SQL

POCKET BOOK

Quick Reference Guide To Master SQL

NOTHING CAN STOP YOU.....YOU'RE ALL THE WAY UP !!!!

Thanks for downloading this SQL pocketbook; I am grateful that you have taken this decision to work on your skills and build an unshakable foundation for your career and projects.
This mini-book will help you in mastering SQL so that you don't have to look things up on google constantly. My best advice is to take printouts of this guide and keep them with you when you're working.

This pocketbook will help you stay focused and not look up things on the shiny internet full of distractions.

I am also constantly updating this pocketbook and adding new stuff and illustrations to help you better.
So, to get the latest stuff, giveaways, tips, tricks, and more helpful content. Please stay connected with me via my email newsletters!

Dane Wade
Author

INDEX

CHAPTER 4

CHAPTER 1

SOME SQL BASICS

OVERVIEW OF SQL

- SQL stands for **Structured Query Language**
- Language to communicate with databases.
- Accepted as Standard language for relational database management systems by ANSI in 1986.
- SQL allows users to define, access and manipulate database and its objects.

RELATIONAL DATABASES

Any database that has the below properties can be considered as a relational database:

- database that stores data in tables.
- follows the relational model proposed by E.F Codd
- Uses SQL as a standardized language to interact with the database can be considered as a relational database.

INVENTERS OF SQL

Donald D. Chamberlin

Raymond F. Boyce

RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)

Relational Database Management Systems or RDBMS is a software that is designed to work on Relational Databases. RDBMS provides a graphical user interface to interact with the relational databases. Users can create, manage and manipulate data/definition of database objects.

TYPES OF SQL COMMANDS IN SQL

DATA DEFINITION LANGUAGE (DDL)

commands that define, creates and modifies the structure of a database and database objects.

Below are the DDL Commands:
- CREATE
- ALTER
- DROP
- TRUNCATE

DATA MANIPULATION LANGUAGE (DML)

Commands that insert, modify and delete the data present in databases.

Below are the DDL Commands:
- INSERT
- UPDATE
- DELETE

DATA CONTROL LANGUAGE (DCL)

commands that grant and revokes privileges and authority to users in a database.

Below are the DDL Commands:
- GRANT
- REVOKE

DATA QUERY LANGUAGE (DQL)

commands that retrieve data from a database.

SELECT command is used to retrieve data and is hence considered as DQL command.

TRANSACTION CONTROL LANGUAGE (TCL)

commands to manage transactions in a database.

Below are the TCL commands:
- COMMIT
- ROLLBACK
- SAVEPOINT

CHAPTER 2

DATA AND DATATYPES

DATATYPES

Choosing the right datatype is essential before creating database objects. Broadly datatypes are divided into 3 categories—Numeric, String and Datetime.

Below are the essential datatypes for SQL Server:

NUMERIC DATATYPES

Numerical Datatypes stores numerical values.

BIT	Stores Integer values— 0,1 and NULL
TINYINT	Stores integer values starting from 0 to 255
SMALLINT	Stores small integer values starting from -32,768 to 32,767
INTEGER	Stores integers values starting from -2,147,483,648 to 2,147,483,647

BIGINT	Stores Big Sized integers values starting from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
DECIMAL	Best for storing Decimal data from -10^38 +1 to 10^38 -1
NUMERIC	Similar to Decimal data type. Better choice for fixed precision and scale. Starting from -10^38 +1 to 10^38 -1.
SMALL MONEY	Stores smaller ranges of monetary values and range starts from -214,748.3648 to 214,748.3647.
MONEY	Stores wider ranges of monetary values and range starts from -922,337,203,685,477.5808 to 922,337,203,685,477.5807
REAL	Best to represent the approximation of real values from -3.40E + 38 to 3.40E + 38
FLOAT	Stores approximate values like real datatype but with better precision. Starts from – 1.79E+308 to 1.79E+308.

STRING DATATYPES

CHAR	Stores fixed length Text and alphanumeric characters. Starts from 0 to 8000 characters.
VARCHAR (n)	Can store numers and characters Length is variable for the character strings and range starts from 0 to 8000. Where n = Length of the string values.
VARCHAR (max)	Variable length character datatype and it can store larger data as compared to varchar (n). Storage capacity upto 2GB.
NCHAR	Stores fixed length UNICODE characters. Starts from 1 to 4000.
NVARCHAR	Stores UNICODE characters of varying length. Storage range starts from 0 till 4000 characters.

BINARY DATATYPES

BINARY(n)	Stores fixed length raw binary data . Storage range starts from 1 till 8000.
VARBINARY (n)	Stores varying length binary values. Storage range starts from 1 till 8000. Where n= any number
VARBINARY (max)	Stores varying length binary values. Storage size ranges is upto 2GB

DATE TIME DATATYPES

DATE	Stores only date values but not time values eg: date, days, month, year etc. Range starts from 0001-01-01 to 9999-12-31
DATETIME	Stores both date and time values. dates range starts from 1753-01-01 to 9999-12-31 time range starts from 00:00:00 to 23:59:59.997
SMALLDATETIME	Stores date and time values similar to DATETIME. Unlike DATETIME datatype Time is stored in minutes and not in seconds

DATETIME2	similar to DATETIME datatype with more precision and storage size. Date Range starts from 0001-01-01 to 9999-12-31 Time Range starts from00:00:00 to 23:59:59.9999999
DATETIMEOFFSET	Stores date and time and "time zone offsets" as well.
TIME	Dedicated datatype to store TIME values.

OTHER DATATYPES

CURSOR	A datatype that is designed for database operations and store reference to a cursor.
XML	A datatype designed to store XML format data and XML variables.
TABLE	Table datatype is designed to store result set that later can be used for some processing .
SQL_VARIANT	SQL_VARIANT datatype can be used to store different SQL Server datatypes

CHAPTER 3

CREATING DATABASES AND TABLES

WORKING WITH DATABASE

CREATING A DATABASE

Syntax: `CREATE DATABASE NameOfTheDatabase;`

Example: `CREATE DATABASE DataCeps ;`

DELETING A DATABASE

Syntax: `DROP DATABASE NameOfTheDatabase;`

Example: `DROP DATABASE DataCeps ;`

DISPLAY CURRENT DATABASE NAME

`DB_NAME()` function helps in finding out the current database name;

Example: `SELECT DB_NAME() as MyCurrentDatabase;`

DISPLAY NAMES OF ALL THE DATABASES

`sys.databases` has the record for each and every database in MS SQL Server.

Selecting the details for all the database:

```
SELECT * FROM sys.databases;
```

WORKING WITH TABLES

CREATING TABLES

CREATE statement can be used to create tables in a database.

Syntax:

```
CREATE TABLE NameOfTheTable (
ColumnNumber1 Datatype,
ColumnNumber2 Datatype,
ColumnNumber3 Datatype,
.
.
.
ColumnNumber(n) Datatype
);
```

Example:

```
CREATE TABLE DatacepsStudents (
StudentID INT,
FirstName VARCHAR(100),
LastName VARCHAR(100),
CourseName VARCHAR(100)
);
```

CREATING TABLES WITH CONSTRAINTS

Constraints helps in establishing rules for the data that is going to be stored in tables.

Syntax:

```
CREATE TABLE NameOfTable (
ColumnNumber1 DataType Constraint,
ColumnNumber2 DataType Constraint,
ColumnNumber2 DataType Constraint,
.
.
.
ColumnNumber(n) DataType Constraint
);
```

Example: Creating a table **DatacepsStudents** with **NOT NULL** Constraint on column **StudentID**

```
CREATE TABLE DatacepsStudents (
StudentID INT NOT NULL,
FirstName VARCHAR(100),
LastName VARCHAR(100),
CourseName VARCHAR(100)
);
```

CREATING TABLES WITH PRIMARY AND FOREIGN KEYS

To create a table with primary key constraint we can use **PRIMARY KEY** constraint with **CREATE TABLE** statement and, mentioning foreign key with the **FOREIGN KEY REFERENCES** with the reference table (child table) helps in adding a foreign key constraint to the table.

Syntax:

```
CREATE TABLE NameOfTheTable
(
PrimaryKeyColumnName DATATYPE OtherConstraints PRIMARY KEY,
ColumnNumber2 DATATYPE (SIZE),
ColumnNumber3 DATATYPE (SIZE),
ColumnNumber4 DATATYPE (SIZE),
ForeignKeyOfParentTable DATATYPE FOREIGN KEY REFERENCES
ChildTable(PrimaryKeyOfChildTable)
);
```

Example:

```
CREATE TABLE DatacepsStudents
(
StudentID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(100),
LastName VARCHAR(100),
CourseName VARCHAR(100),
CID INT FOREIGN KEY REFERENCES DatacepsCourses (CoursesID)
);
```

CREATE TABLES WITH AN AUTOMATICALLY GENERATED FIELD

To create a table with an auto incrementing value for a particular column in sql server, we can add **IDENTITY** property to that column.

