Oracle-- <https://apex.oracle.com/en/>

# Data type

|  |  |  |
| --- | --- | --- |
| **Data type** | **Format** | **Range** |
| Time(SQL 2008) | hh:mm:ss[.nnnnnnn] | 00:00:00.0000000 through 23:59:59.9999999 |
| date(SQL 2008) | YYYY-MM-DD | 0001-01-01 through 9999-12-31 |
| smalldatetime | YYYY-MM-DD hh:mm:ss | 1900-01-01 through 2079-06-06 |
| datetime | YYYY-MM-DD hh:mm:ss[.nnn] | 1753-01-01 through 9999-12-31 |
| datetime2 | YYYY-MM-DD hh:mm:ss[.nnnnnnn] | 0001-01-01 00:00:00.0000000 through 9999-12-31 23:59:59.9999999 |
| tinyint | **1 byte**--2 power(8) | 0-255 |
| smallint | **2 byte**--2 power(16) | (2^16)/2 ( -32,768 to +32,767) |
| int/integer | **4 byte**--2 power(32) | (2^32)/2 (-2,147,483,648 to +2,147,483,647) |
| bigint | **8 byte**--2 power(64) | (2^64)/2 (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807) |
| decimal |  |  |

# Difference between nvarchar and varchar

* **nchar** and **nvarchar** can store **Unicode** characters.
* **Char and varchar can store ASCII character**
* Regarding memory usage, nvarchar uses 2 bytes per character, whereas varchar uses 1.

# Temp Table

These tables can be created at **runtime** and can do the all kinds of operations that one normal table can do. But, based on the table types, the scope is limited. These tables are created inside tempdb database.

### Local Temp Table

Local temp tables are only available to the current connection for the user; and they are automatically deleted when the user disconnects from instances. Local temporary table name is stared with hash ("#") sign.

### Global Temp Table

Global Temporary tables name starts with a double hash ("##"). Once this table has been created by a connection, like a permanent table it is then available to any user by any connection. It can only be deleted once all connections have been closed.

# Declare variable

DECLARE @variable\_name datatype [ = initial\_value ],

@variable\_name datatype [ = initial\_value ],

...;

SET @techonthenet = 'Example showing how to declare variable';

## Example

DECLARE @t1 datetime, @n int;

SET @t1=getdate();

SET @n=10;

WHILE @n>0 begin

SELECT @n As n,@t1 as time;

SET @n=@n-1;

SET @t1=@t1-1;

end

## Declare variable with table type

DECLARE @Pet Table(PetID INT, OwnerID INT , PetName NVARCHAR(50), AnimalType NVARCHAR(10))

INSERT INTO @Pet(PetID, OwnerID, PetName, AnimalType) VALUES(1, 1, 'Sonya', 'Dog')

INSERT INTO @Pet(PetID, OwnerID, PetName, AnimalType) VALUES(2, 2, 'Bessy', 'Cat')

## Difference between table-variable and Temporary Table

* The first one is that transaction logs are not recorded for the table-variables. Hence, they are out of scope of the transaction mechanism

CREATE table #T (s varchar(128))

DECLARE @T table (s varchar(128))

INSERT into #T select 'old value #'

INSERT into @T select 'old value @'

BEGIN transaction

UPDATE #T set s='new value #'

UPDATE @T set s='new value @'

ROLLBACK transaction

SELECT \* from #T

SELECT \* from @T

s

---------------

old value #

s

---------------

new value @

* table-variables exist only in the same scope as variables. Contrary to the temporary tables, they are visible for entire session

# Function

CREATE FUNCTION [schema\_name.]function\_name

( [ @parameter [ AS ] [type\_schema\_name.] datatype

[ = default ] [ READONLY ]

, @parameter [ AS ] [type\_schema\_name.] datatype

[ = default ] [ READONLY ] ]

)

RETURN**S** return\_datatype --Different from oracle

[ AS ]

BEGIN

[declaration\_section] -- This section is before Begin in oracle

executable\_section

RETURN return\_value

END;

## Scalar Function

### without parameter

ALTER FUNCTION Add2() --() is mandatory which is different from oracle

RETURN**S** VARCHAR(20)

AS

BEGIN

RETURN 'Hello';

END;

### Calling Function

SELECT schema\_name.fucntion name

Select dbo.add2();

### with parameter

CREATE FUNCTION fn1(@var VARCHAR(20))

RETURNS VARCHAR(20)

AS

BEGIN

RETURN @var

END;

### Calling Function

SELECT dbo.fn1('Hello WOrld')

## Inline Function

Inline user-defined functions are a subset of user-defined functions that return a **table** data type. Inline functions can be used to achieve the functionality of parameterized views. This view would be better if it were more generalized and let users specify the region they are interested in viewing. Views, however, do not support parameters in the search conditions specified in the WHERE clause. Inline user-defined functions can be used to support parameters in the search conditions specified in the WHERE clause.

This function does not allow declaration and begin-end keyword

ALTER FUNCTION fn\_EMP(@id int)

RETURNS TABLE

AS

RETURN

SELECT \* FROM EMP where id=@id

### Calling Function

SELECT \* FROM fn\_EMP(1)

## Table Valued Function

ALTER FUNCTION table\_valuedfn\_EMP()

RETURNS @table\_valued TABLE

(ID INT,

ename VARCHAR(20)

)

AS

BEGIN

INSERT INTO @table\_valued VALUES(1,'kk1')

INSERT INTO @table\_valued VALUES(2,'kk2')

RETURN;

END;

### Calling Function

SELECT \* FROM table\_valuedfn\_EMP()

# PROCEDURE

CREATE PROCEDURE procedure\_name [ @parameter datatype [OUT |IN], @parameter datatype [OUT |IN]

AS

BEGIN

[declaration\_section]

executable\_section

END;

--Oracle has () which is different

## PROCDEURE with OUT parameter

### Example 1

CREATE PROCEDURE FindSite

@site\_name VARCHAR(50) OUT

AS

BEGIN

DECLARE @site\_id INT;

SET @site\_id = 8;

IF @site\_id < 10

SET @site\_name = 'TechOnTheNet.com';

ELSE

SET @site\_name = 'CheckYourMath.com';

END;

### Calling Procedure

DECLARE @site\_name varchar(50);

EXEC FindSite @site\_name OUT;

PRINT @site\_name;

## PROCDEURE without parameter

### Example 2

CREATE PROCEDURE MyProc

AS

SELECT \* FROM EMP

---EXEC MyProc

## PROCDEURE with IN parameter

Default is IN parameter

CREATE PROCEDURE ADD3(@a INT,@b INT,@c INT OUT)

AS

BEGIN

SET @c=@a + @b

END

### Calling Procedure

It is different from oracle.oracle uses () while sending parameter

DECLARE @c INT

EXEC ADD3 11,12,@c OUT

PRINT @C

# Pivot

SELECT first\_column AS <first\_column\_alias>,

[pivot\_value1], [pivot\_value2], ... [pivot\_value\_n]

