# DATABASES

data type: Is a formatting that strictly enforced in that particular column.

The **main formats** are:

1. strings
2. numeric
3. datetime types

### 1. String Types

A sequence of characters. In MySQL there are 2 common types of strings: varchar and text. **Varchar** columns contain short strings of characters, these can vary in length, and they're ideal for things like product names and movie titles. In the other hand **text** columns are for longer strings like product or movie descriptions.

### 2. Numeric Types

There are **integers**, which are whole numbers.
**Fixed Point** numbers are for example decimals. Good for store money.
**Float Point** an example type is float, which store numbers in a no exact way, where the decimal place is not fixed to a particular precision. The problem with this is when you are recalling a number and you want it exactly a you stored it, for example, you may store a number as 2.3 but the database did it as 2.29999.

### 3. Day & Time Types

Store temporal data or data relating to time. Some may content just the date and some just the time, or a component one called datetime which will store date and time.

## SQL statement to create a table

CREATE TABLE <<nameOfTheTable>> (<<name(columnname)>> VARCHAR(50));
To create a table with **two (2)** columns you must type:
CREATE TABLE movies (title VARCHAR(200), year INTEGER);

### Insert data

INSERT INTO movies VALUES ("Avatar", 2009);

- SELECT * FROM movies ORDER BY year ASC, title DESC;

- SELECT * FROM movies LIMIT 10;

- SELECT * FROM movies LIMIT 10 OFFSET 1;
- SELECT * FROM movies WHERE year IS NULL;
-
- SELECT * FROM movies WHERE year IS NOT NULL ORDER BY year;

**DDL** -> Stands for data definition language, deals with special keywords that create and modify tables and create and modify databases. Deals with schema.
**DML** -> Stands for data manipulation language, and deals with the CRUD.

- CREATE TABLE movies (title VARCHAR(200) NOT NULL, year INTEGER ) ENGINE InnoDB;
- SHOW ENGINES
- INSERT INTO movies VALUES ("Avatar", 2009);
- INSERT INTO movies (year, title) VALUES (2009, "Avatar"), (NULL, "Avatar 2");
- INSERT INTO movies SET title="Back to the Future", year = 1995;
- UPDATE movies SET year=2015 WHERE title = "Avatar 2";
- **SET SQL_SAfE_UPDATES = 0;**
- UPDATE movies SET year=2016, title="Avatar Reloaded" WHERE title = "Avatar 2";
- DELETE FROM movies WHERE title = "Avatar Reloaded" AND year = 2016;

RENAME TABLE movies TO movie_table;
RENAME TABLE actors TO actor_table;

———————-
RENAME TABLE movies TO movie_table, actors TO actor_table;

DROP TABLE IF EXISTS reviews;

TRUCANTE TABLE movie_table; //deletes the table

ALTER TABLE movie_table ADD COLUMN genre VARCHAR(100);

ALTER TABLE actor_table ADD (pob VARCHAR(100), dob DATE);

ALTER TABLE actor_table CHANGE COLUMN pob place_of_birth VARCHAR(100), dob DATE);

ALTER TABLE actor_table DROP date_of_birth;  //DELETE a column!

## DELETE THE DATABASE

DROP DATABASE movie_db_3;

DROP SCHEMA IF EXISTS movie_db_3;

**Normalization:** Describes the process of setting up a table that contains repeated and redundant data from one column of a table and putting that information into another table

## Keys

1. **Primary Keys:** For example an ID. They are used to uniquely define each row in a table.
   - Movies can be named same. We need something unique, like a number.
   - Can't be null.
   - Can't be duplicated.

2. **Unique Keys:** **For example,** an email address (email_address) or a social security number (ssn).
   - Enforce uniqueness
   - Can be null, unless you specify otherwise.
   - Can't be duplicated.

3. **Foreign Keys:** **For example,** a genre_id.
   - Special keys that describe the relationship between data in two tables. Foreign keys also known as reference keys.
   - They can be null
   - They can be duplicated.

