Working with Operators, Constraints and Data Types

simplilearn

Learning Objectives

By the end of this lesson, you will be able to:

- ● Identify the different types of operators in MySQL

- ● Determine the levels of data in SQL

- ● List the different MySQL constraints

- ● Differentiate between the data types in SQL

MySQL Operators

SQL Operators

- An operator is a reserved word or character used with the WHERE clause of an SQL statement.

- It specifies a condition in the SQL statement.

SQL

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations in SQL.

Consider variables: a = 5 and b = 10

Operator	Description	Example
+	Addition of two operands	a + b = 15
-	Subtraction of two operands	a – b = -5
*	Multiplication of two operands	a * b = 50
/	Division of two operands	a / b = 0.5
%	Modulus or remainder from the division	a % b = 0

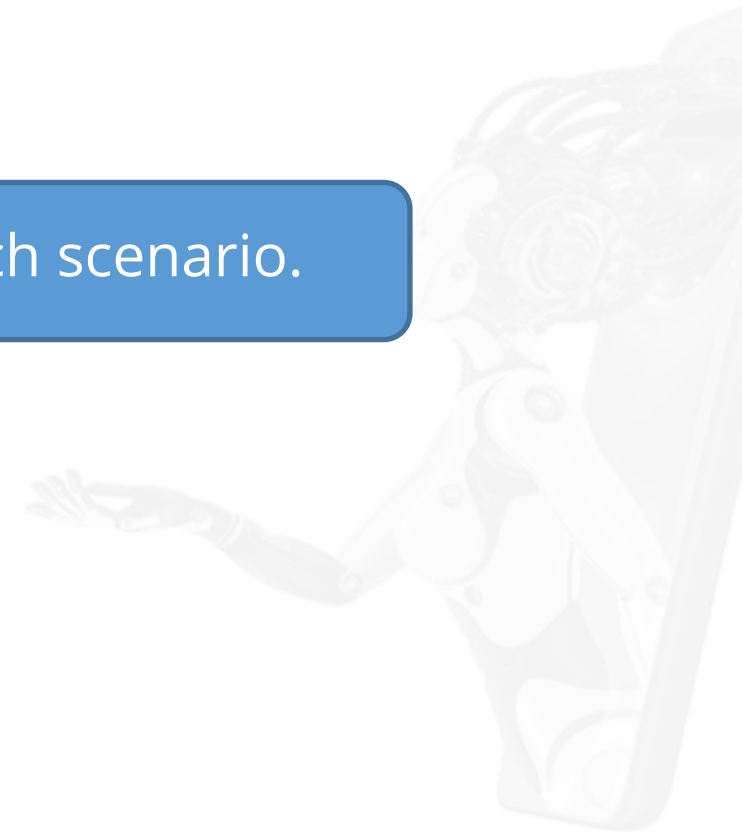# Arithmetic Operators

Syntax

SELECT column 1 + column 2 FROM table 1

SELECT column 1 * column 2 FROM table 2

Arithmetic Operators: Example

Problem Statement: You are a junior analyst in your organization. You need to help the HR team determine salaries of employees for different scenarios.

**Objective:** Use arithmetic operators to determine the new values based on each scenario.

Arithmetic Operators: Example

Consider the employee table given below, which has columns: Emp_ID, Emp_First_Name, Emp_Last_Name, Emp_Salary, and Emp_Annual_Bonus.

Emp_ID	Emp_F_Name	Emp_L_Name	Emp_Salary	Emp_Annual Bonus
1134	Mark	Jacobs	20000	1500
1256	John	Barter	25000	1000
1277	Michael	Scar	22000	1000
1300	Dan	Harris	30000	2000
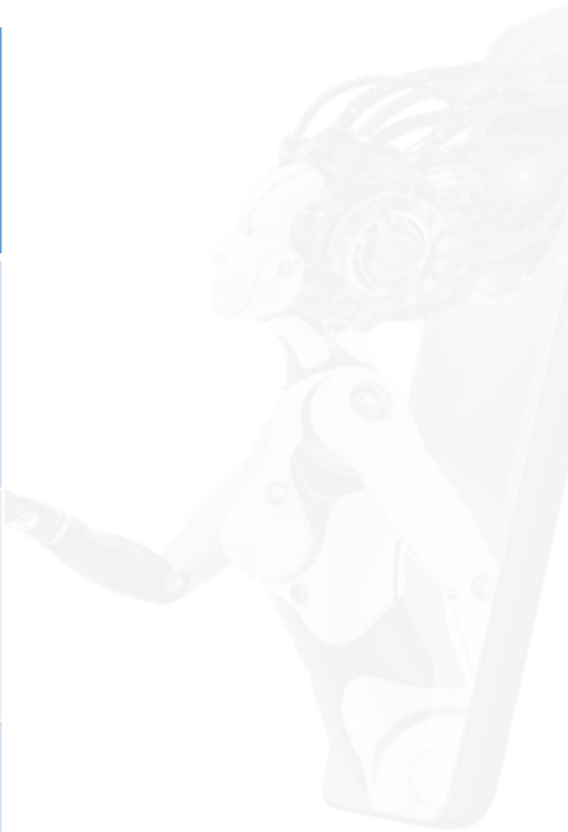
Arithmetic Operators: Example

If you want to add the salary and bonus, use the addition operator. You get the following results.

Syntax

SELECT Emp_ID, Emp_Salary +Emp_Annual_Bonus as Emp_Total_Earning FROM Employee_Records;

Emp_ID	Emp_Total_Earning
1134	21500
1256	26000
1277	23000
1300	32000

Arithmetic Operators: Example

Suppose for the same table there is another column named deductions, and you are required to deduct this amount from the final earning. Here, you should use the subtract operator.

Emp_ID	Deductions
1256	200
1300	150

Syntax

SELECT Emp_ID, Emp_Salary – Deductions as Emp_Final_Earning FROM Employee_Records;

Emp_ID	Emp_Final_Earning
1134	20000
1256	24800
1277	22000
1300	29850

Arithmetic Operators: Example

If you want to increase the salary of employees by two times, then you must use the multiplication operator to get the following results.

Syntax

SELECT Emp_Salary * 2 as New_Salary FROM Employee_Records;

Emp_ID	New_Salary
1134	40000
1256	50000
1277	44000
1300	60000

Arithmetic Operators: Example

Similarly, if you want to reduce the salary of each employee by 50%, then you can use the division operator to obtain the following results.

Syntax

```
SELECT Emp_Salary / 2 as New_Salary FROM Employee_Records;
```

	Emp_ID	New_Salary
▶	1134	10000.0000
	1256	12500.0000
	1277	11000.0000
	1300	15000.0000

Bitwise Operators

Bitwise operators perform bit manipulations between two expressions of integer data type.

Operator	Description
&	AND
|	OR
^	Exclusive OR

They take two integer values, convert them to binary bits, and then apply AND, OR, or NOT operations on each bit.