FROM

(<source\_table>) AS <source\_table\_alias>

PIVOT

(

aggregate\_function(<aggregate\_column>)

FOR <pivot\_column>

IN ([pivot\_value1], [pivot\_value2], ... [pivot\_value\_n])

) AS <pivot\_table\_alias>;

### Different from oracle:

--IN parameter where [] is mandatory

-- aliase table and pivot name is mandatory

### Example

select \* from

(select \* from employees)a

PIVOT(sum(salary)

FOR dept\_id IN ([30],[45]))pv

***Example-2*** *pivot on multiple columns*

SELECT \* FROM (

SELECT Seq, ItemCode, ItemModel, ItemName, ItemColour FROM TblPivot ) AS P

-- For ItemColour

PIVOT ( Count(ItemCode) FOR ItemColour IN ([Red], [Blue], [Green]) ) AS pv1

-- For ItemName

PIVOT ( Count(ItemModel) FOR ItemName IN ([Samsung Mobile], [Nokia Mobile], [Motorola Mobile]) ) AS pv2

## MySql syntax

<https://www.geeksforgeeks.org/how-to-return-pivot-table-output-in-mysql/>

SELECT

non\_pivoted\_column,

**SUM**(

CASE WHEN pivoted\_column = 'value1' THEN aggregate\_column END

) AS value1,

SUM(

CASE WHEN pivoted\_column = 'value2' THEN aggregate\_column END

) AS value2,

-- Additional pivoted columns as needed

FROM your\_table

GROUP BYnon\_pivoted\_column;

**Example**: Below are table details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NAME** | **COLLEGE** | **ROLL NUMBER** | **SUBJECT** | **MARKS** |
| ROMY | BVP | 261150 | DBMS | 90 |
| ROMY | BVP | 261150 | NETWORKING | 87 |
| ROMY | BVP | 261150 | GRAPHICS | 95 |
| PUSHKAR | MSIT | 898888 | DBMS | 91 |
| PUSHKAR | MSIT | 898888 | NETWORKING | 90 |
| PUSHKAR | MSIT | 898888 | GRAPHICS | 78 |

Output should be as below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **NAME** | **COLLEGE** | **ROLL NUMBER** | **DBMS** | **NETWORKING** | **GRAPHICS** |
| ROMY | BVP | 261150 | 90 | 87 | 95 |
| PUSHKAR | MSIT | 898888 | 91 | 90 | 78 |

SELECT NAME,COLLEGE,ROLL\_NO,

SUM(CASE WHEN SUBJECT='DBMS' THEN MARKS END) AS 'DBMS',

SUM(CASE WHEN SUBJECT='NETWORKING' THEN MARKS END) AS 'NETWORKING',

SUM(CASE WHEN SUBJECT='GRAPHICS' THEN MARKS END) AS 'GRAPHICS'

FROM pivot1

GROUP BY NAME,COLLEGE,ROLL\_NO

# Constraint

* NOT NULL
* CHECK
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* DEFAULT

# CREATE Table

CREATE TABLE customers( customer\_id DECIMAL(10,2) NOT NULL,

customer\_name varchar(50) NOT NULL,city varchar(50));

## Create table with primary key

CREATE TABLE [dbo].[customers](

[cid] [int] NULL, [customer\_name] [varchar](50) NOT NULL,

[city] [varchar](50) NULL,

[d\_col] VARCHAR(20) DEFAULT ‘Hello’,

[d\_date] datetime DEFAULT getdate(),

[idnt] [int] IDENTITY(1,2) NOT NULL,CONSTRAINT pk1 PRIMARY KEY (idnt))

## Create table with Foreign key

CREATE TABLE customers\_fk(FK\_idnt INT,cname VARCHAR(20),

CONSTRAINT fk1 FOREIGN KEY (FK\_idnt) REFERENCES customers(idnt))

## Create copy from old table

### SELECT INTO

SELECT expressions INTO new\_table FROM tables [WHERE conditions];

expressions

The columns or calculations that you wish to retrieve.

SELECT employee\_id, last\_name **INTO contacts** FROM employees WHERE employee\_id < 1000;

SELECT \* INTO contacts FROM employees WHERE employee\_id < 1000;

**In MySQL**

CREATE TABLE newTable AS SELECT \* FROM OldTable

# INSERT INTO SELECT

INSERT INTO table\_name1 SELECT \* from table\_name2 where col1=1;

INSERT INTO Customers (CustomerName, City, Country) SELECT SupplierName, City, Country FROM Suppliers;

# PRIMARY KEY CONSTRAINT

By default , adding primary key created unique clustered index

ALTER TABLE customers ADD CONSTRAINT pk1 PRIMARY KEY (idnt)

If you want to create non CLUSTERED unique index on primary key then as follow

ALTER TABLE customers ADD CONSTRAINT pk1 PRIMARY KEY NONCLUSTERED (idnt)

### DROP Constraint

ALTER TABLE customers DROP CONSTRAINT pk1

### Disable Constraint

ALTER TABLE customers NOCHECK CONSTRAINT pk1

### Enable Constraint

ALTER TABLE customers CHECK CONSTRAINT pk1

# FOREIGN KEY CONSTRAINT

ALTER TABLE customers\_fk ADD CONSTRAINT fk1 FOREIGN KEY (FK\_idnt) REFERENCES customers(idnt)

### DROP Constraint

ALTER TABLE customers\_fk DROP CONSTRAINT fk1

### Disable Constraint

ALTER TABLE table\_name NOCHECK CONSTRAINT constraint\_name;

ALTER TABLE customers\_fk NOCHECK CONSTRAINT fk1

### Enable Constraint

ALTER TABLE table\_name CHECK CONSTRAINT Constraint\_name;

ALTER TABLE customers\_fk CHECK CONSTRAINT fk1

### foreign key with Cascade DELETE

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete in SQL Server. A foreign key with cascade delete can be created using either a CREATE TABLE statement or an ALTER TABLE statement.

ALTER TABLE customers\_fk ADD CONSTRAINT fk1 FOREIGN KEY (FK\_idnt) REFERENCES customers(idnt) ON DELETE CASCADE

# CHECK Constraint

CREATE TABLE table\_name

(

column1datatype null/not null,

column2datatype null/not null,

...

CONSTRAINT constraint\_name CHECK (column\_name condition) [DISABLE]

);

CREATE TABLE frgn(id1 int,name1 varchar(10),CONSTRAINT ckCHECK(id in (1,2,3))

CREATE TABLE frgn(id1 int,name1 varchar(10),CONSTRAINT ckCHECK(id >=10)

CREATE TABLE frgn(id1 int,name1 varchar(10),CONSTRAINT ckCHECK(id between 10 and 100)

Alter table table\_nameADD CONSTRAINTconstraint\_name CHECK (col\_name>=10)

ALTER TABLE [TableName] DROP CONSTRAINT [CONSTRAINT\_NAME]

# DEFAULT CONSTRAINT

A default is bound to a column or alias data type and specifies a default value for the column or columns, when no value is supplied.

