



Introduction to DBMS

Objectives

01

DBMS & RDMS

Introduction to
DBMS and
RDBMS

02

Key Attributes

Understand the
key attributes of
DBMS

03

SQL

Introduction to
SQL & Data
types

04

SQL Commands

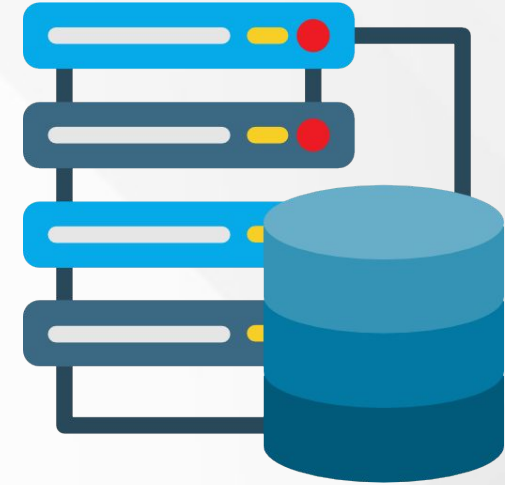
Understand SQL
Commands like:
DDL, DML, DQL,
SELECT
Statements

Introduction to DBMS and RDBMS



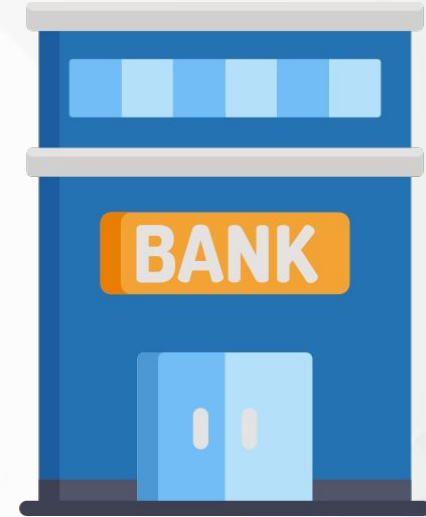
What is DBMS?

- DBMS stands for **Database Management Systems**.
- These are systems that **store, retrieve** and **manipulate** data.
- DBMS is developed to handle **large** amounts of **data**.



Why DBMS?

- Consider a bank that maintains customer's account details, employee details, bank device details, etc.
- This details needs to be stored in such a way that it can be **added**, **deleted**, **updated** and **retrieved** from one place.
- DBMS is a software designed for this type of operations.

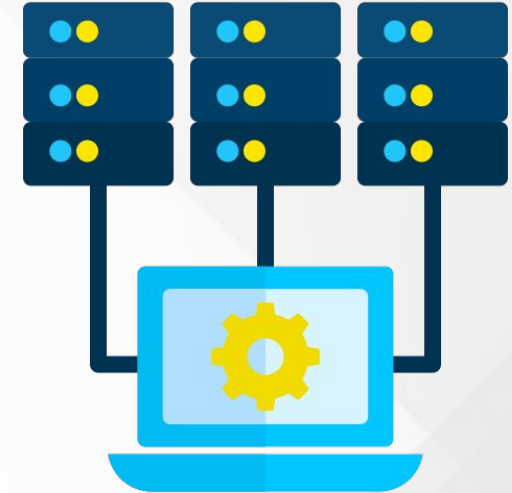




RDBMS

What is RDBMS?

- RDBMS stands for Relational Database Management Systems.
- RDBMS allows to **store, retrieve** or **manipulate** data, but in a more **efficient** way than DBMS.
- Apart from rows and columns the RDBMS table has following components:
 - Domain
 - Instance
 - Schema
 - Keys



A Database Table

- A database consists of one or more tables.
- A table is the most significant component in an RDBMS.
- It is where all data is stored.
- It constitutes of **rows & columns**.
- Each **column represents attributes** of an entity.

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

A Record in a Table

- A record/tuple is a row in a table.
- Each record contains all the data about a certain item, such as a person or a product.

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

A Column in a Table

- In a table, each column represents an attribute.
- This provides a single piece of data regarding the characteristic.
- For instance, a customer's last name.

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28



Keys

- A key is a **data item** (a column or a combination of columns) that allows a table record to be **uniquely identified**.
- It's used to get a single record or a group of records from a table.
- Keys may also be used to provide a variety of helpful limitations.
- A unique key restriction, for example, may assist you avoid entering the same value again.



Keys

A database supports various types of keys. Some of them are:

- Candidate key
- Primary key
- Foreign key
- Unique key
- Alternate key



Keys

- **Candidate identifier**: An attribute (column) of a group of attributes that identifies a record in a unique way.
- In a client transaction database, for example, Customer ID + Store ID + Location ID.
- **Primary key**: Uniquely identifies each record in a table; it must never be the same for two entries in the same table.
- Customer ID, for example, is a unique identifier for a customer in a customer table.



Candidate Key

- Candidate Key can be any column or a combination of columns that can qualify as unique key in database.
- There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as Primary Key.

Primary Key

- A Primary Key is a column or a combination of columns that uniquely identify a record.
- There is only one Primary Key in a table.

Keys

- A **foreign key** is one of a table's columns that points to **the primary key of another table**.
- CustomerID in the customer transaction database, for example, corresponds to CustomerID in the customer tables.



