MySQL II

Objectives

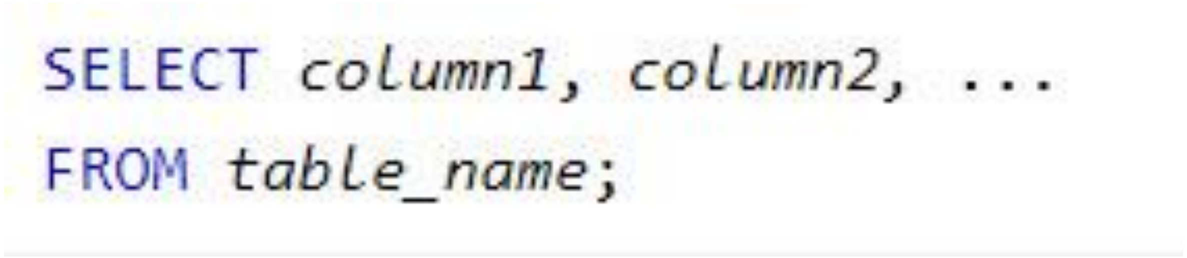
- Introduction to SQL
- SQL Statements
- Normalization

Introduction to SQL

- SQL is a standard language for accessing and manipulating databases.
- What is SQL?
  - SQL stands for Structured Query Language
  - SQL lets you access and manipulate databases
  - SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- To build a web site that shows data from a database, you will need
  - An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
  - To use a server-side scripting language, like PHP or ASP
  - To use SQL to get the data you want
  - To use HTML / CSS to style the page

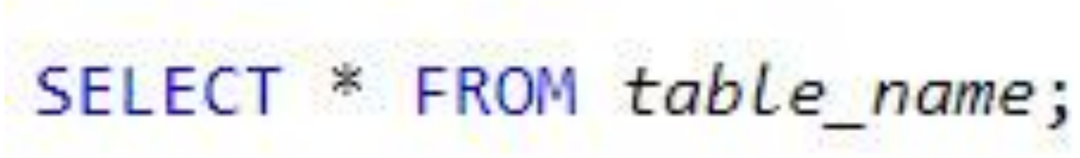
SQL SELECT

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



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

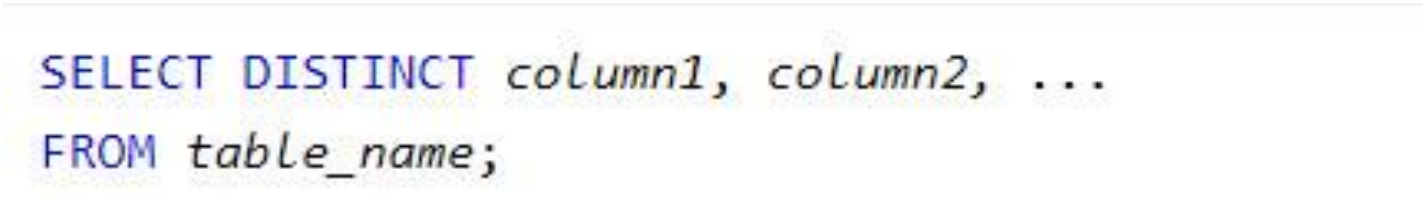
- column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax



```
SELECT * FROM table_name;
```

SQL SELECT DISTINCT

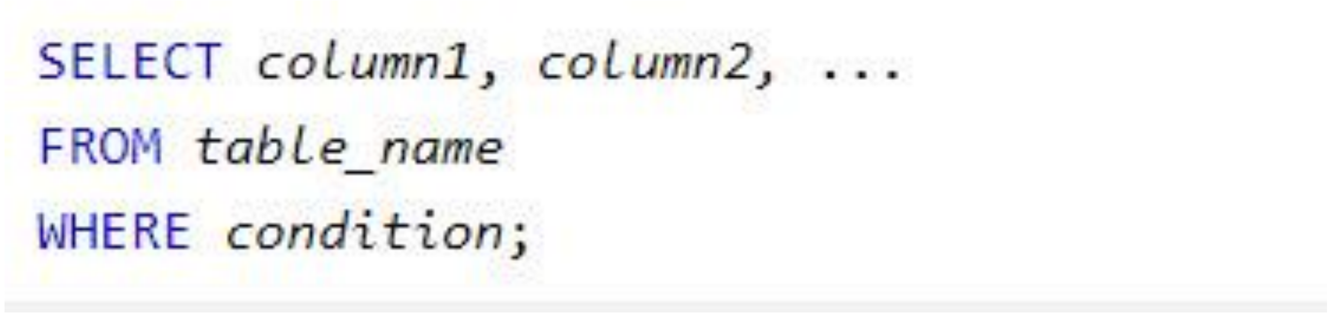
- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.
- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Syntax



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

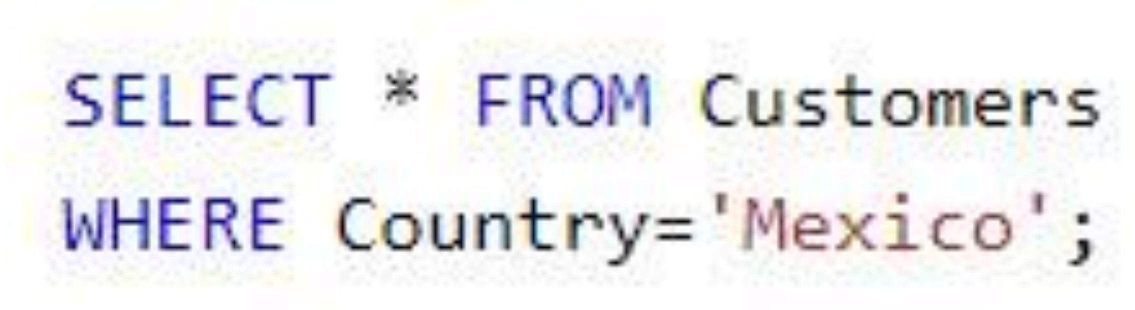
SQL WHERE Clause

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.
- The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE statement, etc.
- Syntax