CREATE TABLE table\_Name (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) DEFAULT 'Sandnes',

OrderDate datetime DEFAULT GETDATE());

ALTER TABLE table\_name ADD CONSTRAINT constraint\_name DEFAULT GETDATE() for column\_name

ALTER TABLE table\_name DROP CONSTRAINT Contraint\_name

# IDENTITY

* The identity must only contain integer values but the column can be of any numeric datatype (bigint, int, tinyint, numeric, decimal).
* An identity column is not guaranteed to be unique nor consecutive
* Index can be created on identity column
* Constraint can be added on identity column like primary key

## CREATE TABLE with IDENTITY

CREATE TABLE tbl\_Indentity(ID INT IDENTITY(1,1) NOT NULL,NAME1 VARCHAR(20))

### In Mysql

We should use AUTO\_INCREMENT keyword and column must be part of key

CREATE Table IdentityEx(id int auto\_increment, name varchar(20),PRIMARY KEY(id));

## ALTER TABLE with IDENTITY

ALTER TABLE table\_name ADD column\_name data\_type IDENTITY(starting value,Increment value)

ALTER TABLE customers ADD idnt INT IDENTITY(1,2)

## Find current identity value

IDENT\_CURRENT(‘table\_name’)—must include quotation

SELECT IDENT\_CURRENT('tbl\_Indentity')

## Changing Current Seed

DBCC checkident(table\_name,reseed,1)

--Table can have only one identity column

--We can also reset the current seed to it’s original value via a truncate table. Delete does not do.

# SEQUENCES

This feature is available from SQL server 2012 onwards.

A sequence is a user defined schema bound object that **generates a sequence of numeric values**.

Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.

The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when max\_value exceeds.

**Syntax:**

CREATE SEQUENCE sequence\_name

START WITH initial\_value

INCREMENT BY increment\_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE|NOCYCLE ;

**sequence\_name:** Name of the sequence.

**initial\_value:** starting value from where the sequence starts.

Initial\_value should be greater than or equal

to minimum value and less than equal to maximum value.

**increment\_value:** Value by which sequence will increment itself.

Increment\_value can be positive or negative.

**minimum\_value:** Minimum value of the sequence.

**maximum\_value:** Maximum value of the sequence.

**cycle:** When sequence reaches its set\_limit

it starts from beginning.

## Difference between identity and Sequence

An **identity** is a sequence that's automatically created and associated with a table column. Identity properties are tied to a specific table and can't be shared with other tables.

**Sequences** are user-defined database objects that can generate a series of numeric values. Sequences are independent objects that can be used across multiple tables.

# UNIQUE CONSTRAINT

alter table customers

ADD CONSTRAINT uq UNIQUE (cid)

# Alter Table

## Rename table

It is different from oracle.( Alter table table\_name RENAME TO new\_table\_name)

SP\_RENAME old\_table\_name,new\_table\_name;

SP\_RENAME test,test\_new;

## Rename Column

sp\_rename 'table\_name.old\_column\_name', 'new\_column\_name', 'COLUMN';

quotation is mandatory

SP\_RENAME 'test.ID','IDD','COLUMN'

## ADD Column

Alter table prmry ADD name varchar(10),name2 varchar(20) --oracle has () for list of columns

## Data type change Column

ALTER TABLE customers ALTER COLUMN customer\_name varchar2(100) --only single column

## DROP COLUMN

ALTER TABLE table\_name DROP COLUMN column\_name;

# TRIGGER

A trigger is a kind of stored procedure that executes when an event occurs in the server.

## Inserted

It is table used in trigger for UPDATE/INSERT trigger

## Deleted

It is table used in trigger for DELETE trigger

CREATE [ OR ALTER ] TRIGGER [ schema\_name . ]trigger\_name

ON { table | view }

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

AS { sql\_statement}

There are two types of triggers

## DML Trigger

Data Manipulation Language (DML) triggers execute when a user tries to modify data. DML triggers are carried out on DML events such as INSERT,UPDATE, or DELETE statements

DROP TRIGGER Trigger\_name

## DDL Trigger(ON Database)

It includes creating and dropping database objects such as tables, views, procedures, and logins. DDL triggers can be associated with CREATE, ALTER, and DROP statements.

It is viewed by sys.triggers.

CREATE TRIGGER NoTableUpdate

ON DATABASE FOR DROP\_TABLE, ALTER\_TABLE

AS

PRINT 'DROP TABLE and ALTER TABLE statements are not allowed'

ROLLBACK

#### Example-2

CREATE TRIGGER DDLTrigger\_Sample

ON DATABASE

FOR CREATE\_PROCEDURE, ALTER\_PROCEDURE, DROP\_PROCEDURE

AS

BEGIN

END

### ALTER Trigger

ALTER TRIGGER NoTableUpdate

ON DATABASE FOR DROP\_TABLE, ALTER\_TABLE

AS

PRINT 'DROP TABLE and ALTER TABLE statements are not allowed'

ROLLBACK

### DROP Trigger

DROP TRIGGER NoTableUpdate on DATABASE

### DISABLE Trigger

DISABLE TRIGGER NoTableUpdate on DATABASE

### ENABLE Trigger

ENABLE TRIGGER NoTableUpdate on DATABASE

## DDL Trigger (ON ALL Server)

CREATE TRIGGER ddl\_trig\_database

ON ALL SERVER

FOR CREATE\_DATABASE

AS

PRINT 'Database Created.'

SELECT EVENTDATA().value('(/EVENT\_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(max)')

### Drop DDL Trigger

DROP TRIGGER ddl\_trig\_database ON ALL SERVER

--

FOR | AFTER  
AFTER specifies that the DML trigger is fired only when all operations specified in the triggering SQL statement have executed successfully. All referential cascade actions and constraint checks also must succeed before this trigger fires.

AFTER is the default when FOR is the only keyword specified.2

AFTER triggers cannot be defined on views.

## AFTER INSERT

Statement in trigger executed first then insert or update statement is executed

CREATE TRIGGER Tr\_AFTER ON t\_FOR\_TRIGGER

FOR INSERT

AS

INSERT INTO t\_FOR\_TRIGGER\_Audit SELECT EID,[name],salary,'After Insert',getdate() from inserted

PRINT 'AFTER INSERT trigger fired.'

## AFTER UPDATE

CREATE TRIGGER Tr\_AFTER\_UPDATE ON t\_FOR\_TRIGGER

AFTER UPDATE

AS

INSERT INTO t\_FOR\_TRIGGER\_Audit SELECT EID,[name],salary,'AFter UPDATE',getdate() from inserted

PRINT 'AFTER update trigger fired.'

## AFTER DELETE

CREATE TRIGGER Tr\_AFTER\_DELETE ON t\_FOR\_TRIGGER

AFTER DELETE

AS

INSERT INTO t\_FOR\_TRIGGER\_Audit SELECT EID,[name],salary,'AFter DELETE',getdate() from deleted

PRINT 'AFTER DELETE trigger fired.'