Syntax:

```
IDENTITY [ (seed , increment) ]
```

Where **seed** ="value that will be used for the first record in the table";
And **increment** ="value that will be added to the previous row value and helps in incrementing";

Example:

```
CREATE TABLE DatacepsStudents
(
StudentID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
FirstName VARCHAR(100),
LastName VARCHAR(100),
CourseName VARCHAR(100),
);
```

In the above query the seed for **IDENTITY** column will be 1 and the increment is also 1. Therefore, the **StudentID** will start from 1 and will increment by 1.

CREATE A TABLE THAT DOES NOT ALREADY EXIST

To check if a table already exists or not before even executing a create a statement to create the table, we can use **IF** with **NOT EXISTS** operator.

Example:

```
IF NOT EXISTS
      (

        SELECT *
        FROM INFORMATION_SCHEMA.TABLES
        WHERE
        TABLE_NAME = 'DataCepsStudents'
      )

BEGIN

      CREATE TABLE DatacepsStudents
      (
      StudentID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
      FirstName VARCHAR (100),
      LastName VARCHAR (100),
      CourseName VARCHAR (100),
      );

END
```

ADDING A COLUMN IN A TABLE USING ALTER

ALTER TABLE can be used to ADD a new column in an existing table.

Syntax:

```
ALTER TABLE NameOfTheTable ADD NameOfTheColumn Datatype;
```

Example:

```
ALTER TABLE DatacepsStudents ADD CourseID SMALLINT;
```

MODIFYING COLUMN'S DATATYPE IN A TABLE

ALTER TABLE command can be used to modify datatype of a column in a table.

Syntax:

```
ALTER TABLE NameOfTheTable
ALTER COLUMN NameOfTheColumn NewDataType(size);
```

Example:

```
ALTER TABLE DatacepsStudents
ALTER COLUMN CourseID INT;
```

RENAMING COLUMN IN A TABLE

ALTER TABLE command can be used to modify name of a column in a table.

Syntax:

```
ALTER TABLE NameOfTheTable
ALTER COLUMN NewNameOfTheColumn DATATYPE(size);
```

Example:

```
ALTER TABLE DatacepsStudents
ALTER COLUMN CourseNumber INT;
```

DELETING/DROPPING COLUMN FROM A TABLE

DROP COLUMN command can be used to Delete a column from a table.

Example:

```
ALTER TABLE DatacepsStudents
DROP COLUMN CourseID;
```

RENAMING A TABLE IN SQL

ALTER TABLE can be used to Rename a table.

Syntax:

```
ALTER TABLE OldTableName RENAME TO NewTableName;
```

Example:

```
ALTER TABLE DatacepsStudents RENAME TO NewDatacepsStudents ;
```

CHAPTER 4

UPDATING AND INSERTING DATA

INSERT INTO TABLES

SQL **INSERT INTO** Statement helps in inserting records in a table.

INSERTING SINGLE RECORD IN A TABLE

Syntax:

```
INSERT INTO NameOfTheTable
(
ColumnNumber1,ColumnNumber2....ColumnNumber (n)
)
VALUES ( ValueNumber1, ValueNumber2.....ValueNumber (n));
```

Example:

```
INSERT INTO DatacepsStudents (
StudentID,FirstName,LastName,Cour
seName
)
VALUES ('111','Dane','Wade','SQL
Course');
```

MULTI ROW INSERTS

Syntax:

```
INSERT INTO NameOfTheTable
(
ColumnNumber1,ColumnNumber2. . . CourseName (n)
)
VALUES ('ValueForRecord1Column1'... 'ValueForRecord1Column (n)'),
 ('ValueForRecord2Column1'... 'ValueForRecord2Column (n)'),
 ('ValueForRecord3Column1'... 'ValueForRecord3Column (n)');
```

Example:

```
INSERT INTO DatacepsStudents
(StudentID,FirstName,LastName,CourseName)
VALUES ('112','John','Lloyd','Python'),
     ('113', 'Jane','Robbins','R'),
     ('114', 'Stefan','Mikkelson','Data Analytics');
```

INSERT THE RESULTS OF A QUERY INTO A TABLE USING SUBQUERIES

Syntax:

```
INSERT INTO NameOfTable (ColumnNumber1.....ColumnNumber (n))
VALUES (SELECT Column1..... Column(n)
FROM NameOfTheTable2);
```

Example:

```
INSERT INTO NewDataCepsStudents
(StudentID,FirstName,LastName,CourseName)
VALUES (SELECT
StudentID,FirstName,LastName,CourseName
FROM DataCepsStudents);
```

UPDATING TABLES

UPDATE STATEMENT: **UPDATE** statement helps in updating/modifying the column values in a table.

Syntax:

```
UPDATE NameOfTheTable
SET ColumnNumber1 = Value1, ColumnNumber2 = Value2, ...
ColumnNumber(n)=Value3(n)
WHERE condition;
```

SAMPLE DATA TABLES

DatacepsStudents

StudentID	FirstName	LastName	CourseName
111	Dane	Wade	SQL Course
112	Max	Blaine	Python Mastery
113	Sara	James	C++

NewDatacepsStudents

StudentID	FirstName	LastName	CourseName
111	Daneil	Wade	SQL Course
112	Max	Blaine	Python Mastery
113	Sara	James	C++
114	Jake	Otusanaya	C
115	Mark	Z	Data Structures
116	Bill	Adam	C++

Examples:

UPDATING A SINGLE COLUMN

```
UPDATE DatacepsStudents
SET FirstName = 'Rob'
WHERE StudentID='116';
```

UPDATING MULTIPLE COLUMNS

```
UPDATE DatacepsStudents
SET FirstName = 'Dane', LastName ='S Wade'
WHERE StudentID='111';
```

UPDATE NEW VALUES FROM A SUBQUERY

```
UPDATE DatacepsStudents
SET FirstName= (SELECT FirstName
FROM NewDataCepsStudents Where StudentID='111')
WHERE StudentID='111';
```

UPDATING VIEWS

```
UPDATE NameOfTheView
SET ColumnNumber1 = Value1, ColumnNumber2 = Value2, ...
ColumnNumber(n)=Value3(n)
WHERE condition;
```

CHAPTER 5

DELETING DATA AND DATABASE OBJECTS

DELETE Statement: **DELETE** statement is used to Delete database objects and records from a table.

DROP command : **DROP** command is used to completely destroy database objects like—tables, indexes , views etc. along with its data.

SAMPLE DATA TABLES

DatacepsStudents

StudentID	FirstName	LastName	CourseName
111	Dane	Wade	SQL Course
112	Max	Blaine	Python Mastery
113	Sara	James	C++

NewDatacepsStudents

StudentID	FirstName	LastName	CourseName
111	Daneil	Wade	SQL Course
112	Max	Blaine	Python Mastery
113	Sara	James	C++
114	Jake	Otusanaya	C
115	Mark	Z	Data Structures
116	Bill	Adam	C++

DELETING A SINGLE RECORD

Syntax:

```
DELETE FROM NameOfTheTable
WHERE condition;
```

Example:

```
DELETE FROM DatacepsStudents
WHERE StudentID='113';
```

DELETING A TABLE

Syntax:

```
DROP TABLE NameOfTheTable;
```

Example:

Delete table NewDatacepsStudents permanently along with its data.

```
DROP TABLE NewDatacepsStudents;
```

DELETING ALL RECORDS

Syntax:

```
DELETE FROM NameOfTheTable;
```

Example:

```
DELETE FROM DatacepsStudents;
```

CHAPTER 6

QUERYING BASICS

SAMPLE DATA TABLES

DatacepsStudents

StudentID	FirstName	LastName	CourseName	CourseJoining Date	Total Fees Paid
111	Dane	Wade	SQL Course	2019-01-12	99.99
112	Max	Blaine	Python Mastery	2021-01-04	
113	Sara	James	C++	2019-01-04	

NewDatacepsStudents

StudentID	FirstName	LastName	CourseName
111	Daneil	Wade	SQL Course
112	Max	Blaine	Python Mastery
113	Sara	James	C++
114	Jake	Otusanaya	C
115	Mark	Z	Data Structures
116	Bill	Adam	C++

The SELECT Statement

SELECT statement helps in retrieving data from databases. Data displayed by SELECT statement is displayed in tabular format.

RETRIEVING DATA FROM SELECTED COLUMNS

Syntax:

```
SELECT ColumnName1, ColumnName2...ColumnName(n)
FROM TableName;
```

RETRIEVING DATA FROM ALL COLUMNS

Syntax:

```
SELECT * FROM TableName;
```

CLAUSES IN SQL

In built functions of SQL server that helps in filtering and analyzing data.

The FROM Clause: FROM clause is used to specify and indicate the source from where we are pulling the dataset. The source is usually database objects like table, view , CTE etc.

Syntax:

```
SELECT ColumnName1, ColumnName2,
ColumnName3...ColumnName(n)
FROM TableName;
```

TOP Clause: TOP clause is used after **SELECT** and helps in fetch thing the Top (n) records. Where n can be any number or percentage you want in the result set.

Syntax:

```
SELECT TOP N FROM NameOfTheTable;
```

Where, N= Number or percentage of records you want in the output result set.