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

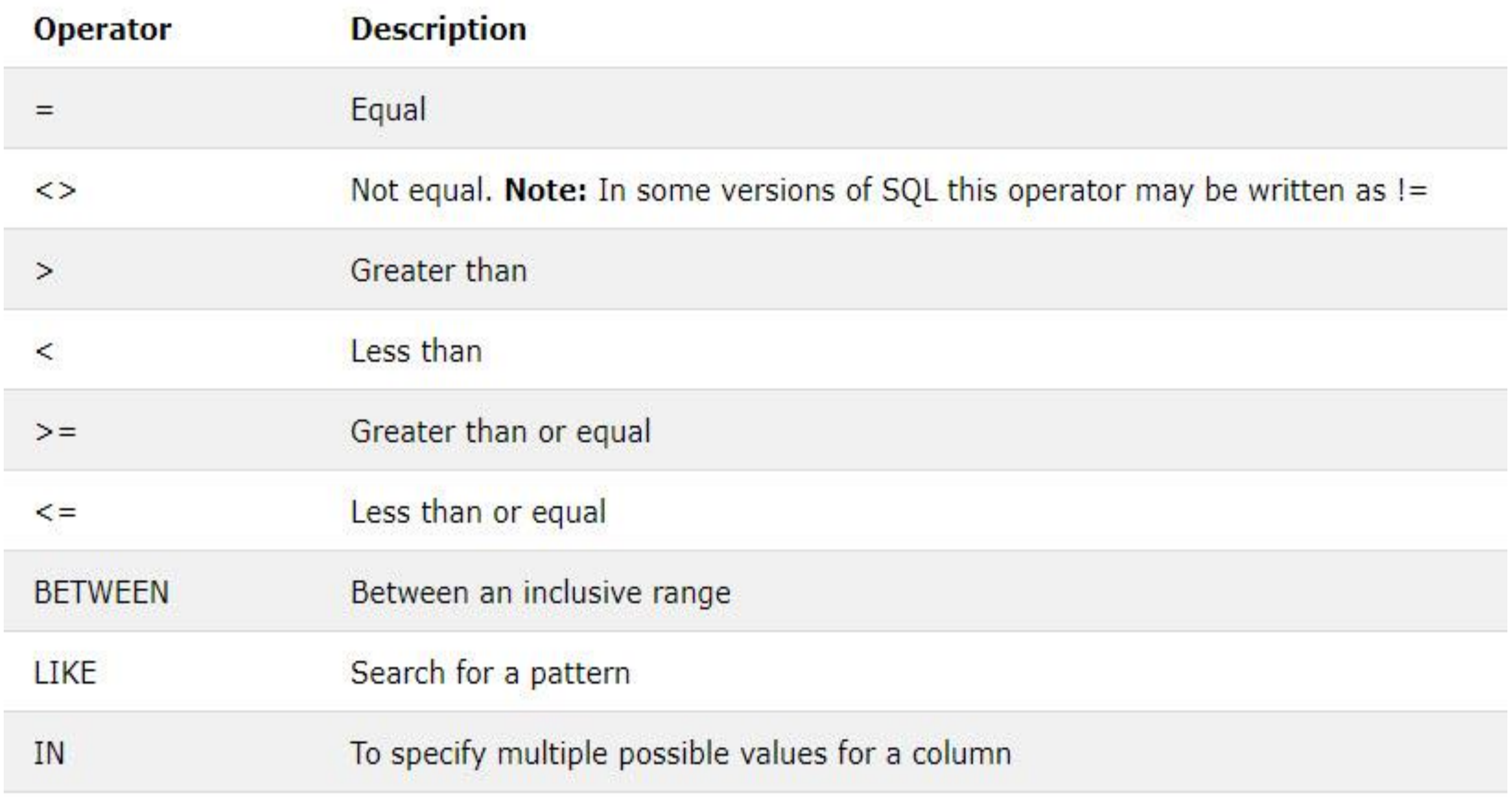
- Example



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

WHERE Clause

- The following operators can be used in the WHERE clause



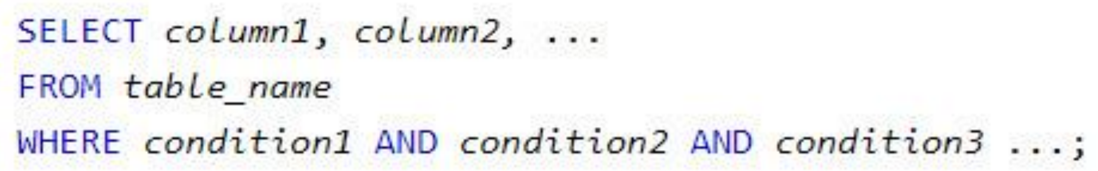
Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

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 is 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.

