SQL Database Management: Table Operations, Constraints, and Schema Modifications - p1 - Lecture - 4

SQL Database Management: Table Operations, Constraints, and Schema Modifications - p1

```
SELECT column, another_column, …
FROM mytable;
```

Exercise

We will be using a database with data about some of Pixar's classic movies for most of our exercises. This first exercise will only involve the **Movies** table, and the default query below currently shows all the properties of each movie. To continue onto the next lesson, alter the query to find the exact information we need for each task.

Table: Movies

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115
9	WALL-E	Andrew Stanton	2008	104
10	Up	Pete Docter	2009	101

Exercise 1 -
1. Find the
2. Find the
3. Find the
4. Find the
5. Find all

Exercise 1 — Tasks

1. Find the **title** of each film `SELECT title FROM movies;`
2. Find the **director** of each film `SELECT Director FROM movies;`
3. Find the **title** and **director** of each film `SELECT title, director FROM movies;`
4. Find the **title** and **year** of each film `SELECT title, year FROM movies;`
5. Find **all** the information about each film `SELECT * FROM movies;`

Select All ★

Problem | Submissions | Leaderboard | Discussions | Editorial

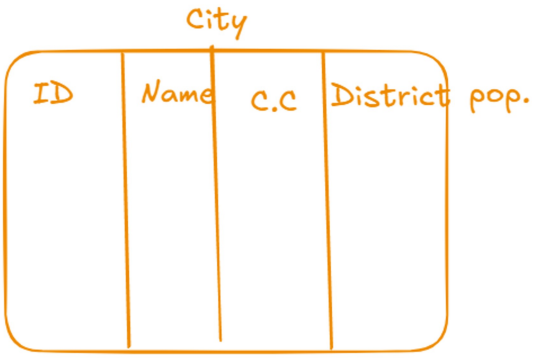
Query all columns (attributes) for every row in the **CITY** table.

The **CITY** table is described as follows:

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

```
1  SELECT * FROM City;
```



Your Output (stdout)

```
6 Rotterdam NLD Zuid-Holland 593321
3878 Scottsdale USA Arizona 202705
3965 Corona USA California 124966
3973 Concord USA California 121780
3977 Cedar Rapids USA Iowa 120758
3982 Coral Springs USA Florida 117549
4054 Fairfield USA California 92256
4058 Boulder USA Colorado 91238
4061 Fall River USA Massachusetts 90555
```

Expected Output

```
6 Rotterdam NLD Zuid-Holland 593321
3878 Scottsdale USA Arizona 202705
3965 Corona USA California 124966
3973 Concord USA California 121780
3977 Cedar Rapids USA Iowa 120758
3982 Coral Springs USA Florida 117549
4054 Fairfield USA California 92256
4058 Boulder USA Colorado 91238
4061 Fall River USA Massachusetts 90555
```

varchar varchar float/double/decimal Boolean - T/F

ProductID	ProductName	Price	QuantityInStock	DateAdded	IsAvailable
101	Wireless Mouse	25.99	150	2023-06-01	TRUE
ABC	12345	Twenty Five	Fifty	June First, 2023	Yes
103	Keyboard	49.99	75	2023-06-03	TRUE
104	Monitor	Fifty-Nine	Thirty	2023/06/04	No
105	Mouse Pad	5.99	300	2023-06-05	TRUE
			int	date	boolean

Define the Data-type at the time of creation

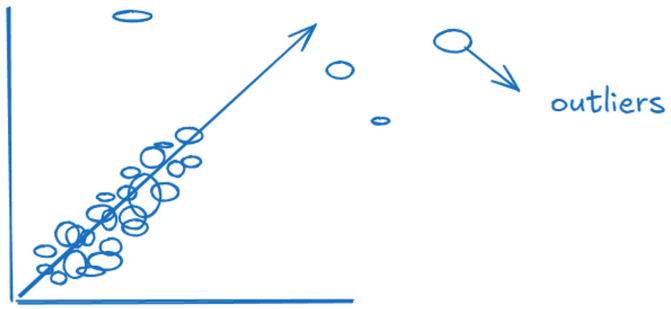
*Storing information in a tabular format is making the data more structural. Making sure right data type is aligned with the column and it follow as well *

Select * from Random_table where isAvailable = "TRUE"