INSTEAD OF  
Specifies that the DML trigger is executed instead of the triggering SQL statement, therefore, overriding the actions of the triggering statements. INSTEAD OF cannot be specified for DDL or logon triggers.

# INDEX

An index is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns. Indexes can be created on **tables**, **views**, and **temporary tables**. An index can even be created before there is data in the table. Table can have only one **clustered index** and 249 **non-clustered index**.

CREATE [ **UNIQUE** ] [ **CLUSTERED** | **NONCLUSTERED** ] INDEX index\_name

**ON** table\_name ( column1 [ASC | DESC ], ... column\_n [ ASC | DESC ] )

## Clustered Index

A clustered index is actually a table where the data for the rows are stored. It defines the order of the table data based on the key values that can be sorted in only one way. In the database, each table can have only one clustered index. In a relational database, if the table column contains a primary key or unique key, [MySQL](https://www.javatpoint.com/mysql-tutorial) allows you to create a clustered index named **PRIMARY** based on that specific column.

## Disadvantages

The main disadvantages of the clustered index are as follows:

* It contains many insert records in a non-sequential order.
* It always takes a long time to update the records.
* It needs extra work for SQL queries, such as insert, updates, and deletes.

## Non-Clustered Index

Indicates that the logical order does not determine the physical order of the rows in the table.

CREATE INDEX contacts\_idx ON contacts (last\_name, first\_name);

Since we have not specified ASC | DESC to each of the columns, the index is created with each of the fields in ascending order. We could modify our example and change the sort orders to descending as follows: CREATE INDEX contacts\_idx ON contacts (last\_name, first\_name DESC);

## UNIQUE Index (UNIQUE NON CLUSTERED Index)

It is also called **non clustered unique** index.

Indicates that the combination of values in the indexed columns must be unique.

CREATE UNIQUE INDEX contacts\_uidx ON contacts (last\_name, first\_name);

This example would create an index called contacts\_uidx on that contacts table that consists of the last\_name and first\_namefields, but also ensures that the there are only unique combinations of the two fields.

## UNIQUE CLUSTERED Index

You could modify this example further to make the unique index also clustered so that the physical order of the rows in the table is determined by the logical order of the index.

CREATE UNIQUE CLUSTERED INDEX contacts\_uidx ON contacts (last\_name, first\_name);

## RENAME INDEX

sp\_rename 'table\_name.index\_name', 'new\_index\_name', 'INDEX';

SP\_RENAME 'customers.idx1','index\_name','INDEX'

## Drop Index

DROP INDEX table\_name.index\_name; --idex column is not required here

# VIEW

*Views* are virtual tables that allow you to represent data in an alternate way. You can create a view only in the current database.

CREATE VIEW StorePersonnel

AS SELECT SalesPersonID, Name FROM AdventureWorks.Sales.Store WHERE SalesPersonID >250

# Conditional Statement

## IF ELSE

IF condition

{...statements to execute when condition is TRUE...}

[ ELSE

{...statements to execute when condition is FALSE...} ]

Note: Oracle has THEN key; in the last oracle has end if

### Example 1

DECLARE @site\_value INT;

SET @site\_value = 50;

IF @site\_value < 25

PRINT 'TechOnTheNet.com';

ELSE

BEGIN

IF @site\_value < 50

PRINT 'CheckYourMath.com';

ELSE

PRINT 'BigActivities.com';

END;

### Example 2

DECLARE @v INT

SET @v=10

IF @v>5

PRINT 'variable has value greater than 5'

ELSE

PRINT 'variable has value less than 5'

# While Loop

WHILE condition

BEGIN

{...statements...}

END;

### Example1

DECLARE @v INT

SET @v=1

WHILE @v<5

BEGIN

PRINT @v

SET @v=@v+1

END

## For Loop

SQL server does not support for loop

## Break

DECLARE @intFlag INT

SET @intFlag = 1

WHILE (@intFlag <=5)

BEGIN

PRINT @intFlag

SET @intFlag = @intFlag + 1

IF @intFlag = 4

BREAK;

END

## Continue

In SQL Server, the CONTINUE statement is used when you are want a WHILE LOOP to execute again. It will ignore any statements after the CONTINUE statement

DECLARE @site\_value INT;

SET @site\_value = 0;

WHILE @site\_value <= 10

BEGIN

IF @site\_value = 2

BREAK;

ELSE

BEGIN

SET @site\_value = @site\_value + 1;

PRINT 'Inside WHILE LOOP at ' + CAST(@site\_value AS VARCHAR(5));

CONTINUE;

END;

END;

# Function

Built-in Function

1. Aggregate Function—Count,Sum,AVG,MAX,MIN
2. Numeric Function
3. Conversion Function
4. String Function
5. Date and Time Function

User Defined Function

## Numeric Function

ABS(number)—Absolute value of number

SELECT ABS(-35.6) ---35.6

SELECT ROUND(35.6) –36

SELECT FLOOR(35.6) —36 which ignore decinmal value

SELECT MOD(11,3) --2 Remainder MOD(dividend,divisor)

SELECT POWER(3,2) --9 POWER(number,power)

SELECT SQRT(16) --4 Square root

SELECT GREATEST(16,25,3) –-25 maximum among list GREATED(col1,col2,…)

SELECT LEAST(16,25,3) –-3 minimm among list LEAST(col1,col2,…)

SELECT RAND() --Random number for every execution,value from 0 to 1, e.g 0.988888989

## String Function

## CHARINDEX

Not available in MySql

CHARINDEX( Searching\_string, string, [start\_position] ) –Similar in oracle: INSTR

SELECT CHARINDEX('t','Teachert',1),CHARINDEX('t','Teachert',7)

## LEN, Replace,..

LEN(STRING) -- similar in oracle/MySql LENGTH(string)

<https://www.w3schools.com/mysql/mysql_ref_functions.asp>

SELECT LEN('Teachert')

SELECT LEFT('SQL Tutorial.com', 3) AS "LFT",RIGHT('SQL Tutorial.com', 3)"RHT"

SELECT LOWER('THE khalid')"lower",UPPER('THE khalid')"upper"

SELECT REPLACE('Hello Mr Khalid', 'Mr', 'Mrs');-- Hello mrs Khalid

SELECT COALESCE(col1, col2, col3);-- if col1 is null then col2 will be returned

SELECT CONCAT("SQL ", "Tutorial ", "is ", "fun!") AS ConcatenatedString;

SELECT CONCAT\_WS(" ","SQL ", "Tutorial ", "is ", "fun!")