Bitwise Operators: Example

Consider the same employee table used for arithmetic operators. If you want to apply Bitwise AND on salary and annual bonus, use the syntax below.

Syntax

```
SELECT Emp_Salary & Emp_Annual_Bonus from Employee_Records;
```

	Emp_Salary & Emp_Annual_Bonus
▶	1024
	424
	480
	1296

Bitwise Operators: Example

For Bitwise OR operation, use the syntax given below.

Syntax

SELECT Emp_Salary | Emp_Annual_Bonus from Employee_Records;

	Emp_Salary | Emp_Annual_Bonus
▶	20476
	25576
	22520
	30704

simplilearn

Comparison Operators

Comparison operators compare values between operands and return TRUE or FALSE based on the condition.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Syntax

```
SELECT * FROM table1 WHERE column1 >= Condition
```

Comparison Operators: Example

The example that was shown before has been used here. If you want to find the employees whose salary is more than 2500, then you use the *greater than* operator.

Syntax

SELECT Emp_ID, Emp_Salary from Employee_Records WHERE Emp_Salary > 25000;

	Emp_ID	Emp_Salary
▶	1300	30000

Similarly, other comparison operators like less than, equal to, greater than equal to, or less than equal to can be used based on the requirement.

Compound Operators

Compound operators operate where variables are operated upon and assigned in the same line.

Compound Operators	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise exclusive OR equals

Syntax

SELECT column1+= condition FROM table1;

Logical Operators

Logical operators compare two conditions at a time to determine whether a row can be selected for an output or not.

Logical Operators	Description
AND	Add equals
OR	Subtract equals
BETWEEN	Multiply equals
NOT	Divide equals
LIKE	Modulo equals

Syntax

SELECT column1, column 2 FROM table1 WHERE logical condition;

Logical Operators: Example

Consider the same employee records table. If you want to extract data based on two conditions, that are salary and location, then you use the AND operator.

Syntax

SELECT Emp_ID, Emp_Salary, Emp_Location from Employee_Records WHERE Emp_Salary > 1000 AND Emp_Location = 'California';

Emp_ID	Emp_Salary	Emp_Location
1256	25000	California

Logical Operators: Example

If you want to extract data based on any one of the two conditions mentioned, that are salary or annual bonus, then you use OR operator.

Syntax

SELECT Emp_ID, Emp_Salary from Employee_Records WHERE Emp_Salary > 22000 OR Emp_Annual_Bonus <1000;

	Emp_ID	Emp_Salary
▶	1256	25000
	1300	30000

Logical Operators: Example

If you want to extract data by excluding a certain data record, that is, excluding employees from New York, then you use the NOT operator.

Syntax

SELECT Emp_ID, Emp_F_Name, Emp_L_Name from Employee_Records WHERE Emp_Location NOT = 'New York';

Emp_ID	Emp_F_Name	Emp_L_Name
1256	John	Barter
1277	Michael	Scar
1300	Dan	Harris

Logical Operators: Example

If you want to extract data starting with a specific character or ending with a specific character, that is employees whose name starts with M, then the LIKE operator is used.

Syntax

SELECT Emp_ID, Emp_F_Name from Employee_Records WHERE Emp_F_Name LIKE 'M%';

	Emp_ID	Emp_F_Name
▶	1134	Mark
	1277	Michael

Duration: 20 Min.

Problem Statement: You are required to use a logical operator to identify the candidates in the age group of 22 to 35 from the created table in the MySQL Workbench.

Assisted Practice: Logical Operator

Steps to be performed:

1. Create a database **example**, then make a table **candidates** that has columns **FirstName**, **LastName**, and **Age**.

> **TABLE CREATION**
>
> ```
> CREATE TABLE `example`.`candidates` (
> `FirstName` VARCHAR(255) NOT NULL,
> `LastName` VARCHAR(255) NOT NULL,
> `Age` INT NOT NULL);
> ```

Assisted Practice: Logical Operator

Steps to be performed:

2. Insert values in the table **candidates**.

VALUE INSERTION

```
INSERT INTO `example`.`candidates` (`FirstName`, `LastName`, `Age`)
VALUES ('James', 'Smith', '23'),
('Maria ', 'Gracia', '21'),
('Michael ', 'Rodriguez', '27'),
('Robert ', 'Johnson', '41'),
('David', 'Hernandez', '27');
```

ASSISTED PRACTICE

3. Write a query to select all the people in the age group of 22 to 35.

QUERY

SELECT * FROM example.candidates

WHERE Age BETWEEN 22 and 35;

Assisted Practice: Lab Output

	FirstName	LastName	Age
▶	James	Smith	23
	Michael	Rodriguez	27
	David	Hernandez	27

Result Grid | Filter Rows: | Export:

Indexing in MySQL

Indexing in MySQL

MySQL indexes sort data in a logical and sequential order.

Indexes are used to find rows with specific column values quickly.

Indexing in MySQL

There are two ways to create an index:

Before the table is created

Create Index

After the table is created
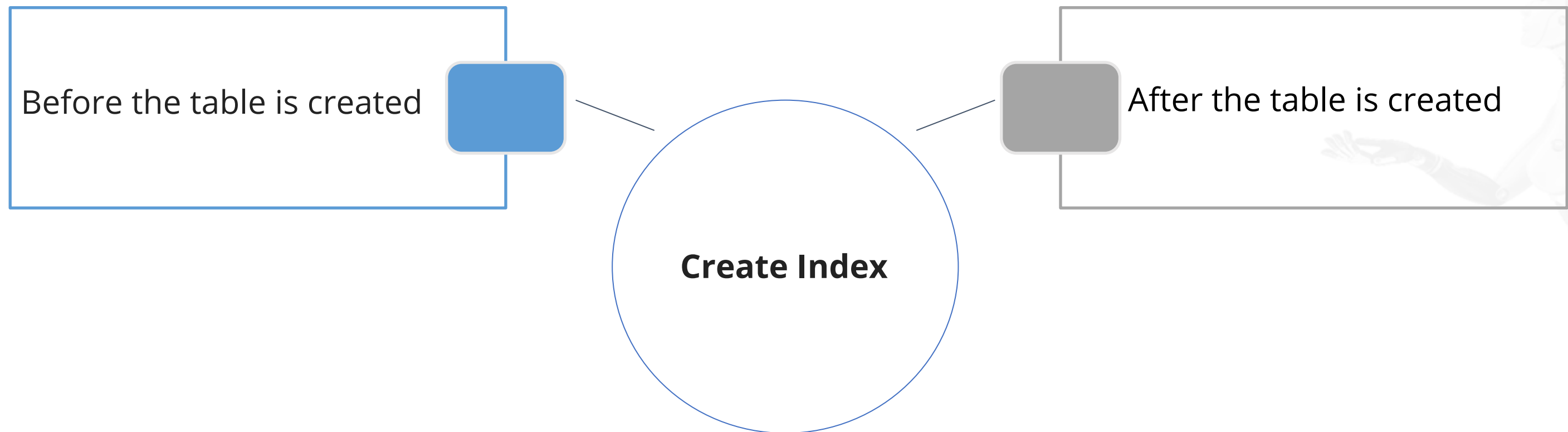
Indexing in MySQL

Before creating the table, use the following syntax:

Syntax

```
CREATE TABLE t_index
(
    col1 INT PRIMARY KEY,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 VARCHAR(20),
   INDEX (col2,col3)
);
```

Indexing in MySQL

After creating the table, use the following syntax:

Syntax

CREATE INDEX id_index ON table_name(column_name);

Indexing in MySQL

Problem statement: Consider the junior DBA wants to improve the speed and result of the query by adding an index.

