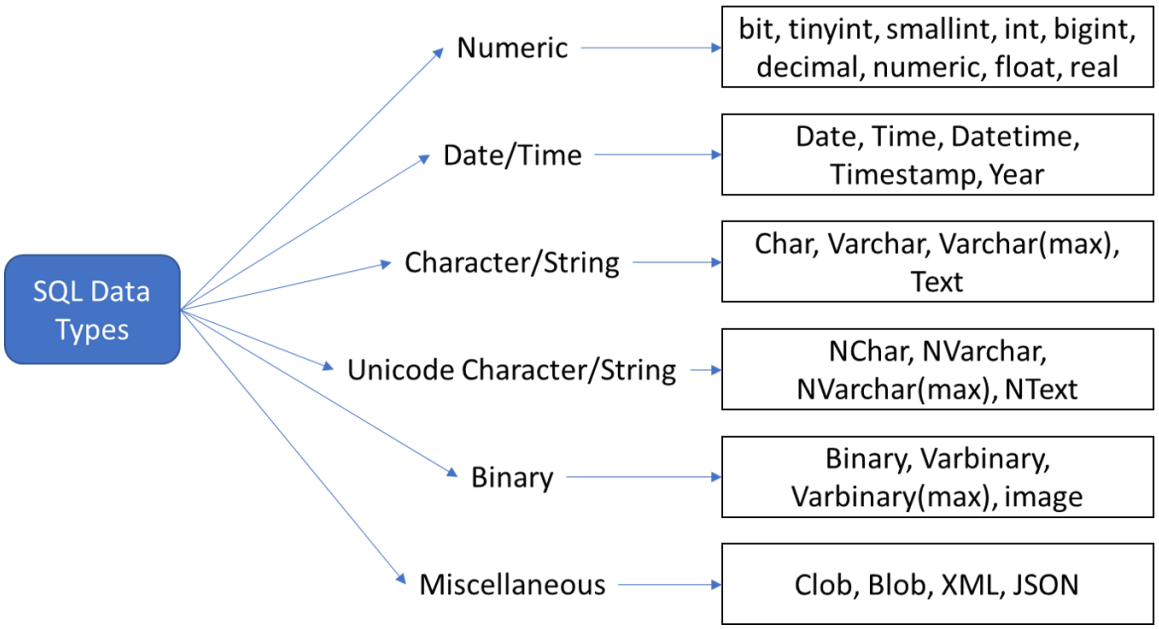
SQL Data Types :

The table `products` stores information about products with columns for `id`, `name`, `price`, and `quantity`.

```
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY, -- Unique identifier for each product
    name VARCHAR(100) NOT NULL,        -- Name of the product (up to 100 cha
racters)
    price DECIMAL(10, 2) NOT NULL,     -- Price with 2 decimal places
    quantity INT DEFAULT 0             -- Quantity with a default value of 0
);
```

Data Types :



String :

Types of String Data Types in SQL

1. CHAR

- **Description**: Stores fixed-length strings. If the stored string is shorter than the defined size, it is padded with spaces.
- **Usage**: When the data always has a fixed length.
- **Size**: Up to 255 characters.

Example:

```
sql
Copy code
CREATE TABLE Users (
    Username CHAR(10)
);
-- 'Alex' is stored as 'Alex      ' (padded with spaces)
```

---

2. VARCHAR

- **Description**: Stores variable-length strings. It only uses as much space as the actual string length, plus 1 or 2 bytes for length storage.
- **Usage**: When the data length varies.
- **Size**: Up to 65,535 characters (depending on row size and database).

Example:

```
sql
Copy code
CREATE TABLE Users (
    Email VARCHAR(100)
);
-- 'Alex' is stored as 'Alex' (no padding)
```

ENUM

- **Description**: Stores a predefined list of string values. It allows only one value from the list.
- **Usage**: When the column should have a restricted set of string values.
- **Size**: Up to 65,535 distinct values.

