Security

Christian Rudder

August 2024

Contents

This page is left intentionally blank.

1 Introduction

Hackers, Security issues, and exploits all exists, because the software we use isn't perfect. It's written by humans. One or two edge cases from the best programmers might slip through. This doesn't mean just software, but languages, and even hardware.

The languages we use are just solutions we thought would work. Binary is a solution, it's not the intrinsic solution, but it's a solution. So we built assembly on top, then C, Java, and so on abstracting the difficult parts. But the more we abstract, the more we lose control. A vulnerability can quickly cascade up a chain of abstractions.

2 SQL Basics

2.1 Creating Database & Tables

SQL stands for "***Structured Query Language***," used to query against databases with tables containing columns of data, which most often relate to each other.

Using key words like `SELECT`, `FROM`, `WHERE`, ignoring case. It's good practice to use **all caps for SQL keywords**, and **lowercase for table and column names**. Here's a simple example:

```sql
SELECT * FROM my_table
```

Selects all (*) columns from the table `my_table`.

Now, we are a record company with bands, albums, and songs:

```sql
CREATE DATABASE test; -- creating a test database
DROP DATABASE test; -- deleting the test database


CREATE DATABASE concise_records; -- creating our database
USE concise_records; -- selecting our database to run commands on it


CREATE TABLE bandds (); -- creating a table for our bands
DROP TABLE bandds; -- deleting it because of our typo


-- Create bands table: artist name (at most 255 characters), CANNOT be NULL/EMPTY
CREATE TABLE bands (
    name VARCHAR(255) NOT NULL
);


-- Add id column to bands, auto increment, not NULL, make this column important
ALTER TABLE bands
ADD COLUMN id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
```

We created the database, `concise_records`, and a table `bands` with two columns: `name` and `id`.

The PRIMARY KEY acts as an ID for each row, useful for drawing a thread of relationships between tables where the ID is present.

Definition 2.1: Primary Key

A column which identifies each row in a table. It must be unique, and it cannot be NULL.

To create our albums table:

```sql
-- Create albums table: album id, album names, release dates (optional)
CREATE TABLE albums (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    release_date DATE,
    PRIMARY KEY (id)
);


-- We need a way to link bands to albums
-- Create a column in albums pointing to bands' id column
ALTER TABLE albums
ADD COLUMN band_id INT NOT NULL FOREIGN KEY REFERENCES bands(id);
```

We created the albums table with three columns: id, name, and release_date, and band_id. A FOREIGN KEY uses the PRIMARY KEY of another table to establish a relationship between them.

Definition 2.2: Foreign Key

A column that references another table's PRIMARY KEY.

So far we have:

```sql
-- DB: concise_records
--
--      bands
--      +----+-----------------+
--      | id | name            |
--      +----+-----------------+
--      |    |                 |
--      +----+-----------------+
--
--      albums
--      +----+-----------------+--------------+---------+
--      | id | name            | release_date | band_id |
--      +----+-----------------+--------------+---------+
--      |    |                 |              |         |
--      +----+-----------------+--------------+---------+
```

The database concise_records with our tables bands and albums.

Let's begin to add data to our tables:

```sql
-- Insert 'The Beatles', 'The Rolling Stones', 'The Who' into bands
INSERT INTO bands (name) VALUES ('The Beatles');
INSERT INTO bands (name) VALUES ('The Rolling Stones'), ('The Who');

-- DB: concise_records
--
--       bands
--       +----+----------------------+
--       | id | name                 |
--       +----+----------------------+
--       | 1  | The Beatles          |
--       | 2  | The Rolling Stones   |
--       | 3  | The Who              |
--       +----+----------------------+

-- Insert 'Abbey Road', 'Let It Bleed', 'Who's Next' into albums

INSERT INTO albums (name, release_date, band_id) VALUES ('Abbey Road', '1969', 1);
INSERT INTO albums (name, release_date, band_id) VALUES ('Let Be', '1970', 1);
INSERT INTO albums (name, band_id) VALUES ('Who''s Next', 3);

-- DB: concise_records
--
--       albums
--       +----+----------------+--------------+---------+
--       | id | name           | release_date | band_id |
--       +----+----------------+--------------+---------+
--       | 1  | Abbey Road     | 1969         | 1       |
--       | 2  | Let It Be      | 1970         | 1       |
--       | 3  | Who's Next     |              | 3       |
--       +----+----------------+--------------+---------+
```

Single quotes denote strings. Double single quotes in strings act as single quotes, seen in the above with `'Who''s next'`.