Objective: Implement indexing and get the desire result.

Instructions: Refer the emp_data table which was created and shown before.

Table Description

Field Name	Description
EMP_ID	Employee ID
FIRST_NAME	First name of the employee
LAST_NAME	Last name of the employee
GENDER	Gender of the employee (M/F)
ROLE	Designation of the employee (Junior, Senior, Lead, and Associate Data Scientist)
DEPT	Name of the department (Retail, Finance, Automotive, and Healthcare)

Table Description

Field Name	Description
EXP	Experience of the employee
COUNTRY	Country where the employee lives
CONTINENT	Continent based on the country
SALARY	Salary of the employee per month
EMP_RATING	Rating for the employee (1: Not Achieved Any Goals, 2: Below Expectations, 3: Meeting Expectations, 4: Excellent Performance, 5: Overachiever
MANAGER_ID	Employee ID for the manager

Indexing in MySQL

Use the following emp_data:

```
1 •    SELECT * FROM sys.emp_data;
```

EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	EXP	COUNTRY	CONTINENT	SALARY	EMP_RATIN
E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	7	INDIA	ASIA	7000	3
E245	Nian	Zhen	M	SENIOR DATA SCIENTIST	RETAIL	6	CHINA	ASIA	6500	2
E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	2	INDIA	ASIA	3000	1
E640	Jenifer	Jhones	F	JUNIOR DATA SCIENTIST	RETAIL	1	COLOMBIA	SOUTH AMERICA	2800	4
E403	Steve	Hoffman	M	ASSOCIATE DATA SCIENTIST	FINANCE	4	USA	NORTH AMERICA	5000	3
E204	Karene	Nowak	F	SENIOR DATA SCIENTIST	AUTOMOTIVE	8	GERMANY	EUROPE	7500	5
E204	Karene	Nowak	F	SENIOR DATA SCIENTIST	AUTOMOTIVE	8	GERMANY	EUROPE	7500	5
E010	William	Butler	M	LEAD DATA SCIENTIST	AUTOMOTIVE	12	FRANCE	EUROPE	9000	2
E478	David	Smith	M	ASSOCIATE DATA SCIENTIST	RETAIL	3	COLOMBIA	SOUTH AMERICA	4000	4
E005	Eric	Hoffman	M	LEAD DATA SCIENTIST	FINANCE	11	USA	NORTH AMERICA	8500	3
E532	Claire	Brennan	F	ASSOCIATE DATA SCIENTIST	AUTOMOTIVE	3	GERMANY	EUROPE	4300	1
E583	Janet	Hale	F	MANAGER	RETAIL	14	COLOMBIA	SOUTH AMERICA	10000	2
E103	Emily	Grove	F	MANAGER	FINANCE	14	CANADA	NORTH AMERICA	10500	4
E612	Tracy	Norris	F	MANAGER	RETAIL	13	INDIA	ASIA	8500	4
E428	Pete	Allen	M	MANAGER	AUTOMOTIVE	14	GERMANY	EUROPE	11000	4
E002	Cynthia	Brooks	F	PRESIDENT	ALL	17	CANADA	NORTH AMERICA	14500	5
E002	Cynthia	Brooks	F	PRESIDENT	ALL	17	CANADA	NORTH AMERICA	14500	5

Indexing in MySQL

Execute the following statement to return the result of the employee who is a manager:

EMP_ID	FIRST_NAME	LAST_NAME
▶ E583	Janet	Hale
E103	Emily	Grove
E612	Tracy	Norris
E428	Pete	Allen

Example:

SELECT EMP_ID, FIRST_NAME, LAST_NAME
FROM sys.emp_data WHERE ROLE
'MANAGER';

Indexing in MySQL

If you want to check how MySQL performs the previous query internally, execute the following query:

Example:

EXPLAIN SELECT EMP_ID, FIRST_NAME, LAST_NAME FROM sys.emp_data WHERE
ROLE = 'MANAGER';

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

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	emp_data	NULL	ALL	NULL	NULL	NULL	NULL	18	10.00	Using where

Indexing in MySQL

Create an index for a class column using the following query:

Example:

```
CREATE INDEX indx ON sys.emp_data (role);


EXPLAIN SELECT EMP_ID, FIRST_NAME, LAST_NAME FROM  sys.emp_data WHERE
ROLE = 'MANAGER';
```

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

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	emp_data	NULL	ref	indx	indx	183	const	4	100.00	NULL

Indexing in MySQL

If you want to show the indexes of a table, execute the following query:

Example:

```
SHOW INDEXES FROM sys.emp_data;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expr
▶	emp_data	1	IX_emp_data_EMP_ID	1	EMP_ID	A	17	NULL	NULL	YES	BTREE			YES	NULL
	emp_data	1	indx	1	ROLE	A	8	NULL	NULL	YES	BTREE			YES	NULL

Result Grid | Filter Rows: | Export: | Wrap Cell Content: ‡A

Indexing in MySQL

If you want to drop the index of a table, execute the following query:

Example:

```
DROP INDEX `indx` ON `emp_data`;