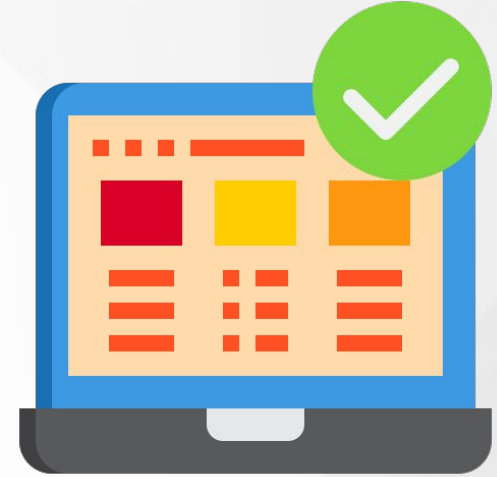
Keys

- A candidate key that is not deemed a main key is referred to as an **alternative key**.
- In a shop information table, for example, store name + store location.
- A unique key is a characteristic or a group of properties that allows a record in a database to be uniquely identified. This is comparable to a main key, which may be empty.
- **Store Name**, for example, in a store information table.



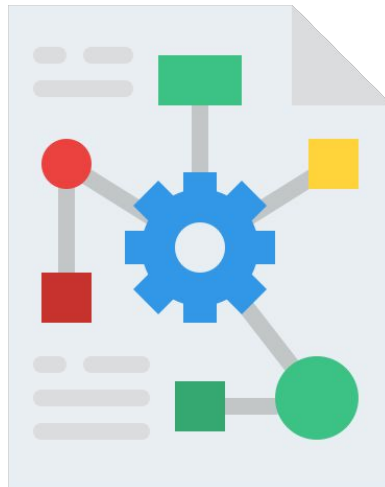
Domain

- A domain is a collection of possible **values for an attribute**.
- Any value **outside** of an attribute's **domain** would be **rejected**.
- For example, if the domain of the field is integer, the field "account no" in a bank customer database will only take integer entries.
- You may also apply limitations to the attributes in addition to the data type.
- Domain Requirements are a kind of constraint that combines many constraints.



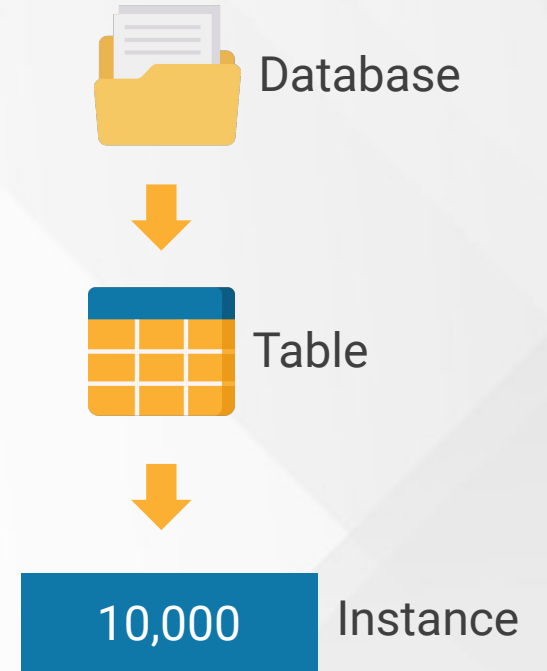
Schemas

- A database schema is a blueprint that depicts the database's logical perspective.
- It establishes the link between tables.
- Physical Structure: This is the most basic kind of database schema. This level describes how data is saved in physical storage.
- The logical restrictions (design) that must be imposed to the data stored are referred to as a logical schema.



Instance

- Consider the 'customer' table in the database 'bank,' which contains 10,000 entries; the database instance at this point is 10,000.
- Assume we're going to add 1000 additional entries to the same table the next day.
- As a result, the database instance will be 11,000 tomorrow.



Instance

- A table in a relational database management system (RDBMS) undergoes a lot of modifications over time.
- In parallel, data is entered, altered, and removed.
- The data saved in a database at a certain point in time is referred to as an **instance**.

