ALL_ABOUT MYSQL

INTRODUCTION

- MySQL is a relational database management system
- MySQL is open-source and free
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, scalable, and easy to use
- MySQL is cross-platform
- MySQL is compliant with the ANSI SQL standard
- MySQL was first released in 1995
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

Entity Relationship Diagram

What is ER Diagram?

- ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database.

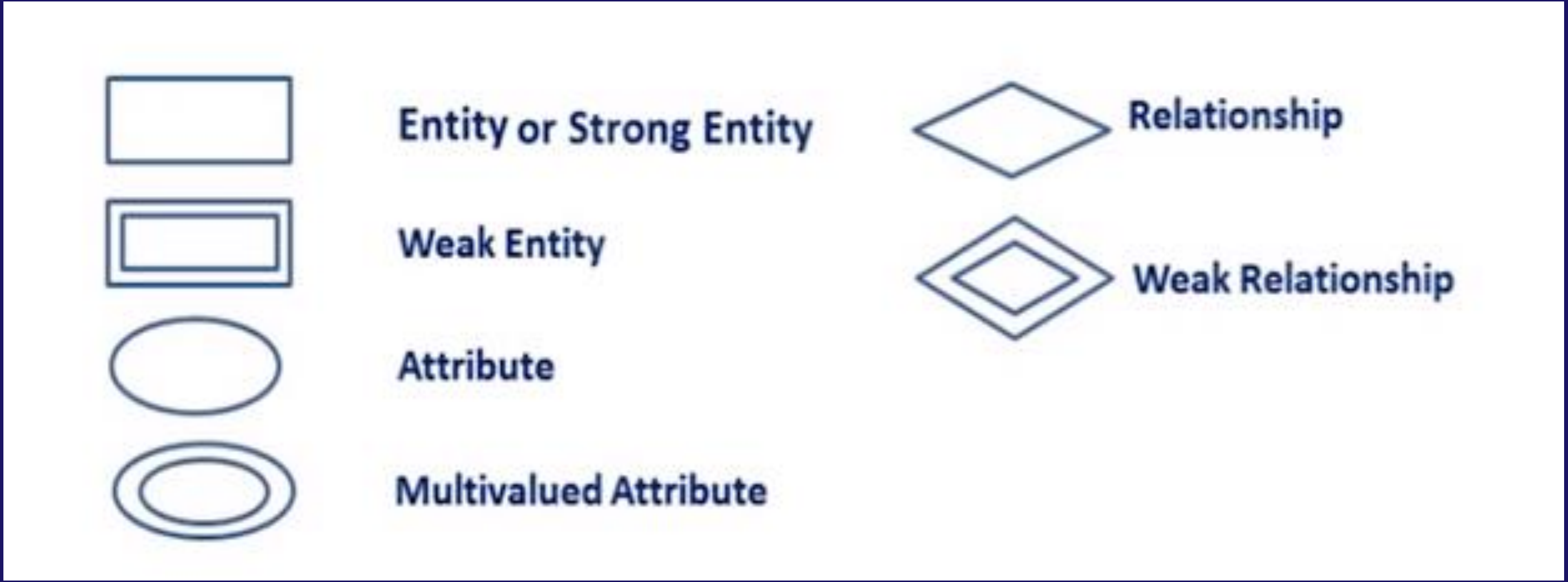
- ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

Why use ER Diagrams?

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications

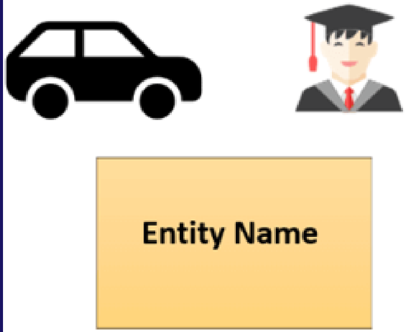
ER Diagrams Symbols & Notations

- Rectangles: This Entity Relationship Diagram symbol represents entity types
- Ellipses : Symbol represent attributes
- Diamonds: This symbol represents relationship types
- Lines: It links attributes to entity types and entity types with other relationship types
- Primary key: attributes are underlined
- Double Ellipses: Represent multi-valued attributes


Entity or Strong Entity
Weak Entity
Attribute
Multivalued Attribute
Relationship
Weak Relationship

Components of the ER Diagram

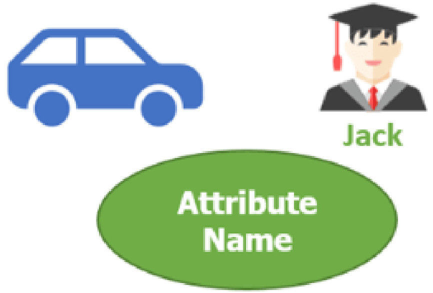
- Entities
- Attributes
- Relationships



Entity

Person,place,object,event or concept about which data is to be maintained

Example: Car, Student

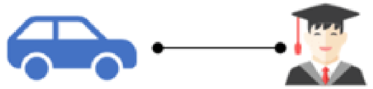


Attribute

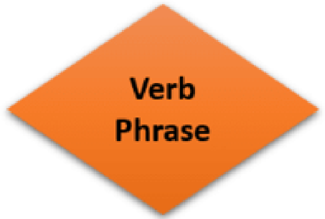
Property or characteristic of an entity

Example: Color of car Entity

Name of Student Entity



Relation



Association between the instances of one or more entity types

Example: Blue Car Belongs to Student Jack

WHAT IS ENTITY?

- An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some ‘attributes’ which represent that entity.

WHAT IS Attributes?

- It is a single-valued property of either an entity-type or a relationship-type.
- For example, a lecture might have attributes: time, date, duration, place, etc.
- An attribute in ER Diagram examples, is represented by an Ellipse

Types of Attributes	Description
● Simple attribute	Simple attributes can’t be divided any further. For example, a student’s contact number. It is also called an atomic value.
● Composite attribute	It is possible to break down composite attribute. For example, a student’s full name may be further divided into first name, second name, and last name.
● Derived attribute	This type of attribute does not include in the physical database. However, their values are derived from other attributes present in the database. For example, age should not be stored directly. Instead, it should be derived from the DOB of that employee.
● Multivalued attribute	Multivalued attributes can have more than one values. For example, a student can have more than one mobile number, email address, etc.

WHAT IS Cardinality?

- Defines the numerical attributes of the relationship between two entities or entity sets.