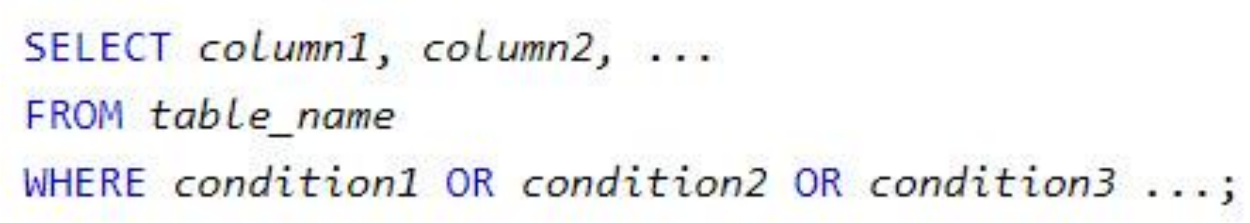
AND, OR and NOT Operators

- AND Syntax



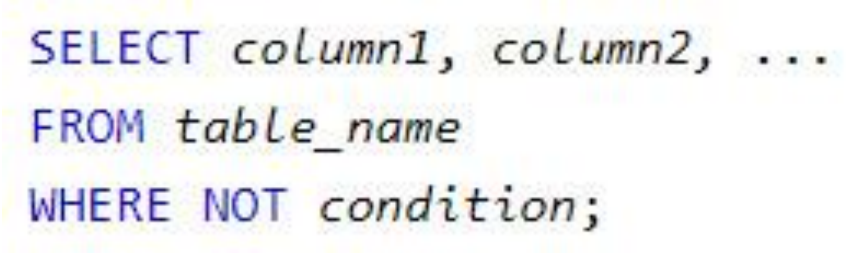
```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

- OR Syntax



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

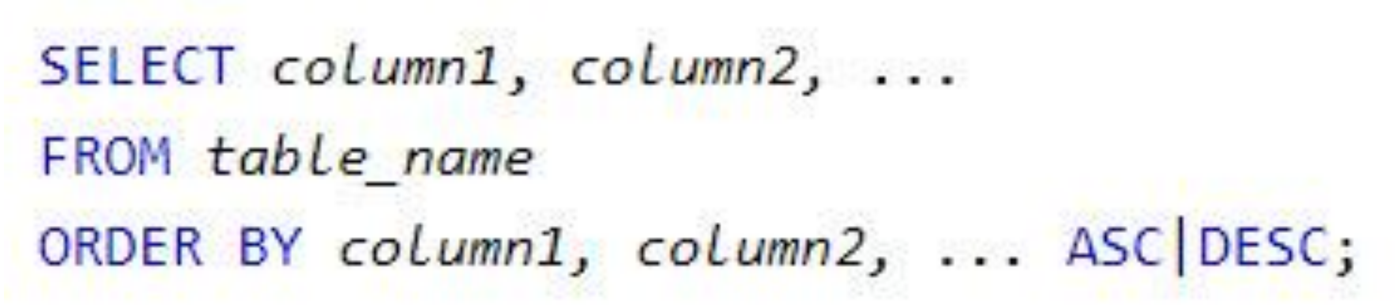
- NOT Syntax



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

SQL ORDER BY

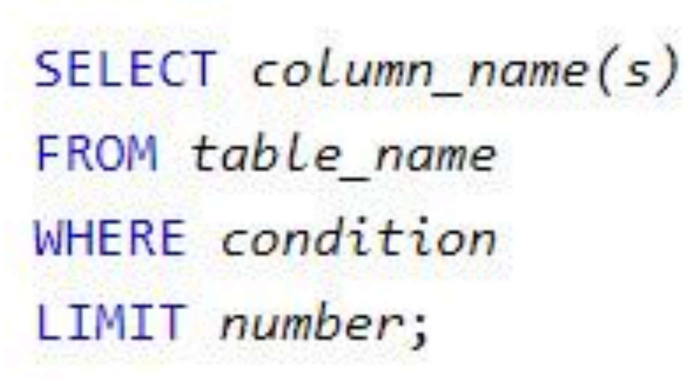
- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.
- Syntax



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

SELECT TOP Clause

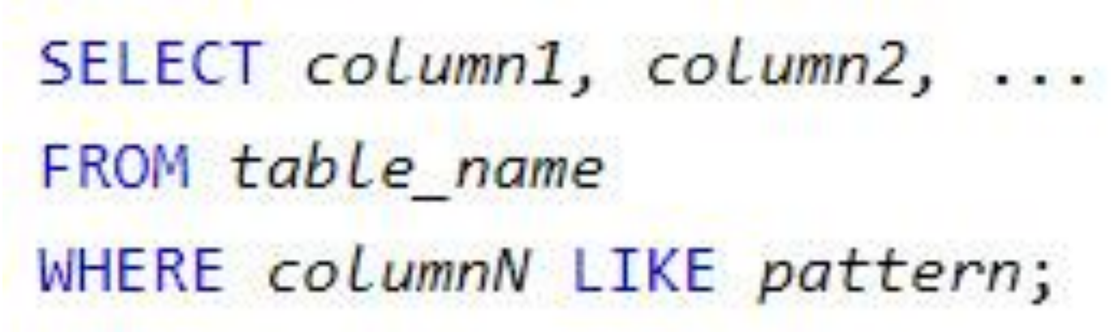
- The SELECT TOP clause is used to specify the number of records to return.
- The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact on performance.
- **Note:** Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses ROWNUM.
- MySQL Syntax



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