Note

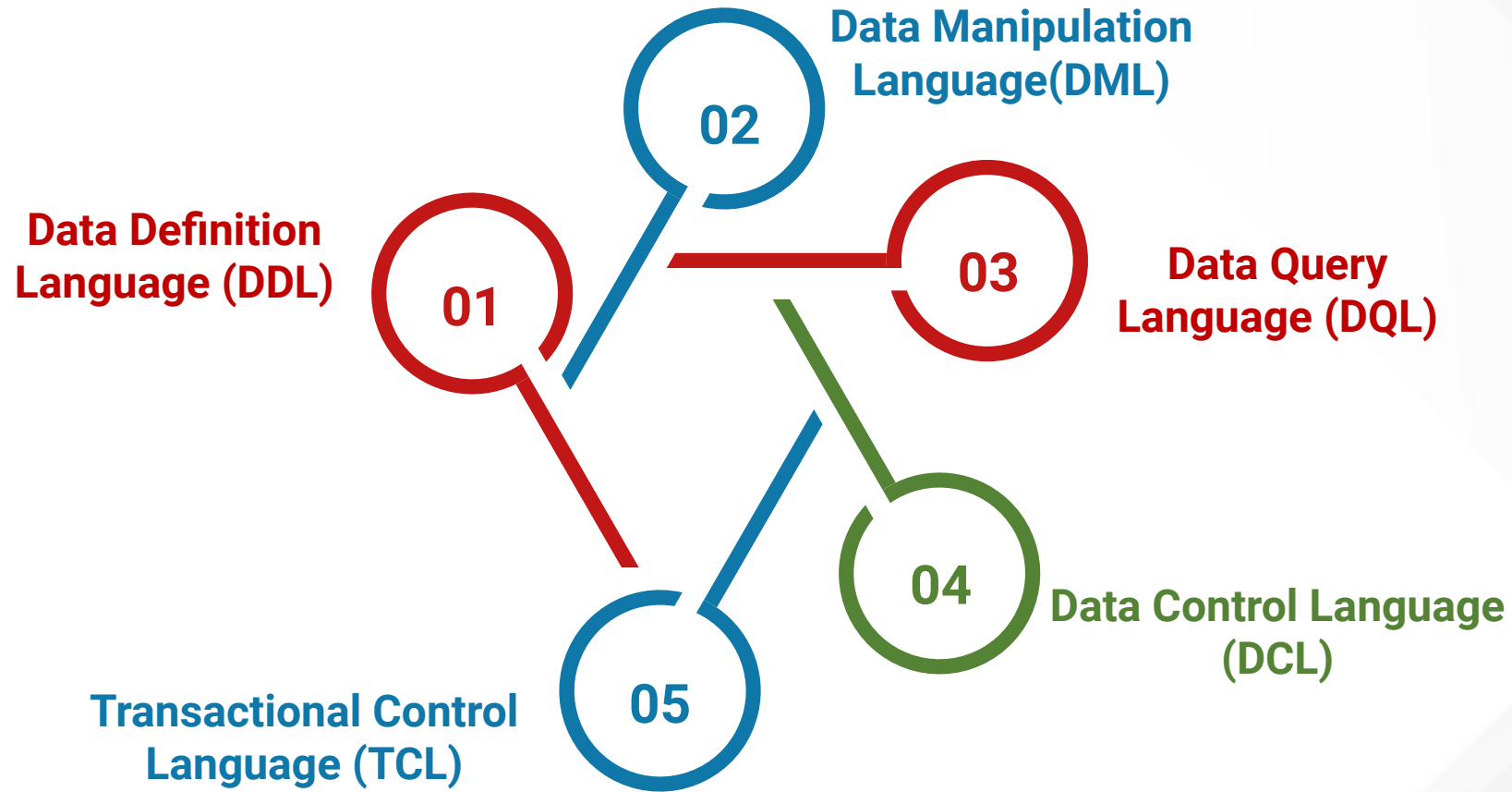
- One of the most crucial advantages of having a DBMS is the protection it gives to your company or organization.
- End users and programmers may access the same data using a database management system without affecting its integrity.