SELECT SUBSTRING('MdKhalid', 1, 3);-- start at position 1, extract 3 characters

SELECT CONCAT(LTRIM(" SQL"),RTRIM("Tutorial "),TRIM(" Fun "))

|  |  |
| --- | --- |
| **Function** | **Description** |
| ASCII | Returns the number code that represents the specific character |
| CHAR | Returns the ASCII character based on the number code |
| CHARINDEX | Returns the location of a substring in a string |
| Concat | Concatenates two or more strings together;Concat(str1,str2,str3,…) |
| Concat\_WS | Concate with separator;concat(separator,str1,str2,str3) |
| DATALENGTH | Returns the length of an expression (in bytes) |
| LEN | Returns the length of the specified string |
| LOWER | Converts a string to lower-case |
| UPPER | Converts a string to upper-case |
| LTRIM | Removes leading spaces from a string |
| RTRIM | Removes trailing spaces from a string |
| NCHAR | Returns the Unicode character based on the number code |
| PATINDEX | Returns the location of a pattern in a string |
| REPLACE | Replaces a sequence of characters in a string with another set of characters  REPLACE(string1, **string\_to\_replace**, **string\_with**) |
| REVERSE | It will reverse string; reverse(“StringValue”) |
| SPACE | Returns a string with a specified number of spaces |
| STR | Returns a string representation of a number |
| LEFT | Extracts a substring from a string (starting from left) |
| RIGHT | Extracts a substring from a string (starting from right) |
| SUBSTRING | Extracts a substring from a string |

### [How to count number of occurrence of character in string field](https://stackoverflow.com/questions/1860457/how-to-count-instances-of-character-in-sql-column)

This will return number of occurance of N

select ColumnName, **LEN**(ColumnName)- **LEN**(**REPLACE**(ColumnName, 'N', '')) from Table

## IFNULL(), ISNULL(), COALESCE(), and NVL()

**My SQL** --IFNULL and COALESC is same, if first expression is null then second value

**Oracle**—NVL works similar

**MS SQL**—ISNULL works similar

# Conversion Function

Number🡨🡪Character🡨🡪date

## CAST

The CAST() function converts a value (of any type) into the specified datatype.

CAST( expression **AS** Data\_type)

SELECT CAST(14.85 AS int);

SELECT CAST(14.85 AS float);

SELECT CAST(15.6 AS varchar);

SELECT CAST(15.6 AS varchar(4));

SELECT CAST('15.6' AS float);

SELECT CAST('2014-05-02' AS datetime);-- Format: "YYYY-MM-DD"

SELECT CAST("14:06:10" AS TIME); --

## CONVERT

The CONVERT function converts an expression from one datatype to another datatype

CAST( expression **,** Data\_type)

SELECT CAST(14.85 AS int),CONVERT(int, 14.85);

SELECT CAST(14.85 AS float),CONVERT(float,14.85)

SELECT CAST(15.6 AS varchar), CONVERT(varchar, 15.6)

SELECT CAST(15.6 AS varchar(4)), CONVERT(varchar(4), 15.6)

SELECT CAST('15.6' AS float), CONVERT(float, '15.6')

SELECT CAST('2014-05-02' AS datetime),CONVERT(datetime, '2014-05-02')

**Different from CAST**

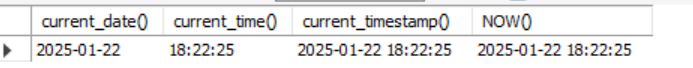
CONVERT accepts an optional style parameter used for formatting.  Other than that, they are pretty much the same as CAST.

SELECT CONVERT(VARCHAR, getdate(), 101) as MMDDYYYY,

CONVERT(VARCHAR, getdate(), 111) as YYYYMMDD;

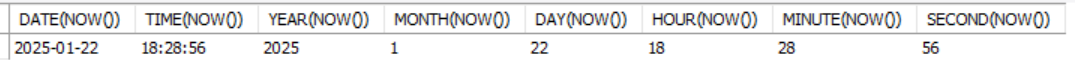
# DATE function

In MySql

SELECT current\_date(),current\_time(),current\_timestamp(),NOW()

SELECT **DATE**(NOW()),**TIME**(NOW()),**YEAR**(NOW()),**MONTH**(NOW()),

**DAY**(NOW()),**HOUR**(NOW()),**MINUTE**(NOW()),**SECOND**(NOW()),**MONTHNAME**(NOW()) --January



## DATEADD

DATEADD( interval, number, date )

SELECT DATEADD(year, 1, '2014/04/28'),DATEADD(quarter, 1, '2014-04-28')

SELECT DATEADD(month, 1, '2014/04/28'),DATEADD(week, 1, '2014-04-28'),DATEADD(week, 1, '2014-03-28')

SELECT DATEADD(day, 1, '2014/04/28'),DATEADD(hour, 1, '2014-04-28'),DATEADD(minute, 1, '2014-04-28')

In My SQL we have Date\_Add(dateColumn, INTERVAL unit Interval\_unit

SELECT **Date\_add**(now(),**INTERVAL** 1 month)

|  |  |
| --- | --- |
| Interval Value (any one of) | Explanation |
| year, yyyy, yy | Year interval |
| quarter, qq, q | Quarter interval |
| month, mm, m | Month interval |
| dayofyear | Day of year interval |
| day, dy, y | Day interval |
| week, ww, wk | Week interval |
| weekday, dw, w | Weekday interval |
| hour, hh | Hour interval |
| minute, mi, n | Minute interval |
| second, ss, s | Second interval |
| millisecond, ms | Millisecond interval |

## DATEDIFF

DATEDIFF( interval, date1, date2 ) -- date2 minus date1

SELECT DATEDIFF(**year**,'2014/04/28','2015/04/28'),DATEDIFF(**quarter**,'2014-04-28','2015-04-28')

SELECT DATEDIFF(**month**,'2014-04-28','2015/04/28'),DATEDIFF(**week**, '2014-04-28', '2015-04-28')

SELECT DATEDIFF(**day**, '2014-04-28', '2014/04/30'),DATEDIFF(**hour**, '2014-04-28 00:00:00', '2014-04-28 01:00:00'),DATEDIFF(**minute**,'2014-04-28 00:00:00', '2014-04-28 01:00:00')

In **MySQL** , we have DATEDIFF (date1,date2) -- Default interval is day .

## DATEPART

In MSSQL

the DATEPART function returns a specified part of a given date, as an integer value.

DATEPART( interval, date )

SELECT DATEPART(month, '2014/04/28'),DATEPART(mm, '28 APR 2014')

SELECT DATEPART(minute, '2014/04/28 09:49'),DATEPART(second, '2014/04/28 09:49:12')

SELECT DATEPART(m, '2014/04/28 09:49'),DATEPART(s, '2014/04/28 09:49:12')

In **MySQL** , we have EXTRACT(component from date)

SELECT **EXTRACT**(WEEK FROM NOW()),**EXTRACT**(QUARTER FROM NOW())

Componenet as below

|  |  |  |
| --- | --- | --- |
| YEAR |  | HOUR |
| MONTH |  | MINUTE |
| DAY |  | SECOND |
| WEEK |  | QUARTER |

## DATENAME

the DATEPART function returns a specified part of a given date, as a string value.