SQL LIKE Operator

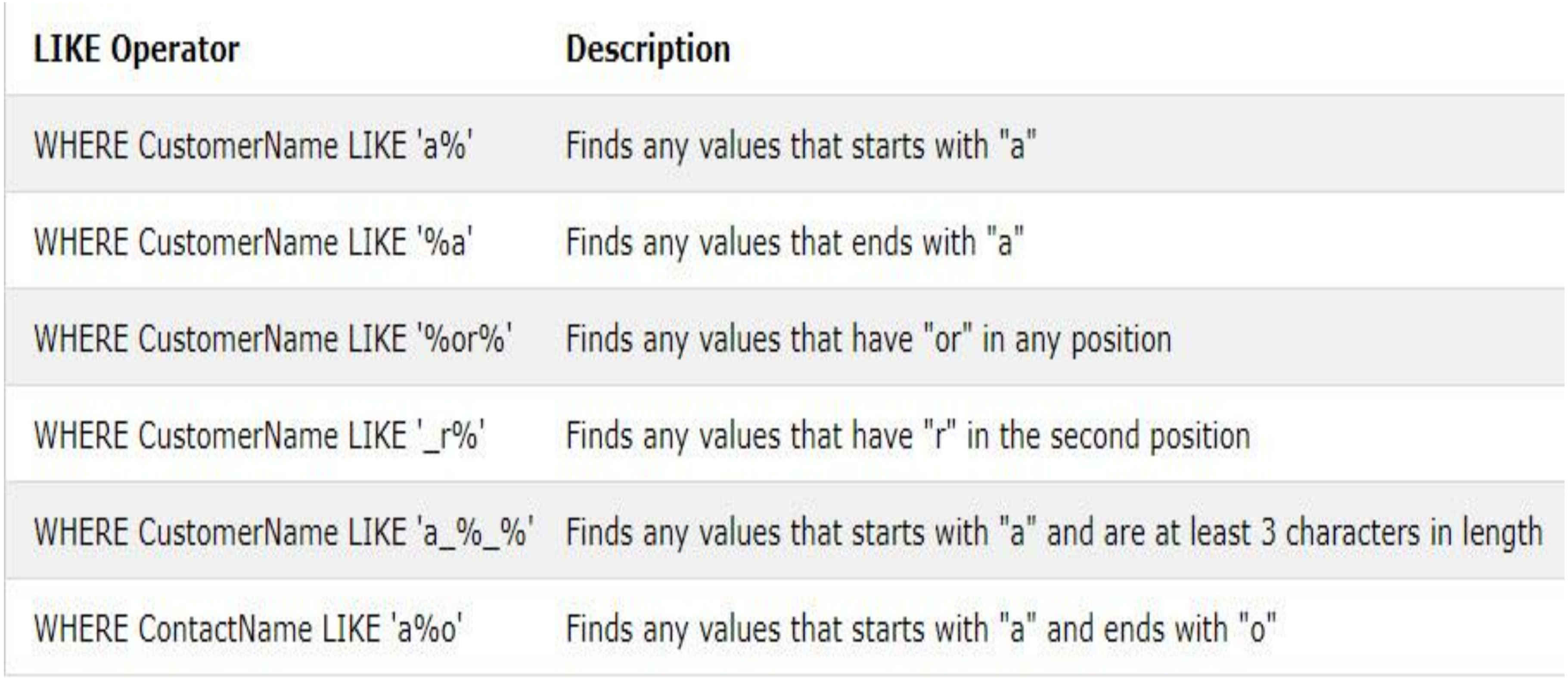
- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards used in conjunction with the LIKE operator:
  - % - The percent sign represents zero, one, or multiple characters
  - _ - The underscore represents a single character
- You can also combine any number of conditions using AND or OR operators.
- Syntax



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

Example

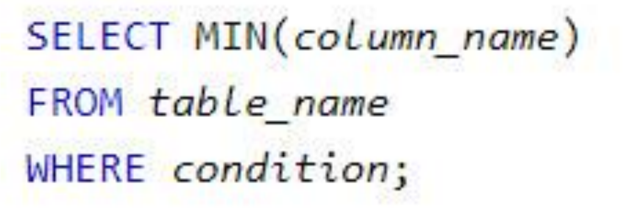
- Here are some examples showing different LIKE operators with '%' and '_' wildcards:



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends 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 starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

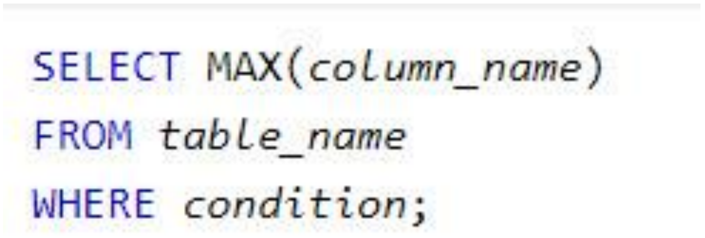
SQL MIN() and MAX()

- 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;
```