Structured Query Language (SQL) Basics

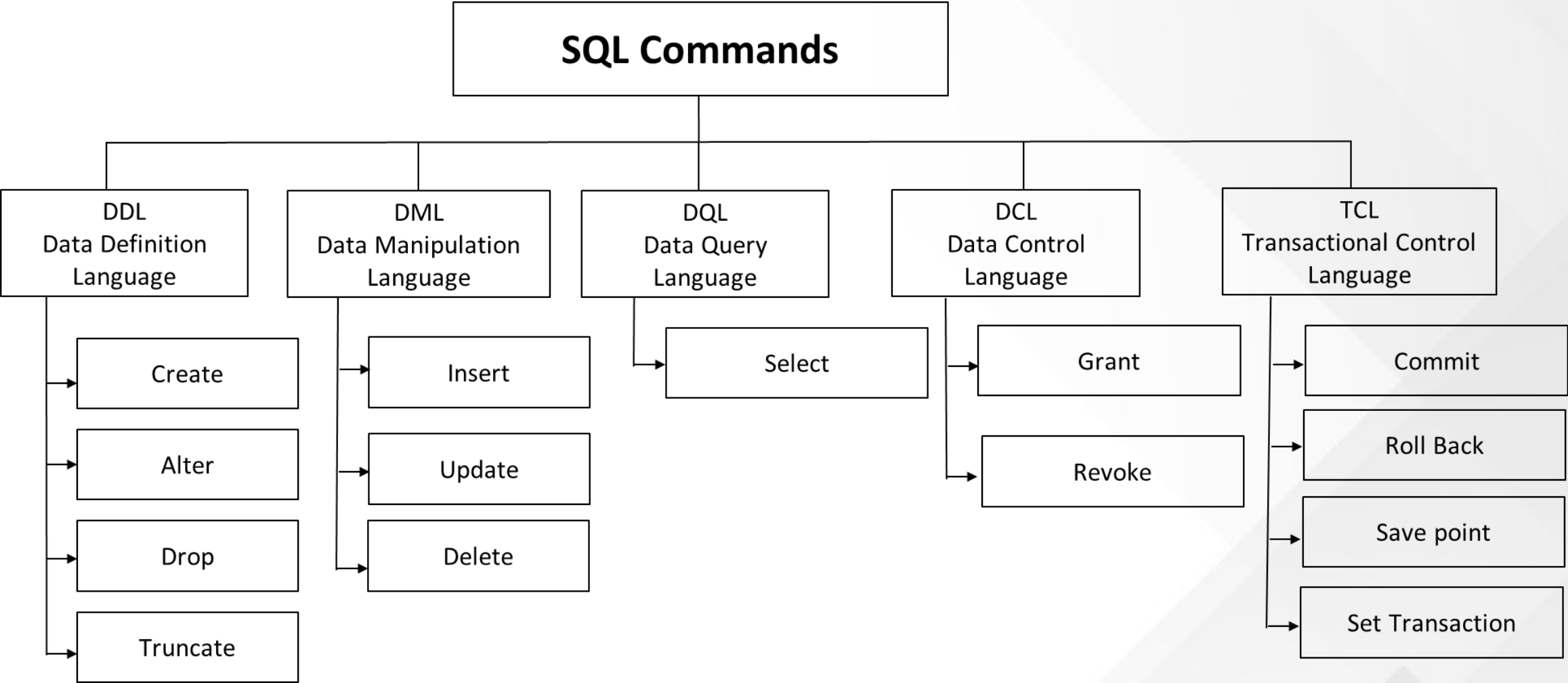


SQL Introduction

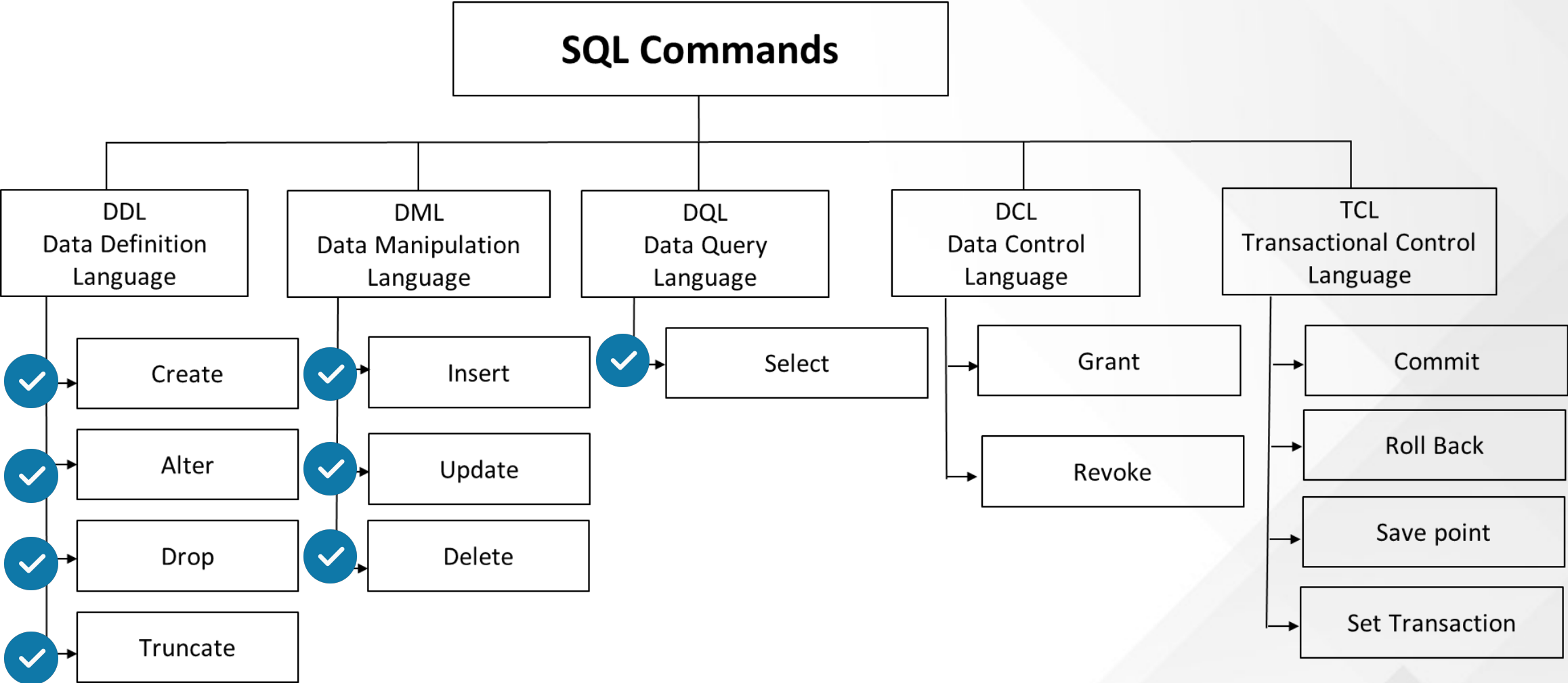
The commands available in SQL can be broadly categorized as follows:



Types of SQL Commands



Types of SQL Commands

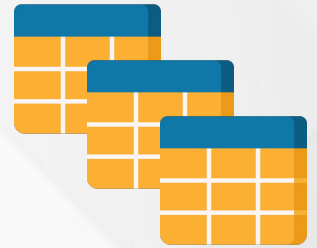


SQL Commands –Data definition Language

A group of five students (three women and two men) are sitting together, smiling, in a casual setting. The image is in grayscale and serves as a background. A solid red horizontal bar is at the bottom. White geometric lines, including a large 'L' shape and a diagonal line, are overlaid on the right side of the image.

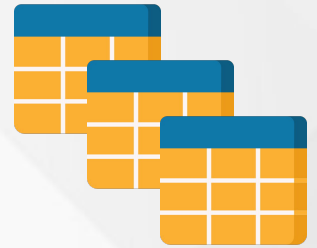
Data Definition Language (DDL)

- A database is a **collection of tables**, and a database server may store a large number of them.
- Tables (specified by columns) → Rows → Database Server → Databases → Tables.
- The terms "**database objects**" and "**tables**" relate to databases and tables, respectively.
- Data Definition Language refers to any process that involves creating, altering, or removing database items (DDL).