Example:

```
SELECT TOP 10 FROM DatacepsStudents;
```

DISTINCT Clause : DISTINCT clause is used to fetch only distinct records from the table. Using **DISTINCT** in **SELECT** query removes duplicate records and only fetches the unique records.

Syntax:

```
SELECT DISTINCT ColumnName FROM NameOfTheTable;
```

Where, **N**= Number or percentage of records you want in the output result set.

Example:

```
SELECT DISTINCT CourseName FROM NewDatacepsStudents;
```

WHERE Clause: WHERE clause helps in filtering records from a table. It filters the data based on the condition we build after the WHERE clause.

Syntax:

```
SELECT ColumnName1, ColumnName2 . . ..ColumnName (n)
FROM NameOfTheTable
WHERE Condition;
```

Example:

```
SELECT StudentID, FirstName, LastName
FROM NewDatacepsStudents
WHERE CourseName='SQL Course';
```

GROUP BY Clause : Groups similar values and data together. Usually **GROUP BY** clause is used with aggregate functions.

Syntax:

```
SELECT
ColumnName1,AGGREGATE_FUNCTION(ColumnName1)
FROM NameOfTheTable
WHERE Condition
GROUP BY ColumnName1 ;
```

Example:

```
SELECT COUNT(CourseName) AS CourseCount, CourseName
FROM DatacepsStudents
GROUP BY CourseName ;
```

HAVING Clause : HAVING clause also helps in filtering data groups. **HAVING** clause is commonly used with aggregate functions. As we can't use **WHERE** clause with aggregate functions.

Syntax:

```
SELECT ColumnName1, ColumnName2
FROM NameOfTheTable
GROUP BY ColumnName1,ColumnName2
HAVING condition
```

Example:

```
SELECT  CourseName, COUNT (CourseName) AS CourseCount
FROM DatacepsStudentsCopy
GROUP BY CourseName
HAVING CourseName='SQL Course';
```

ORDER BY Clause : ORDER BY Clause is used for sorting the result set of a query.

Syntax:

```
SELECT Column1, Column2, Column3 . . . . . Column(n)
FROM NameOfTheTable
ORDER BY Column1;
```

Example:

```
SELECT FirstName,CourseName
FROM DatacepsStudents
ORDER BY FirstName;
```

ORDER EXECUTION OF SQL QUERY

A SQL query is executed and evaluated in the below mentioned sequence:

- **The FROM clause**: The FROM clause along with the JOINS are evaluated on the first place. It gives the total working dataset.
- **The WHERE clause:** Filters the working dataset after applying the conditions and constraints and discards the unnecessary data.
- **The GROUP BY clause:** The dataset is aggregated and grouped using the GROUP BY clause.
- **The HAVING clause:** HAVING clause helps in filtering the records that are grouped and aggregated using GROUP BY clause.
- **The SELECT clause:** Displays or Returns the final dataset.

SQL EXPRESSIONS

SQL Expressions can be considered as formulas. These expressions help in evaluating a value by performing some operations. To build a SQL Expression we can use different SQL Operators and Functions.

Below are the different types of SQL Expressions:

- **Boolean Expressions:** Boolean Expressions in SQL fetches records based on single value matching.

Syntax:

```
SELECT ColumnName1,ColumnName2
FROM NameOfTheTable
WHERE Boolean condition;
```

Numeric Expressions: Numerical Expressions performs mathematical operations on a SQL query.

Syntax:

```
SELECT ColumnName1,
ColumnName2, ColumnName3...
ColumnName (n)
FROM NameOfTheTable
WHERE Mathematical conditions
or expressions;
```

OR

```
SELECT Mathematical Condition;
```

OR

```
SELECT Mathematical
Condition,ColumnName1,
ColumnName2... ColumnName (n)
FROM NameOfTheTable
```

Example:

```
SELECT (15 * 6) AS Multiplication ;
```

Date Expressions: Date and Time related values are compared with the help of SQL Date expressions.

Syntax:

```
SELECT ColumnName1, ColumnName2,
ColumnName3... ColumnName (n)
FROM NameOfTheTable
WHERE Date conditions;
```

Example:

```
SELECT StudentID ,FirstName
FROM DatacepsStudents
WHERE CourseJoiningDate> '2019-11-11';
```

CHAPTER 6

OPERATOR AND FUNCTIONS

OPERATORS IN SQL

Operators in SQL helps in doing complex comparisons and creating conditions for a given data set.
Below are the operators in SQL:

SAMPLE DATA TABLES

DatacepsStudents

StudentID	FirstName	LastName	CourseName	CourseJoining Date	Total Fees Paid	CourseID
111	Dane	Wade	SQL Course	2019-01-12	99.99	C1
112	Max	Blaine	Python Mastery	2021-01-04	NULL	C2
113	Sara	James	C++	2019-01-04	NULL	C3

CourseDetails

CID	CourseName	CoursePrice
C1	SQL Course	$99.0
C2	Python Mastery	$149.0
C3	C++	$49.0

ARITHMETIC OPERATORS

OPERATOR	OPERATION	DESCRIPTION	EXAMPLE
+	ADD	Adds two numbers present on either side of operator	**SELECT 60 + 40;** **OUTPUT :** 100
-	SUBTRACT	Subtracts RHS from LHS value	**SELECT 60 - 40;** **OUTPUT : 20**
/	DIVIDE	Divides LHS value by RHS value	**SELECT 60 / 3;** **OUTPUT : 20**
*	MULTIPLY	Multiplies LHS with RHS	**SELECT 60 * 40;** **OUTPUT : 2400**
%	MODULUS	Returns remainder, after dividing LHS value by RHS value.	**SELECT 60 % 40;** **OUTPUT : 20**

COMPARISON OPERATORS

OPERATOR	OPERATION	DESCRIPTION	EXAMPLE
=	EQUAL TO	If both LHS and RHS are equal, then only condition become true	SELECT * FROM DATACEPSSTUDENTS WHERE COURSEJOININGDATE ='2021-01-04';
!= OR <>	NOT EQUAL TO	If both LHS and RHS are NOT equal, then only condition become true	SELECT * FROM DATACEPSSTUDENTS WHERE COURSEJOININGDATE <>'2021-01-04';
<	LESS THAN	If LHS value is less than the RHS value, only then the condition is considered as true.	SELECT * FROM DATACEPSSTUDENTS WHERE STUDENTID <'112';
>	GREATER THAN	If LHS value is greater than the RHS value, then the condition is considered as true.	SELECT * FROM DATACEPSSTUDENTS WHERE STUDENTID >'112';
<=	LESS THAN EQUAL TO	If LHS value is greater than the RHS value, then the condition is considered as true.	SELECT * FROM DATACEPSSTUDENTS WHERE STUDENTID >'112';
>=	GREATER THAN EQUAL TO	If LHS value is greater than OR equal to the RHS value, then the condition is considered as true.	SELECT * FROM DATACEPSSTUDENTS WHERE STUDENTID <='111';

!<	NOT LESS THAN	If LHS value is NOT less than the RHS value, then only the condition is considered as true.	SELECT * FROM DATACEPSSTUDENTS WHERE COURSEJOININGDATE !<'2019-01-12';
! >	NOT GREATER THAN	If LHS value is NOT greater than the RHS value, then only the condition is considered as true.	SELECT * FROM DATACEPSSTUDENTS WHERE STUDENTID ! >'111';

LOGICAL OPERATORS

OPERATOR	DESCRIPTION	EXAMPLE
ALL	If the values in the subquery meets the condition, then it returns TRUE	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSEJOININGDATE ='2021-01-04';`
ANY	If any of the values in the subquery satisfies the condition then it returns TRUE.	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSENAME = ANY (SELECT COURSENAME FROM COURSEDETAILS WHERE COURSEPRICE <'149');`
AND	If all the condition present either side of the operator is satisfied and meets the condition, then only it is returns TRUE	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSENAME='SQL COURSE' AND FIRSTNAME='DANE'`
BETWEEN	Filter the query results and only presents the results that are in the mentioned range.	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSEJOININGDATE BETWEEN '2019-01-01' AND '2019-12-12'`
EXISTS	Checks if the results of subquery exists in the main query results.	`SELECT * FROM DATACEPSSTUDENTS WHERE EXISTS (SELECT COURSENAME FROM COURSEDETAILS WHERE COURSEPRICE='49' AND DATACEPSSTUDENTS.COURSENAME=CO URSEDETAILS.COURSENAME);`
IN	It check and return rows that matches with the expressions or values present IN the list mentioned.	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSENAME IN ('PYTHON MASTERY','C++');`
LIKE	LIKE operator checks for the mentioned pattern in the result set of the query.	`SELECT * FROM DATACEPSSTUDENTS WHERE FIRSTNAME LIKE '%SA%';`

NOT	NOT returns the results that for which the given condition is FALSE or NOT TRUE.	`SELECT * FROM DATACEPSSTUDENTS WHERE FIRSTNAME NOT LIKE '%SA%';`
OR	OR returns the records that matches any of the condition separated by the OR operator	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSEJOININGDATE >='2019-01-12' OR FIRSTNAME LIKE 'M%';`
SOME	SOME operator returns records when any of the values present in the subquery meets the condition.	`SELECT * FROM DATACEPSSTUDENTS WHERE COURSENAME = SOME (SELECT COURSENAME FROM COURSEDETAILS WHERE COURSEPRICE >'49');`

Wildcards for LIKE operator

LIKE operator can be combined with below wildcards:

- **Percentage Sign (%):** % can represent 0, 1 or even multiple characters.
- **Underscore sign (_):** _ represents a single character.