  One-to-One Relationships

  One-to-Many Relationships

  May to One Relationships

  Many-to-Many Relationships

DBMS Normalization

What is Normalization?

- Normalization is a database design technique that reduces data redundancy and eliminates unwanted characteristics.
- Normalization rules divides larger table into smaller tables and links them using relationships.

Purpose of Normalization?

- The purpose of Normalization in SQL is to eliminate redundant data and ensure data is stored logically.
- Edgar Codd inventor of Relational Model proposed the theory of normalization with 1NF.

1NF Normalization

- In 1NF a table’s attribute would not be able to hold various values it will only be able to hold an attribute of single Value.
- Each record needs to be unique.

EXAMPLE

Stu_No	Name	Courses
11	Sourav	Web, Android
12	Shiran	C++
13	Kishon	C++, Java

There are some multiple values in courses column. We use 1NF method to resolve it as follows.

EXAMPLE

Stu_No	Name	Courses
11	Sourav	Web
11	Sourav	Android
12	Shiran	C++
13	Kishon	C++
13	Kishon	Java

There are some values getting repeated but there is just one value in every column.

2NF Normalization

- A relation is said to be in 2NF when it exists in 1NF, while the relation's every non-prime attribute depends on every candidate key as a whole.
- If a relation is in 1NF and all the attributes of the non-primary keys are fully dependent on primary keys, then this relation is known to be in the 2NF or the Second Normal Form.

EXAMPLE

Lecturer_Table

Lecturer_ID	Course	Lecturer_Age
1001	Java	34
1001	C++	34
1204	Web	29
1212	Android	32
1212	Python	32

EXAMPLE

Lecturer_Detail_Table

Lecturer_ID	Lecturer_Age
1001	34
1204	29
1212	32

EXAMPLE

Lecturer_Course_Table

Lecturer_ID	Course
1001	Java
1001	C++
1204	Web
1212	Android
1212	Python

3NF Normalization

- In a relation that is in 1NF or 2NF, when none of the non-primary key attributes transitively depend on their primary keys, then we can say that the relation is in the third normal form of 3NF.

- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

EXAMPLE

Student_Table

Stu_No	**Name**	**Postcode**	**City**	**Province**
11	Sourav	40000	Jaffna	North
12	Shiran	31000	Trincomalee	East
13	Kishon	90000	Badulla	Uva
14	Stephan	00800	Borella	West
15	Biet	20400	Peradeniya	Central

EXAMPLE

Student_Table

Stu_No	**Name**	**Postcode**
11	Sourav	40000
12	Shiran	31000
13	Kishon	90000
14	Stephan	00800
15	Biet	20400

EXAMPLE

Student_City_Table

<u>**Postcode**</u>	**City**	**Province**
40000	Jaffna	North
31000	Trincomalee	East
90000	Badulla	Uva
00800	Borella	West
20400	Peradeniya	Central

MYSQL DATA TYPES

String Data Types

Data type	Description
● CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
● VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535

String Data Types

Data type	Description
● BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
● VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
● TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
● TINYTEXT	Holds a string with a maximum length of 255 characters
● TEXT(size)	Holds a string with a maximum length of 65,535 bytes

String Data Types

Data type	Description
● BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
● MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
● MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
● LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
● LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data

String Data Types

Data type	Description
● ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
● SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types

Data type	Description
● BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
● TINYINT(size)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
● BOOL	Zero is considered as false, nonzero values are considered as true.

Numeric Data Types

Data type	Description
● BOOLEAN	Equal to BOOL
● SMALLINT(size)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)
● MEDIUMINT(size)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)

Numeric Data Types

Data type	Description
● INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
● INTEGER(size)	Equal to INT(size)
● BIGINT(size)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)

Numeric Data Types

Data type	Description
● FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
● FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()

Numeric Data Types

Data type	Description
● DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
● DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.DEC(size, d) Equal to DECIMAL(size,d)

Numeric Data Types

Data type	Description
● DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
● DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.DEC(size, d) Equal to DECIMAL(size,d)

Date and Time Data Types

Data type	Description
● DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
● DATETIME(fsp)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time

Date and Time Data Types

Data type	Description
● TIMESTAMP(fsp)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
● TIME(fsp)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'

Date and Time Data Types

Data type	Description
● YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

MySQL Database

01

MySQL CREATE DATABASE

MySQL CREATE DATABASE Statement

- The CREATE DATABASE statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

EXAMPLE

The following SQL statement creates a database called "testDB":

```
CREATE DATABASE testDB;
```

02

MySQL DROP DATABASE

MySQL DROP DATABASE Statement

- The DROP DATABASE statement is used to drop an existing SQL database.

Syntax

```
DROP DATABASE databasename;
```

EXAMPLE

The following SQL statement drops the existing database "testDB":

```
DROP DATABASE testDB;
```

03

MySQL CREATE TABLE

MySQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

EXAMPLE

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255) );
```

OUTPUT

PersonID	LastName	FirstName	Address	City

04

MySQL DROP TABLE

MySQL DROP TABLE Statement

- The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

EXAMPLE

The following SQL statement drops the existing table "Shippers":

```
DROP TABLE Shippers;
```

MySQL TRUNCATE TABLE

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE *table_name*;
```

05

MySQL ALTER TABLE

MySQL ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD COLUMN

Syntax

```
ALTER TABLE table_name
ADD column_name datatype;
```

ALTER TABLE - DROP COLUMN

Syntax

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

ALTER TABLE - MODIFY COLUMN

Syntax

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

EXAMPLE

The following SQL statement to change the data type DATE of the column named "DateOfBirth" in the "Persons" table.

```
ALTER TABLE Persons

MODIFY COLUMN DateOfBirth year;
```

06

MySQL Constraints

MySQL Constraints

- SQL constraints are used to specify rules for data in a table.

Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Create Constraints Syntax

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

07

MySQL NOT NULL Constraint

MySQL NOT NULL Constraint

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

EXAMPLE

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL,
Age int
);
```

EXAMPLE

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

08

MySQL UNIQUE Constraint

MySQL UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

EXAMPLE
CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
UNIQUE (ID) );
```

EXAMPLE
CREATE TABLE

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
CONSTRAINT UC_Person UNIQUE (ID,LastName) );
```

EXAMPLE
ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons

ADD UNIQUE (ID);
```

EXAMPLE
ALTER TABLE

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons

ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

EXAMPLE
DROP TABLE

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons
DROP INDEX UC_Person;
```

09

MySQL PRIMARY KEY Constraint

MySQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

EXAMPLE
CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
PRIMARY KEY (ID) );
```

10

MySQL FOREIGN KEY Constraint

MySQL FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.
- The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

EXAMPLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int,
PRIMARY KEY (OrderID),
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID) );
```

11

MySQL CHECK Constraint

MySQL CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

EXAMPLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

CHECK (Age>=18) );
```

EXAMPLE

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255),
CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes’) );
```

12

MySQL DEFAULT Constraint

MySQL DEFAULT Constraint

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.

EXAMPLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255) DEFAULT 'Sandnes’ );
```

EXAMPLE

The DEFAULT constraint can also be used to insert system values, by using functions like CURRENT_DATE():

```
CREATE TABLE Orders (
ID int NOT NULL,
OrderNumber int NOT NULL,
OrderDate date DEFAULT CURRENT_DATE() );
```

13

MySQL CREATE INDEX Constraint

MySQL CREATE INDEX Constraint

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note : Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

```
CREATE INDEX index_name
ON table_name (column1, column2,
...);
```

CREATE UNIQUE INDEX

Syntax

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2,
...);
```

EXAMPLE

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname

ON Persons (LastName);
```

14

MySQL AUTO INCREMENT Constraint

MySQL AUTO INCREMENT Constraint

- MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

EXAMPLE

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons (
Personid int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
PRIMARY KEY (Personid) );
```

EXAMPLE

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

15

MySQL Dates

MySQL Dates

- The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

- As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

MySQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

●	DATE -	format YYYY-MM-DD
●	DATETIME -	format: YYYY-MM-DD HH:MI:SS
●	TIMESTAMP -	format: YYYY-MM-DD HH:MI:SS
●	YEAR -	format YYYY or YY

Note :- The date data type are set for a column when you create a new table in your database!

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-08	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

to select the records with an OrderDate of "1996-07-08" from the table above.
We use the following SELECT statement:

```
SELECT * FROM Orders WHERE OrderDate='1996-07-08’;
```

OUTPUT

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-08	F-95852	France

16

MySQL Views

MySQL Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
- A view is created with the CREATE VIEW statement.

Note :- A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

EXAMPLE

The following SQL creates a view that shows all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

MySQL Updating a View

- A view can be updated with the CREATE OR REPLACE VIEW statement.

CREATE OR REPLACE VIEW

Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

EXAMPLE

The following SQL adds the "City" column to the "Brazil Customers" view:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

SQL VIEW to fetch all records of a table

- It is the simplest form of a VIEW. Usually, we do not use a VIEW in SQL Server to fetch all records from a single table.

EXAMPLE

```
CREATE VIEW EmployeeRecords
AS
     SELECT *
     FROM
[HumanResources].[Employee];
```

SQL VIEW to fetch a few columns of a table

- We might not be interested in all columns of a table. We can specify required column names in the select statement to fetch those fields only from the table.

EXAMPLE

```
CREATE VIEW EmployeeRecords
            AS
SELECT NationalIDNumber,LoginID,
        JobTitle
FROM [HumanResources].[Employee];
```

SQL VIEW to fetch a few columns of a table and filter results using WHERE clause

- SQL VIEW to fetch a few columns of a table and filter results using WHERE clause

EXAMPLE

```
CREATE VIEW EmployeeRecords AS
  SELECT NationalIDNumber,LoginID,
           JobTitle,MaritalStatus
  FROM [HumanResources].[Employee]
      WHERE MaritalStatus = 'M';
```

SQL VIEW to fetch specific column

- Once we have a view, it is not required to fetch all columns from the view. We can select few columns as well from a VIEW in SQL Server similar to a relational table.

EXAMPLE

```
      SELECT Name,ContactType
FROM [Sales].[vStoreWithContacts];
```

BASIC MYSQL STATEMENT

01

MySQL SELECT Statement

MYSQL SELECT

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...
FROM table_name;


SELECT * FROM table_name;
```

MySQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsväge n 8	Luleå	S-958 22	Sweden

EXAMPLE

The following SQL statement selects the "CustomerName", "City", and "Country" columns from the "Customers" table:

```
SELECT CustomerName, City,
Country FROM Customers;
```

OUTPUT

CustomerName	City	Country
Alfreds Futterkiste	Berlin	Germany
Ana Trujillo	México D.F.	Mexico
Antonio Moreno	México D.F.	Mexico
Around the Horn	London	UK
Berglunds snabbköp	Luleå	Sweden

02

MySQL WHERE Clause

MYSQL WHERE

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

WHERE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Operators in The WHERE Clause

Operator	Description
● >	Greater than
● <	Less than
● >=	Greater than or equal
● <=	Less than or equal
● <>	Not equal. Note: In some versions of SQL this operator may be written as !=
● BETWEEN	Between a certain range
● LIKE	Search for a pattern
● IN	To specify multiple possible values for a column

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

EXAMPLE

The following SQL statement selects all the customers from "Mexico":

```
SELECT * FROM Customers
WHERE Country = 'Mexico';
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

03 MySQL AND, OR and NOT Operators

MYSQL AND, OR AND NOT OPERATORS

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition:
    - The AND operator displays a record if all the conditions separated by AND are TRUE.
    - The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2
AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2
OR condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

EXAMPLE

selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR “Mannheim"

```
SELECT * FROM Customers
WHERE Country = 'Germany' AND (City = 'Berlin' OR
City = ‘Mannheim');
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

04

ORDER BY Keyword

The MySQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword..

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ...
ASC|DESC;
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

EXAMPLE

selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

05
INSERT INTO Statement

INSERT INTO Statement

- The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

```
INSERT INTO table_name
(column1, column2, column3,
...)
VALUES (value1, value2, value3,
...);
```

INSERT INTO Syntax 2

```
INSERT INTO table_name
VALUES (value1, value2, value3,
...);
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

EXAMPLE

inserts a new record in the "Customers" table:

```
INSERT INTO Customers (CustomerName,
ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21',
'Stavanger', '4006', 'Norway');
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Cardinal	Tom B. Erichsen	Skagen 21	Stavan ger	4006	Norway

06
NULL Values

NULL Values

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

How to Test for NULL Values?

- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Cardinal	Tom B. Erichsen	Skagen 21	Stavan ger	4006	Norway

EXAMPLE

lists all customers with a NULL value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

OUTPUT

CustomerName	ContactName	Address

EXAMPLE

lists all customers with a value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

OUTPUT

CustomerName	ContactName	Address
Alfreds Futterkiste	Maria Anders	Obere Str. 57
Ana Trujillo	Ana Trujillo	Avda. de la Constitución
Antonio Moreno	Antonio Moreno	Mataderos 2312
Around the Horn	Thomas Hardy	120 Hanover Sq.
Cardinal	Tom B. Erichsen	Skagen 21

07 UPDATE Statement

UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.
- Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 =
value2, ...
WHERE condition;
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

EXAMPLE

updates the first customer (CustomerID = 1) with a new contact person and a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
WHERE CustomerID = 1;
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Cardinal	Tom B. Erichsen	Skagen 21	Stavan ger	4006	Norway

08 DELETE Statement

UPDATE Statement

- The DELETE statement is used to delete existing records in a table.
- Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

DELETE Syntax

```
DELETE FROM table_name WHERE
condition;
```

CUSTOMER TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

EXAMPLE

deletes the customer "Alfreds Futterkiste" from the "Customers" table:

```
DELETE FROM Customers WHERE
CustomerName='Alfreds Futterkiste';
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Cardinal	Tom B. Erichsen	Skagen 21	Stavan ger	4006	Norway

09
LIMIT Clause

LIMIT Clause

- The LIMIT clause is used to specify the number of records to return.
- The LIMIT clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

CUSTOMER TABLE

Customer ID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

EXAMPLE

selects the first three records from the "Customers" table:

```
SELECT * FROM Customers
LIMIT 3;
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

10

MIN() and MAX() Functions

MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

11

COUNT(), AVG() and SUM() Functions

COUNT(), AVG() and SUM() Functions

- The COUNT() function returns the number of rows that matches a specified criterion.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

12

LIKE Operator

LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character
- The percent sign and the underscore can also be used in combinations!

LIKE Operator

LIKE Operator	Description
WHERE CustomerName LIKE 'a%’	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

13

Wildcard Characters

Wildcard Characters

- A wildcard character is used to substitute one or more characters in a string.

- Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Wildcard Characters

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit

14

IN Operator

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,
value2, ...);
```

IN Syntax

```
SELECT * FROM Customers
WHERE Country IN
(SELECT Country FROM Suppliers);
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsväge n 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement selects all customers that are from the same countries as the orders:

```
SELECT * FROM Customers
WHERE Country IN
(SELECT Country FROM Orders);
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

15

BETWEEN Operator

BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

- The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value
1 AND value2;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

EXAMPLE

The following SQL statement selects all customers with a customerID 2 and 4:

```
SELECT * FROM Customers
WHERE CustomerID BETWEEN 2 AND 4;
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

16

MySQL Aliases

MySQL Aliases

- Aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

```
SELECT c.CustomerName AS Name, o.OrderID, o.OrderDate,
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND
c.CustomerID=o.CustomerID;
```

OUTPUT

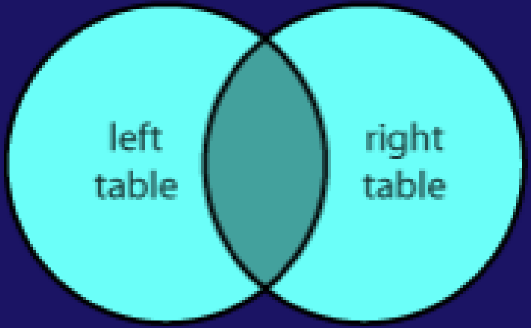
Name	OrderID	OrderDate
Around the Horn	10251	1996-07-08

17

MySQL INNER JOIN

MySQL INNER JOIN

- The INNER JOIN keyword selects records that have matching values in both tables.



INNER JOIN

Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.
column_name;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement selects all orders with customer information:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON
Orders.CustomerID = Customers.CustomerID;
```

OUTPUT

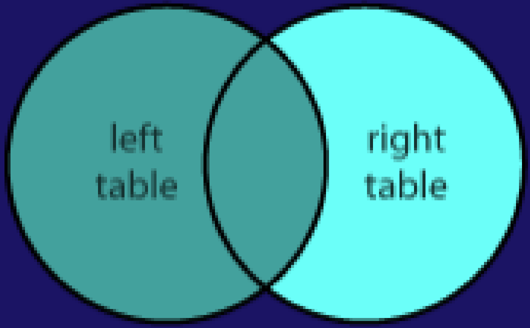
CustomerName	OrderID
Alfreds Futterkiste	10248
Ana Trujillo	10249
Antonio Moreno	10250
Around the Horn	10251
Berglunds snabbköp	10252

18

MySQL LEFT JOIN

MySQL LEFT JOIN

- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2).



LEFT JOIN
Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.
column_name;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement will select all customers, and any orders they might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
ORDER BY Customers.CustomerName;
```

OUTPUT

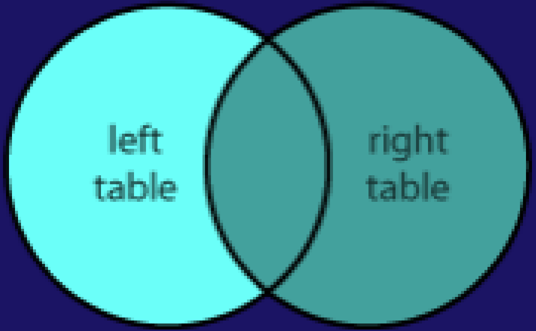
CustomerName	OrderID
Alfreds Futterkiste	10248
Ana Trujillo	10249
Antonio Moreno	10250
Around the Horn	10251
Berglunds snabbköp	10252

19

MySQL RIGHT JOIN

MySQL RIGHT JOIN

- The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).