Data Definition Language (DDL)

- DDL is used to **create** a new schema as well as to **modify** an existing schema.
- The typical commands available in DDL are:
 - CREATE
 - ALTER
 - DROP
 - TRUNCATE

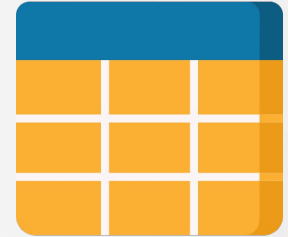


Create Tables



Prerequisites for Creating Tables

- A database is required to **build** and **manage** tables.
- You should provide the following information when creating columns in a table: the column's name, datatype (integer, floating point, string, and so on), and Let's look at data types before we establish a table.

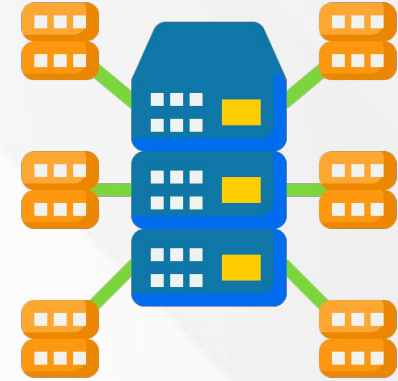


Data Types



Built-in Data Types

- A database is required to **build** and **manage** tables.
- You should provide the following information when creating columns in a table: the column's **name**, **datatype** (integer, floating point, string, and so on), and let's look at data types before we establish a table.



Note

- You don't need to memorize the maximum and minimum size limits for various data types since every relational database contains them.
- The idea is to know what data type should be utilized in a certain case.

CREATE TABLE - Syntax

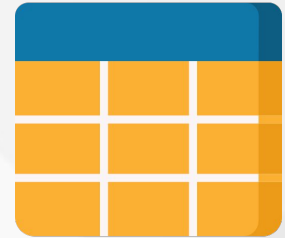
- The CREATE TABLE statement is used to create a new table in a database.
- Syntax:

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

- The table's column names are specified by the column parameters.
- The datatype parameter defines the type of data that may be stored in the column (e.g. varchar, integer, date, etc.)

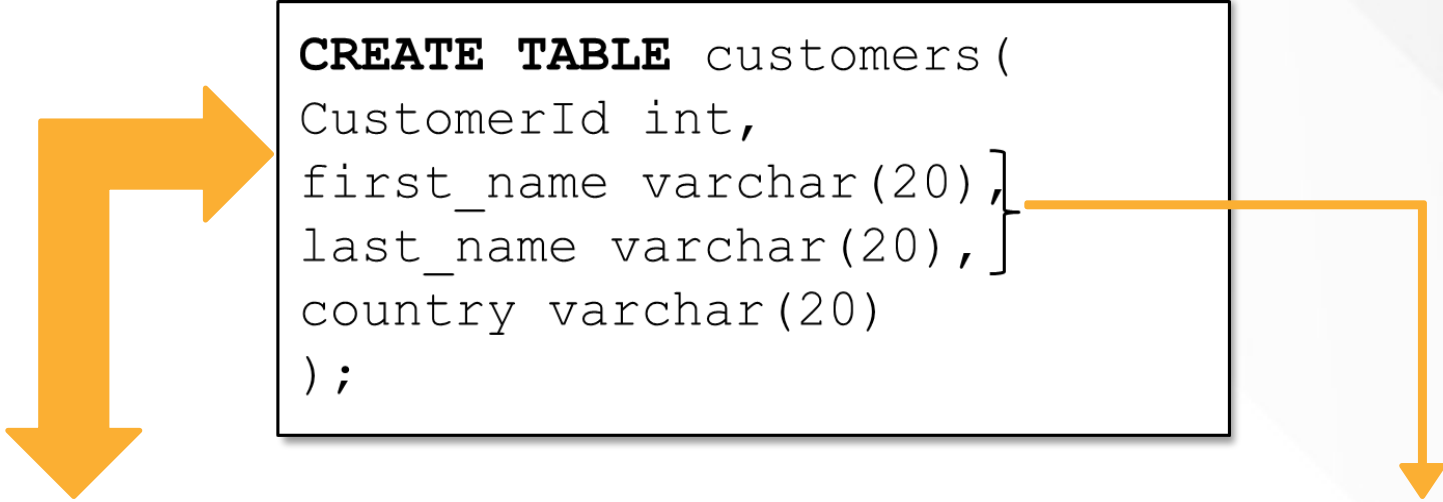
CREATE TABLE - Example

- We will create a customers table, which will hold the customers information.
- The table will have the following columns in it:
 - CustomerID
 - FirstName
 - LastName and
 - Country



CREATE TABLE - Example

- We will create a table “customers” in the company database. Select the database ‘company’ using USE company command.



```
CREATE TABLE customers(  
CustomerId int,  
first_name varchar(20),  
last_name varchar(20),  
country varchar(20)  
);
```

It is declared as an integer since it contains only integers

First_name, last_name, and country: They contain strings, so they are defined as varchar

Drop Table



CREATE TABLE - Syntax

- To remove an existing table from a database, use the DROP TABLE command.
- Syntax:

```
DROP TABLE table_name;
```

Note

- When dropping a table, be cautious.
- If you delete a table, you will lose all the information it contains!