Example #1:

```
SELECT ColumnName1,
ColumnName2 . . . . ColumnName (n)
WHERE ColumnName1 LIKE 'z%' ;
```

The above query will find values in column name that starts with letter 'z'

Example #2:

```
SELECT ColumnName1,
ColumnName2 . . . . ColumnName (n)
WHERE ColumnName1 LIKE '_a%'
```

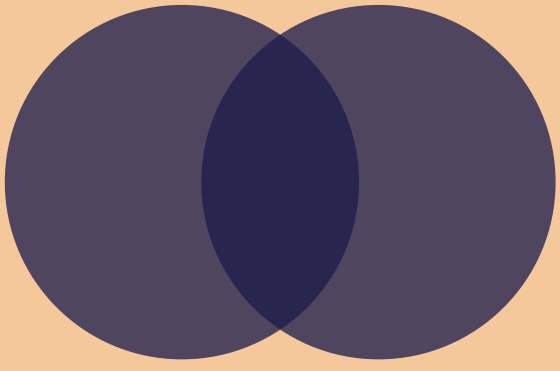
The above query will find values in column name where the second letter is 'a'

SET OPERATORS

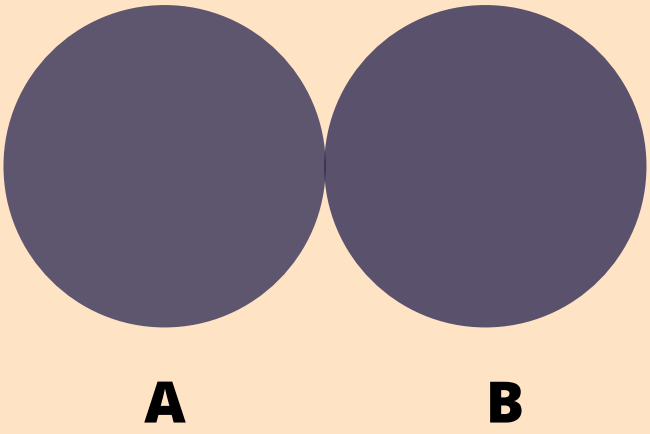
OPERATOR	OPERATION	SYNTAX
UNION	Combines result set of two or more queries into one. Duplicates are eliminated.	`SELECT COLUMNNAME1 FROM NAMEOFTABLE2 UNION SELECT COLUMNNAME2 FROM NAMEOFTABLE2;`
UNION ALL	Combines result set of two or more queries into one. But Duplicates are not eliminated	`SELECT COLUMNNAME1 FROM NAMEOFTABLE2 UNION SELECT COLUMNNAME2 FROM NAMEOFTABLE2;`
INTERSECT	Returns the data that is present in both the result set.	`SELECT COLUMNNAME1 FROM NAMEOFTABLE2 INTERSECT SELECT COLUMNNAME2 FROM NAMEOFTABLE2;`
MINUS	Displays results that are present in first result set and are not present in the second one.	`SELECT COLUMNNAME1 FROM NAMEOFTABLE2 MINUS SELECT COLUMNNAME2 FROM NAMEOFTABLE2;`

SET OPERATORS (ILLUSTRATIONS)

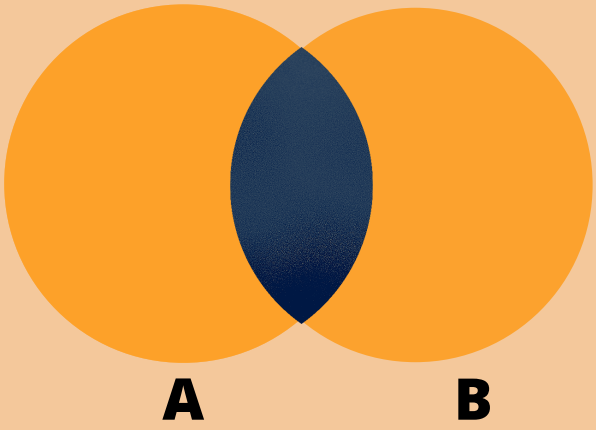
A UNION **B**



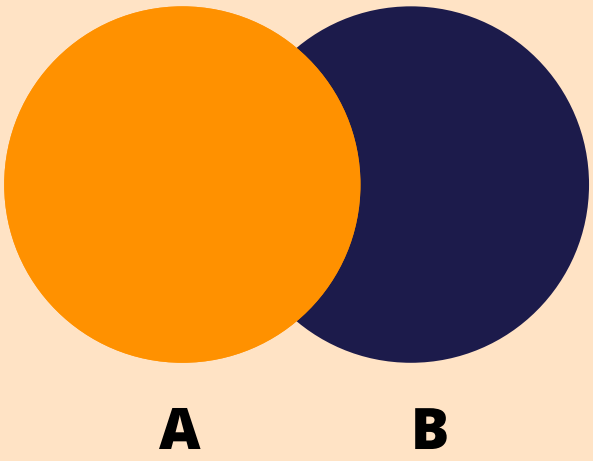
A UNION ALL **B**



A INTERSECT **B**



A MINUS **B**



FUNCTIONS IN SQL

SQL FUNCTIONS basically reusable subprograms that helps in achieving a particular operation like—concatenation, calculation and string operations.
Following are the different types of SQL Functions that can be used:

SQL AGGREGATE FUNCTIONS

Aggregate functions are used for doing calculations on multiple rows of a column and returns a single output.

Below are some aggregate functions:

COUNT (): Counts the number of records present in a column.

Syntax:

```
SELECT
COUNT(NameOfTheColumn)
FROM NameOfTheTable;
```

SUM () : Calculates the sum of all the values of the selected column on which sum () function is applied.

Syntax:

```
SELECT SUM(NameOfTheColumn)
FROM NameOfTheTable;
```

AVG () : Calculates the average of all the values of the selected column on which avg() function is applied.

Syntax:

```
SELECT AVG(NameOfTheColumn)
FROM NameOfTheTable;
```

MAX () : Calculates the maximum value present in the selected column on which the max () function is applied.

Syntax:

```
SELECT MAX(NameOfTheColumn)
FROM NameOfTheTable;
```

MIN () : Calculates the minimum value present in the selected column on which the min () function is applied.

Syntax:

```
SELECT MIN(NameOfTheColumn)
FROM NameOfTheTable;
```

SQL STRING FUNCTIONS

OPERATOR

ASCII

OPERATION

Returns the ASCII value of the character in the function

SYNTAX

```
ASCII ('Character');
```

EXAMPLE

```
SELECT ASCII('S');
```

RESULT: ASCII value of 'S' is 83

OPERATOR

CHAR

OPERATION

Returns the character value of an ASCII value.

SYNTAX

```
CHAR ( ASCII Code )
```

EXAMPLE

```
SELECT CHAR(83);
```

RESULT: CHAR value of ASCII 83 is S.

OPERATOR

UNICODE

OPERATION

Returns the character value of an ASCII value.

SYNTAX

```
UNICODE ('Character')
```

EXAMPLE

```
SELECT UNICODE ( 'B' );
```

RESULT: 66

OPERATOR

LEN

OPERATION

Returns the length of a given string.

SYNTAX

```
LEN('String Value')
```

EXAMPLE

```
SELECT LEN ('Dataceps') AS CharacterLegth;
```

RESULT: 8

OPERATOR

CHARINDEX

OPERATION

Searches a substring inside a given string and returns the position of that substring.

SYNTAX

```
CHARINDEX ('Substring you want to search',
'String where you want to search')
```

EXAMPLE

```
SELECT CHARINDEX ('e', 'DataCeps') AS
CharacterLocation;
```

RESULT: 6

OPERATOR

PATINDEX

OPERATION

Returns to position of a given string pattern.

SYNTAX

```
PATINDEX ( %StringPattern%, 'String')
```

WHERE:

%StringPattern**%** =>String pattern that you want to search.
'String' =>String where you want to search your pattern

EXAMPLE

```
SELECT PATINDEX('%Ceps%', 'DataCeps.com');
```

RESULT: 5

OPERATOR

LEFT

OPERATION

Returns the characters from a string starting from left to right, based on the value in count.

SYNTAX

```
LEFT ('String', CharacterCount)
```

EXAMPLE

```
SELECT LEFT('DataCeps', 3);
```

RESULT: Dat

OPERATOR

RIGHT

OPERATION

Returns the characters from a string starting from right to left, based on the value in count.

SYNTAX

```
RIGHT('String', CharacterCount)
```

EXAMPLE

```
SELECT RIGHT('DataCeps', 3);
```

RESULT: eps

OPERATOR

LTRIM

OPERATION

Removes the extra spaces from the left hand side (LHS) of a string value.