RIGHT JOIN

Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.
column_name;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsväge n 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement will select all customers, and any orders they might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
RIGHT JOIN Orders ON Customers.CustomerID =
Orders.CustomerID
ORDER BY Customers.CustomerName;
```

OUTPUT

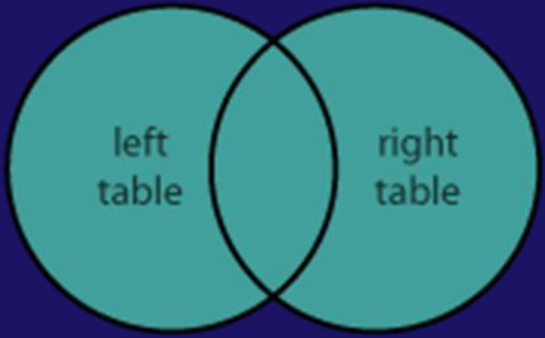
CustomerName	OrderID
Alfreds Futterkiste	10248
Ana Trujillo	10249
Antonio Moreno	10250
Around the Horn	10251
Berglunds snabbköp	10252
NULL	10254

20

MySQL CROSS JOIN

MySQL CROSS JOIN

- The CROSS JOIN keyword returns all records from both tables (table1 and table2).



CROSS JOIN
Syntax

```
SELECT column_name(s)
FROM table1
CROSS JOIN table2;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka

EXAMPLE

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
CROSS JOIN Orders
ORDER BY CustomerName;
```

OUTPUT

CustomerName	OrderID
Alfreds Futterkiste	10248
Alfreds Futterkiste	10249
Alfreds Futterkiste	10250
Ana Trujillo	10248
Ana Trujillo	10249
Ana Trujillo	10250

21

MySQL Self Join

MySQL Self Join

- A self join is a regular join, but the table is joined with itself.

Self Join Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and T2 are different table aliases for the same table.

STUDENT TABLE

Student_ID	Name	Course_ID	Duration
1	Adam	1	3
2	Peter	2	4
1	Aam	2	4
3	Brian	3	2
2	Shane	3	5

EXAMPLE

get all the result (student_id and name) from the table where student_id is equal, and course_id is not equal.

```
SELECT  s1.student_id, s1.name
FROM student AS s1, student s2
WHERE s1.student_id=s2.student_id
AND s1.course_id<>s2.course_id;
```

STUDENT TABLE

Student_ID	Name
1	Adam
2	Shane
1	Adam
2	Peter

22

MySQL UNION Operator

MySQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2
;
```

MySQL UNION ALL Statement

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

UNION ALL Syntax

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2 ;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsväge n 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement returns the cities (only distinct values) from both the "Customers" and the “Orders" table:

```
SELECT Country FROM Customers

UNION

SELECT Country FROM Orders

ORDER BY Country;
```

CUSTOMERS TABLE

Country
Germany
UK

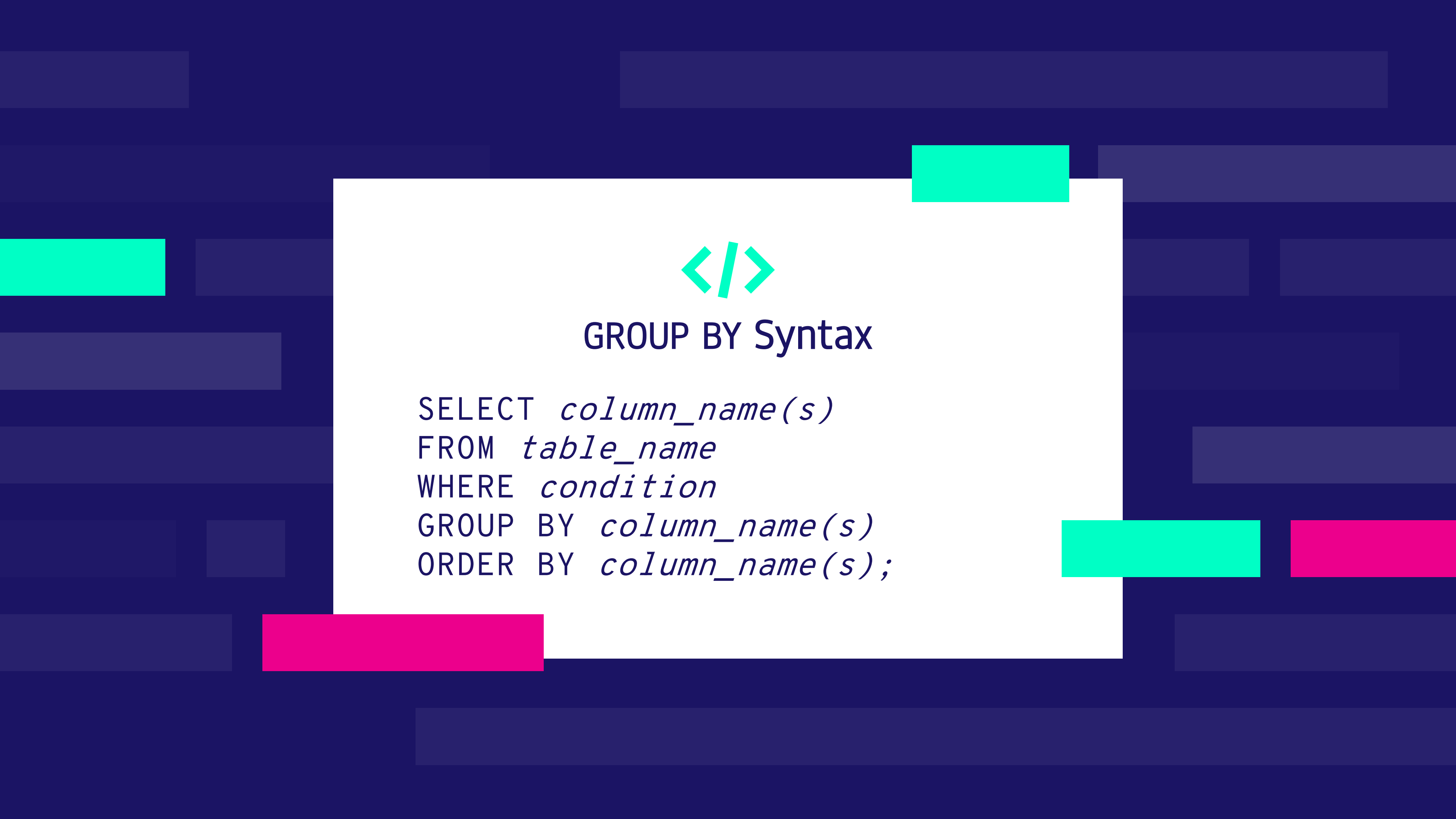
23

MySQL GROUP BY Statement

MySQL GROUP BY Statement

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

- The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.



GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsväge n 8	Luleå	S-958 22	Sweden

EXAMPLE

The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

OUTPUT

Count(CustomerID)	Country
1	Germany
2	Mexico
1	UK
1	Sweden

24

MySQL HAVING Clause

MySQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsväge n 8	Luleå	S-958 22	Sweden

EXAMPLE

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 1;
```

OUTPUT

Count(CustomerID)	Country
2	Mexcio

25

MySQL EXISTS Operator

MySQL EXISTS Operator

- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name
 WHERE condition);
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement returns TRUE and lists the Customers with a product Country is SriLanka:

```
SELECT *
FROM Customers
WHERE EXISTS
(SELECT OrderID FROM Orders WHERE
Customers.CustomerID= Orders.OrderID AND Country = SriLanka);
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

26

MySQL ANY and ALL Operators

MySQL ANY and ALL Operators

- The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

Note :- The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

The ANY Operator

- The ANY operator:
    - returns a boolean value as a result
    - returns TRUE if ANY of the subquery values meet the condition
- ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

The ALL Operator

- The ALL operator:

  - returns a boolean value as a result
  - returns TRUE if ALL of the subquery values meet the condition
  - is used with SELECT, WHERE and HAVING statements

- ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
   (SELECT column_name
   FROM table_name
   WHERE condition);
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

ORDERS TABLE

OrderID	CustomerID	EmployeeID	OrderDate	PostalCode	Country
10248	1	5	1996-07-04	12209	Germany
10249	2	6	1996-07-05	40000	SriLanka
10250	3	4	1996-07-08	40000	SriLanka
10251	4	3	1996-07-08	WA1 1DP	UK
10252	5	4	1996-07-09	F-95852	France
10254	6	5	1996-07-11	WA1 1DP	UK

EXAMPLE

The following SQL statement lists the All if it finds ANY records in the Orders table has PostalCode equal to 40000 (this will return TRUE because the PostalCode column has some values of 40000):

```
SELECT *

FROM Customer

WHERE Country = ANY

(SELECT Country

FROM Orders

WHERE PostalCode = 40000);
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

27

MySQL INSERT INTO SELECT Statement

MySQL INSERT INTO SELECT Statement

- The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
- The INSERT INTO SELECT statement requires that the data types in source and target tables matches.

Note :- The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all Columns

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

INSERT INTO SELECT Syntax

Copy Some Columns

```
INSERT INTO table2 (column1, column
2, column3, ...)
SELECT column1, column2, column3,
...
FROM table1 WHERE condition;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

ORDERS TABLE

OrderID	CustomerID	OrderName	OrderDate	City	Country
10248	1	Tom	1996-07-04	Berlin	Germany
10249	2	Parker	1996-07-05	Trinco	SriLanka
10250	3	Jonny	1996-07-08	Kandy	SriLanka
10251	4	Peter	1996-07-08	London	UK

EXAMPLE

The following SQL statement copies “Orders" into "Customers" (the columns that are not filled with data, will contain NULL):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT OrderName, City, Country FROM Orders;
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
Null	Tom	Null	Null	Berlin	Null	Germany
Null	Parker	Null	Null	Trinco	Null	SriLanka
Null	Jonny	Null	Null	Kandy	Null	SriLanka
Null	Peter	Null	Null	London	Null	UK

28

MySQL CASE Statement

MySQL CASE Statement

- The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

- If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

CUSTOMERS TABLE

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo	Ana Trujillo	Avda. de la Constitución	México D.F.	05021	Mexico
3	Antonio Moreno	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Queens	S-958 22	UK

EXAMPLE

The following SQL goes through conditions and returns a value when the first condition is met:

```
SELECT CustomerID, City,
CASE
WHEN City = México D.F. THEN 'The country is Mexico'
WHEN City = Berlin THEN 'The country is Germany'
ELSE 'The country is UK'
END AS CountryName
FROM Customers;
```

OUTPUT

CustomerID	City	CountryName
1	Berlin	The Country is Germany
2	México D.F.	The Country is Mexico
3	México D.F.	The Country is Mexico
4	London	The Country is UK
5	Queens	The Country is UK

29

NULL Functions

IFNULL() Function

- The MySQL IFNULL() function lets you return an alternative value if an expression is NULL.

PRODUCT TABLE

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	

IFNULL() Syntax

```
SELECT ProductName, UnitPrice *
(UnitsInStock +
IFNULL(UnitsOnOrder, 0)) AS Digit
FROM Products;
```

PRODUCT TABLE

ProductName	Digit
Jarlsberg	323.95
Mascarpone	748.88
Gorgonzola	141.03

COALESCE() Syntax

```
SELECT ProductName, UnitPrice *
(UnitsInStock +
COALESCE(UnitsOnOrder, 0))
FROM Products;
```

30

MySQL Comments

MySQL Comments

- Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments

- Single line comments start with --.
- Any text between -- and the end of the line will be ignored (will not be executed).

EXAMPLE

The following example uses a single-line comment as an explanation:

```
-- Select all:
SELECT * FROM Customers;
```

MySQL Comments

- Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Multi-line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored.

EXAMPLE

The following example uses a multi-line comment as an explanation:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

31

MySQL Operators

MySQL Arithmetic Operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

MySQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

MySQL Comparison Operators

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

MySQL Compound Operators

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

MySQL Logical Operators

Operator	Description
● ALL	TRUE if all of the subquery values meet the condition
● AND	TRUE if all the conditions separated by AND is TRUE
● ANY	TRUE if any of the subquery values meet the condition
● BETWEEN	TRUE if the operand is within the range of comparisons
● EXISTS	TRUE if the subquery returns one or more records
● IN	TRUE if the operand is equal to one of a list of expressions
● LIKE	TRUE if the operand matches a pattern

MySQL Logical Operators

Operator	Description
● NOT	Displays a record if the condition(s) is NOT TRUE
● OR	TRUE if any of the conditions separated by OR is TRUE
● SOME	TRUE if any of the subquery values meet the condition

MySQL Functions

MySQL String Functions

Function	Description
● ASCII	Returns the ASCII value for the specific character
● CHAR_LENGTH	Returns the length of a string (in characters)
● CHARACTER_LENGTH	Returns the length of a string (in characters)
● CONCAT	Adds two or more expressions together
● CONCAT_WS	Adds two or more expressions together with a separator
● FIELD	Returns the index position of a value in a list of values
● FIND_IN_SET	Returns the position of a string within a list of strings

MySQL String Functions

Function	Description
● FORMAT	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
● INSERT	Inserts a string within a string at the specified position and for a certain number of characters
● INSTR	Returns the position of the first occurrence of a string in another string
● LCASE	Converts a string to lower-case

