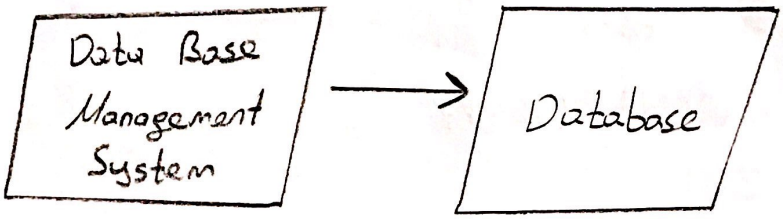
The Ultimate MySQL Bootcamp

MS 08.07.2021

- SELECT * FROM customers;
  " " " orders;
- SELECT * FROM products ORDER BY Price DESC;

Section 2 – Overview and Installation:

- Database = Collection of data, "Giant book"
-



PostgreSQL
Oracle Database

MySQL
SQLite

Section 3 – Creating Databases and Tables

- "mysql-ctl cli"
- "show databases;"
- "CREATE DATABASE name;" //name = hello_world_db
- "DROP DATABASE name;"
  ↳ "Delete"
- "USE database_name;"
- current use database → "SELECT database();"
- Tables hold the data
- Table → columns (headers) & Rows (the actual data)

- Data Types

  ↳ INT , CHAR , VARCHAR

  ⇒ Username with max 15 chars , varchar(15)

- CREATE TABLE tablename
  (
  column_name data_type,
  column_name data_type
  );

Example:
```
CREATE TABLE cats
(
    name VARCHAR(100),
    age  INT
);
```

- "SHOW TABLES;"
- "SHOW COLUMNS FROM <table-name>;"
- ↳ "DESC <table-name>;"
  ↳ Describe
- "DROP TABLE <table-name>;"

Section 4 - Insert Data:

- INSERT INTO cats(name, age)
  VALUES ("Jetson", 7); ⇒ order matters!

- SELECT * <table-name>;

- INSERT INTO cats (name, age)
  VALUES ('Charlie', 10),
  ('Sadie', 3),
  ('Lazy Bear', 1);

MS 12.07.2021

- "SHOW WARNINGS;"

- INSERT INTO cats()
  VALUES ();

↳ CREATE TABLE cats2
(
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL
);

↳ CREATE TABLE cats3
(
    name VARCHAR(100) DEFAULT 'no name provided'
    age INT DEFAULT 99
);

↳ " " cats4
(
    " " NOT NULL DEFAULT 'no name'
    " " NOT NULL DEFAULT 99

- Primary key, ID

AUTO_INCREMENT

↳ CREATE TABLE unique_cats (cat_id INT NOT NULL ,
    name VARCHAR(100),
    age INT,
    PRIMARY KEY (cat_id));

Section 5 - CRUD Commands:

- Create, Read, Update, Delete
- Read: "SELECT * FROM cats;" / "SELECT <column-name> FROM cats;"

↳ "SELECT name, age FROM cats;"

- SELECT * FROM cats WHERE age = 4;
  or
  name = 'Egg';
  ↳ case is not matter!

- SELECT cat_id AS id, name FROM cats;

id	name
:	:

- UPDATE cats SET breed = 'Shorthair' WHERE breed = 'Tabby';
- "DELETE FROM cats WHERE name = 'Egg';"

MS 25.07.2021

Section 7 - The World of String Functions:

- New file ⇒ first_file.sql
- source file_name.sql
- source folder_name/file_name.sql
- CONCAT(column , anotherColumn); ⇒ connector
  - ↳ CONCAT(first_name, ' ' , last_name);
  - ↳ SELECT
    CONCAT (author_Fname, ' ' , author_lname)
    FROM books;
- CONCAT_WS ('-', title , name);
  ↳ Seperator
- SELECT SUBSTRING ('Hello World', 1, 4);

  1 2 3 4 5 6 7 8 9 10 11
  Hello World
  ↳ Hell
  - ↳ SELECT SUBSTRING ('Hello World', -3)

    Hello World
    ↳ rld
  - ↳ " " " , 7)