SYNTAX

```
LTRIM(' String')
```

EXAMPLE

```
SELECT LTRIM(' DataCeps');
```

RESULT: DataCeps

OPERATOR

RTRIM

OPERATION

Removes the extra spaces from the right hand side (RHS) of a string value.

SYNTAX

```
RTRIM ( 'String' )
```

EXAMPLE

```
SELECT RTRIM('DataCeps   ');
```

RESULT: DataCeps

OPERATOR

REPLACE

OPERATION

Replaces all occurrences of string value within a string with another value.

SYNTAX

```
REPLACE('MainString', 'OldStringValue',
        'NewStringValue')
```

EXAMPLE

```
SELECT REPLACE('DataCeps', 'a', 'e');
```

RESULT: DeteCeps

OPERATOR

REPLICATE

OPERATION

Returns string values based on the number value given in the function.

SYNTAX

REPLICATE ('String ', Number)

Number => Number of times you want the string to repeat.

EXAMPLE

```
SELECT REPLICATE('DataCeps.com',3);
```

RESULT:

DataCeps.comDataCeps.comDataCeps.com

OPERATOR

REVERSE

OPERATION

Returns the reversed value of the string that's mentioned in the function.

SYNTAX

REVERSE ('String') ;

EXAMPLE

```
SELECT REVERSE ('DataCeps');
```

RESULT: **speCataD**

OPERATOR

QUOTENAME

OPERATION

Returns the string after adding square bracket after and before the string value.

SYNTAX

```
QUOTENAME('String')
```

OR

```
QUOTENAME('String', 'QuoteCharacter');
```

Here, The value of **QuoteCharacter** is optional.
Below are the options from where you can choose the values for it:

- **Square brackets ([ ])**
- **Parenthesis or round brackets (())**
- **Angle brackets (< >)**
- **curly brackets ({ })**

EXAMPLE

```
SELECT QUOTENAME('DataCeps');
```

RESULT:

[DataCeps]

OPERATOR

SPACE

OPERATION

Returns the spaces based on the number mentioned in the function

SYNTAX

```
SPACE(Number)
```

EXAMPLE

```
SELECT SPACE(5);
```

RESULT:

5 spaces. (*because we can't spaces in this book*)

OPERATOR

STR

OPERATION

Coverts a numerical value to a string value.

SYNTAX

```
STR(NUMBER)
```

EXAMPLE

```
SELECT 'DataCeps'+STR(2022);
```

RESULT: DataCeps 2022

OPERATOR

STUFF

OPERATION

Removes a particular part of the main string and inserts the desired new string in that position.

SYNTAX

```
STUFF('MainString', StartingPosition,
TotalLength, 'NewString');
```

Where, MainString => String where you want to make changes or do the stuffing.

StartingPosition =>The place from where your new string will start replacing the main string.

TotalLength =>Total length your new string will cover in the existing main string space.

NewString => The new string that you want to stuff or replace in the existing one.

EXAMPLE

```
SELECT STUFF('.com', 1, 0, 'DataCeps');
```

RESULT: **DataCeps .com**

OPERATOR

SUBSTRING

OPERATION

Returns a substring from a main string.

SYNTAX

```
SUBSTRING('MainString',
StartingPostion, TotalLength)
```

Where,

'MainString' => is the string from where you want to extract the substring.

StartingPostion => is the Starting point in the main string from where you want to extract your substring

TotalLength => is the total length of the characters you want to extract from the main string for your substring.

OPERATOR

LOWER

OPERATION

Converts the characters of a mentioned string in lower case

SYNTAX

```
LOWER('String')
```

EXAMPLE

```
SELECT LOWER('DataCeps');
```

RESULT: **dataceps**

OPERATOR

UPPER

OPERATION

Converts the characters of a mentioned string in upper case

SYNTAX

```
UPPER ('String')
```

EXAMPLE

```
SELECT UPPER('DataCeps');
```

RESULT: **DATACEPS**

SOUNDEX

OPERATION

Returns a 4 character code for a given string value, based on the SOUND of the string.

SYNTAX

```
SOUNDEX('String')
```

EXAMPLE

```
SELECT SOUNDEX('DataCeps');
```

RESULT:

D321

OPERATOR

DIFFERENCE

OPERATION

Returns the difference between the string SOUNDEX values on the scale of 0 to 4.

SYNTAX

```
DIFFERENCE('String1','String2')
```

Where, in results the value 0 represents lowest similarity.
And, 4 represents highest similarity.

EXAMPLE

```
SELECT
DIFFERENCE('DataCeps','DataCaps');
```

RESULT: 4

OPERATOR

CONCAT

OPERATION

Returns the addition/concatenation of two or more string values

SYNTAX

```
CONCAT('String1', 'String2'..... 'String (n)')
```

EXAMPLE

```
SELECT CONCAT('Data', 'Ceps', '.com');
```

RESULT: DataCeps.com

OPERATOR

TRIM

OPERATION

Removes extra spaces from both LHS and RHS of a given string value

SYNTAX

```
TRIM('    String    ')
```

EXAMPLE

```
SELECT TRIM ('  DataCeps   ')
```

RESULT: DataCeps

OPERATOR

TRANSLATE

OPERATION

Returns the string values present in the 1st string after Translating/Replacing the string values present in the 2nd argument with the string values present in the 3rd argument.

SYNTAX

```
TRANSLATE ('Main String',
'StringToReplace', 'ReplaceWith')
```

Where,

'Main String' => is the main string where you want to make changes/replacement.

'StringToReplace' => String value that you want to replace from the main string.

'ReplaceWith'=> Values that you want replace in the main string.

EXAMPLE

```
SELECT TRANSLATE('[DataCeps]', '[]', '{}');
```

RESULT: **{DataCeps}**

OPERATOR

REPLICATE

OPERATION

Replicates or Repeats a given string based on the input value we provide in the arguments.

SYNTAX

```
REPLICATE('String',ReplicationNumber);
```

Where,

'String'=>The string value that you want to replicate.

ReplicationNumber=> The number of times you want to replicate the string value

EXAMPLE

```
SELECT REPLICATE('DataCeps.com ', 3);
```

RESULT:

DataCeps.com
DataCeps.com
DataCeps.com

OPERATOR

NCHAR

OPERATION

Returns the UNICODE character of a specified number code.

SYNTAX

```
NCHAR(NumberCode)
```

Where,
NumberCode=> Unicode standard number code.

EXAMPLE

```
SELECT NCHAR(87);
```

RESULT: W

OPERATOR

DATALENGTH

OPERATION

Returns the number of bytes the value or expression is taking.

SYNTAX

```
DATALENGTH('StringValue'/Expression);
```

EXAMPLE

```
SELECT DATALENGTH('www.DataCeps.com');
```

RESULT: 16

OPERATOR

CONCAT_WS

OPERATION

Returns the addition/ concatenation of two or more string values.
But separated by a given separator

SYNTAX

```
CONCAT_WS('Separator', 'String1',
'String1','String1'. . . . 'String
(n)')
```

EXAMPLE

```
SELECT
CONCAT_WS ('+', 'SQL',
'PRACTICE','REPETITION');
```

RESULT: **SQL+PRACTICE+REPETITION**

OPERATOR

CONCAT USING +

OPERATION

Returns the addition/
concatenation of two or more string values by using + operator.

SYNTAX

```
'String1' + 'String2' +
'String3'+.....+ 'String (n)';
```

EXAMPLE

```
SELECT 'Data' + 'Ceps' + '.com';
```

RESULT: **DataCeps.com**

DATE AND TIME FUNCTIONS

OPERATOR

```
GETDATE();
```

OPERATION

Returns the current system date along with time in the below format:

YYYY-MM-DD hh:mm:ss.mmm

EXAMPLE

```
SELECT GETDATE();
```

RESULT: **2021-11-28 17:32:54.087**

OPERATOR

CURRENT_TIMESTAMP

OPERATION

Returns the current system date along with time in the below format:

YYYY-MM-DD hh:mm:ss.mmm

EXAMPLE

```
SELECT CURRENT_TIMESTAMP;
```

RESULT: **2021-11-28 17:35:33.930**

OPERATOR

SYSDATETIME()

OPERATION

Displays the system date along with time of the system where SQL server instance is running.

SYNTAX

```
SELECT SYSDATETIME();
```

RESULT: **2021-11-28 17:37:20.2256749**

OPERATOR

GETUTCDATE()

OPERATION

Returns the current system UTC date along with time in the below format:

YYYY-MM-DD hh:mm:ss.mmm

EXAMPLE

```
SELECT GETUTCDATE();
```

RESULT: **2021-11-28 12:10:45.000**

OPERATOR

SYSDATETIMEOFFSET()

OPERATION

Returns the current date along with time and the time zone of the system on which the SQL server is running.

EXAMPLE

```
SELECT SYSDATETIMEOFFSET();
```

RESULT: **2021-11-28 17:44:20.4460466 -05:00**

OPERATOR

SYSUTCDATETIME()

OPERATION

Returns the current date along with time of the system on which the SQL server is running, in DATETIME2 format.