SELECT DATENAME(month, '2014/04/28')

Result=April

## MONTH(date)

In both MS SQL and MySQL

It is equivalent to DATEPART(MONTH,date)

SELECT MONTH('2014/04/28');

Result=4

SELECT MONTH('2014/03/31 10:05');

Result=3

## DAY(date)

In both **MS SQL** and **MySQL**

It is equivalent to DATEPART(DAY,date)

SELECT DAY('2014/04/28');

Result: 28

SELECT DAY('2014/03/31 10:05');

Result: 31

SELECT DAY('2014/04/01 10:05:18.621');

Result: 1

## YEAR(date)

In both **MS SQL** and **MySQL**

It is equivalent to DATEPART(YEAR,date)

SELECT YEAR('2014/04/28');

Result: 2014

SELECT YEAR('2013/03/31 10:05');

Result: 2013

SELECT YEAR('2015/12/01 10:05:18.621');

Result: 2015

## Calculate Age

SELECT a.\*,

CASE WHEN MONTH(today\_date)>MONTH(dob) OR

(

MONTH(today\_date)=MONTH(dob) AND DAY(today\_date)>DAY(dob)

) THEN

DATEDIFF(year,dob,today\_date) ELSE DATEDIFF(year,dob,today\_date)-1 END AS 'AGE'

FROM t\_Date a

## Calculate days

SELECT datediff(day, getdate(), dateadd(month, 1, getdate()))—current month

SELECT datediff(day, '2018-03-15', dateadd(month, 1, '2018-03-15'))—particular

Caution

SELECT DATEDIFF(DAY,'2018-01-30',DATEADD(MONTH,1,'2018-01-30'))—this gives 29 as february is of 28. So it is better to be careful while selecting that. It should be 1 to 28 .

# CASE

## Case-Type1

CASE expression

WHEN value\_1 THEN result\_1

WHEN value\_2 THEN result\_2

WHEN value\_n THEN result\_n

ELSE result

END

## Case-Type2

CASE

WHEN condition\_1 THEN result\_1

WHEN condition\_2 THEN result\_2

WHEN condition\_n THEN result\_n

ELSE result

END

# COALESCE

the COALESCE function returns the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function will return null.

COALESCE( expression1, expression2, ... expression\_n )—exp must be in same data type

SELECT COALESCE(NULL, NULL, 1, 2, 3, NULL, 4);

SELECT COALESCE(NULL,'Hello',1) --Gives error

# ****ISDATE****

* The ISDATE function returns 1, if the *expression* is a valid date.
* The ISDATE function returns 0, if the *expression* is NOT a valid date

SELECT ISDATE('2014-05-01'),ISDATE('2014-05-01 10:03:32.001'),ISDATE('techonthenet.com'),ISDATE(123)

# ISNULL(oracle=NVL)

the ISNULL function lets you return an alternative value when an expression is NULL.

ISNULL( expression, alternative\_value )

SELECT ISNULL(NULL, 'Te'),ISNULL('xyz', 'yzx')

# ISNUMERIC

 the ISNUMERIC function returns 1 if the expression is a valid number. Otherwise, it returns 0.

ISNUMERIC( expression )

SELECT ISNUMERIC(1234),ISNUMERIC('1234'),ISNUMERIC('te'),ISNUMERIC('2014-05-01');

For both integer and decimal datatype, isnumeric returns 1

# CEILING FLOOR

The CEILING function returns the smallest integer greater than or equal to the specified numeric expression. The FLOOR function returns the largest integer less than or equal to the specified numeric expression. For example, in considering a numeric expression of 12.9273, CEILING returns 13 and FLOOR returns 12. The return value of both FLOOR and CEILING has the same data type as the input numeric expression.

|  |  |  |
| --- | --- | --- |
| EID | CEILING | FLOOR |
| 11 | 11 | 11 |
| 11.11 | 12 | 11 |
| 13 | 13 | 13 |
| 13.13 | 14 | 13 |
| 13.59 | 14 | 13 |
| 13.99 | 14 | 13 |
| 13.01 | 14 | 13 |
| 13.5 | 14 | 13 |
| -13.59 | -13 | -14 |
| -13.99 | -13 | -14 |
| -13.01 | -13 | -14 |
| -13.5 | -13 | -14 |

# NULLIF

 the NULLIF function compares expression1 and expression2. If expression1 and expression2are equal, the NULLIF function returns NULL. Otherwise, it returns the first expression which is expression1.

NULLIF( expression1, expression2 )

SELECT NULLIF(12, 12),NULLIF(12, 45),NULLIF('2014-05-01', '2014-05-01'),NULLIF('A','a')

# GREATEST(NA in SQL) --in MySql

There is no such function in SQL server. We can use **case** function for workaround.

Maximum of all subject as below.

SELECT CASE

WHEN MATH>=phy AND MATH>=Che AND MATH>=Hindi THEN MATH

WHEN PHY>=MATH AND PHY>=Che AND PHY>=Hindi THEN PHY

WHEN CHE>=MATH AND CHE>=PHY AND CHE>HINDI THEN CHE

WHEN HINDI>=MATH AND HINDI>=PHY AND HINDI>CHE THEN HINDI

END AS maxsubject FROM STUDENt2

# LEAST(NA in SQL) --In MySql

There is no such function in SQL server. We can use **case** function for workaround.

SELECT CASE

WHEN MATH<=phy AND MATH<=Che AND MATH<=Hindi THEN MATH

WHEN PHY<=MATH AND PHY<=Che AND PHY<=Hindi THEN PHY

WHEN CHE<=MATH AND CHE<=PHY AND CHE<HINDI THEN CHE

WHEN HINDI<=MATH AND HINDI<=PHY AND HINDI<CHE THEN HINDI

END AS minsubject FROM STUDENt2

# ANY

The ANY operator returns TRUE if any of the subquery values meet the condition. The ANY comparison condition is used to compare a value to a list or subquery. It must be preceded by =, !=, >, <, <=, >= and followed by a list or subquery.

If ANY= then it is equivalent to IN as below

SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

SELECT ProductName FROM Products WHERE ProductID >= ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

# ALL

The ALL operator returns TRUE if all of the subquery values meet the condition. The ALL comparison condition is used to compare a value to a list or subquery. It must be preceded by =, !=, >, <, <=, >= and followed by a list or subquery.

SELECT ProductName FROM Products WHERE ProductID >= ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

# UNION

In order to perform a UNION, the data being combined must meet two conditions.First, the number and the order of the columns must be the same. Next, the data types must be compatible.  They need not be the same exact same type, but they must be of a type that SQL Server can implicitly convert.

* Compatible like smallint,Float,Decimal,INT are compatible
* **Either the datatype used should be same in two table**(or)**you should use cast or convert function to match the datatypes in those two tables**.

# CURSOR

Cursor is a Database object which allows us to process each row and manipulate its data. A Cursor is always associated with a Select Query and it will process each row returned by the Select Query one by one.