SHOW INDEXES FROM sys.emp_data;
```

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

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expr
emp_data	1	IX_emp_data_EMP_ID	1	EMP_ID	A	17	NULL	NULL	YES	BTREE			YES	NULL

Order of Execution in SQL

Order of Execution in SQL

FROM
Selects the table from which the data is obtained

WHERE
Applies a condition to filter data

HAVING
Filters the aggregated data
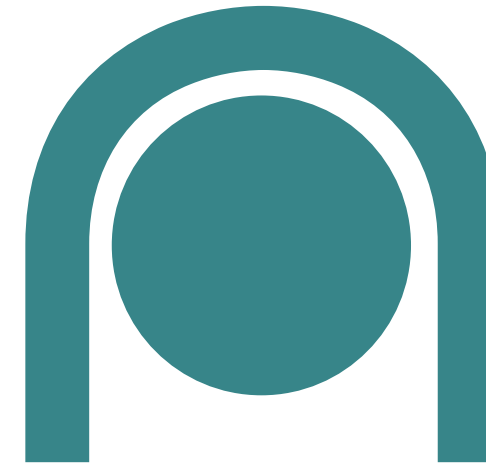
JOIN
Derives common data from a different table

GROUP BY
Aggregates the data

SELECT
Returns or derives the required data

simplilearn

Order of Execution: Example

Consider the following syntax taken from the example of logical operators.

Syntax
SELECT Emp_ID, Emp_F_Name from Employee_Records WHERE Emp_F_Name LIKE 'M%';

Here, the **FROM** clause is executed first to determine the table. Next, the **WHERE** clause is executed to determine the condition. The **SELECT** statement is executed to extract the data that satisfies this condition in the table.

Order of Execution: Example

Consider that you have a **customers** table with customer ID, customer name, and their location.

If you want to identify the locations of more than five customers, then use the following syntax.

Syntax

SELECT COUNT(CustomerID), Location FROM Customers GROUP BY Location,
HAVING COUNT(CustomerID) > 5;

Here, the **FROM** clause is executed first to determine the table. Next, the **GROUP BY** clause aggregates the same location records. It is followed by the **HAVING** clause that determines the condition. The **SELECT** statement is executed to extract the data that satisfies this condition in the table.

MySQL Constraints

MySQL Constraints

Constraint is a condition that specifies the type of data that can be entered into a table.
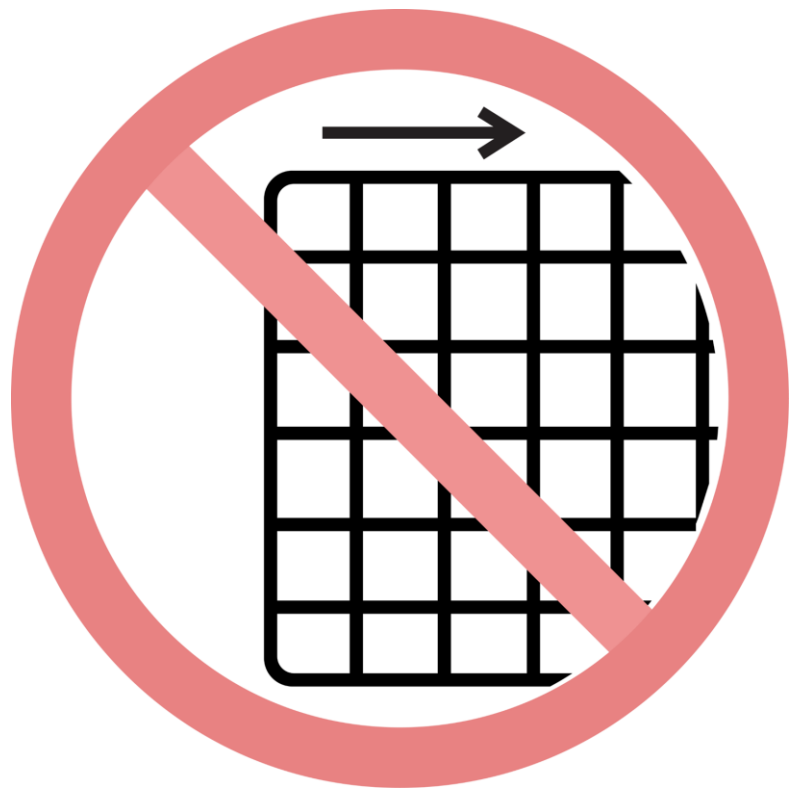
There are two types of constraints in MySQL:

- Column level restrictions

- Table level restrictions
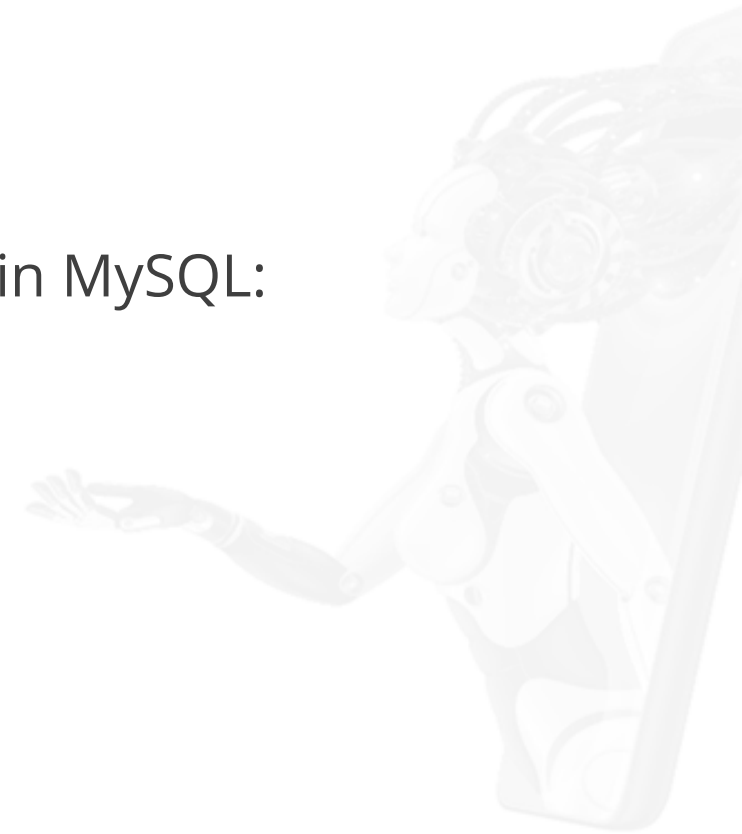
NOT NULL Constraint

NOT NULL constraint prevents the column from having NULL or empty values.

Example

CREATE table Employee (ID int, First_Name text NOT NULL, Last_Name text NOT_NULL, City VARCHAR(30))

Primary Constraint

Primary constraint provides a distinct identity to each record in a table. A table can only have one primary key.

Example

CREATE table People (ID int Primary Key, Name varchar (30) NOT NULL, Age int)

Primary and NOT NULL Constraints: Example

Problem Statement: As a product manager, you are required to create a table with product details, such as product ID which is the primary key, product name, and date of manufacturing which is not a not value.

Example