EXAMPLE

```
SELECT SYSUTCDATETIME();
```

RESULT: **2021-11-28 12:18:57.4662726**

FUNCTIONS THAT RETURN PART OF A DATE AND TIME

OPERATOR

DATENAME

OPERATION

Returns a part of the date, in STRING datatype.

SYNTAX

```
DATENAME (PartOfDate, YourDate)
```

Where, **PartOfDate =** The part of date that you want to display or return.
It can be from below mentioned values**:**

yy,yyyy year ➜ To get year from the date.

mm,m,month ➜ To get the month from the date.

dy,y,day ➜ To get day from the date.

qq,q, quarter ➜ To get the quarter from the mentioned date.

dayofyear ➜ To get the day of the current year for the mentioned date.

wk,ww,week ➜ to get the week number for the date mentioned.

dw,w, weekday ➜ To get the weekday of the date mentioned.

hh,hour ➜ To get the hour from mentioned time.

n, mi, minute ➜ To get the minute from the mentioned time.

s, ss, second ➜ To get the seconds from the mentioned time.

ms, millisecond ➜ To get the millisecond from the mentioned time.

YourDate ➜ The date you want to get values from.

OPERATOR

DATEPART

OPERATION

Returns a part of the date, in INTEGER datatype.

SYNTAX

```
DATEPART(PartOfDate, YourDate)
```

Where, **PartOfDate =** The part of date that you want to display or return.
It can be from below mentioned values**:**

yy,yyyy year → To get year from the date.

mm,m,month → To get the month from the date.

dy,y,day → To get day from the date.

qq,q, quarter → To get the quarter from the mentioned date.

dayofyear → To get the day of the current year for the mentioned date.

wk,ww,week → to get the week number for the date mentioned.

dw,w, weekday → To get the weekday of the date mentioned.

hh,hour → To get the hour from mentioned time.

n, mi, minute → To get the minute from the mentioned time.

s, ss, second → To get the seconds from the mentioned time.

ms, millisecond → To get the millisecond from the mentioned time.

YourDate → The date you want to get values from.

EXAMPLE #1

```
SELECT DATEPART (YYYY,'2020-10-25 19:28:05.200');
```

RESULT: 2020

EXAMPLE #2

```
SELECT DATEPART (Q,'2020-10-25 19:28:05.200');
```

RESULT: 4

OPERATOR

DAY

OPERATION

Returns the day from the mentioned date.

SYNTAX

```
DAY(date)
```

EXAMPLE

```
SELECT DAY('2020-10-25');
```

RESULT: 25

OPERATOR

MONTH

OPERATION

Returns the month from the mentioned date.

SYNTAX

```
MONTH (date)
```

EXAMPLE

```
SELECT MONTH('2020-10-25');
```

RESULT: 10

OPERATOR

YEAR

OPERATION

Returns the year from the mentioned date.

SYNTAX

```
YEAR(date)
```

EXAMPLE

```
SELECT YEAR('2020-10-25');
```

RESULT: 2020

FUNCTIONS TO MODIFY DATES

OPERATOR

DATEADD

OPERATION

Adds number to the part of a date to increase the interval.

SYNTAX

```
DATEADD (PartOfDate,value, YourDate)
```

Where,

PartOfDate: The part of date that you want to display or return.
It can be from below mentioned values:

yy,yyyy year → To get year from the date.

mm,m,month → To get the month from the date.

dy,y,day → To get day from the date.

qq,q, quarter → To get the quarter from the mentioned date.

dayofyear → To get the day of the current year for the mentioned date.

wk,ww,week → to get the week number for the date mentioned.

dw,w, weekday → To get the weekday of the date mentioned.

hh,hour → To get the hour from mentioned time.

n, mi, minute → To get the minute from the mentioned time.

s, ss, second → To get the seconds from the mentioned time.

ms, millisecond → To get the millisecond from the mentioned time.

YourDate → The date you want to get values from.

EXAMPLE #1

```
SELECT DATEADD(yyyy, 3, '2020-10-25');
```

RESULT: 2023-10-25 00:00:00.000

EXAMPLE #2

```
SELECT DATEADD(mm, 3, '2020-10-25');
```

RESULT: 2021-01-25 00:00:00.000

EXAMPLE #3

```
SELECT DATEADD(d, 3, '2020-10-25');
```

RESULT: 2020-10-28 00:00:00.000

OPERATOR

DATEDIFF

OPERATION

Returns the difference between the two mentioned date values

SYNTAX

```
DATEDIFF( PartOfDate , Startdate , Enddate)
```

Where,

Startdate , Enddate : The dates to calculate the difference between

PartOfDate: The part of date that you want to display or return.

It can be from below mentioned values:

yy,yyyy year → To get year from the date.

mm,m,month → To get the month from the date.

dy,y,day → To get day from the date.

qq,q, quarter → To get the quarter from the mentioned date.

dayofyear → To get the day of the current year for the mentioned date.

wk,ww,week → to get the week number for the date mentioned.

dw,w, weekday → To get the weekday of the date mentioned.

hh,hour → To get the hour from mentioned time.

n, mi, minute → To get the minute from the mentioned time.

s, ss, second → To get the seconds from the mentioned time.

ms, millisecond → To get the millisecond from the mentioned time.

YourDate → The date you want to get values from.

EXAMPLE #1

```
SELECT DATEDIFF( YYYY , '2020-10-25' , '2021-10-29')
```

RESULT: 1

EXAMPLE #2

```
SELECT DATEDIFF( YYYY , '2020-10-25' , '2021-10-29')
```

RESULT: 12

NUMERIC FUNCTIONS/MATH FUNCTIONS

SAMPLE DATA
CourseDetails

CID	CourseName	CoursePrice
C1	SQL Course	$99.0
C2	Python Mastery	$149.0
C3	C++	$49.0

OPERATOR

ACOS

OPERATION

Returns the Arc Cosine of the mentioned number.

SYNTAX

```
ACOS (Number)
```

Where, The Range of number you want to find the arc cosine starts from -1 to 1.

EXAMPLE

```
SELECT ACOS(-1);
```

RESULT: 3.14159265358979

OPERATOR

ASIN

OPERATION

Returns the Arc sine of the mentioned number.

SYNTAX

```
ASIN (Number)
```

Where, The Range of number you want to find the arc cosine starts from -1 to 1.

EXAMPLE

```
SELECT ASIN(-1);
```

RESULT: -1.5707963267949

OPERATOR

ATAN

OPERATION

Returns the Arc tangent of the mentioned number.

OPERATION

```
ATAN (Number)
```

Where, The Range of number you want to find the arc cosine starts from -1 to 1.

EXAMPLE

```
SELECT ATAN(-1);
```

RESULT: -0.785398163397448

OPERATOR

AVG

OPERATION

Returns the average of the value mentioned in the parameter of the function

SYNTAX

```
AVG (value)
```

Where ,
Value= any expression, formula or a column

EXAMPLE

```
SELECT AVG(CoursePrice) FROM CourseDetails;
```

RESULT: 99.00

OPERATOR

ATN2

OPERATION

Returns the Arc tangent of the two mentioned number.

OPERATION

```
ATN2(FirstNumber, SecondNumber)
```

Where, The Range of number you want to find the arc tangent . starts from -1 to 1.

EXAMPLE

```
SELECT ATN2(1, -1);
```

RESULT: 2.35619449019234

OPERATOR

CEILING

OPERATION

Returns an integer value that is bigger than or equal to the mentioned value in the parameter.

SYNTAX

```
CEILING (Number)
```

EXAMPLE

```
SELECT CEILING(7.75);
```

RESULT: 8

OPERATOR

COUNT

OPERATION

Returns the total number of records present in a table for a column.

SYNTAX

```
COUNT (Expression or Column);
```

EXAMPLE

```
SELECT COUNT (CID) FROM CourseDetails;
```

RESULT: 3

OPERATOR

COS

OPERATION

Returns the cosine of the mentioned number.

SYNTAX

```
COS (Number)
```

EXAMPLE

```
SELECT COS(1);
```

RESULT: 0.540302305868140

OPERATOR

COT

OPERATION

Returns the cotangent of the mentioned number.

SYNTAX

```
COT(Number)
```

EXAMPLE

```
SELECT COT(1);
```

RESULT: 0.642092615934331

OPERATOR

DEGREES

OPERATION

Converts the mentioned value from radians to degrees.

SYNTAX

```
DEGREES (Number)
```

EXAMPLE

```
SELECT DEGREES(3);
```

RESULT: 171

OPERATOR

EXP

OPERATION

Returns the exponential value of a mentioned value

SYNTAX

```
EXP(Number)
```

Where, **Number** will be used to get the exponential value. By determining e raised to the power **Number**.
e is a mathematical constant value — 2.71828182845905

EXAMPLE

```
SELECT EXP(1);
```

That means **e** raised to power 1.

RESULT: 2.71828182845905

OPERATOR

FLOOR

OPERATION

Returns biggest integer value that is smaller than or equal to the mentioned value in the parameter.