### Example1

DECLARE @ename VARCHAR(20)

DECLARE c CURSOR FOR SELECT EMPNAME FROm STUDENT2

OPEN c

FETCH NEXT FROM c INTO @ename --if not that @@FETCH\_STATUS=-1

WHILE @@FETCH\_STATUS=0

BEGIN

PRINT @ename

FETCH NEXT FROM c INTO @ename

END

CLOSE c

DEALLOCATE c

### Exampl2

DECLARE cu CURSOR FOR SELECT EMPNAME,MATH FROm STUDENT2

DECLARE @ename VARCHAR(20),@marks INT;

OPEN cu

FETCH NEXT FROM cu INTO @ename,@marks

WHILE @@FETCH\_STATUS=0

BEGIN

print @ename + ' has obtained markes ' + CAST(@marks As varchar(5))

FETCH NEXT FROM cu INTO @ename,@marks

END

CLOSE cu

DEALLOCATE cu

# Renaming Columns with AS

SELECT DepartmentID As ID, Name As Title FROM HumanResources.Department Where DepartmentID BETWEEN 5 AND 10

# Filter Operator

|  |  |
| --- | --- |
| **Condition** | **Description** |
| **=** | Tests for an equal condition. |
| **<>** | Tests for a not-equal condition. |
| **!=** | Tests for a not-equal condition. |
| **>** | Tests for a greater-than condition. |
| **>=** | Tests for a greater-than or equal-to condition. |
| **!>** | Tests for a not-greater-than condition. |
| **<** | Tests for a less-than condition. |
| **<=** | Tests for a less-than or equal-to condition. |
| **!<** | Tests for a not-less-then condition. |
| **[ NOT ] LIKE** | Tests for a matching pattern. |
| **ESCAPE ‘escape\_character’** | Allows a wildcard character to be searched for. |
| **[ NOT ] BETWEEN** | Tests for a between condition. The AND keyword separates the starting and ending values. |
| **IS [ NOT ] NULL** | Tests for a null or optionally a not-null condition. |
| **CONTAINS** | Tests for fuzzy matching or words or phrases. |
| **[ NOT ] IN** | Tests if a value is included or excluded from a list. The list can be a set of constants enclosed in parentheses or a subquery. |

# ESCAPE

Wildcard Character s

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| % | \_ | [ | ] | [] | ^ |

You can search for character strings that include one or more of the special wildcard characters. For example, the discounts table in a customers database may store discount values that include a percent sign (%). To search for the percent sign as a character instead of as a wildcard character, the ESCAPE keyword and escape character must be provided. For example, a sample database contains a column named comment that contains the text 30%. To search for any rows that contain the string 30% anywhere in the comment column, specify a WHERE clause such as WHERE comment LIKE '%30!%%' ESCAPE '!'. If ESCAPE and the escape character are not specified, the Database Engine returns any rows with the string 30.

If there is no character after an escape character in the LIKE pattern, the pattern is not valid and the LIKE returns FALSE. If the character after an escape character is not a wildcard character, the escape character is discarded and the character following the escape is treated as a regular character in the pattern. This includes the percent sign (%), underscore (\_), and left bracket ([) wildcard characters when they are enclosed in double brackets ([ ]). Also, within the double bracket characters ([ ]), escape characters can be used and the caret (^), hyphen (-), and right bracket (]) can be escaped.

To search string containg %

SELECT \* FROM wildcard where name like '%|%%' ESCAPE '|'

SELECT \* FROM wildcard where name like '%[%]%'

**In MySQL**

For searching % character, we can use escape character **\**

where name like '%**\%**%'

or name like '%**\.**%' # searching .

or name like '%**\\_**%' # searching \_

# Using Wildcard Characters As Literals

|  |  |
| --- | --- |
| Symbol | Meaning |
| LIKE '5[%]' | 5% |
| LIKE '[\_]n' | \_n |
| LIKE '[a-cdf]' | a, b, c, d, or f |
| LIKE '[-acdf]' | -, a, c, d, or f |
| LIKE '[ [ ]' | [ |
| LIKE ']' | ] |
| LIKE 'abc[\_]d%' | abc\_d and abc\_de |
| LIKE 'abc[def]' | abcd, abce, and abcf |

# SP\_Help

It returns about bject details in tabular form with few column. View,DML Trigger or table in current database.

Sp\_help object\_name

# SP\_HELPTRIGGER

Returns the type or types of DML triggers defined on the specified table for the current database. sp\_helptrigger cannot be used with DDL triggers.

SP\_HELPTRIGGER table\_name

# SP\_HELPTEXT

Displays the definition of a user-defined “Procedure”,Function,View,Table,trigger

SP\_HELPTEXT Tr\_AFTER

# SP\_EXECUTESQL

It accept sql statement in data type ntext/nchar/nvarchar

EXEC SP\_EXECUTESQL N'SELECT \* FROM DATA2'

--

DECLARE @str nvarchar(100);

SET @str='SELECT ''khalid'' as kk UNION SELECT ''Anwar'' as kk'

EXEC SP\_EXECUTESQL @str

# JOINS

Equi Join

# Natural Join

The SQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with the same name of associated tables will appear once only.

* The associated tables have one or more pairs of identically named columns.
* The columns must be the same data type.
* **Don’t use ON** clause in a natural join.

SELECT \* FROM table1 NATURAL JOIN table2;

*The INNER JOIN using ON clause do the same job as natural Join.*

A NATURAL JOIN can be an INNER join, a LEFT OUTER join, or a RIGHT OUTER join. The default is INNER join.

## Syntax

[***TableExpression***](https://docs.oracle.com/javadb/10.6.2.1/ref/rreftableexpression.html#rreftableexpression) **NATURAL [ { LEFT | RIGHT } [ OUTER ] | INNER ] JOIN {** [**TableViewOrFunctionExpression**](https://docs.oracle.com/javadb/10.6.2.1/ref/rrefsqlj33215.html#rrefsqlj33215) **| (** [***TableExpression***](https://docs.oracle.com/javadb/10.6.2.1/ref/rreftableexpression.html#rreftableexpression) **) }**

Difference between natural join and inner join

There is one significant difference between INNER JOIN and NATURAL JOIN is the number of columns returned. Joining key column will not come twice as it come in inner join.