INT - Numerical Value [123 , 321 , -21332, 123693,] [whole Number]
Decimal - 10digit , 2 decimal [Decimal(10,2)] - 1235.97 , 12238.99 , 3187877.12

Corrected Table with Datatypes:

Column Name	Datatype	Example Value
ProductID	INT	101
ProductName	VARCHAR(100)	"Wireless Mouse"
Price	DECIMAL(10,2)	25.99
QuantityInStock	INT	150
DateAdded	DATE	2023-06-01
IsAvailable	BOOLEAN	TRUE



[123 , 12345 , 1324, 5445 , 353, 121 , 3434, 12134, -123423]

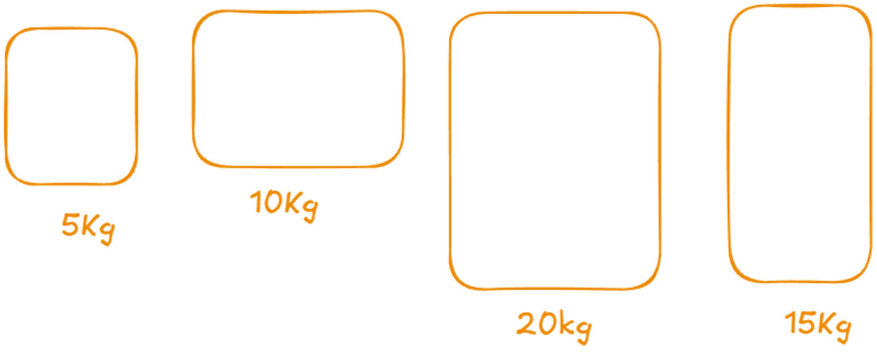
outlier [Anamolies]

INTEGER Variants

- **INT**: Can store a wide range of whole numbers. If you define a column as `INT`, it can store values like `-2147483648` to `2147483647` (for a typical 4-byte integer).
- **SMALLINT**: Designed for smaller integers. A `SMALLINT` column can hold values from `32768` to `32767`.
- **TINYINT**: Even smaller range, suitable for very small numbers or boolean values in some systems. A `TINYINT` can store values from `0` to `255` (unsigned) or `-128` to `127` (signed).
- **BIGINT**: For very large integer values. A `BIGINT` column can hold values from `-9223372036854775808` to `9223372036854775807`.

8 byte

1MB = 1024 Kb
1GB = 1024 MB



DECIMAL (or NUMERIC)

- **DECIMAL(5,2)**: This definition means the column can store numbers with up to 5 digits, with 2 of those digits after the decimal point. For example, `123.45` fits this format. If you try to insert `1234.567`, it will be rounded or truncated to fit, depending on the system, possibly resulting in `1234.57`.

Decimal Value - Price , Profit

FLOAT and DOUBLE

- **FLOAT**: A single-precision floating-point number. If you define a column as `FLOAT` and insert `123456.789`, the actual stored value might be approximately `123456.7891` due to the way floating-point arithmetic is handled.

4 bytes

- **DOUBLE**: A double-precision floating-point number offers more precision than `FLOAT`. If you insert `123456789.12345` into a `DOUBLE` column, it retains more of the original value's precision, storing it as approximately `123456789.12345`, depending on the exact limits of the system's floating-point precision.

8 bytes

Scientific value

"Double is more accurate in terms of storing important data but costly in terms of space."
"Scientific Value , Business Deal - GDP , Value need to be important."

2. Character and String Data Types

These data types store text and are used for names, descriptions, and any other alphanumeric data.

- **CHAR** (Character): Stores fixed-length strings (e.g., `CHAR(5)`). CHAR(5) can store up to 5 characters. If you insert 'Hi', it will be stored as 'Hi '.
- **VARCHAR** (Variable Character): Stores variable-length strings up to a specified length (e.g., `VARCHAR(255)`). This column can store strings up to 255 characters long. If you insert `'Hello World'`, it will store exactly `'Hello World'` without additional padding, unlike `CHAR`.
- **TEXT**: Stores long text data. It's suitable for large amounts of text, like articles or comments. The exact size limit varies by SQL database system. Capable of storing long text data, such as full articles. The exact limit can vary, but it is generally much larger than `VARCHAR`, often up to 65,535 characters (for `TEXT` in MySQL, for example). If you insert a paragraph or an entire article, it will store the entire text as is, subject to the maximum length limit.