SYNTAX

```
FLOOR (Number)
```

EXAMPLE

```
SELECT FLOOR(77.34);
```

RESULT: 77

OPERATOR

LOG

OPERATION

Returns the logarithm of a mentioned number.

SYNTAX

```
LOG (Number, BaseForLog)
```

EXAMPLE

```
SELECT LOG (4, 16);
```

RESULT: 0.5

OPERATOR

LOG10

OPERATION

Returns the base 10 logarithm of a mentioned number.

SYNTAX

```
LOG10 (Number)
```

EXAMPLE

```
SELECT LOG10(10000);
```

RESULT: 4

OPERATOR

PI

OPERATION

Returns the value of PI.

SYNTAX

```
PI()
```

EXAMPLE

```
SELECT PI();
```

RESULT: 3.141592653589793

OPERATOR

POWER

OPERATION

Returns the value of mentioned base number raised to the power of mentioned exponent number.

SYNTAX

```
POWER (BaseNumber,ExponentNumber)
```

EXAMPLE

```
SELECT POWER(2, 2);
```

RESULT: 49.00

OPERATOR

RADIANS

OPERATION

Returns radians by converting from degree value.

SYNTAX

```
RADIANS (Number)
```

EXAMPLE

```
SELECT RADIANS(90);
```

RESULT: 1

OPERATOR

RAND

OPERATION

Returns a random decimal value.

SYNTAX

```
RAND()
```

EXAMPLE

```
SELECT RAND();
```

RESULT: 0.291524901550145

OPERATOR

ROUND

OPERATION

Returns a rounded value of a number based on mentioned decimal places.

SYNTAX

```
ROUND(Number, DecimalPlace, Operation)
```

Where, **Number**= The number you want to round.

DecimalPlace= Decimal places you want to round your number.

Operation= Default value is 0. That means, it rounds the result to the number of decimal.
If other than 0, then truncates the results to the number of decimals.

EXAMPLE

```
SELECT ROUND(123.321252123, 3, 0);
```

RESULT: **123.321000000**

OPERATOR

SIGN

OPERATION

Returns the sign of the mentioned value.

When the mentioned value is >1, then it returns 1.

When the mentioned value is <1, then it returns -1.

When the mentioned value is equal 1, then it returns 1.

SYNTAX

```
SIGN (Number)
```

EXAMPLE

RESULT: `SELECT SIGN (-234);`

OPERATOR

SIN

OPERATION

Returns sine value of mentioned number.

SYNTAX

```
SIN (Number)
```

EXAMPLE

RESULT: `SELECT SIN(-3);`

OPERATOR

SQRT

OPERATION

Returns the square root of a mentioned number.

SYNTAX

```
SQRT (Number)
```

EXAMPLE

```
SELECT SQRT(144);
```

RESULT: 12

OPERATOR

SQUARE

OPERATION

Returns the square of the mentioned number.

SYNTAX

```
SQUARE (Number)
```

EXAMPLE

```
SELECT SQUARE(12);
```

RESULT: 144

OPERATOR

SUM

OPERATION

Returns the sum of the mentioned column or numbers or expressions.

SYNTAX

```
SUM (Number or Expressions)
```

EXAMPLE

```
SELECT SUM(CoursePrice) FROM CourseDetails;
```

RESULT: 297.00

OPERATOR

TAN

OPERATION

Returns Tangent of the mentioned number.

SYNTAX

```
TAN (Number)
```

EXAMPLE

```
SELECT TAN(-7);
```

RESULT: -0.871447982724319

ADVANCED FUNCTIONS

OPERATOR

CAST

OPERATION

Can convert one type datatype to another datatype.

SYNTAX

```
CAST (Expression AS NewDatatype(length))
```

EXAMPLE

```
SELECT CAST(46.12 AS INT);
```

RESULT : 46

OPERATOR

TRY_CAST

OPERATOR

Works similar to the CAST function. That means also converts one type of datatype to another one.
The only difference is that if the TRY_CAST function fails to convert then it returns NULL value.

SYNTAX

```
TRY_CAST (Expression AS NewDatatype(length))
```

EXAMPLE

```
SELECT TRY_CAST (46.12 AS INT);
```

RESULT: 46

OPERATOR

COALESCE

OPERATION

This function helps in handling NULL values. It evaluates given values and expressions , figures out the first NOT NULL value in it and returns that NOT NULL value.

SYNTAX

```
COALESCE(Expression1,Expression2,Expression3,....
                .. Expression(n))
```

EXAMPLE

```
SELECT COALESCE (NULL,NULL,'Welcome
To',NULL,'DataCeps','SQL','Book');
```

RESULT: Welcome To

OPERATOR

CONVERT

OPERATION

Converts the value of any datatype to the desired datatype.

SYNTAX

```
CONVERT(NewDatatype(length), Expression, style);
```

Where, **NewDatatype**= The desired new datatype you want to convert your values into.
Expression= The actual value whose datatype you wish to convert.

Style= Its Optional , used to identify how the function will convert the expression.

EXAMPLE

```
SELECT CONVERT(INT, 425.134);
```

RESULT: 425

OPERATOR

TRY_CONVERT

OPERATION

Converts the value of any datatype to the desired datatype.
if the TRY_CONVERT function fails to convert then it returns NULL value.

Where, **NewDatatype**= The desired new datatype you want to convert your values into.
Expression= The actual value whose datatype you wish to convert.

Style= Its Optional , used to identify how the function will convert the expression.

EXAMPLE

```
SELECT TRY_CONVERT(INT, 425.134);
```

RESULT: 425

EXAMPLE

CURRENT_USER

OPERATION

Returns the name of the user that is currently working on the SQL Server.

SYNTAX

```
SELECT CURRENT_USER;
```

EXAMPLE

```
SELECT CURRENT_USER;
```

RESULT: dane

OPERATOR

IIF

OPERATION

IFF function returns TRUE if the condition mentioned inside the function is TRUE.
If the condition is false then the function returns false

SYNTAX

```
IIF (Condtion, Value_IfTrue, Value_IfTrue)
```

EXAMPLE

```
SELECT IIF (1300>100,'True','False');
```

RESULT: True

OPERATOR

ISNULL

OPERATION

If the value in the in the ISNULL function is NULL then it returns the expression. If the expression has some value then the function returns the expression itself.

SYNTAX

```
ISNULL ( MainExpression, Value)
```

EXAMPLE

```
SELECT ISNULL('SQL Course', 'DataCeps.com');
```

OPERATOR

ISNUMERIC

OPERATION

Returns 1 if the expression inside the function is a numerical value. Returns 0 when the expression inside the function is not a numerical value.

SYNTAX

```
ISNUMERIC (expression)
```

EXAMPLE

```
SELECT ISNUMERIC ('string');
```

RESULT: 0

OPERATOR

NULLIF

OPERATION

Returns NULL if both the expression inside the function are equal. If the expression inside the function are not equal then it returns the first expression present in the function.

SYNTAX

```
NULLIF (Expression1, Expression2)
```

EXAMPLE #1

```
SELECT NULLIF('DataCeps.com', 'DataCeps.com');
```

RESULT: NULL

EXAMPLE #2

```
SELECT NULLIF (112,223);
```

RESULT: 112

OPERATOR

SESSIONPROPERTY

OPERATION

Returns the settings of the mentioned option.

Returns 1 if the mentioned session property is on, if not returns 0.

SYNTAX

SESSIONPROPERTY (OptionSettings)

Where,
Values for OptionSettings can be any one of the below mentioned options:

- ANSI_PADDING
- ANSI_WARNINGS
- ARITHABORT
- NUMERIC_ROUNDABOUT
- QUOTED_IDENTIFIER
- CONCAT_NULL_YIELDS_NULL
- ANSI_NULLS

EXAMPLE

```
SELECT SESSIONPROPERTY ('ANSI_NULLS');
```

RESULT:

```
1
```

OPERATOR

SYSTEM_USER

OPERATION

Returns the login name of the user that's currently logged into SQL server.

SYNTAX

```
SYSTEM_USER
```

EXAMPLE

```
SELECT SYSTEM_USER;
```

RESULT: **LP-DP5M2DAN\danew**

OPERATOR

SESSION_USER

OPERATION

Returns the username of the current user in the SQL server instance.

SYNTAX

```
SESSION_USER
```

EXAMPLE

```
SELECT SESSION_USER;
```

RESULT: dane

OPERATOR

USER_NAME

OPERATION

Returns the database username

SYNTAX

```
USER_NAME()
```

EXAMPLE

```
SELECT USER_NAME();
```

RESULT: dane

CHAPTER 8

WORKING WITH MULTIPLE TABLES

JOINS

Joins helps in combining two or more table's data or records based on a related field between the tables

INNER JOIN

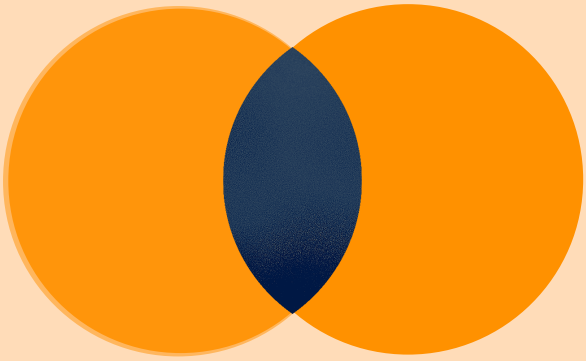
Tables with INNER JOIN selects the records that are common in both the tables.