DROP and TRUNCATE TABLE - Example

- The following SQL statement drops the existing table "company":

```
DROP TABLE Customers;
```

- The TRUNCATE TABLE command deletes the contents of a table but not the table itself.

```
TRUNCATE TABLE Customers;
```


Data Manipulation Language (DML)



Data Manipulation Language (DML)

- As a data analyst, you will spend the bulk of your time generating insights and using DML commands.
- The typical commands available in DML are:
 - INSERT
 - UPDATE
 - DELETE
 - SELECT



Insert



SQL INSERT - Syntax

- To add new records to a table, use the **INSERT INTO** command.
- There are two methods to construct the **INSERT INTO** statement.
- The first method defines the column names as well as the values that will be added.
- Syntax:

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

SQL INSERT - Syntax

- Another method for inserting data that does not utilize the column names is as follows:
- Syntax :

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

- Make sure the order of the values is in the same order as the columns in the table.

SQL INSERT - Example

- In this step, we'll add data to the customers table that we generated before.
- The **INSERT** statement is used to populate a table with new entries.

```
INSERT INTO customers (CustomerId, first_name, last_name, country)
VALUES
(1, 'Mike', 'Christensen', 'USA'),
(2, 'Andy', 'Hollands', 'Australia'),
(3, 'Ravi', 'Vedantam', 'India');
```

INSERT - Example

- The "customer" table will now look like this:

CustomerID	first_name	last_name	Country
1	Mike	Christensen	USA
2	Andy	Hollands	Australia
3	Ravi	Vedantam	India



Update

SQL UPDATE - Syntax

- The **UPDATE** statement is used to change the contents of a table's existing records.
- Syntax:

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

Note

- When changing records in a table, be cautious! In the **UPDATE** statement, take note of the **WHERE** clause.
- The WHERE clause determines which records should be changed. If you don't utilize the WHERE clause, all of the table's records will be changed!

SQL UPDATE - Example

- We are updating the first row of the customer's table.
- The first_name and the last_name will be updated to Oho, Honey from the country 'India'.
- Syntax:

```
UPDATE customers  
SET first_name = 'Oho', last_name= 'Honey'  
WHERE country = 'India';
```

UPDATE - Example

- The "customer" table will now look like this:

CustomerID	first_name	last_name	Country
1	John	Kent	USA
2	Andy	Hollands	Australia
3	Ravi	Vedantam	India

Delete



SQL DELETE - Syntax

- To delete existing records in a table, use the **DELETE** command.
- Syntax:

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

Note

- Exercise caution while removing records from a table! Take note of the DELETE statement's WHERE clause.
- The WHERE clause identifies the record(s) to remove. If the WHERE clause is omitted, the whole table will be erased!

SQL DELETE - Example

- We are deleting the first row where, first name is Honey.
- Syntax:

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

- The "customer" table will now look like this:

CustomerID	first_name	last_name	Country
2	Andy	Hollands	Australia
3	Ravi	Vedantam	India

Data Query Language (DQL)

A smiling woman with long dark hair is giving a thumbs up in a modern office setting. The image is darkened to serve as a background. A solid red horizontal bar is at the bottom. White geometric lines, including a large rounded rectangle and a diagonal line, are overlaid on the right side of the image.

SELECT Statement - Syntax

- The **SELECT** statement is used to **access data** contained in a table.
- The returned data is saved in a result table referred to as the result-set.
- Syntax:

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

SELECT Statement - Syntax

- column1, column2,... are the names of the fields in the table from which you wish to choose data.
- To select all fields in the table, use the following syntax:

```
SELECT * FROM table_name;
```

SELECT Statement - Data

- Below is a selection from the "Customers" table in the "company" database:

CustomerID	first_name	last_name	Country
1	Mike	Christensen	USA
2	Andy	Hollands	Australia
3	Rahul	Vedantam	India
4	Jeevan	Sharma	India



SELECT Column - Example

- The following SQL query retrieves data from the "Customers" table's "first name" and "country" columns:

```
SELECT first_name, Country FROM Customers;
```

Output:

first_name	Country
Mike	USA
Andy	Australia
Rahul	India
Jeevan	India

Selecting all Columns - Example

- The following SQL statement selects all the columns from the "Customers" table:

```
SELECT * FROM Customers;
```

Output:

CustomerID	first_name	last_name	Country
1	Mike	Christensen	USA
2	Andy	Hollands	Australia
3	Rahul	Vedantam	India
4	Jeevan	Sharma	India



WHERE Clause- Syntax

- The WHERE clause is used to restrict the number of records returned.
- The WHERE clause is used to restrict the extraction to records that meet a specific criterion.
- Syntax:

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

Note

- Not only is the WHERE clause utilized in SELECT statements, but also in UPDATE and DELETE statements!

WHERE Clause- Example

- The following SQL statement selects all the customers from the country "India", in the "Customers" table:

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

- Output:

CustomerID	first_name	last_name	Country
3	Rahul	Vedantam	India
4	Jeevan	Sharma	India

Operators in the WHERE Clause

- The following operators can be used in WHERE clause

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Operators in the WHERE Clause

- The following operators can be used in WHERE clause

Operator	Description
<>	Not equal. In some versions of SQL the operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Did You Know?

- By default, **SQL keywords** (SELECT, FROM, WHERE, AS, etc.) are **case-insensitive**, however they are often expressed in all caps.