2.2 Queries

SELECT

We can also retrieve data from our tables using the `SELECT`:

```sql
-- Retrieve all columns from bands
SELECT * FROM bands;

-- Query Result:
--       +----+----------------------+
--       | id | name                 |
--       +----+----------------------+
--       | 1  | The Beatles          |
--       | 2  | The Rolling Stones   |
--       | 3  | The Who              |
--       +----+----------------------+
```

Queries to a table return another table.

To list a few commands:

```
-- Retrieve the name column from bands
SELECT name FROM bands;

-- Query Result:
--     +-----------------------+
--     | name                  |
--     +-----------------------+
--     | The Beatles           |
--     | The Rolling Stones    |
--     | The Who               |
--     +-----------------------+
```

```
-- Retrieve the name column from bands, limit to 2
SELECT name FROM bands LIMIT 1;

-- Query Result:
--     +-----------------------+
--     | name                  |
--     +-----------------------+
--     | The Beatles           |
--     +-----------------------+
```

```
-- Retrieve and give aliases to id and name columns from bands
SELECT id AS 'ID', name AS 'Band Name'

-- Query Result:
--     +----+-----------------------+
--     | ID | Band Name             |
--     +----+-----------------------+
--     | 1  | The Beatles           |
--     | 2  | The Rolling Stones    |
--     | 3  | The Who               |
--     +----+-----------------------+
```

```
-- Order bands by name in descending order
SELECT * FROM bands ORDER BY name DESC;

-- Query Result:
--     +----+-----------------------+
--     | id | name                  |
--     +----+-----------------------+
--     | 1  | The Who               |
--     | 2  | The Rolling Stones    |
--     | 3  | The Beatles           |
--     +----+-----------------------+

-- Order bands in ascending order
SELECT * FROM bands ORDER BY name ASC;
-- which can be shortened to
SELECT * FROM bands ORDER BY name; -- as ASC is the default
```

Here we used the `LIMIT`, `AS`, and `ORDER BY (ASC/DESC)` commands.

Say we had the table with the following data:

```
-- DB: school_table
--
--     students
--     +----+----------------------+
--     | id | name                 |
--     +----+----------------------+
--     | 1  | Joe                  |
--     | 2  | Joe                  |
--     | 3  | Joe                  |
--     | 4  | Alvin                |
--     +----+----------------------+

-- Retrieve all unique names from students
USE school_table;
SELECT DISTINCT name FROM students;

-- Query Result:
--     +----------------------+
--     | name                 |
--     +----------------------+
--     | Joe                  |
--     | Alvin                |
--     +----------------------+
```

UPDATE

To visit our `concise_records` example again:

```
-- DB: concise_records
--
--     bands
--     +----+----------------------+
--     | id | name                 |
--     +----+----------------------+
--     | 1  | The Beatles          |
--     | 2  | The Rolling Stones   |
--     | 3  | The Who              |
--     +----+----------------------+
--
--     albums
--     +----+----------------+--------------+---------+
--     | id | name           | release_date | band_id |
--     +----+----------------+--------------+---------+
--     | 1  | Abbey Road     | 1969         | 1       |
--     | 2  | Let It Be      | 1970         | 1       |
--     | 3  | Who's Next     |              | 3       |
--     +----+----------------+--------------+---------+
```

We will run the `UPDATE` command to change the `release_date`.

To Change the release date of `Who's Next` to 1971:

```sql
-- We could do
UPDATE albums
SET release_date = '1971'

-- But that would result in all albums having the same release date
-- instead we use WHERE
UPDATE albums SET release_date = '1971' WHERE name = 'Who''s Next';

-- DB: concise_records
--
--     albums
--     +----+----------------+--------------+---------+
--     | id | name           | release_date | band_id |
--     +----+----------------+--------------+---------+
--     | 1  | Abbey Road     | 1969         | 1       |
--     | 2  | Let It Be      | 1970         | 1       |
--     | 3  | Who's Next     | 1971         | 3       |
--     +----+----------------+--------------+---------+
```