SYNTAX

```
SELECT ColumnNames
FROM TableA
INNER JOIN TableB
ON TableA.Column1=TableB.Column2;
```

ILLUSTRATION

TableA INNER JOIN **TableB**



TableA **TableB**

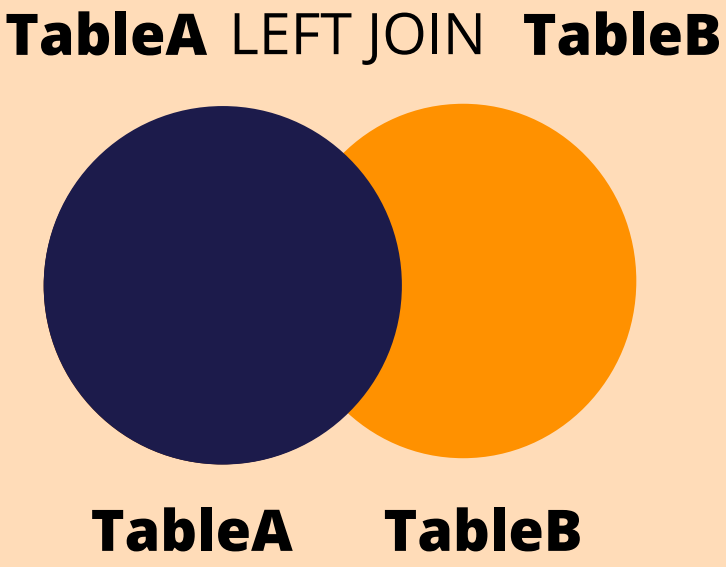
LEFT JOIN or LEFT OUTER JOIN

When tables are in LEFT JOIN then it selects and displays all the data from the left table and only the matching records from the right table.

SYNTAX

```
SELECT ColumnNames
FROM TableA
LEFT JOIN TableB
ON TableA.Column1=TableB.Column2;
```

ILLUSTRATION



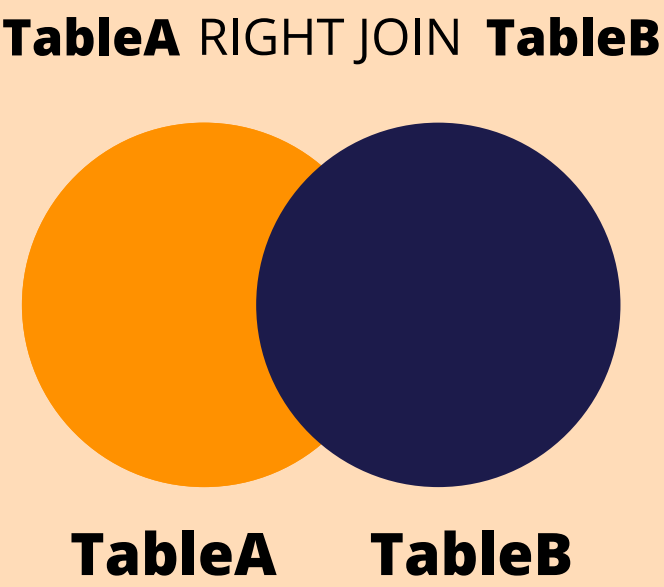
RIGHT JOIN or RIGHT OUTER JOIN

When the tables are in RIGHT JOIN then it selects and displays all the data from the right table and only the matching rows in the left table.

SYNTAX

```
SELECT ColumnNames
FROM TableA
RIGHT JOIN TableB
ON TableA.Column1=TableB.Column2;
```

ILLUSTRATION



FULL JOIN or FULL OUTER JOIN

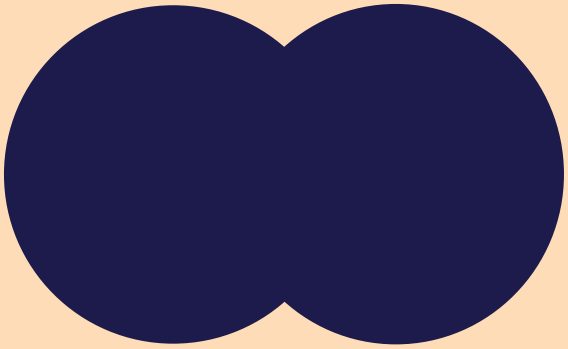
When the tables are in FULL OUTER JOIN then it selects and displays all combinations of LEFT JOIN and RIGHT JOINS.

SYNTAX

```
SELECT ColumnNames
FROM TableA
FULL JOIN TableB
ON TableA.Column1=TableB.Column2;
```

ILLUSTRATION

TableA FULL JOIN **TableB**



TableA **TableB**

SELF JOIN

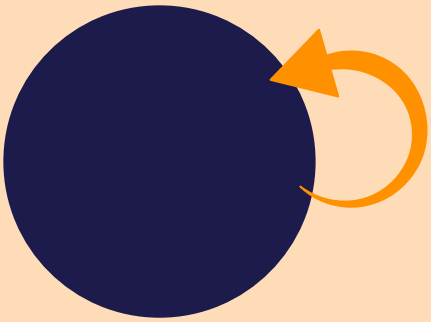
Self-join is a simple join in which a table is joined with itself.

SYNTAX

```
SELECT T1.ColumnA,T2.ColumnA
FROM TableA T1, TableA T2
WHERE T1.ColumnA=T2.ColumnA;
```

ILLUSTRATION

TableA SELF JOIN **TableA**



TableA

COMMON TABLE EXPRESSIONS (CTE)

Common Table Expression (CTE) creates a temporary result set that can be then referenced in SQL statements like—INSERT, UPDATE, DELETE and SELECT etc.
CTE helps in increasing the code readability and maintenance.
Helps in creating recursive queries.

SYNTAX

```
WITH CTEName (Column1, Column2, Column3. . . . Column(n))
AS (Query_For_CTE)
INSERT, UPDATE, SELECT or DELETE Statements
```

RECURSIVE COMMON TABLE EXPRESSIONS

A recursive Common Table Expression has ability to reference itself.
There are 3 main parts to a recursive CTE:

Invocation: This is the initial part of the CTE that returns the base result of a query.
Recursion: The part of the CTE that calls itself recursively.
Termination condition: The part of CTE that keeps on checking if the recursion needs to execute one more time or needs to be stopped.

SYNTAX

```
WITH CTEExpression (Column1,Column2,Column3. . . . Column(n))
AS
(
    InitialQuery  -- Anchor member
    UNION ALL
    -- Recursive member that references CTEExpression.
    RecursiveQuery
)
-- References expression name
SELECT *
FROM   CTEExpression;
```

CHAPTER 9

VIEWS AND INDEXES

VIEWS

View is a virtual table that has the dataset that is extracted from underlying tables based on the predefined conditions and requirements. It only presents us the data that we want to see. The multiple underlying tables might have multiple joins or functions that helps in creating a dataset that we want to see. However, the end user still might think that the data is coming from a single database object.
The data that is presented in the view is always up-to-date.

CREATE VIEWS

SYNTAX

```
CREATE VIEW ViewName AS
SELECT Column1, Column2, ...Column(n)
FROM TableName
WHERE Condition;
```

UPDATING VIEW

SYNTAX

```
CREATE OR REPLACE VIEW ViewName AS
SELECT Column1, Column2, ...Column(n)
FROM TableName
WHERE Condition;
```

DROPPING VIEW

SYNTAX

```
DROP VIEW ViewName;
```

INDEXES

Indexes helps in extracting data from databases quickly. Indexes are not visible to the end users. Basically, an index can be considered as a pointer that points to a data present in a table.

CREATE INDEX

SYNTAX

```
CREATE INDEX NameOfIndex ON
NameOfTable(Column1,Column2,Column3. . . . Column(n));
```

TYPES OF INDEXES

UNIQUE INDEX :

Helps in maintaining data integrity and ensures that there are no duplicate values present in the index key. UNIQUE index is automatically created on the PRIMARY KEY columns.

CREATE VIEW

SYNTAX

```
CREATE UNIQUE INDEX NameOfIndex ON NameOfTable
(Column1,Column2,Column3. . . . Column(n));
```

SINGLE-COLUMN INDEX

Index created on a table column is considered as a single column index.

CREATE VIEW

SYNTAX

```
CREATE INDEX NameOfIndex ON NameOfTable (Column);
```

COMPOSITE INDEX

Index created on multiple columns is considered as a single column index

CREATE VIEW

SYNTAX

```
CREATE UNIQUE INDEX NameOfIndex ON NameOfTable
(Column1,Column2,Column3. . . . Column(n));
```

REMOVING INDEXES

SYNTAX

```
DROP INDEX TableName.IndexName;
```