```
CREATE table Product_Details (Pro_ID int Primary Key, Pro_Name varchar (30) NOT NULL, Date_Manf
DATE);
```

Primary and NOT NULL Constraints: Example

After creating the table, if there is no record for the **NOT NULL** field as shown below, then you are prompted with an error.

Example

insert into Product_Details (Pro_ID, Date_Manf) values (151, "2021-02-24");

Error Code: 1364. Field 'Pro_Name' doesn't have a default value

Foreign Key Constraint
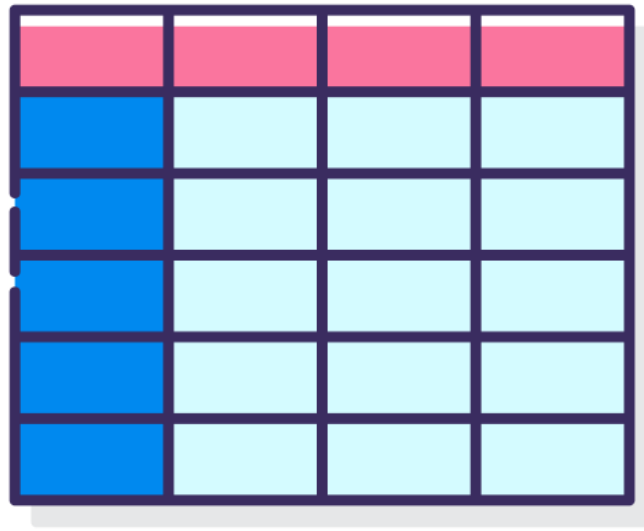
Foreign key constraint is used to connect two tables. It corresponds to the primary key of a different table.

Example

CREATE table Teachers (Teacher_ID int Primary Key, Name varchar (30) NOT NULL, Age int, College_ID int Foreign Key)

Disable Foreign Key Check

Foreign key check is a feature in SQL that prevents us from making any changes to tables that have foreign keys in them.

- If you want to make changes to the table, disable the foreign key check, make changes, and enable the key again.

- You can disable it by assigning zero to foreign key check.

Example

SET foreign_key_checks = 0

Foreign Key Constraint: Example

Problem statement: You are the sales manager of a store. You have data of your customers and their orders in two different tables. You must ensure that the customer data added to the table on orders is not different from the original data.

Objective: Use a foreign key to specify the column that must contain only the data present in the primary table.
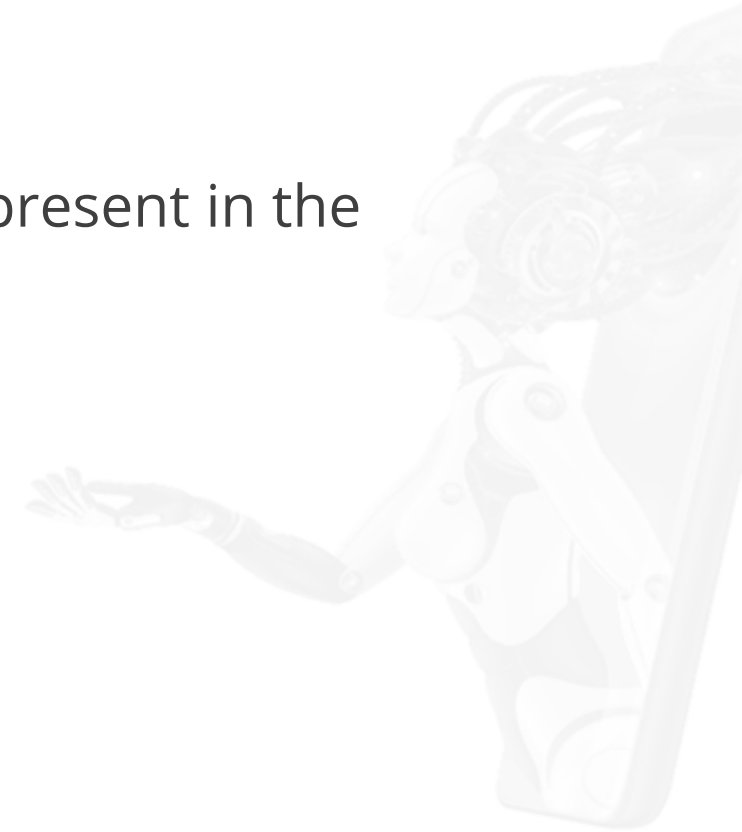
Foreign Key Constraint: Example

Steps to perform:

- Create a table with data on customers, like customer name, last name, age, and customer ID as primary key

- Create a table with data on orders, with order ID, order number, and person ID as the foreign key

- Set foreign key to zero to ensure that there are no external changes
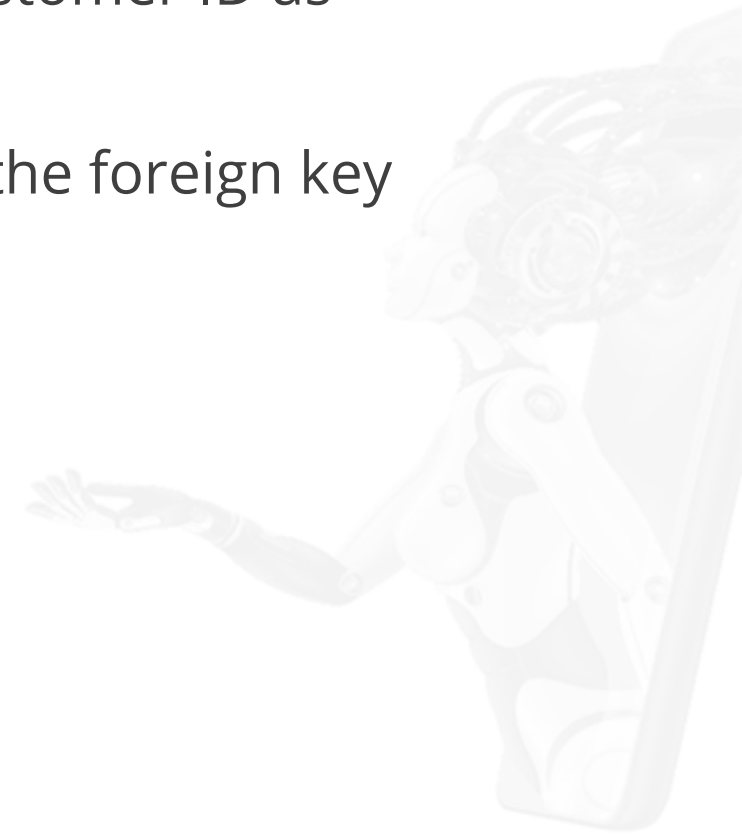
Foreign Key Constraint: Example

After creating the two tables, set foreign check to zero and insert them with the following data:

Example

insert into customers (Customer_ID, First_Name, Last_Name, Age) values (1,'Mark', 'Bouncer', 23),(2,'Max','Hussey',34),(3,'Harry','James',44);
insert into orders (Order_ID, Order_Number, Customer_ID) VALUES (1,7765,3),(2,7734,3),(3,7789,2)

If you try to enter any customer ID that is not present in the customers table, MySQL will prompt an error.

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails

Unique Constraint

Unique constraint ensures that there are no entries with the same value in a column.

Example

Create table Names (ID int NOT NULL, Name varchar (30), Age int, UNIQUE (ID))

Unique Constraint: Example

Create a table named **names** using the syntax shown before. Add values to this table.

Example

insert into names (ID, Name, Age) values (1, 'George', 35),(2, 'Lily', 28);

You can see that there is an ID with 2 as value. If you try to enter a new record with the same ID, you will be prompted with the following error.

Error Code: 1062. Duplicate entry '2' for key 'names.ID'

Check Constraints

Check constraint can be used to verify the value being entered into a record.

Example

Create table Tenants (ID int NOT NULL, Name varchar (30), Age int, Check (Age >=18))

Check Constraints: Example

Consider the example shown before for unique constraint. Add the check condition to verify that ID is not more than 10.

Example

Create table Names (ID int NOT NULL, Name varchar (30), Age int, UNIQUE (ID), Check (ID<=10));

When the ID entered is more than 10, the you will get the following error prompt.

Error Code: 3819. Check constraint 'names_chk_1' is violated.

Check Constraints Emulation

These constraints are used to emulate the CHECK constraints.

The two MySQL triggers used are:

BEFORE INSERT

BEFORE UPDATE

Duration: 10 Min.

Problem Statement: You are required to create a new table with constraints and assign **Candidate_No.** as the **primary key** in the MySQL Workbench.

Steps to be performed:

1. Create a table named **candidates**, name columns as **Candidate_No.**, **FirstName**, **LastName**, and **Age**, and assign **Candidate_No.** as the **primary key**.

TABLE CREATION