MySQL String Functions

Function	Description
● LEFT	Extracts a number of characters from a string (starting from left)
● LENGTH	Returns the length of a string (in bytes)
● LOCATE	Returns the position of the first occurrence of a substring in a string
● LOWER	Converts a string to lower-case
● LPAD	Left-pads a string with another string, to a certain length
● LTRIM	Removes leading spaces from a string

MySQL String Functions

Function	Description
● MID	Extracts a substring from a string (starting at any position)
● POSITION	Returns the position of the first occurrence of a substring in a string
● REPEAT	Repeats a string as many times as specified
● REPLACE	Replaces all occurrences of a substring within a string, with a new substring
● REVERSE	Reverses a string and returns the result
● RIGHT	Extracts a number of characters from a string (starting from right)

MySQL String Functions

Function	Description
MID	Extracts a substring from a string (starting at any position)
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string as many times as specified
REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)

MySQL String Functions

Function	Description
● RPAD	Right-pads a string with another string, to a certain length
● RTRIM	Removes trailing spaces from a string
● SPACE	Returns a string of the specified number of space characters
● STRCMP	Compares two strings
● SUBSTR	Extracts a substring from a string (starting at any position)
● SUBSTRING	Extracts a substring from a string (starting at any position)

MySQL String Functions

Function	Description
• SUBSTRING_INDEX	Returns a substring of a string before a specified number of delimiter occurs
• TRIM	Removes leading and trailing spaces from a string
• UCASE	Converts a string to upper-case
• UPPER	Converts a string to upper-case

MySQL Numeric Functions

Function	Description
● ABS	Returns the absolute value of a number
● ACOS	Returns the arc cosine of a number
● ASIN	Returns the arc sine of a number
● ATAN	Returns the arc tangent of one or two numbers
● ATAN2	Returns the arc tangent of two numbers
● AVG	Returns the average value of an expression

MySQL Numeric Functions

Function	Description
● CEIL	Returns the smallest integer value that is >= to a number
● CEILING	Returns the smallest integer value that is >= to a number
● COS	Returns the cosine of a number
● COT	Returns the cotangent of a number
● COUNT	Returns the number of records returned by a select query
● DEGREES	Converts a value in radians to degrees

MySQL Numeric Functions

Function	Description
• DIV	Used for integer division
• EXP	Returns e raised to the power of a specified number
• FLOOR	Returns the largest integer value that is <= to a number
• GREATEST	Returns the greatest value of the list of arguments
• LEAST	Returns the smallest value of the list of arguments
• LN	Returns the natural logarithm of a number

MySQL Numeric Functions

Function	Description
• LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
• LOG10	Returns the natural logarithm of a number to base 10
• LOG2	Returns the natural logarithm of a number to base 2
• MAX	Returns the maximum value in a set of values
• MIN	Returns the minimum value in a set of values
• MOD	Returns the remainder of a number divided by another number

MySQL Numeric Functions

Function	Description
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
POWER	Returns the value of a number raised to the power of another number
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places

MySQL Numeric Functions

Function	Description
• SIGN	Returns the sign of a number
• SIN	Returns the sine of a number
• SQRT	Returns the square root of a number
• SUM	Calculates the sum of a set of values
• TAN	Returns the tangent of a number
• TRUNCATE	Truncates a number to the specified number of decimal places

DATE & TIME FUNCTIONS

```
SELECT ADDDATE("2017-
06-15",
INTERVAL 10 DAY);
```

ADDDATE() function

```
SELECT ADDTIME("2017-
06-15
09:34:21", "2");
```

ADDTIME() function

```
SELECT CURDATE();
```

CURDATE() Function

```
SELECT CURRENT_DATE()
;
```

CURRENT_DATE()
Function

```
SELECT CURRENT_TIME();
```

CURRENT_TIME()
Function

```
SELECT CURRENT_TIMES
TAMP();
```

CURRENT_TIMESTAMP()
Function

DATE & TIME FUNCTIONS

```
SELECT CURTIME();
```

CURTIME() function

```
SELECT DATE("2017-06-
15");
```

DATE() Function

```
SELECT DATEDIFF
("2017-06-25",
"2017-06-15");
```

DATEDIFF() Function

```
SELECT DATE_ADD("2017
-06-15",
INTERVAL 10 DAY);
```

DATE_ADD() Function

```
SELECT DATE_FORMAT("20
17-06-15", "%Y");
```

DATE_FORMAT() Function

```
SELECT DATE_SUB("201
7-06-15",
INTERVAL 10 DAY);
```

DATE_SUB() Function

DATE & TIME FUNCTIONS

```
SELECT DAY("2017-06-15");
```

DAY() Function

```
SELECT DAYNAME("2017-06-15");
```

DAYNAME() Function

```
SELECT DAYOFMONTH("2017-06-15");
```

DAYOFMONTH() Function

```
SELECT DAYOFWEEK("2017-06-15");
```

DAYOFWEEK() Function

```
SELECT DAYOFYEAR("2017-06-15");
```

DAYOFYEAR() Function

```
SELECT EXTRACT(MONTH FROM "2017-06-15");
```

EXTRACT() Function

DATE & TIME FUNCTIONS

```
SELECT MAKETIME(11,
35, 4);
```

MAKETIME()

```
SELECT MAKEDATE(2017,
3);
```

MAKEDATE()

```
SELECT
LOCALTIMESTAMP();
```

LOCALTIMESTAMP()

```
SELECT LOCALTIME();
```

LOCALTIME()

```
SELECT LAST_DAY("2017-
06-20");
```

LAST_DAY()

```
SELECT HOUR("2017-
06-20 09:34:00");
```

HOUR()

DATE & TIME FUNCTIONS

```
SELECT MAKETIME(11, 35, 4);
```

MAKETIME()

```
SELECT MAKEDATE(2017, 3);
```

MAKEDATE()

```
SELECT LOCALTIMESTAMP();
```

LOCALTIMESTAMP()

```
SELECT LOCALTIME();
```

LOCALTIME()

```
SELECT LAST_DAY("2017-06-20");
```

LAST_DAY()

```
SELECT HOUR("2017-06-20 09:34:00");
```

HOUR()