    Hello World
    ↳ World

- SELECT SUBSTRING (TEXT , 1, 10)

- SELECT
    CONCAT
    (
        SUBSTRING(title, 1, 10), '...'
) AS 'short title'
  FROM books;

- SELECT REPLACE('Hello World', 'Hell', 'Heaven');
  ↳ Case sensitive
- SELECT REVERSE ('Hello World');
- SELECT CHAR_LENGTH ('Hello World');
  ↳ 11
- SELECT UPPER('Hello World'); ⇒ One parameter
  ↳ HELLO WORLD
  ↳ LOWER()

Section 8 - Refining Our Selections:

- SELECT DISTINCT author_lname FROM books;
  ↳ Every name will be mentioned just once!

- SELECT author_lname FROM books ORDER BY author_lname;
  ↳ ascending by default
  ↳ " " " " DESC; ⇒ descending

↳ SELECT ① title, ② author_fname, ③ author_lname
FROM books ORDER BY 2;

↳ " " " ORDER BY author_lname, author_fname
FIRST → SECOND →

- SELECT title FROM books LIMIT 10;
  ↳ First 10 elements (rows)

- WHERE author_fname LIKE '%da%';
  ↳ contains "da"
  ↳ 'da%' → starts with da, not case sensitive
  ↳ '%the' → Ends with the, " " "

- WHERE title LIKE '%_%'

- WHERE stock_quantity LIKE '____'

↳ 4 character

SECTION 9 - Aggregate Functions:

- Counting rows in the database:

  SELECT COUNT(*) FROM books;

↳ Counting distinct names:

SELECT COUNT(DISTINCT author_fname) FROM books;

(DISTINCT author_fname, author_lname)

- GROUP BY creates mega rows

  SELECT released_year, COUNT(*) FROM books GROUP BY ;

released_year

- SELECT Min(released_year) FROM books;

↳ Also "Max"

, title

- SELECT * FROM books
  WHERE pages = (SELECT Min(pages)
  FROM books);

↳ not a good solution

→ SELECT title, pages FROM books ORDER BY pages ASC LIMIT 1;

↳ Better solution

- SELECT Sum(pages) FROM books;

Sum of all pages

- SELECT author_fname, Sum(released_year) FROM books GROUP BY author_lname, author_fname

author_lname

- SELECT AVG(released_year) FROM books;

[MS 26.07.2021]

Section 10 - Data Types:

- CHAR ⇒ has fixed length

- DECIMAL(5, 2) — 5: Total number of digits; 2: Digits after decimal

  ←5→
  999.99
  ←2→

- FLOAT and DOUBLE, not specific

- DATETIME: 'YYYY-MM-DD HH:MM:SS' format
  (YYYY-MM-DD: DATE; HH:MM:SS: TIME)

- CURDATE(), CURTIME(), NOW() → current datetime

- INSERT INTO people (name, birthdate, birthtime, birthdt)
  VALUES ('Microwave', CURDATE(), CURTIME(), NOW());

- SELECT name, birthdate, DAYNAME(birthdate) FROM people;
  ↳ There are others

- SELECT DATE_FORMAT(birthdt, '%m/%d/%Y') FROM people
  ↳ at %h:%m

- SELECT DATEDIFF(NOW(), birthdate) FROM people

- CREATE TABLE comments (
    content VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW()
    ↳ ON UPDATE CURRENT_TIMESTAMP
);

Section 11 - Logical Operators:

- SELECT title, released_year FROM books WHERE released_year = 2017;
    ↳ released_year != 2017
- SELECT title FROM books WHERE title NOT LIKE 'W%';
    ↳ Do not start with "W" will be selected.
- SELECT * FROM books
    WHERE released_year > 2000;
    ↳ released_year >= 2000
- SELECT 98 > 1;
    Output = 1 (true)

- SELECT 'a' = 'A'; ⇒ Output: 1 ⇒ Not case sensitive

- <=, <

- AND ⇒ &&

- SELECT * FROM books WHERE author_lname = 'Eggers' AND released_year > 2010;

- OR ⇒ ||

- BETWEEN

  ↳ SELECT * FROM books
  WHERE released_year (NOT) BETWEEN 2004 AND 2005;

- IN

  ↳ SELECT title, author_lname FROM books
  WHERE author_lname IN ('Carver', 'Lahiri', 'Smith');

  ↳ Reduces the usage of or.

