DBMS: SQL Injection

Table of contents

1 Relational Databases

A **relational database** is a system for storing data in **tables** made up of **rows** and **columns**. Each row in a table is a record with a unique ID. The unique ID is often called a primary key. Tables can be linked to each other using foreign keys, allowing for complex queries and relationships between data.

Table 1: A database table made from a Student model with sample entries. If the student ID is unique, it can serve as a primary key.

StudentID	FirstName	LastName	Major	GraduationYear
1001	Alice	Johnson	Computer Sci	2025
1002	Bob	Smith	Math	2024
1003	Carla	Nguyen	Biology	2026
1004	David	Lee	History	2025

Keys in one table can be paired or matched with keys in other tables to create **foreign key relationships**. As an example, a City might have a Country that it exists in. Thus, you might have a Country entry in the City table that just points to an entry of the Country table.

In order to prevent duplicate data, ensure data consistency, and make our databases easier to maintain, we go through a **database normalization** process.

2 MySQL

MySQL (frequently pronounced `My sequel`), is an open source database management system. `My` is the daughter of one of the cofounders of MySQL. SQL is a database querying language.

2.1 Installing MySQL Server

If you have not already installed MySQL server on your virtual machine, you can do so by running the following command:

```
sudo apt install mysql-server
```

2.2 Getting to Know SQL

1. Login to MySQL Server as root using the following command. The `-u` flag is for user and the `-p` flag is for password. If the command is executed successfully, you will be prompted for a password. If a password has not been set previously, leave the prompt blank.

   ```
   sudo mysql -u root -p
   ```

2. Create a database named `testdb`

```
CREATE DATABASE testdb;
```

3. Make `testdb` the default database for future commands in this session.

```
USE testdb;
```

4. Create a table for a `team` model. For us a team will be defined as follows:

   - Every team will have an ID. This ID will serve as our primary key. That is, it will be unique for every team that gets created. We will not allow this value to be a NULL value. Also, we will make sure the id will increment automatically when new teams are created, that way we don't need to worrying about knowing what the previously created team's id was. The id will be limited to values between 0 and 65,535 by specifying the type as `smallint(5) unsigned`. Note that the digit 5 is for display width when using the zerofill option.
   - Every team will have a name. The name will not be allowed to be a NULL value. If a name is not provided, then by default, an empty string will be stored. The name allows a variable length string of up to 50 characters.
   - Every team will have a score. The score can be an integer from -2,147,483,648 to 2,147,483,647. The display width for the score should be 10 and the score cannot be null.

```
CREATE TABLE `team` (
    `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    `name` varchar(50) NOT NULL DEFAULT '',
    `score` int(10) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

   Note that the last line with `ENGINE=InnoDB DEFAULT CHARSET=latin1` means that the table sohuld use the storage engine InnoDB (which is the default for MySQL) and to use the character set that covers Western European languages. For better international support, you can use `utf8` for the characterset.

5. Look at the details of the table we created using the following command:

```
DESCRIBE team;
```

   The output should be similar to the following:

```
+--------+-------------------+---------+--------+---------+----------------+
| Field  | Type              | Null    | Key    | Default | Extra          |
+--------+-------------------+---------+--------+---------+----------------+
| id     | smallint unsigned | NO      | PRI    | NULL    | auto_increment |
| name   | varchar(50)       | NO      |        |         |                |
```

```
| score  | int                | NO      |        | NULL    |                 |
+--------+--------------------+---------+--------+---------+-----------------+
```

6. Create a team named `AFC Richmond`.

```
INSERT INTO `team` (name, score)
VALUES ('AFC Richmond', 0);
```

7. Create another team named `Man. City`

```
INSERT INTO `team` (name, score)
VALUES ('Man. City', 0);
```

8. Make a query for all the teams with the `*` symbol.

```
SELECT * FROM team;
```

   The output should be a table with the teams you have created so far. Note that the primary key `id` was assigned for us automatically.

9. You can delete a database with the DROP command.

```
DROP DATABASE testdb;
```

3 SQL Injection

In this section, you will setup a database and a website to interact with that database. After setup, you will perform SQL injection to modify the database in ways not intended by the developers of the webapp.

3.1 Setting up the database

1. If you are not already logged in to MySQL server as the root user, do so now.

```
sudo mysql -u root -p
```

2. Create a database named `cyberdb`.

```
CREATE DATABASE cyberdb;
```

3. It is bad practice to use the root user for everything. Create a user with username `cyberadmin` and password `Cyber!1234`. Note that depending on your setup, MySQL may require you to have a stronger password.