```
CREATE TABLE `example`.`candidates` (

`Candidate_No.` INT NOT NULL,

`FirstName` VARCHAR(255) NOT NULL,

`LastName` VARCHAR(255) NOT NULL,

`Age` INT NOT NULL,  PRIMARY KEY (`Candidate_No.`));
```

Assisted Practice: Lab Output

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
🔑 Candidate_No.	INT	☑	☑	☐	☐	☐	☐	☐	☐	
◇ FirstName	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐	
◇ LastName	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	☐	
◇ Age	INT	☐	☑	☐	☐	☐	☐	☐	☐	
◇ DOB	DATE	☐	☑	☐	☐	☐	☐	☐	☐	

Column Name: Candidate_No.

Data Type: INT

Charset/Collation: Default Charset   Default Collation

Default:

Comments:

Storage: ○ Virtual   ○ Stored

☑ Primary Key   ☑ Not Null   ☐ Unique

☐ Binary   ☐ Unsigned   ☐ Zero Fill
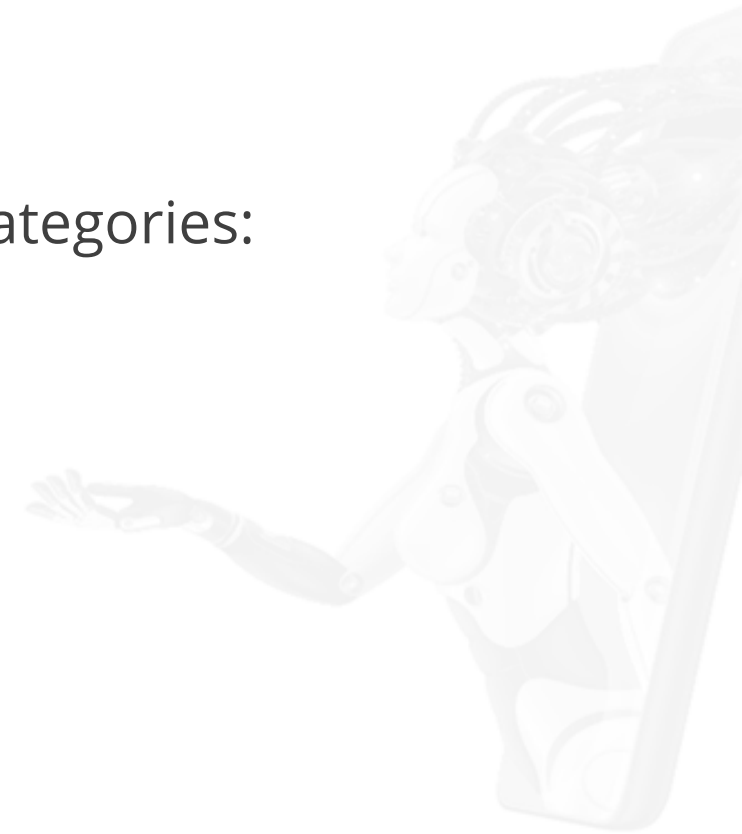
☐ Auto Increment   ☐ Generated

SQL Data Types

Data Types in SQL

Data type refers to the nature or format of data that is entered into the database.

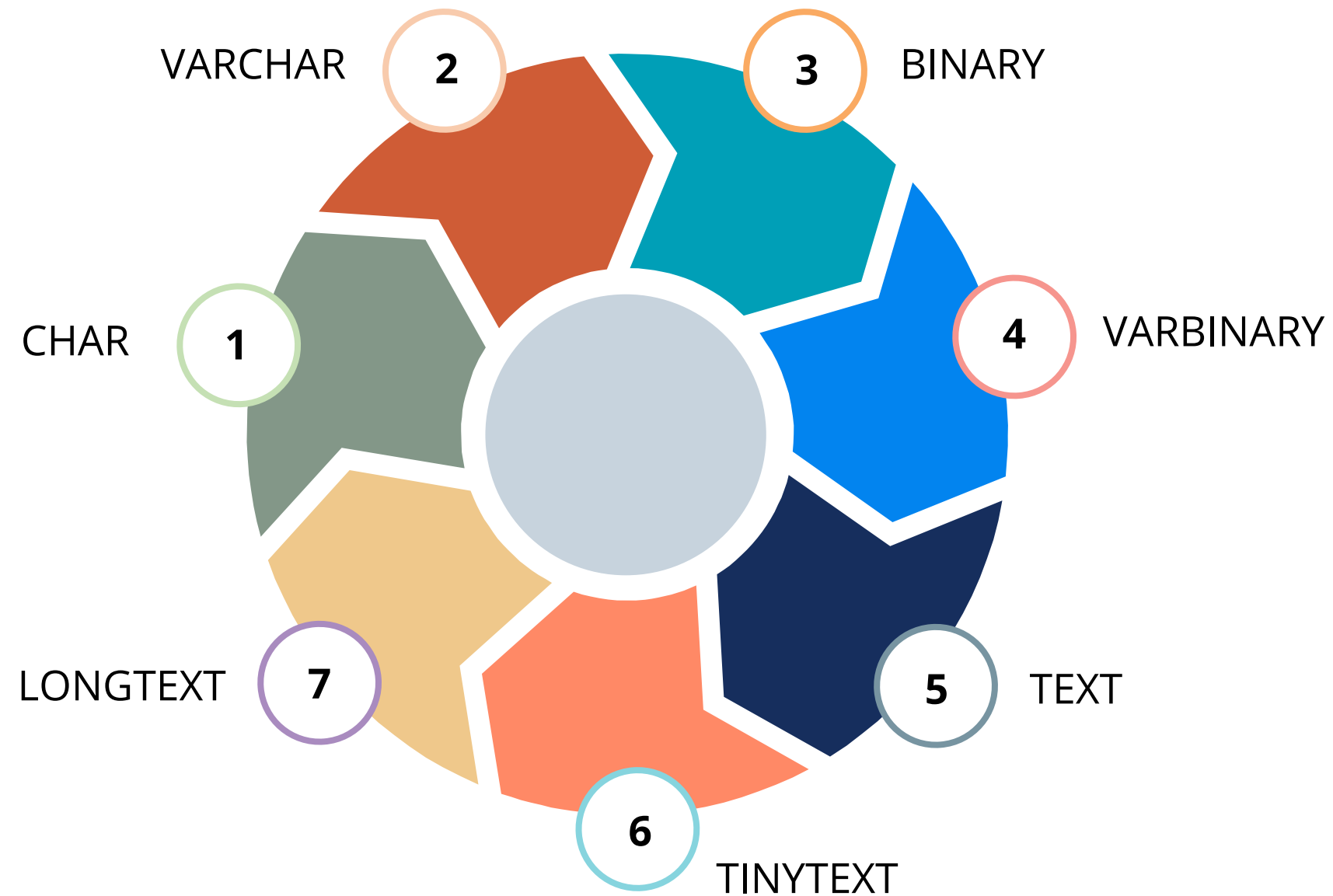Data types are divided into three categories:

- String
- Numeric
- Time and Date

String Data Types in SQL

VARCHAR 2

BINARY 3

CHAR 1

VARBINARY 4

LONGTEXT 7

TEXT 5

TINYTEXT 6

String Data Type: Example

Problem Statement: You are an IT administrator and want to create a table that shows the office assets assigned to each employee, with the employee ID, employee Name with a restriction of number of characters, and asset name which does not have any character limit.

Objective: Create a table with employee name of char data type and asset name of varchar data type.

String Data Type: Example

CREATE TABLE Asset_Tracker (Emp_ID int, Emp_Name char (7), Asset_Name varchar (255);

Employee name has a character limit. When you enter a longer name as shown below in the third instance, you will get an error notification.