Compound Search Conditions



Compound Search Conditions

- The compound conditions are constructed from a collection of basic conditions that are joined by **AND** or **OR**.
- The number of basic criteria that may be included in a single query is limitless.
- They allow you to set compound search criteria in order to **more precisely define your data retrieval needs**.

Compound Search Conditions

- Use following table for further operations:

distributor_id	distributor_name	city	state
1000	aakash_sales	pune	maharashtra
2000	priya_marketing	nagpur	maharashtra
3000	ramesh_sales	chandigadh	punjab
4000	bansal_electrics	indore	MP
5000	deshpandes	patna	bihar
6000	das_sales	lucknow	UP
7000	vijay_sales	rajkot	gujrat



Compound Search Conditions

- Use of the "AND" and "OR" Conditions with the SELECT Statement.

```
SELECT * FROM employee.distributor  
WHERE (state = 'maharashtra' AND distributor_id <> 7000)  
OR (distributor_id = 1000);
```

- Output:

distributor_id	distributor_name	city	state
1000	aakash_sales	pune	maharashtra
2000	priya_marketing	nagpur	maharashtra

Explanation

- The below query will return all distributors located in Maharashtra but without a distributor id equal to 7000.
- Additionally, the query will return distributors with a distributor id of 1000.

Compound Search Conditions

- Use of AND and OR with UPDATE statement.

```
UPDATE employeee.distributor SET state = 'rajasthan'  
WHERE distributor_id = 6000 OR (distributor_id > 5000 AND  
city <> 'lucknow');
```

- Output:

distributor_id	distributor_name	city	state
1000	aakash_sales	pune	maharashtra
2000	priya_marketing	nagpur	maharashtra
3000	ramesh_sales	chandigadh	punjab
4000	bansal_electrics	indore	MP
5000	deshpandes	patna	bihar
6000	das_sales	lucknow	UP
7000	vijay_sales	rajkot	gujrat

Missing Data



Checking Missing Data

- The SQL **NULL** specifies a value that is not existent.
- A NULL value in a database is a value that seems to be **blank** in a field.
- To check for a NULL value, you must use the **IS NULL** or **IS NOT NULL** operators.



Checking Missing Data

- Consider the following Employee table having the records as shown below:

ID	NAME	AGE	ADDRESS	SALARY
1	Kellie	32	California	2000
2	Pete	25	Texas	1500
3	Popy	23	Boston	2000
4	Sam	25	Florida	
5	Jhon	27	Hawaii	



Is Not Null Operator

- Use of AND and OR with UPDATE statement:

```
SELECT ID, NAME, AGE, ADDRESS, SALARY FROM Employee  
WHERE SALARY IS NOT NULL;
```

- Output:

ID	NAME	AGE	ADDRESS	SALARY
1	Kellie	32	California	2000
2	Pete	25	Texas	1500
3	Popy	23	Boston	2000

Is Null Operator

- Syntax

```
SELECT ID, NAME, AGE, ADDRESS, SALARY FROM Employee  
WHERE SALARY IS NULL;
```

- Output:

ID	NAME	AGE	ADDRESS	SALARY
4	Sam	25	Florida	
5	Jhon	27	Hawaii	

Understanding DISTINCT,ORDER BY(Ascending and Descending), BETWEEN Operator

- SQLite's **DISTINCT** keyword is used in conjunction with the **SELECT** command to filter out duplicate records and retrieve only unique ones.
- There may be instances where a table has several duplicate records. When retrieving such records, it makes more sense to retrieve just unique records rather than duplicates.

Syntax

Following is the basic syntax of DISTINCT keyword to eliminate duplicate records.

```
SELECT DISTINCT column1, column2,.....columnN  
FROM table_name  
WHERE [condition]
```


Understanding DISTINCT,ORDER BY(Ascending and Descending), BETWEEN Operator

- The **ORDER BY** clause in SQLite is used to arrange data ascending or descending by one or more columns.

Syntax

Following is the basic syntax of ORDER BY clause.

```
SELECT column-list  
FROM table_name  
[WHERE condition]  
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

Understanding DISTINCT, ORDER BY (Ascending and Descending), BETWEEN Operator

- The **BETWEEN** logical operator determines if a value is inside a specified range of values. The BETWEEN operator **returns true if the value is inside the provided range**.
- The BETWEEN operator is applicable to the WHERE clauses of SELECT, DELETE, UPDATE, and REPLACE statements.

66

67