"Varchar is suggested if you don't know the size of the input."
"Char is faster in fixed size". [Space Allocation] [Wastage of space]

3. Date and Time Data Types

Important

Date and time data types represent dates, times, or both. They are crucial for recording events, scheduling, and temporal analysis.

- **DATE**: Stores calendar dates (year, month, and day). If you define a column as `DATE` and insert `2023-03-15`, it stores precisely that date with no time component.
- **TIME**: Stores time of day. Defining a column as `TIME` and inserting `14:30:00` means it stores the time as exactly `14:30:00`.
- **DATETIME** and **TIMESTAMP**: Store both date and time. The difference between them often involves time zone handling and precision. A `DATETIME` column with an inserted value of `2023-03-15 14:30:00` stores both the date and time exactly as `2023-03-15 14:30:00`. Similar to `DATETIME`, but in systems like MySQL, it also considers time zones. If you insert `2023-03-15 14:30:00 UTC` into a `TIMESTAMP` column, the stored value might be converted based on the time zone settings of your database server

yyyy-mm-dd

hh:mm:ss

date time

24-10-2024 21:43:21

"Timestamp is just putting the current time"

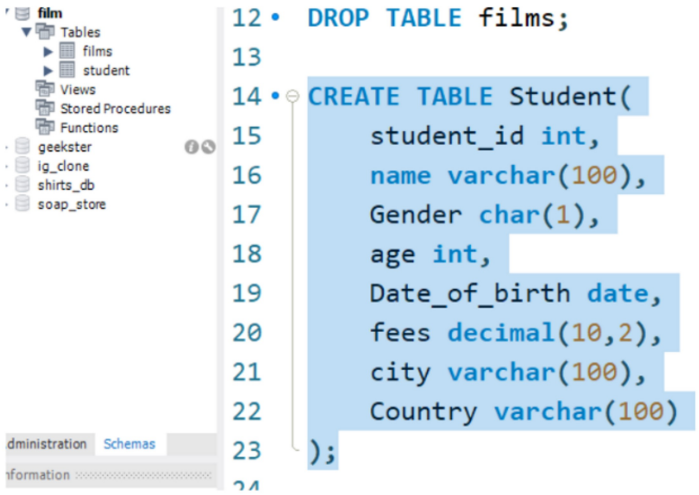
When you like some post, when you delete post.
Every crime that you do, timestamp is marked.

"Surveillance"

online website. ide.goorm.io → MySQL Workbench, Client Side [Terminal]

- Stores JSON (JavaScript Object Notation) data.
- **Column Type:** `user_profile`, `settings`, `metadata`
- Example: `{"name": "John", "age": 30}` → dictionary

Key : Value
name : John
age : 30
gender : 'M'




```
25 • DESC Student;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

Field	Type	Null	Key	Default	Extra
student_id	int	YES		NULL	
name	varchar(100)	YES		NULL	
Gender	char(1)	YES		NULL	
age	int	YES		NULL	
Date_of_birth	date	YES		NULL	
fees	decimal(10,2)	YES		NULL	
city	varchar(100)	YES		NULL	
Country	varchar(100)	YES		NULL	

```
mysql> DROP TABLE student;
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_film |
+----------------+
| films          |
+----------------+
```

```
mysql> CREATE TABLE Student(
    ->  student_id int,
    ->      name varchar(100),
    ->      Gender char(1),
    ->      age int,
    ->      Date_of_birth date,
    ->      fees decimal(10,2),
    ->      city varchar(100),
    ->      Country varchar(100)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_film |
+----------------+
| films          |
| student        |
+----------------+
2 rows in set (0.00 sec)

mysql> DESC Student;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| student_id    | int           | YES  |     | NULL    |       |
| name          | varchar(100)  | YES  |     | NULL    |       |
| Gender        | char(1)       | YES  |     | NULL    |       |
| age           | int           | YES  |     | NULL    |       |
| Date_of_birth | date          | YES  |     | NULL    |       |
| fees          | decimal(10,2) | YES  |     | NULL    |       |
| city          | varchar(100)  | YES  |     | NULL    |       |
| Country       | varchar(100)  | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```