into a single results
  - e.g. a list of drawings with the names of the students who drew them
- This re-united of linked records is known as a join

(Inner) joins

```
SELECT Student.given, Student.family, Drawing.title
    FROM Student, Drawing
    WHERE Student.year_level = 5
    AND Drawing.student_id = Student.id;
```

- Joins are implemented using the SELECT statement.
- We specify which table each field comes from
- . . . list the tables
- . . . and link up foreign and primary key:
  - Here, Drawing.student_id = Student.id

(Technically, this is known as an inner join. There are other join types, but they are rare in practice.)

Demo: join query

- *Select the “Student” table*
- *Select the “SQL” tab*
- *Enter the previous query*

Building database applications

- PhpMySql is little more than an administrative interface
- . . . and no user is going to use SQL directly
- Even for data entry, more support is needed (for instance, checking field types; selecting foreign keys)
- For a user-facing application, of course, a much richer interface is required

The three-tier architecture

- Most web applications follow a three-tier architecture:

  Presentation tier The browser window, running on the client side (HTML, CSS, perhaps some client-side programming)

  Logic tier An application, written in a programming language, running on the server side (PHP, Java, ASP . . .)

  Data tier The database, running behind the server side.

The web application

- We need to implement the logic tier, as a web application or program
- This talks HTML over HTTP to the browser at the front end
- And talks SQL to the database on the backend

Object-relational mapping (ORM)

- In the application program, we often use a layer (library) that “hides” the SQL from us,
- . . . lets us deal instead with simpler (though less powerful) representations of the database, in the idiom of the programming language
- One such representation is the Object-Relational Mapping, which wraps relational SQL in an object-oriented model

We’ll look in more detail at ORM later in the course . . .

Take-away points: RDBMS

- A (relational) database management system ((R)DBMS) manages our access to a (relational) database
- Full-scale RDBMS run as independent servers, often on separate computers
- RDBMS provides additional services such as transactions, parallelization, redundancy, etc.

But, we need an interface to talk to a RDBMS.

Take-away points: MySQL and PhpMyAdmin

- MySQL is a widely-used RDBMS, particularly in web applications
- PhpMyAdmin provides an administration-level interface to it
- MS Access is an alternative for a client-only, single-user systems

But, generic admin interfaces are not adequate for real users.

Take-away points: SQL

- The Structured Query Language (SQL) is the standard language for communicating with relational databases.
- Provides expressions for defining the schema, inserting and updating data, and querying that data
  - including joins to reunite records separated in the schema
- Primarily useful as a programmatic interface, or an emergency admin interface in the hands of power users.

But, no user is going to talk SQL to the database.

Take-away points: Database architecture

- Generally, an application needs to be developed to interface between user and database
- In web (and other) development, a three-tier model of presentation, logic, and data is used
- The logic layer talks HTML to the browser, SQL to the database

We'll start looking at the application layer next week.