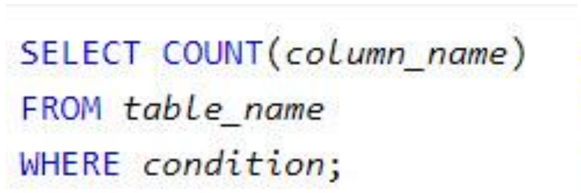
- MAX() Syntax



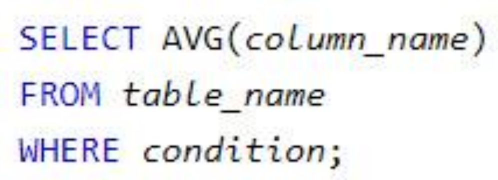
```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

COUNT(),AVG(),SUM()

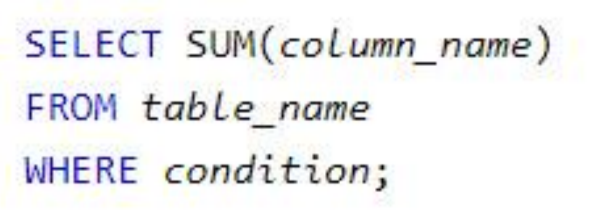
- The COUNT() function returns the number of rows that matches a specified criteria.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.
- Syntax



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



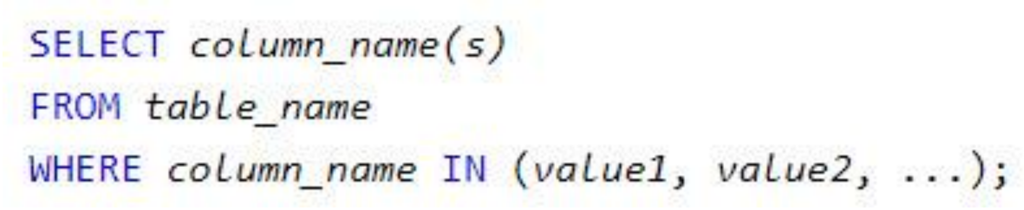
```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```



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

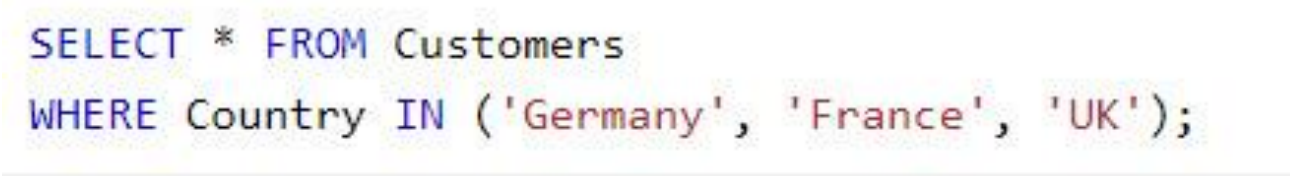
SQL IN Operator

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



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

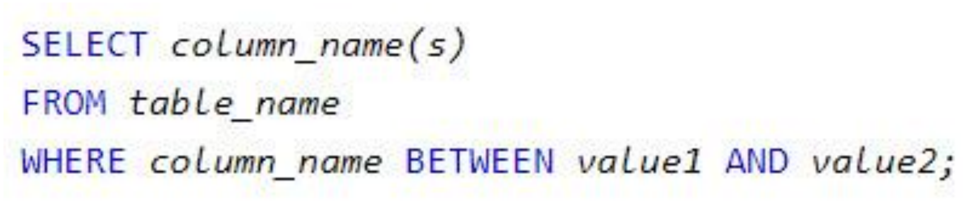
- Example



```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

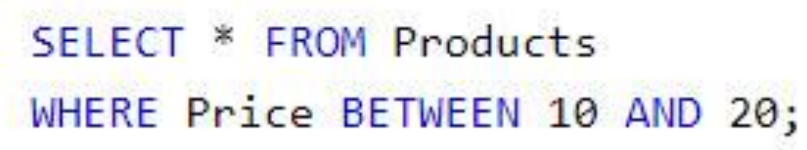
BETWEEN Operator

- 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.
- Syntax



```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

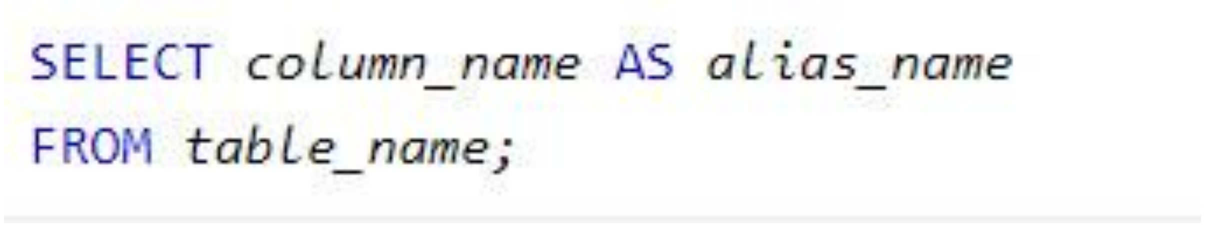
- Example



```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

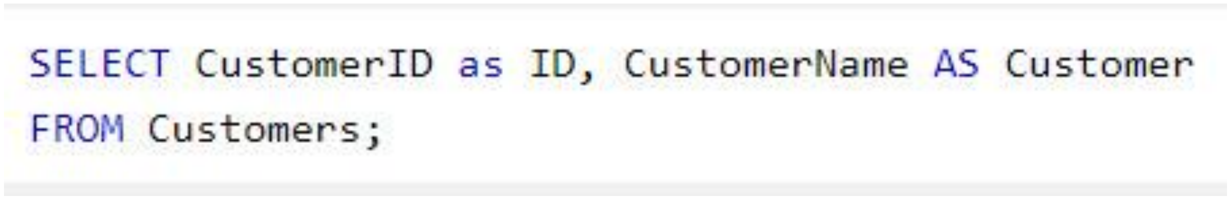
SQL Aliases

- SQL 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 the query.
  - **Note:** It requires double quotation marks or square brackets if the alias name contains spaces
- Syntax



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

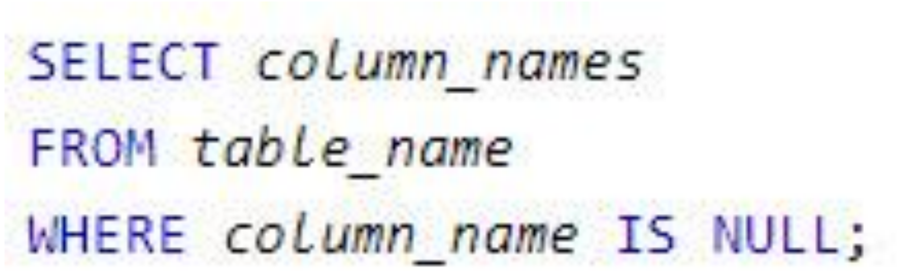
- Example