DATE & TIME FUNCTIONS

```
SELECT
PERIOD_ADD(201703,
5);
```

PERIOD_ADD()

```
SELECT NOW();
```

NOW()

```
SELECT
MONTHNAME("2017-06-
15");
```

MONTHNAME()

```
SELECT MONTH("2017-
06-15");
```

MONTH()

```
ELECT MINUTE("2017-06-
20 09:34:00");
```

MINUTE()

```
SELECT
MICROSECOND("2017-
06-20
09:34:00.000023");
```

MICROSECOND()

DATE & TIME FUNCTIONS

```
SELECT SUBDATE("2017-06-15", INTERVAL 10 DAY);
```

SUBDATE()

```
SELECT STR_TO_DATE("August 10 2017", "%M %d %Y");
```

STR_TO_DATE()

```
SELECT SEC_TO_TIME(1);
```

SEC_TO_TIME()

```
SELECT SECOND("2017-06-20 09:34:00.000023");
```

SECOND()

```
SELECT QUARTER("2017-06-15");
```

QUARTER()

```
SELECT PERIOD_DIFF(201710, 201703);
```

PERIOD_DIFF()

DATE & TIME FUNCTIONS

```
SELECT
TIMESTAMP("2017-07-
23",  "13:10:11");
```

TIMESTAMP()

```
SELECT
TIMEDIFF("13:10:11",
"13:10:10");
```

TIMEDIFF()

```
SELECT
TIME_TO_SEC("19:30:1
0");
```

TIME_TO_SEC()

```
SELECT
TIME_FORMAT("19:30:10
", "%H %i %s");
```

TIME_FORMAT()

```
SELECT SYSDATE();
```

SYSDATE()

```
SELECT
SUBTIME("2017-06-15
10:24:21.000004",
"5.000001");
```

SUBTIME()

DATE & TIME FUNCTIONS

```
SELECT YEARWEEK("2017-06-15");
```

YEARWEEK()

```
SELECT YEAR("2017-06-15");
```

YEAR()

```
SELECT WEEKOFYEAR("2017-06-15");
```

WEEKOFYEAR()

```
SELECT WEEKDAY("2017-06-15");
```

WEEKDAY()

```
SELECT WEEK("2017-06-15");
```

WEEK()

```
SELECT TO_DAYS("2017-06-20");
```

TO_DAYS()

MySQL Date Functions

Function	Description
● ADDDATE	Adds a time/date interval to a date and then returns the date
● ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
● CURDATE	Returns the current date
● CURRENT_DATE	Returns the current date
● CURRENT_TIME	Returns the current time
● CURRENT_TIMESTAMP	Returns the current date and time
● CURTIME	Returns the current time

MySQL Date Functions

Function	Description
● DATE	Extracts the date part from a datetime expression
● DATEDIFF	Returns the number of days between two date values
● DATE_ADD	Adds a time/date interval to a date and then returns the date
● DATE_FORMAT	Formats a date
● DATE_SUB	Subtracts a time/date interval from a date and then returns the date
● DAY	Returns the day of the month for a given date
● DAYNAME	Returns the weekday name for a given date

MySQL Date Functions

Function	Description
● DAYOFMONTH	Returns the day of the month for a given date
● DAYOFWEEK	Returns the weekday index for a given date
● DAYOFYEAR	Returns the day of the year for a given date
● EXTRACT	Extracts a part from a given date
● FROM_DAYS	Returns a date from a numeric datevalue
● HOUR	Returns the hour part for a given date
● LAST_DAY	Extracts the last day of the month for a given date

MySQL Date Functions

Function	Description
● LOCALTIME	Returns the current date and time
● LOCALTIMESTAMP	Returns the current date and time
● MAKEDATE	Creates and returns a date based on a year and a number of days value
● MAKETIME	Creates and returns a time based on an hour, minute, and second value
● MICROSECOND	Returns the microsecond part of a time/datetime
● MINUTE	Returns the minute part of a time/datetime

MySQL Date Functions

Function	Description
● MONTH	Returns the month part for a given date
● MONTHNAME	Returns the name of the month for a given date
● NOW	Returns the current date and time
● PERIOD_ADD	Adds a specified number of months to a period
● PERIOD_DIFF	Returns the difference between two periods
● QUARTER	Returns the quarter of the year for a given date value

MySQL Date Functions

Function	Description
● SECOND	Returns the seconds part of a time/datetime
● SEC_TO_TIME	Returns a time value based on the specified seconds
● STR_TO_DATE	Returns a date based on a string and a format
● SUBDATE	Subtracts a time/date interval from a date and then returns the date
● SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime
● SYSDATE	Returns the current date and time

MySQL Date Functions

Function	Description
● TIME	Extracts the time part from a given time/datetime
● TIME_FORMAT	Formats a time by a specified format
● TIME_TO_SEC	Converts a time value into seconds
● TIMEDIFF	Returns the difference between two time/datetime expressions
● TIMESTAMP	Returns a datetime value based on a date or datetime value
● TO_DAYS	Returns the number of days between a date and date "0000-00-00"

MySQL Date Functions

Function	Description
● WEEK	Returns the week number for a given date
● WEEKDAY	Returns the weekday number for a given date
● WEEKOFYEAR	Returns the week number for a given date
● YEAR	Returns the year part for a given date
● YEARWEEK	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
BIN	Returns a binary representation of a number
BINARY	Converts a value to a binary string
CASE	Goes through conditions and return a value when the first condition is met
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONNECTION_ID	Returns the unique connection ID for the current connection
CONV	Converts a number from one numeric base system to another

MySQL Advanced Functions

Function	Description
● CONVERT	Converts a value into the specified datatype or character set
● CURRENT_USER	Returns the user name and host name for the MySQL account that the server used to authenticate the current client
● DATABASE	Returns the name of the current database
● IF	Returns a value if a condition is TRUE, or another value if a condition is FALSE
● IFNULL	Return a specified value if the expression is NULL, otherwise return the expression

MySQL Advanced Functions

Function	Description
● ISNULL	Returns 1 or 0 depending on whether an expression is NULL
● LAST_INSERT_ID	Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table
● NULLIF	Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned
● SESSION_USER	Returns the current MySQL user name and host name

MySQL Advanced Functions

Function	Description
● SYSTEM_USER	Returns the current MySQL user name and host name
● USER	Returns the current MySQL user name and host name
● VERSION	Returns the current version of the MySQL database

Reference

- https://www.w3schools.com/
- https://www.tutorialspoint.com/
- https://www.guru99.com/
- https://dev.mysql.com/

THANK YOU

FOR YOUR TIME