```
CREATE USER 'cyberadmin'@'localhost' IDENTIFIED BY 'Cyber!1234';
```

4. Grant all permissions to the `cyberadmin` user to modify any cyberdb table (i.e. `cyberdb.*`) when logged in locally (i.e. accessing `localhost`).

```
GRANT ALL PRIVILEGES ON cyberdb.* TO 'cyberadmin'@'localhost';
```

5. Disconnect from MySQL server with `ctrl+d`.

6. Note that the format of the next command we enter is as follows:

```
mysql -u <username> -p<password> <database>
```

   In particular, notice there is not a space between `-p` and `password`. If you put a space, then the next assumed value is the database you are connecting to and not the password. The password will be given via a prompt if a space is provided.

   Login to MySQL server with the new user. Notice that we are putting a space after `-p`, thus `cyberdb` is the database, not the password. When you submit this command, you will then be prompted for the password.

```
mysql -u cyberadmin -p cyberdb
```

7. View which databases you have access to.

```
SHOW DATABASES;
```

   At a minimum, you should see `cyberdb` in the output.

8. Make cyberdb the active database.

```
USE cyberdb;
```

9. Create a table named `teams`. This will be similar to the table named `team` in the previous section of this document. **Notice the backtick symbols around the words teams, id, name, and score.**

```
CREATE TABLE `teams` (
    `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    `name` varchar(50) NOT NULL DEFAULT '',
    `score` int(10) NOT NULL,
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

10. Create a table named `acl`. This will hold the entries perataining to access control information for a team in the `teams` table, in particular, the password for a team. This also uses a foreign key relationship to point to the `team_id` that a particular acl entry might be connected to in the `teams` table.

```
CREATE TABLE `acl` (
    `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
    `team_id` smallint(5) unsigned NOT NULL DEFAULT '0',
    `password` varchar(50) NOT NULL DEFAULT '',
    PRIMARY KEY (`id`),
    KEY `acl_team_id` (`team_id`),
    CONSTRAINT `acl_team_id`
        FOREIGN KEY (`team_id`)
        REFERENCES `teams` (`id`)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

11. Verify the entry of the two tables.

```
DESCRIBE teams;
```

```
DESCRIBE acl;
```

12. Create two teams.

```
INSERT INTO `teams` (name, score)
VALUES ('WHALE',0);
```

```
INSERT INTO `teams` (name, score)
VALUES ('SEAHORSE',0);
```

13. Verify the entry of the teams in the database. Note the ids of the teams.

```
 SELECT * FROM `teams`;
```

14. Create acl information for the team with id 1 (team WHALE).

```
INSERT INTO `acl` (team_id, password)
VALUES (1,sha('password'));
```

15. Create acl information for the team with id 2 (team SEAHORSE).

```
INSERT INTO `acl` (team_id, password)
VALUES (2,sha('123456'));
```

16. View every entry of acl to see the hashes of the passwords ‘password’ and ‘123456’.

```
SELECT * FROM acl;
```

3.2 Setup the Webapp

1. Install PHP and the Apache2 PHP module.

```
sudo apt update
sudo apt install php libapache2-mod-php php-mysql
```

2. Navigate to `/var/www/html` in a terminal.

```
cd /var/www/html
```

3. If a file named `index.html` exists in this terminal, remove it.

```
sudo rm index.html
```

4. Go to canvas, download the following files:

   - `db.php` - this file contains uses database functions for php
   - `config.php` - this file contains the configuration to access cyberdb.
   - `index.php` - this file contains the

5. Move the files to `/var/www/html`:

```
sudo mv ~/Downloads/db.php /var/www/html/
sudo mv ~/Downloads/config.php /var/www/html/
sudo mv ~/Downloads/index.php /var/www/html/
```

6. Open the config.php file and set the values for `S_ADMIN['MYSQL_USER']`, `S_ADMIN['MYSQL_PASS']` and `S_ADMIN['MYSQL_DATABASE']` to match the values for the user and database you created earlier.

```
...
S_ADMIN['MYSQL_USER'] = "cyberadmin"
S_ADMIN['MYSQL_PASS'] = "Cyber!1234"
S_ADMIN['MYSQL_DATABASE'] = "cyberdb"
...
```

7. Open a web browser and navigate to http://localhost. You should see a webpage with a form expecting a Team Name.

8. In the Team Name field, type one of the team's names to see the score of that team (ex: type `WHALE` or `SEAHORSE`).

9. Click the "Submit Query" button or hit enter to see the score.

3.3 Injecting a Query

As you can imagine, when you submit the form, there is a query running in the background that looks something like the following:

```
SELECT `name`,`score` FROM teams WHERE `name`='<whatever the user entered>'
```

So if we enter WHALE, then the query turns into the following:

```
SELECT `name`,`score` FROM teams WHERE `name`='WHALE'
```

Ideally, a user of a webpage would not be able to submit SQL queries into a webform, however the website you are now hosting at http://localhost is vulnerable to SQL injection attacks.

Instead of intering a valid name, we will perform an sql injection attack. Enter **`name' or 'a'='a`** as the team name. Note the lack of opening and close quotes.

This will turn the full query into the following:

```
SELECT `name`,`score` FROM teams WHERE `name`='name' or 'a'='a';
```

Since there is no team named **`name`**, that portion of the query returns false. However, **`'a'='a'`** is always true. Thus the entire WHERE clause becomes true. The result of this is that SQL will return the entire list of teams with their scores.

There are many situations where you would not want to list all entries of a table for privacy concerns.

To prevent SQL injections, you should always sanitize user input and check for invalid inputs, for example, those that could alter SQL queries.

Many moder web dev frameworks, such as Django, tend to have their own Object Relational Model that handles the prevention of SQL injection for developers under most cases.

3.4 Older Injections

Modern versions of PHP prevents multiple SQL queries from being performed in a single submission, but older versions did not. What do you think would have happened if **`'; delete from teams; --`** were submitted?

This would form the string of queries:

```
SELECT `name`,`score` FROM teams WHERE `name`='blah';
delete from teams; -- ' The -- creates a comment
```

The SELECT query returns nothing since there is no team named blah, but the second SQL query would delete everything from the teams table.

Another injection would involve getting the hashes of passwords from another table. Submitting **`blah'; select * from acl; --`** would form the string of queries:

```
SELECT `name`,`score` FROM teams WHERE `name`='blah';
select * from acl; --
```

This would give every entry of the acl table which stores the hashes of passwords.

4 Exporting the Contents of Your Database

This section shows you how to export the contents of your database in a format that you can use to recreate the database easily.

1. Login to the sql server as the root user again.
   ```
   sudo mysql -u root -p
   ```
2. Grant the process privileges to the cyberadmin user.
   ```
   GRANT PROCESS ON *.* TO 'cyberadmin'@'localhost';
   ```
3. You may need to run the following to apply the privileges:
   ```
   FLUSH PRIVILEGES;
   ```
4. Disconnect from mysql server with `ctrl + d`.
5. In the terminal, you can export the contents of your database to a file named using the following command. Note that you may want to place it in a different location than `~/Documents` so modify as desired.
   ```
   mysqldump -u cyberadmin -p --skip-extended-insert \
   --add-drop-database --add-drop-table \
   --databases cyberdb > ~/Documents/cyberdb.sql
   ```
6. Confirm the file named `cyberdb.sql` was created by navigating to where you saved it and using the `ls` command.
7. Confirm the file contains the contents concerning the database you've created by viewing it with `cat` or another desired command.

5 Importing the Contents of A Database

1. Before importing the data, let's remove what you currently have. Login to mysql and remove the database.

```
mysql -u cyberadmin -p
```

2. Remove the database.

```
DROP DATABASE cyberdb;
```

3. Confirm its removal.

```
SHOW DATABASES;
```

4. Disconnect from mysql server with `ctrl + d`
5. Import the database with the following command.

```
mysql -u cyberadmin -p < cyberdb.sql
```

6. Log back into mysql server.

```
mysql -u cyberadmin -p cyberdb
```

7. Confirm the tables are there.

```
SHOW TABLES;
```

8. Confirm the data is there.

```
SELECT * FROM teams;
```

6 More Sample SQL Queries

The following queries are included as additional examples of sql queries you could perform on you database for reference purposes.

1. A query for the hashed password of the `WHALE` team.

```
SELECT `password` FROM acl
WHERE `team_id`=(
    SELECT `id` FROM teams
    WHERE `name`='WHALE'
);
```

The above query has two parts. The inner part gets the id of the team named WHALE. The outerpart uses that id to get the acl corresponding to Whale. The below query is an equivalent query.

```
SELECT `password` FROM acl, teams
WHERE acl.`team_id`=teams.`id` AND teams.`name`='WHALE';
```

A shorter version of the previous form of the query uses variables in place of acl and teams.

```
SELECT `password` FROM acl a, teams t
WHERE a.`team_id`=t.`id` AND t.`name`='WHALE';
```

2. A query using the % symbol. The % symbol is a wildcard in SQL like the * in bash.

```
SELECT `score` FROM teams WHERE `name` LIKE 'WH%';
```

Note that it is case insensitive.

```
SELECT `score` FROM teams WHERE `name` LIKE 'wh%';
```

3. A query that adds 100 points to the WHALE team.

```
UPDATE teams SET `score`=`score`+100 WHERE `name` LIKE '%al%';
```