```
SELECT CustomerID as ID, CustomerName AS Customer
FROM Customers;
```

SQL 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.
  - It is very important to understand that a NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation
- 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.
- Syntax



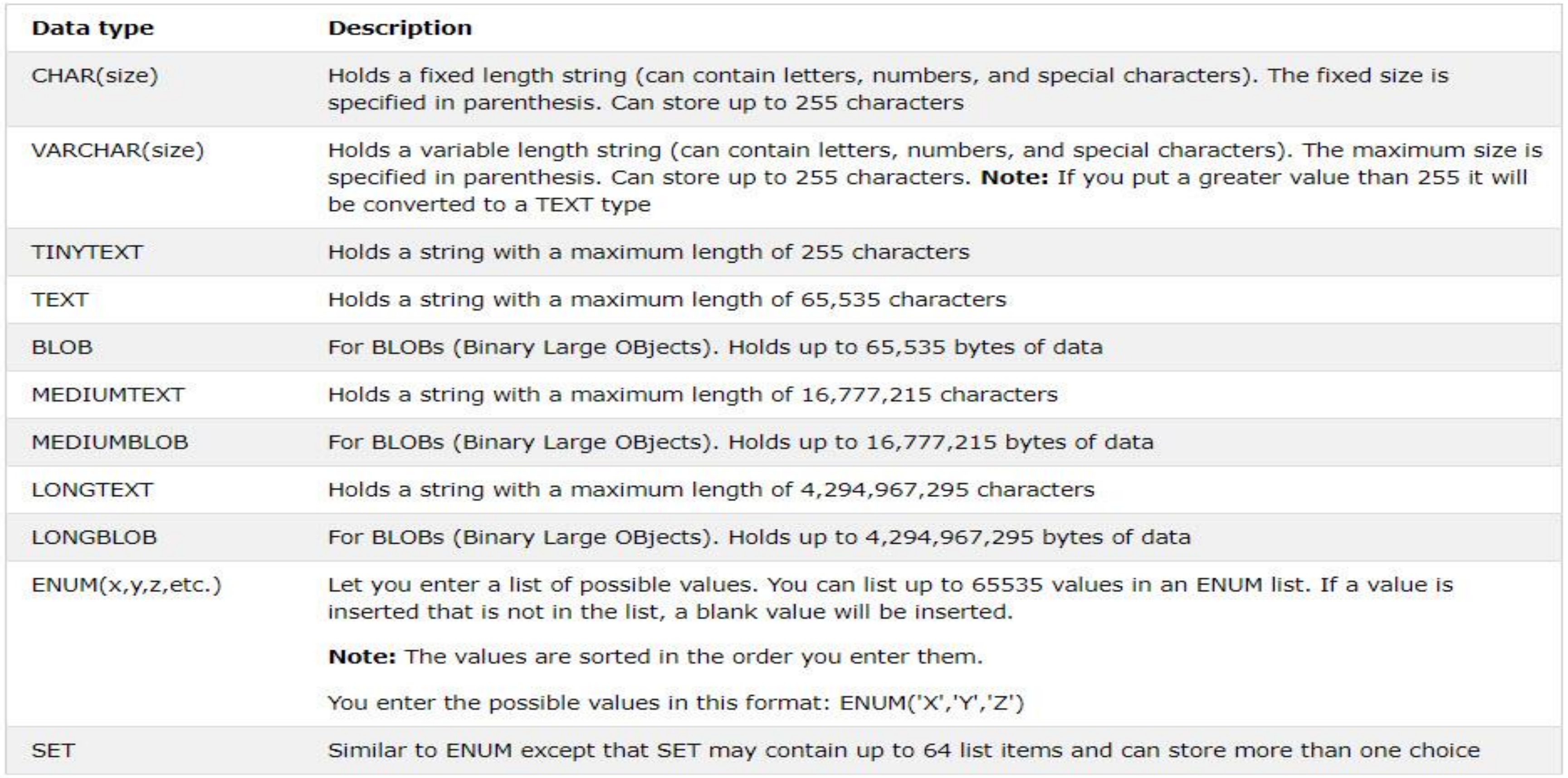
```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

SQL Data Types

- A data type defines what kind of value a column can hold: integer data, character data, monetary data, date and time data, binary strings, and so on.
- Each column in a database table is required to have a name and a data type.
- An SQL developer must decide what type of data that will be stored inside each column when creating a table.
- The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.
  - Data types might have different names in different database. And even if the name is the same, the size and other details may be different