- Modulo ⇒ released_year % 2 != 0;

- SELECT title, released_year,
  CASE
  WHEN released_year >= 2000 THEN 'Modern Lit'
  ELSE '20th Century Lit'
  END AS GENRE
  FROM books;

Section 12 - One to Many:

- Creating the customers and orders tables

```
CREATE TABLE customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE
    amount DECIMAL(8, 2),
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

- Inserting some customers and orders

```
INSERT INTO customers (first_name, last_name, email)
VALUES ('A', 'B', 'C'),
       ('D', 'E', 'F');

INSERT INTO orders (order_date, amount, customer_id)
VALUES (' ', ' ', ' '),
       (' ', ' ', ' ');
```

[MS 27.07.2021]

- Cross join:

```
SELECT * FROM orders WHERE customer_id =
    (
        SELECT id FROM customers
        WHERE last_name = 'George'
    );
```

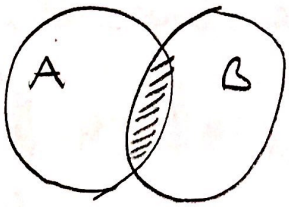
- Inner Join:

```
SELECT * FROM customers, orders WHERE
  customers.id = orders.customer_id;
```

} Implicit inner join

```
SELECT * FROM customers
  JOIN orders
    ON customers.id = orders.customer_id;
```

} Explicit inner join



*

- Finding biggest money spender

```
SELECT
    first_name,
    last_name,
    order_date,
    SUM(amount) AS total_spent
FROM customers
JOIN orders
    ON customers.id = orders.customer_id
GROUP BY orders.customer_id;
ORDER BY total_spent DESC;
```

- Left join ⇒

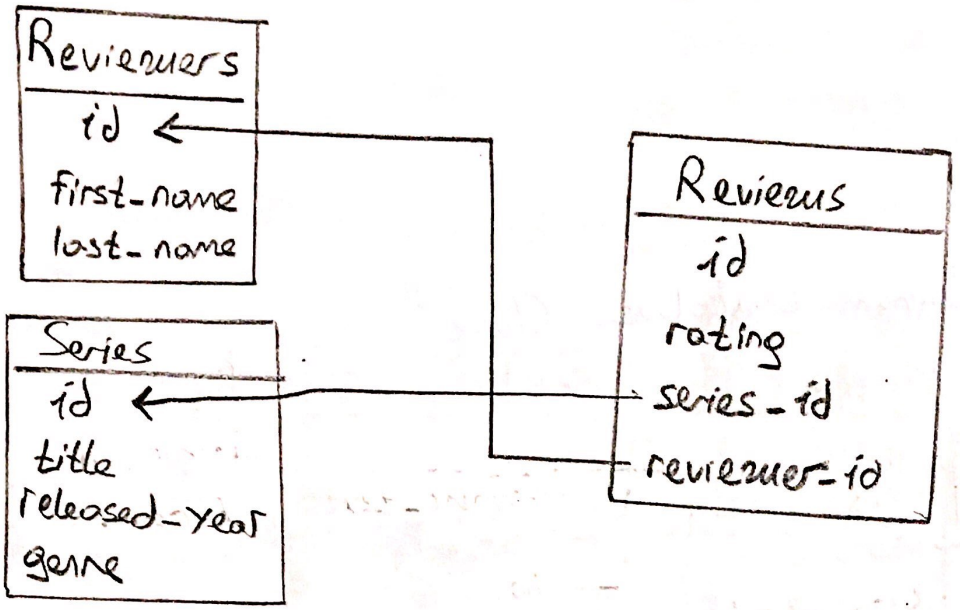