CREATE TABLE genres (id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY, name VARCHAR(200) NOT NULL UNIQUE KEY);

**INSERT INTO genres (name) VALUES ("Sci Fi");**

**ALTER TABLE movies ADD COLUMN id INTEGER AUTO_INCREMENT PRIMARY KEY FIRST;**

**ALTER TABLE movies ADD COLUMN genre_id INTEGER NULL, ADD CONSTRAINT FOREIGN KEY (genre_id) REFERENCES genre(id);**

**UPDATE movies SET genre_id = 1 where id = 8 or id=9;**

**SELECT * FROM movies INNER JOIN genres ON movies.genre_id = genres.id;**

**SELECT * FROM movies LEFT OUTER JOIN genres ON movies.genre_id = genre.id;**

**A left** outer join contain all the data from the inner join, and the data from the table on the left, **and a right** outer join shows all the information from the in if join, plus all the data on the right.

```
SELECT movies.title, genres.name AS genre_name
FROM movies LEFT OUTER JOIN genres
ON movies.genre_id = genre.id
WHERE name IS NOT null;
```

```
SELECT movies.title, genres.name AS genre_name
FROM movies LEFT OUTER JOIN genres
ON movies.genre_id = genre.id
WHERE genres.id = 1;
```

```
SELECT COUNT(*) AS review_count FROM reviews WHERE movie_id = 1;
```

```
SELECT MIN(score) AS minimum_score, MAX(score) AS maximun_score FROM reviews WHERE movie_id = 1;
```

```
SELECT MIN(score) AS minimum_score, MAX(score) AS maximun_score, SUM(score) / COUNT(score) FROM reviews WHERE movie_id = 1;
```

```
SELECT MIN(score) AS minimum_score, MAX(score) AS maximun_score,
AVG(score) AS average
FROM reviews;
```

```
SELECT movie_id, MIN(score) AS minimum_score,
MAX(score) AS maximun_score,
AVG(score) AS average
FROM reviews GROUP BY movie_id;
```

```
SELECT title, MIN(score) AS minimum_score,
MAX(score) AS maximun_score,
IFNULL(AVG(score) ,0) AS average
FROM movies LEFT OUTER JOIN reviews
ON movies.id = review.movie_id
GROUP BY movie_id;

SELECT title, MIN(score) AS minimum_score,
MAX(score) AS maximun_score,
IFNULL(AVG(score) ,0) AS average
FROM movies LEFT OUTER JOIN reviews
ON movies.id = review.movie_id
GROUP BY movie_id HAVING average > 3;

SELECT title, MIN(score) AS minimum_score,
MAX(score) AS maximun_score,
IFNULL(AVG(score) ,0) AS average
FROM movies LEFT OUTER JOIN reviews
ON movies.id = review.movie_id
WHERE year_released > 2000
GROUP BY movie_id HAVING average > 3;


SELECT movies.title, IFNULL(AVG(score) ,0) AS average
FROM reviews RIGHT OUTER JOIN movies
ON movies.id = reviews.movie_id
GROUP BY movie_id HAVING average < 2;

SELECT first_name, UPPER(last_name), LOWER(email), LENGTH(username) AS
username_length FROM users;

SELECT first_name,
UPPER(last_name), LOWER(email), LENGTH(username) AS username_length
FROM users HAVING username_length < 19;

SELECT CONCAT(first_name, " ", UPPER(last_name)),
CONCAT(SUBSTRING(LOWER(email), 1, 10), "…") AS partial_email
LENGTH(username) AS username_length
FROM users HAVING username_length < 19;
```

## DATABASE MAINTAINING

```
EXPLAIN SELECT * FROM users WHERE last_name=“chalkley”;

CREATE INDEX last_name_idx On users(last_name);
```

## PERMISSIONS FOR USERS

```
#user1 - read access;

GRANT SELECT
ON treehouse_movie_db.*
TO user1@‘%’
IDENTIFIED BY ‘password’;

FLUSH PRIVILEGES;

INSERT INTO movies (title, genre_id) VALUES (“Hairspray”, 3);

#user2 - read - write access;

GRANT SELECT, INSERT, UPDATE, DELETE
ON treehouse_movie_db.*
TO user2@‘%’
IDENTIFIED BY ‘password’;

FLUSH PRIVILEGES;

UPDATE movies SET year_released = 2002 WHERE id = 9;

#user3 - DDL

GRANT ALTER, CREATE, DROP
ON treehouse_movie_db.*
TO user3@‘%’
IDENTIFIED BY ‘password’;

FLUSH PRIVILEGES;
```