Example:

```
sql
Copy code
CREATE TABLE Products (
    Size ENUM('Small', 'Medium', 'Large')
```

```
);
-- Size can only be 'Small', 'Medium', or 'Large'.
```

INTEGER :

1. TINYINT

- **Description**: A very small integer.
- **Range**:
  - **Signed**: -128 to 127
  - **Unsigned**: 0 to 255
- **Size**: 1 byte
- **Usage**: For very small numeric values like flags, status codes, or boolean-like values.

Example:

```
CREATE TABLE Example (
    Status TINYINT
);
-- Status can store values like -1, 0, 1
```

---

2. SMALLINT

- **Description**: A small integer.
- **Range**:
  - **Signed**: -32,768 to 32,767
  - **Unsigned**: 0 to 65,535
- **Size**: 2 bytes
- **Usage**: For slightly larger numbers, such as short identifiers or small counters.

Example:

```
CREATE TABLE Example (
    Age SMALLINT
);
```

```
-- Age can store values like 25, 30, 45
```

3. MEDIUMINT

- **Description**: A medium-sized integer.
- **Range**:
  - **Signed**: -8,388,608 to 8,388,607
  - **Unsigned**: 0 to 16,777,215
- **Size**: 3 bytes
- **Usage**: For numbers that are larger than `SMALLINT` but smaller than `INT`.

Example:

```
CREATE TABLE Example (
    Population MEDIUMINT
);
-- Population can store values like 1,000,
```

4. INT (INTEGER)

- **Description**: A standard integer data type.
- **Range**:
  - **Signed**: -2,147,483,648 to 2,147,483,647
  - **Unsigned**: 0 to 4,294,967,295
- **Size**: 4 bytes
- **Usage**: For most general-purpose numeric fields, like IDs or counters.

Example:

```
CREATE TABLE Example (
    UserID INT
);
-- UserID can store large numbers for identification
```

5. BIGINT

- **Description**: A large integer.
- **Range**:
  - **Signed**: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
  - **Unsigned**: 0 to 18,446,744,073,709,551,615
- **Size**: 8 bytes
- **Usage**: For very large numbers, like financial data, scientific computations, or large counters.

Example:

```
CREATE TABLE Example (
    NationalDebt BIGINT
);
-- NationalDebt can store massive numbers
```

Comparison Table of Integer Data Types

Data Type	Size	Signed Range	Unsigned Range	Use Case
TINYINT	1 byte	-128 to 127	0 to 255	Flags, small numeric values
SMALLINT	2 bytes	-32,768 to 32,767	0 to 65,535	Small counters, short IDs
MEDIUMINT	3 bytes	-8,388,608 to 8,388,607	0 to 16,777,215	Medium-sized values like population
INT	4 bytes	-2,147,483,648 to 2,147,483,647	0 to 4,294,967,295	Most common for IDs or counters
BIGINT	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615	Large numbers like financial data

Choosing the Right Integer Data Type

- Use **TINYINT** for small values such as binary flags or status indicators.
- Use **SMALLINT** for small-range numeric data such as age or short counters.
- Use **MEDIUMINT** when values are expected to fall between the ranges of `SMALLINT` and `INT`.
- Use **INT** for most general-purpose numeric data.

- Use **BIGINT** for very large values, such as global counters, financial data, or scientific measurements.

Concepts :

- **Precision**:

  Precision is the **total number of significant digits** that a numeric data type can store.

  - It includes all digits both **before and after the decimal point**.
  - For example, in `45.3343`, the **precision** is `5` because there are 5 total digits.

- **Scale**:

  Scale is the **number of digits that can appear after the decimal point** in a numeric value.

  - For example, in `45.3343`, the **scale** is `4` because there are 4 digits after the decimal.

For `45.3343` :

- **Precision**: Total number of digits in the number.Example: `45.3343` has **5 digits** (2 before the decimal + 3 after the decimal).
- **Scale**: Number of digits to the right of the decimal point.Example: `45.3343` has a **scale of 4**.
-

1. Fixed-Point Data Types

Fixed-point numbers are used when precise decimal values are required. They store numbers with an exact number of digits before and after the decimal point. These are commonly used for financial calculations.

