Contents

1 Setup

1.1 Docker image

```
# Build image
docker run --name mysql_server -p 3306:3306 -e MYSQL_ROOT_PASSWORD=secret
    -d mysql -v ~/Programming/Sandbox/SQLSanbox/image
# Start image
docker start mysql_server
# Check status
docker ps
# Access
docker exec -it mysql_server mysql -p
# Stop image
docker stop mysql_server
```

1.2 Connecting with SQLTools

MySQL8 requires making a user (not just connecting with root).

```
-- Access through terminal £\to£
CREATE USER 'sqluser'%'' IDENTIFIED WITH mysql_native_password BY 'password';
GRANT ALL PRIVILEGES ON *.* TO 'sqluser'%'';
FLUSH PRIVILEGES;
```

Now it should be simple to connect with SQLTools.

```
{
  "mysqlOptions": {
    "authProtocol": "default"
  },
  "previewLimit": 50,
  "server": "localhost",
  "port": 3306,
  "driver": "MySQL",
  "name": "localhost",
  "username": "sqluser",
  "password": "password",
  "database": "Testing"
}
```

2 Commands

Commands are all-caps by convention. Semicolon is required to close lines.

The following is example commands for an airbnb database.

2.1 Creating a Table

```
-- is a comment; use @block to create a ipynb-like cell in SQLTools £\to£
-- @block

-- Create database
CREATE DATABASE Airbnb;
-- Check that it exists
SHOW DATABASES;

-- Must give columns dtypes
CREATE TABLE Users(
```

```
    -- PRIMARY KEY makes id the index of the table
    -- AUTO_INCREMENT automatically increments the id for us
    id INT PRIMARY KEY AUTO_INCREMENT,

    -- 255 (convention): max keys with 8bit number
    -- NOT NULL makes the field required
    -- UNIQUE requires that each email is unique
    email VARCHAR(255) NOT NULL UNIQUE,

    -- TEXT can store unspecified string lenght
    bio TEXT,
    -- Country code is fixed at 2 chars:
    --  CHAR enforces len == 2 exactly
    country CHAR(2)
);
```

2.2 Inserting values (INSERT INTO)

```
-- Inserting values
INSERT INTO Users(email, bio, country)
VALUES (
    -- Don't need ID as we have AUTO_INCREMENT
    'hello@world.com',
    'Biography text!',
    'US'
);

-- Insert multiple rows
INSERT INTO Users(email, bio, country)
VALUES
    ('hola@munda.com', 'foo', 'MX'),
    ('bonjour@monde.com', 'bar', 'FR');
```

2.3 Searching (SELECT, GROUPBY)

```
-- Show entire table
SELECT * FROM Users;

-- Select given columns
SELECT email, id FROM Users
-- Order by id in ascending order
ORDER BY id ASC
-- Print only 2 rows
LIMIT 2;

-- Including conditional statements
SELECT email, id, country FROM Users
WHERE country = 'US'
-- Pattern: all emails that start with h
OR email LIKE 'h%';

-- GROUPBY after a conditional query
-- COUNT number of users & GROUP count BY country
SELECT COUNT(id), country FROM Users
GROUP BY country;
```

2.4 Indexing

SELECT with pattern queries can be slow with large databases. Indexing allows access to important keywords without scanning the entire database, but requires additional memory & makes writing slower.

```
CREATE INDEX email_index ON Users(email);
-- Now pattern queries on email should be faster.
```

2.5 Counting (COUNT())

```
-- Count number of ids in Users
SELECT COUNT(id) FROM Users;

-- Can alias the output column name
SELECT COUNT(id) AS NumberOfUsers in Users;
```

2.6 Updating entries (UPDATE SET)

```
UPDATE Users
-- Which fields to update
SET country = 'UK'
-- Conditions by which to select rows
WHERE id = 3;
```

2.7 Relational Data (FOREIGN KEY)

```
CREATE TABLE Rooms(
    id INT AUTO_INCREMENT,
    street VARCHAR(255),
    owner_id INT NOT NULL,
    -- Alternative way to state id is the index
    PRIMARY KEY (id),
    -- References key in Users table
    -- FOREIGN KEY tells database not to delete related data
    FOREIGN KEY (owner_id) REFERENCES Users(id)
);

INSERT INTO Rooms (owner_id, street)
VALUES
    (1, 'san diego sailboat'),
    (1, 'nantucket cottage'),
    (1, 'vail cabin'),
    (1, 'sf cardboard box'),
    (2, '4sqft apartment');
```

2.8 Relational queries (JOIN)

Reading data from 2 tables.

```
-- Return all rooms' owner_ids associated with ids in Users
SELECT * FROM Users
INNER JOIN Rooms
ON Rooms.owner_id = Users.id;

-- Left join will give ALL Users ids, even those w/o a Room
```

```sql
--  and inserts NULL in unavailable Rooms fields
SELECT * FROM Users
LEFT JOIN Rooms
ON Rooms.owner_id = Users.id;

-- Right join == inner join since all Rooms must have a User id

-- MySQL does not support OUTER JOIN, but others do!

-- Foreign columns are automatically renamed e.g. Users.id
-- Overrride column names by aliasing with AS
SELECT
    -- Output table aliases
    Users.id AS user_id,
    Rooms.id AS room_id,
    email,
    street
FROM Users
INNER JOIN Rooms on Rooms.owner_id = Users.id;
```

2.9 Join with intermediate (third) table

```sql
CREATE TABLE Bookings(
    id INT AUTO_INCREMENT,
    guest_id INT NOT NULL,
    room_id INT NOT NULL,
    check_in DATETIME,
    PRIMARY KEY (id),
    -- Reference the two other tables
    FOREIGN KEY(guest_id) REFERENCES Users(id),
    FOREIGN KEY(room_id) REFERENCES Rooms(id)
);

-- Dummy data
INSERT INTO Bookings(guest_id, room_id, check_in)
VALUES
    -- NOW() gets current datetime
    (3,1, NOW()),
```

```
    -- Quotations to specify a datetime
    (2,1, '2022-12-10 15:06:25')
    -- Timestamp not needed
    (2,2, '2022-10-15');

-- Select booking by a particular individual
SELECT guest_id, street, check_in
FROM Bookings
-- Rooms which are booked by a guest
INNER JOIN Rooms ON Rooms.owner_id = guest_id
-- All rooms which guest 2 has booked
WHERE guest_id = 2;

-- History of all guests who have stayed in a room
SELECT room_id, guest_id, email, bio
FROM Bookings
-- All users with a guest_id
INNER JOIN Users ON Users.id = guest_id
-- All guests which have booked room 1
WHERE room_id = 1;
```

2.10 Dropping (`DROP TABLE;`)

Delete individual data, table or database.

```
-- Bookings must be dropped first as the others are referenced by it
DROP TABLE Bookings;
-- Rooms must be dropped second as it references by Users
DROP TABLE Rooms;
DROP TABLE Users;
DROP DATABASE Airbnb;
```