```
insert into Asset_Tracker (Emp_ID, Emp_Name, Asset_Name)
values (23, 'Michael', 'Printer'),(46, 'John', 'Laptop'),(36,
'Samantha', 'Desktop Printer');
```

Error Code: 1406. Data too long for column 'Emp_Name' at row 3

Numeric Data Types in SQL

INT
Used to represent integer value

FLOATING
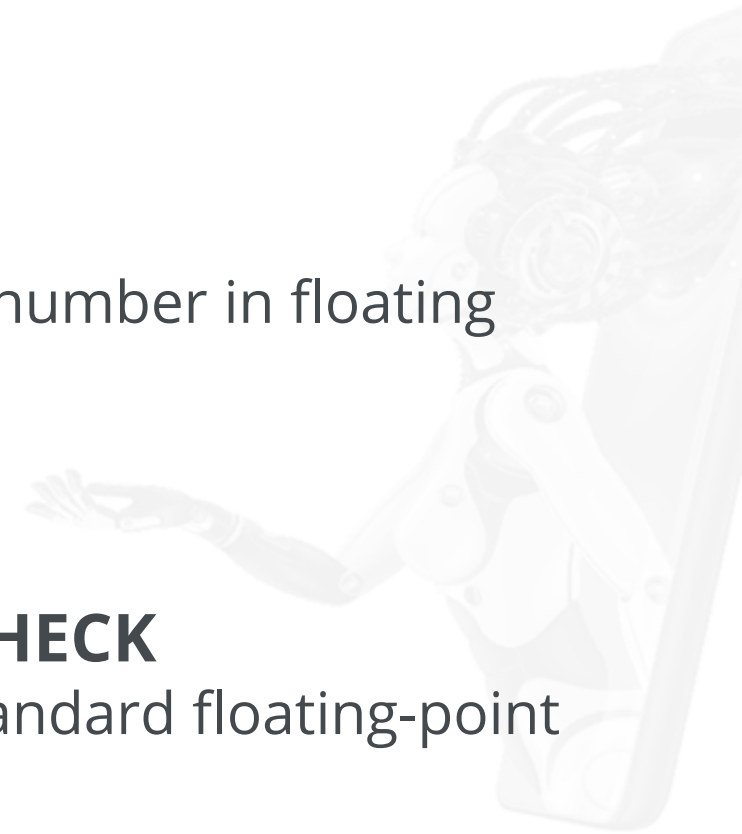Used to specify a number in floating point format

BIT
Used to denote bit-value type
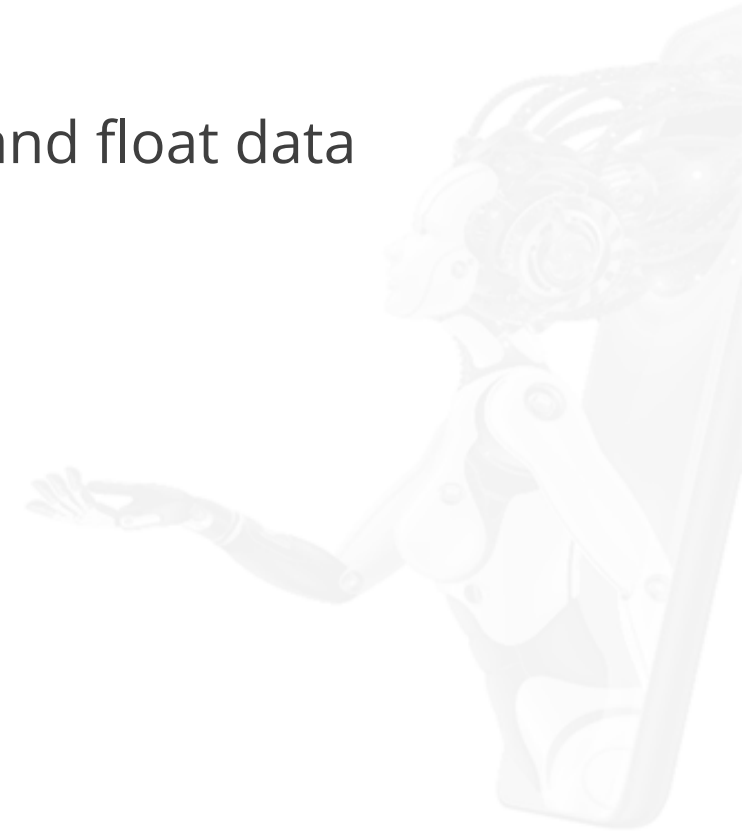
DOUBLE CHECK
Used as a standard floating-point number

Numeric Data Type: Example

Problem Statement: You are a sales manager who wants to create a table with price and quantity of each item that has been sold.

Objective: Create a table with product name, quantity, and price with varchar, int, and float data type respectively.

Numeric Data Type: Example

Syntax

CREATE TABLE Sales_Tracker (Pro_Name varchar(255), Pro_Price float, Pro_Quantity int);

Price is a float data type, because prices can have decimal values; quantity is always an integer.

Syntax