MySQL Data Types

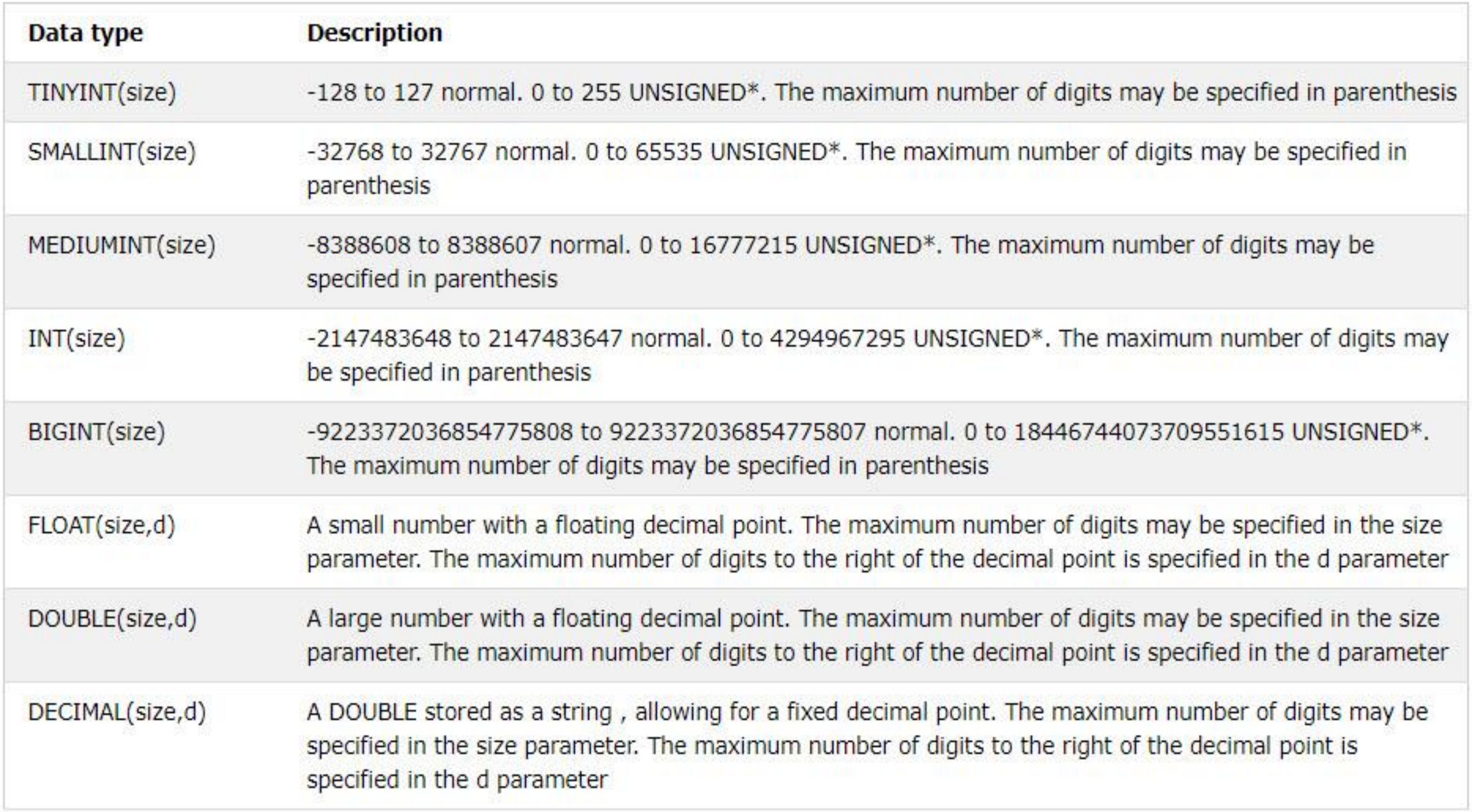
- In MySQL there are three main data types: text, number, and date.
- Text data types



Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	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
ENUM(x,y,z,etc.)	Let you enter 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. **Note:** The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