```
SELECT * FROM customers
  LEFT JOIN
      ON customers.id = orders.customer_id;
```

People that do not have orders will be displayed as NULL's.

will be replaced

```
IFNULL (SUM(amount), 0) AS total_spent
```

Section 13 - Many to Many:



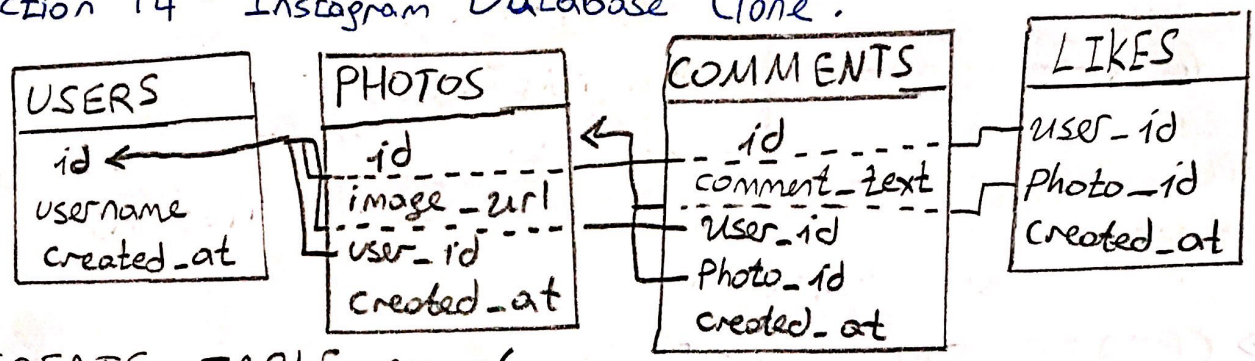
→
```
CREATE TABLE reviewers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100)
);

CREATE TABLE series(
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    released_year YEAR(4),
    genre VARCHAR(100)
);

CREATE TABLE reviews(
    id INT AUTO_INCREMENT PRIMARY KEY,
    rating DECIMAL(2,1),
    series_id INT,
    reviewer_id INT,
    FOREIGN KEY(series_id) REFERENCES series(id),
    FOREIGN KEY(reviewer_id) REFERENCES reviewers(id)
);
```

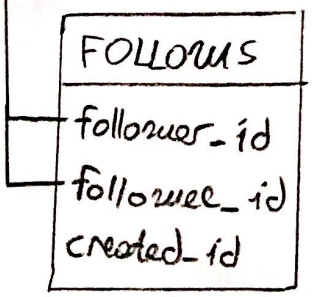
- SELECT * FROM series
  JOIN reviews
    ON series.id = reviews.series_id;

Section 14 - Instagram Database Clone:




```
CREATE TABLE users(
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE photos(
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    image_url VARCHAR(255) NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id)
);
```



```
CREATE TABLE comments (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    comment_text VARCHAR(255) NOT NULL,
    user_id INTEGER NOT NULL,
    photo_id INTEGER NOT NULL,
    created_id TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (photo_id) REFERENCES photos(id)
);

CREATE TABLE likes(
    user_id INTEGER NOT NULL,
    photo_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (photo_id) REFERENCES photos(id),
    PRIMARY KEY (user_id, photo_id)
);

CREATE TABLE follows(
        follower_id INTEGER NOT NULL,
        followee_id INTEGER NOT NULL,
        created_at TIMESTAMP DEFAULT NOW(),
        FOREIGN KEY (follower_id) REFERENCES users(id),
        FOREIGN KEY (followee_id) REFERENCES users(id),
        PRIMARY KEY (follower_id, followee_id)
);
```

```
CREATE TABLE tags(
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    tag_name VARCHAR(255) UNIQUE,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE photo_tags (
    photo_id INTEGER NOT NULL,
    tag_id INTEGER NOT NULL,
    FOREIGN KEY (photo_id) REFERENCES photos(id),
    FOREIGN KEY (tag_id) REFERENCES tags(id),
    PRIMARY KEY(photo_id, tag_id)
);
```

Section 15 - Lots of Instagram Data