```
insert into Sales_Tracker (Pro_Name, Pro_Price, Pro_Quantity) values ('Mobiles',
8999.99, 26),('Laptops',24455.77, 48),('Washing_Machines',2344.55, 34);
```

	Pro_Name	Pro_Price	Pro_Quantity
▶	Mobiles	8999.99	26
	Laptops	24455.7	48
	Washing_Machines	2344.55	34

Date and Time Data Types in SQL

DATE

DATETIME

TIME

TIMESTAMP
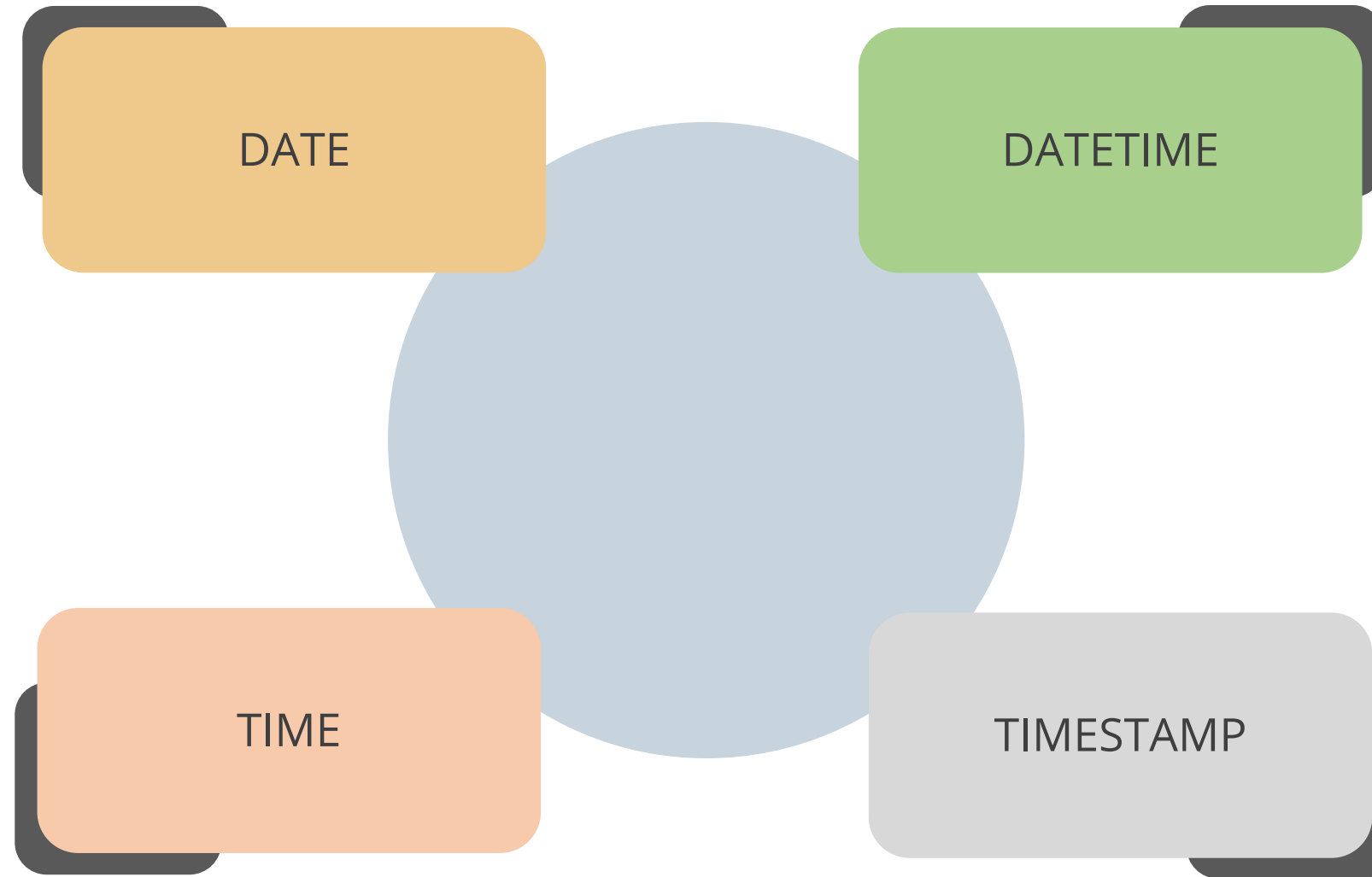
Assisted Practice: Data Type

Duration: 10 Min.

Problem Statement: You are required to create a new table with a field **DOB** with datatype as **DATE** in the MySQL Workbench.

Assisted Practice: Data Type

Steps to be performed:

1. Create a table named **candidates**; name columns: **Candidate_No.** as **Integer**, **FirstName** as **Varchar**, **LastName** as **Varchar**, **Age** as **Integer**, and **DOB** as **Date**. Assign **Candidate_No.** as the **primary key**.

TABLE CREATION

```
CREATE TABLE `example`.`candidates` (

`Candidate_No.` INT NOT NULL,

`FirstName` VARCHAR(255) NOT NULL,

`LastName` VARCHAR(255) NOT NULL,

`Age` INT NOT NULL,

`DOB` DATE NOT NULL,

PRIMARY KEY (`Candidate_No.`));
```

ASSISTED PRACTICE

Assisted Practice: Lab Output

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	Candidate_No.	example	candidates	INT	binary	11	0	0
2	FirstName	example	candidates	VARCHAR	utf8mb4	255	0	0
3	LastName	example	candidates	VARCHAR	utf8mb4	255	0	0
4	Age	example	candidates	INT	binary	11	0	0
5	DOB	example	candidates	DATE	binary	10	0	0

Key Takeaways

◉ SQL operators are used to specify a condition in an SQL statement.

◉ Arithmetic operators are used to perform arithmetic operations in SQL query.

◉ Join clause is used to derive common data from another table in the database.

◉ Foreign key check is a feature in SQL that prevents you from making any changes to tables that have foreign keys in them.

simplilearn

Knowledge Check

Knowledge Check

1

Which of the following operators is used to compare two conditions?

A.   Comparison operators

B.   Compound operators

C.   Logical operators

D.   Arithmetic operators

Knowledge Check

1

Which of the following operators is used to compare two conditions?

A. Comparison operators

B. Compound operators

C. Logical operators

D. Arithmetic operators

The correct answer is **C**

Logical operators are used to compare two conditions in an SQL query.

Which of the following clauses comes first in the order of execution?

A. WHERE

B. FROM

C. SELECT

D. HAVING

Which of the following clauses comes first in the order of execution?

A.   WHERE

B.   FROM

C.   SELECT

D.   HAVING

The correct answer is   **B**

FROM is the first clause that will be executed in an SQL query.

Which of the following constraints is used to provide a unique identity to a column in a table?

A. Primary key

B. Foreign key

C. Unique constraint

D. Check constraint

Which of the following constraints is used to provide a unique identity to a column in a table?

A. Primary key

B. Foreign key

C. Unique constraint

D. Check constraint

The correct answer is **A**

Primary key is the constraint that is used to provide a unique identity to a column in a table.