MySQL Data Types

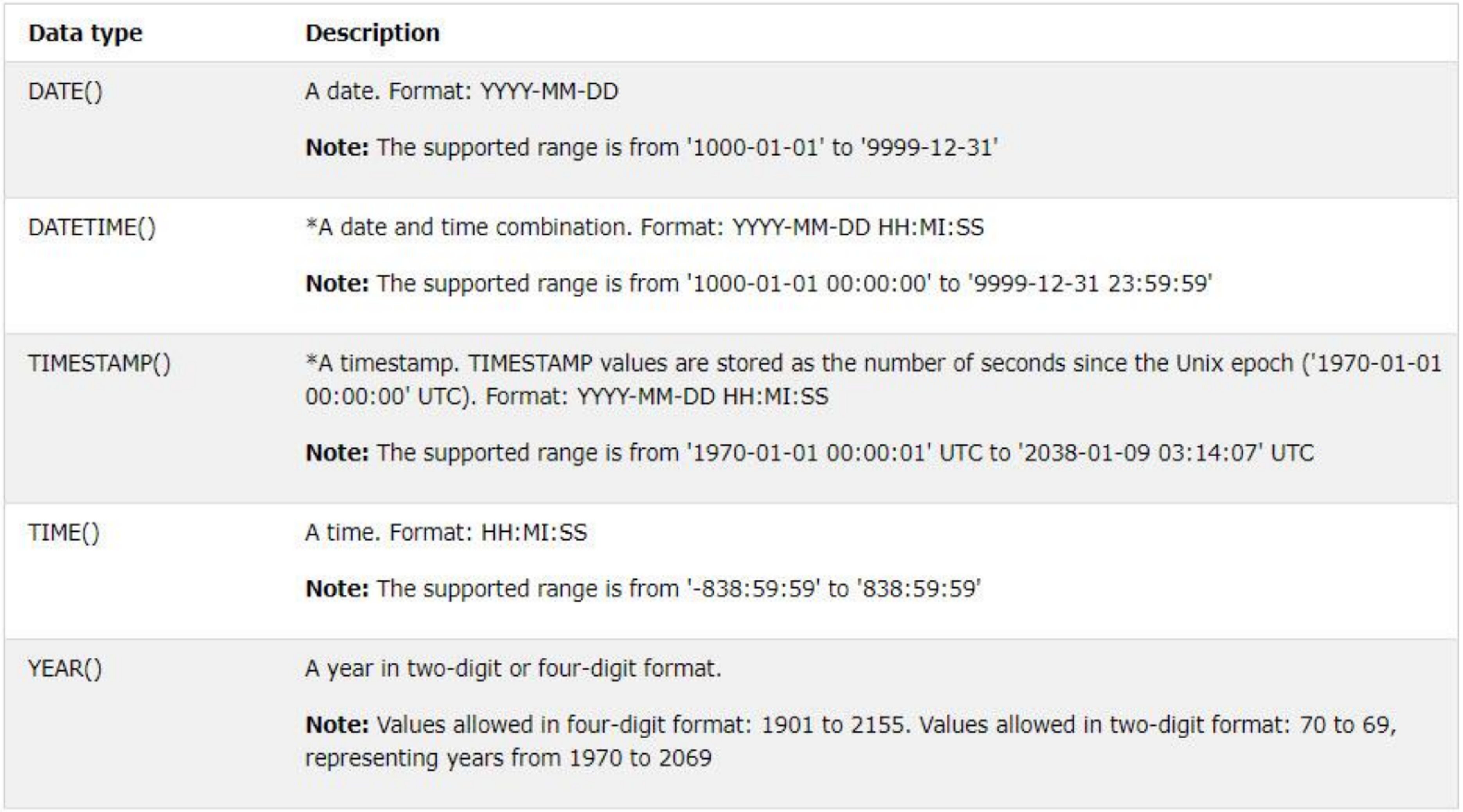
- Number data types



Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

MySQL Data Types

- Date data types



Data type	Description
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*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:MI:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

Demo

- Teacher demo SQL statements for students

Understanding Data Redundancy

- Redundancy:
  - Increases the time involved in updating, adding, and deleting data.
  - Increases the utilization of disk space and hence, disk I/O increases.
- Redundancy can, therefore, lead to:
  - Insertion, modification, and deletion of data, which may cause inconsistencies.
  - Errors, which are more likely to occur when facts are repeated.
  - Unnecessary utilization of extra disk space.

Understanding Data Redundancy

- The STUDENT table contains the values for each attribute, as shown in the following diagram

STUDENT ID	STUDENT NAME	....	STUDENT SEMESTER	STUDENT TEST1	STUDENT TEST2
001	Mary	....	SEM-1	40	65
001	Mary	....	SEM-2	56	48
002	Jake	....	SEM-1	93	84
002	Jake	....	SEM-2	85	90

The details of the students, such as STUDENTID and STUDENTNAME are repeated while recording marks of different semesters.

Definition of Normalization

- Normalization:
  - Is a method of breaking down complex table structures into simple table structures by using certain rules.
  - Has the following benefits:
    - It helps in maintaining data integrity.
    - It helps in simplifying the structure of tables, therefore, making a database more compact.
    - It helps in reducing the null values, which reduces the complexity of data operations.

Definition of Normalization

- Some rules that should be followed to achieve a good database design are:
  - Each table should have an identifier.
  - Each table should store data for a single type of entity.
  - Columns that accept NULL values should be avoided.
  - The repetition of values or columns should be avoided.

Definition of Normalization

- Normalization results in the formation of tables that satisfy certain normal forms.
- The normal forms are used to remove various types of abnormalities and inconsistencies from the database.
- The most important and widely used normal forms are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)

Definition of Normalization

- First Normal Form (1NF):
  - A table is said to be in 1NF when each cell of the table contains precisely one value.
  - The guidelines for converting a table into 1NF are:
    - Place the related data values in a table. Further, define similar data values with the column name.
    - There should be no repeating group in the table.
    - Every table must have a unique primary key.

Definition of Normalization

- Second Normal Form (2NF):
  - A table is said to be in 2NF when:
    - It is in the 1NF, and
    - No partial dependency exists between non-key attributes and key attributes.
  - The guidelines for converting a table into 2NF are:
    - Find and remove attributes that are functionally dependent on only a part of the key and not on the whole key. Place them in a different table.
    - Group the remaining attributes.

Definition of Normalization

- Third Normal Form (3NF):
  - A relation is said to be in 3NF if and only if:
    - It is in 2NF, and
    - No transitive (indirect) dependency exists between non-key attributes and key attributes.
  - The guidelines for converting a table into 3NF are:
    - Find and remove non-key attributes that are functionally dependent on attributes that are not the primary key. Place them in a different table.
    - Group the remaining attributes.

Definition of Normalization

- Boyce-Codd Normal Form (BCNF):
  - The original definition of 3NF was not sufficient in some situations. It was not satisfactory for the tables:
    - That had multiple candidate keys.
    - Where the multiple candidate keys were composite.
    - Where the multiple candidate keys overlapped (had at least one attribute in common).
  - The guidelines for converting a table into BCNF are:
    - Find and remove the overlapping candidate keys. Place the part of the candidate key and the attribute it is functionally dependent on, in a different table.
    - Group the remaining items into a table.

Summary

- Introduction to SQL
- SQL Statements
- Normalization