Fixed-Point Data Types in SQL

1. `NUMERIC(p, s)` **or DECIMAL(p, s)`**
   - **Description**: Used to store exact numeric values with fixed precision.
   - **Precision (`p`)**: Total number of digits (both before and after the decimal point).
   - **Scale (`s`)**: Number of digits after the decimal point.
   - **Storage**: Depends on the precision and scale.

Examples

Table with Fixed-Point Values:

```
sql
Copy code
CREATE TABLE FixedPointExample (
    Price DECIMAL(10, 2),  -- Total 10 digits, 2 digits after the decimal po
int
    InterestRate NUMERIC(5, 3)  -- Total 5 digits, 3 digits after the decima
l point
);
```

Inserting Data:

```
sql
Copy code
INSERT INTO FixedPointExample (Price, InterestRate)
VALUES
(1000.50, 5.123),   -- Valid: Matches the precision and scale
(12345.67, 0.045);  -- Valid: Matches the precision and scale
```

Output:

Price	InterestRate

1000.50	5.123
12345.67	0.045

2. Floating-Point Data Types

Floating-point numbers are used when you need to store very large or very small numbers, and precision can be approximate. These are typically used for scientific calculations or measurements.

Floating-Point Data Types in SQL

1. `FLOAT(p)`
   - **Description**: Approximate numeric data type. Precision depends on the hardware.
   - **Precision (`p`)**: Number of binary digits (not decimal digits) for the mantissa. The actual precision in decimal digits is platform-dependent.
   - **Storage**: Varies depending on precision.
2. `REAL`
   - Alias for `FLOAT` in some SQL implementations.
3. `DOUBLE` **or** `DOUBLE PRECISION`
   - **Description**: Similar to `FLOAT`, but allows for double the precision.

Examples

Table with Floating-Point Values:

```
sql
Copy code
CREATE TABLE FloatingPointExample (
    Temperature FLOAT(24),  -- Approximate precision with single-precision f
loat
    Distance DOUBLE         -- Double-precision float
);
```

Inserting Data:

```
sql
Copy code
INSERT INTO FloatingPointExample (Temperature, Distance)
VALUES
(36.6, 12345.67890),  -- Stores approximate value
```

```
 (98.123456, 0.00005678);
```

Output:

Temperature	Distance
36.6	12345.6789
98.12346	5.678E-05

Comparison Between Fixed-Point and Floating-Point

Feature	**Fixed-Point (`DECIMAL` or `NUMERIC`)**	**Floating-Point (`FLOAT`, `REAL`, `DOUBLE`)**
Precision	Exact and user-defined	Approximate, hardware-dependent
Use Case	Financial data, precise calculations	Scientific data, very large or small values
Performance	Slower due to precision guarantee	Faster for large datasets
Range	Limited by precision and scale	Much larger range
Storage	Fixed size, depends on precision and scale	Varies depending on precision and type

Choosing the Right Data Type

- Use **fixed-point (`DECIMAL` or `NUMERIC`)** when precision is critical, such as for prices, interest rates, and financial data.
- Use **floating-point (`FLOAT`, `DOUBLE`)** when working with scientific or approximate values, such as temperatures, distances, or measurements.

Other DataTypes:

Comparison of Date/Time Data Types

Data Type	**Description**	**Storage**	**Example**
DATE	Only date	3 bytes	`2024-12-11`
TIME	Only time	3 bytes	`14:30:00`
DATETIME	Date and time	8 bytes	`2024-12-11 14:30:00`

TIMESTAMP	Date and time with auto TZ	4 bytes	`2024-12-11 12:00:00`
YEAR	Year only	1 byte	`2024`
INTERVAL	Duration/difference	Varies	`INTERVAL '5 DAYS'`