```
select * from employees where salary between 10000 and 20000;
```

employee_id	first_name	last_name	email	phone_number	hire_date	job_id	salary	manager_id
101	Neena	Kochhar	neena.kochhar@sqltutorial.org	515.123.4568	1989-09-21	5	17000.0	100
102	Lex	De Haan	lex.de haan@sqltutorial.org	515.123.4569	1993-01-13	5	17000.0	100
108	Nancy	Greenberg	nancy.greenberg@sqltutorial.org	515.124.4569	1994-08-17	7	12000.0	101

Understanding the usage of Arithmetic operators in SQL

SQLite Arithmetic Operators

Assume variable **a** holds 10 and variable **b** holds 20, then SQLite arithmetic operators will be used as follows –

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	$a + b$ will give 30
- (Subtraction)	Subtracts the right hand operand from the left hand operand	$a - b$ will give -10
* (Multiplication)	Multiplies values on either side of the operator	$a * b$ will give 200
/ (Division)	Divides the left hand operand by the right hand operand	b / a will give 2
% (Modulus)	Divides the left hand operand by the right hand operand and returns the remainder	$b \% a$ will give 0

Understanding LIKE clause with wildcards(%,*)

- The LIKE operator in SQLite is used to compare text values to a pattern utilizing wildcards. If the search expression matches the pattern expression, the LIKE operator returns true, which is a value of one. Two wildcard characters are used in conjunction with the LIKE operator:
 - The % sign (percent)
 - The _ (underscore character)
- The % sign denotes zero, one, or a sequence of digits or characters. The underscore character is used to denote a single digit or character. These symbols may be used in conjunction with one another.

Understanding LIKE clause with wildcards(%,*)

Syntax

Following is the basic syntax of % and _.

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'
or
SELECT FROM table_name
WHERE column LIKE '%XXXX%'
or
SELECT FROM table_name
WHERE column LIKE 'XXXX_'
or
SELECT FROM table_name
WHERE column LIKE '_XXXX'
or
SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine **N** number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Understanding constraints

(primary key ,check ,unique, not null, default)

- Constraints are the restrictions that apply to the columns of data in a table. These are used to restrict the type of data that a table can include. This guarantees the database's data is accurate and reliable.
- Constraints can be applied at the column or table level. Constraints at the column level are applied to a single column, whereas constraints at the table level are applied to the entire table.

Understanding constraints

(primary key ,check ,unique, not null, default)

Following are commonly used constraints available in SQLite.

NOT NULL Constraint – Ensures that a column cannot have NULL value.

DEFAULT Constraint – Provides a default value for a column when none is specified.

UNIQUE Constraint – Ensures that all values in a column are different.

PRIMARY Key – Uniquely identifies each row/record in a database table.

CHECK Constraint – Ensures that all values in a column satisfies certain conditions.

NOT NULL Constraint:

A column can contain **NULL** values by default. If you do not wish a column to include a NULL value, you must define a constraint on that column saying that NULL values are no longer permitted.

A NULL value is not synonymous with no data. Rather, it indicates **unidentified data**.

Example

For example, the following SQLite statement creates a new table called COMPANY and adds five columns, three of which, ID and NAME and AGE, specifies not to accept NULLs.

```
CREATE TABLE COMPANY (  
    ID INT PRIMARY KEY      NOT NULL,  
    NAME                    TEXT    NOT NULL,  
    AGE                     INT     NOT NULL,  
    ADDRESS                 CHAR(50),  
    SALARY                  REAL  
);
```


DEFAULT Constraint

When the **INSERT INTO** statement does not specify a value for a column, the DEFAULT constraint assigns a default value to the column.

Example

For example, the following SQLite statement creates a new table called COMPANY and adds five columns. Here, SALARY column is set to 5000.00 by default, thus in case INSERT INTO statement does not provide a value for this column, then by default, this column would be set to 5000.00.

```
CREATE TABLE COMPANY (  
    ID INT PRIMARY KEY     NOT NULL,  
    NAME           TEXT     NOT NULL,  
    AGE            INT       NOT NULL,  
    ADDRESS        CHAR(50),  
    SALARY         REAL      DEFAULT 50000.00  
);
```

UNIQUE Constraint

The **UNIQUE** Constraint ensures that no two entries have the same values in a given column. For example, in the COMPANY table, you may wish to prevent two or more people from having the same age.

Example

For example, the following SQLite statement creates a new table called COMPANY and adds five columns. Here, AGE column is set to UNIQUE, so that you cannot have two records with the same age –

```
CREATE TABLE COMPANY (  
    ID INT PRIMARY KEY     NOT NULL,  
    NAME           TEXT     NOT NULL,  
    AGE            INT       NOT NULL UNIQUE,  
    ADDRESS        CHAR(50),  
    SALARY         REAL      DEFAULT 50000.00  
);
```

PRIMARY KEY Constraint

- Each entry in a database table is uniquely identified by the **PRIMARY KEY** constraint.
- There may be additional UNIQUE columns in a table, but only one primary key.
- When building database tables, primary keys are critical. Primary keys are **one-of-a-kind identifiers**.
- They are used to denote tabular rows.
- When relationships between tables are created, primary keys become foreign keys in other tables.
- Due to a '**long-standing coding error**,' primary keys in SQLite can be NULL. Other databases, on the other hand, do not follow this pattern.

PRIMARY KEY Constraint

Example

You already have seen various examples above where we have created COMPANY table with ID as a primary key.

```
CREATE TABLE COMPANY (  
    ID INT PRIMARY KEY      NOT NULL,  
    NAME                    TEXT    NOT NULL,  
    AGE                    INT      NOT NULL,  
    ADDRESS                CHAR(50),  
    SALARY                 REAL  
);
```

CHECK Constraint

- CHECK Constraints allow conditions to validate the value entered into a record. If the condition evaluates to **false**, the record violates the constraint and is therefore **excluded** from the table.

Example

For example, the following SQLite creates a new table called COMPANY and adds five columns. Here, we add a CHECK with SALARY column, so that you cannot have any SALARY Zero.

```
CREATE TABLE COMPANY3 (  
    ID INT PRIMARY KEY      NOT NULL,  
    NAME                    TEXT  NOT NULL,  
    AGE                    INT    NOT NULL,  
    ADDRESS                CHAR (50) ,  
    SALARY                 REAL   CHECK (SALARY > 0)  
);
```



Thank You!

Copyright © HeroX Private Limited, 2022. All rights reserved.