## Find duplicate record

SELECT \* FROM

(SELECT \*,ROW\_NUMBER() OVER(PARTITION BY sn,name,school ORDER BY (SELECT 0)) AS RN

FROM DUPLICATE) TMP

WHERE RN>1

-

SELECT \* FROM

(SELECT \*,ROW\_NUMBER() OVER(PARTITION BY sn,name,school ORDER BY sn) AS RN

FROM DUPLICATE) TMP

WHERE RN>1

## To remove duplicate record

**In MS SQL**

DELETE TMP FROM

(SELECT \*,ROW\_NUMBER() OVER(PARTITION BY sn,name,school ORDER BY sn) AS RN

FROM DUPLICATE) TMP

WHERE RN>1

**Using CTE**

WITH TMP AS

(SELECT \*,ROW\_NUMBER() OVER(PARTITION BY sn,name,school ORDER BY sn) AS RN

FROM DUPLICATE

)

DELETE FROM TMP WHERE RN>1

## Show All Rows with an Above-Average Value

SELECT

  first\_name,

  last\_name,

  salary

FROM employee

WHERE salary > ( SELECT AVG(salary) FROM employee )

# Correlated Subqueries

A correlated subquery is evaluated once for each row processed by the outer query. It is sub query that uses values from outer query. It is top to down approach

The parent statement can be a SELECT, UPDATE, or DELETE statement.

## Select Correlated

SELECT column1, column2,FROM table1 outer WHERE column1 operator

(SELECT column1, column2 FROM table2

WHERE expr1 = outer.expr2);

## Update Correlated

UPDATE table1 **alias1**

SET column = (SELECT expression FROM table2 **alias2** WHERE **alias1**.column =**alias2**.column);

## Delete Correlated

DELETE FROM table1 **alias1**

WHERE column1 operator (SELECT expressionFROM table2 **alias2** WHERE **alias1**.column = **alias2**.column);

<https://www.geeksforgeeks.org/sql-correlated-subqueries/>

## To find Second Highest Salary

1.

SELECT MAX(SALARY) FROM EMPloyees WHERE SALARY not in (SELECT MAX(SALARY) FROM EMPLOYEES)

2.

SELECT name, MAX(salary) AS salary FROM employee WHERE salary < (SELECT MAX(salary) FROM employee);

3.

SELECT \* FROM

(SELECT \*,DENSE\_RANK() OVER(ORDER BY SALARY DESC) AS rnk FROM EMPloyees) T

where rnk=2

4.

Select name,salary from employee A

where n-1 = (Select **count**(**distinct** salary) from employee **B** where **B**.salary>A.salary)

**Department wise maximum salary**

SELECT DEPT,MAX(SALARY) FROM employee2 GROUP BY DEPT

**Department wise 2nd high salary**

SELECT \* FROM EMPLOYEE2 ORDER BY DEPT,SALARY

(

SELECT \*, **dense\_rank**() **OVER**(**PARTITION BY** DEPT **ORDER BY** Salary desc) as drnk FROM EMPLOYEE2 ) A

WHERE drnk=2

--Using correlated sub Query

SELECT a.deptno, a.salary FROM Emp a

WHERE 1 = (SELECT COUNT(DISTINCT salary)

FROM Emp b WHERE b.salary > a.salary AND b.deptno = a.deptno)

group by a.deptno

**Find manager list**

SELECT Distinct e2.employeeNumber As Mgr FROM emp e1 inner JOIN emp e2

on **e2.employeeNumber**=**e1.reportsTo**

**find department with highest average salary for employees who is for more than 2 years.**

SELECT d.DepartmentName, AVG(e.Salary) AS AverageSalary FROM Employees e

INNER JOIN Departments d ON e.DepartmentID = d.DepartmentID

WHERE e.HireDate < DATEADD(year, -2, GETDATE())

GROUP BY d.DepartmentName ORDER BY AverageSalary DESC LIMIT 1;

## Show All Rows with an Above-Average Value

SELECT first\_name,last\_name,salary

FROM employee

WHERE salary > (SELECT **AVG**(salary) FROM employee )

**Employees with Salaries Higher Than Their Departmental Average**

SELECT first\_name,last\_name,salary

FROM employee e1

WHERE salary >

    (SELECT **AVG**(salary)FROM employee e2

     WHERE e1.departmet\_id = e2.department\_id)

**Conditional Sum**

Find sum of salary for group by department

SELECT

  SUM (

CASE WHEN dept\_id IN ('SALES','HUMAN RESOURCES') THEN salary

     ELSE 0 END

) AS **total\_salary\_sales\_and\_hr**,

  SUM (

CASE WHEN dept\_id IN ('IT','SUPPORT') THEN salary

    ELSE 0 END

) AS **total\_salary\_it\_and\_support**

FROM employee

**Group Rows by a Range**

SELECT

CASE

WHEN salary <= 20000 THEN 'low'

WHEN salary > 20000 AND salary <= 50000 THEN 'medium'

WHEN salary > 50000 THEN 'high'

END AS salary\_category,

COUNT(\*) AS number\_of\_employees

FROM Emp

GROUP BY

CASE

WHEN salary <= 20000 THEN 'low'

WHEN salary > 20000 AND salary <= 50000 THEN 'medium'

WHEN salary > 50000 THEN 'high'

END

## EXISTS/NOT EXISTS

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

**Syntax**

SELECT column\_name(s) FROM table\_name  
WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition);

**Example**

If there is no correlated query then it will fetch all record from outer query. So better to use correlated sub query.

SELECT SupplierName FROM Suppliers

WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID);

Output// above query will fetch all matching record where SupplierID is matching in both table

SELECT SupplierName FROM Suppliers

WHERE NOT EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);

Output// above query will fetch all matching record where SupplierID is not matching in both table

## ****How can you optimize a slow-running query?****

* There are several techniques, including:
* Using appropriate indexes
* Avoiding unnecessary joins and subqueries
* Using efficient functions and operators
* Analyzing execution plans to identify bottlenecks

## ****Explain the concept of materialized views and their benefits.****

Materialized views are pre-computed copies of database queries stored as tables. They improve query performance by providing pre-aggregated or filtered data, reducing processing needs for repeated queries.

# Ranking Function

* The ranking functions always assign rank on basis of **ORDER BY** clause.
* These functions are always used with **OVER()** clause.
* The assignment of rank to rows always start with 1 for every new partition.

ROW\_NUMBER will always generate unique values without any gaps, even if there are ties.

RANK can have gaps in its sequence and when values are the same, they get the same rank. **With gaps**

DENSE\_RANK also returns the same rank for ties, but doesn’t have any gaps in the sequence. **Without gaps**

SELECT \*,

**row\_number**() **OVER**(ORDER BY MARKS) As rn,

**RANK**() **OVER**(ORDER BY MARKS) As rnk,

**dense\_rank**() **OVER**(ORDER BY MARKS) As drnk

FROM pivot1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NAME | COLLEGE | ROLL\_NO | SUBJECT | MARKS | rn | rnk | drnk |
| PUSHKAR | MSIT | 898888 | GRAPHICS | 78 | 1 | 1 | 1 |
| ROMY | BVP | 261150 | NETWORKING | 87 | 2 | 2 | 2 |
| ROMY | BVP | 261150 | DBMS | 90 | 3 | 3 | 3 |
| PUSHKAR | MSIT | 898888 | NETWORKING | 90 | 4 | 3 | 3 |
| PUSHKAR | MSIT | 898888 | DBMS | 91 | 5 | 5 | 4 |
| ROMY | BVP | 261150 | GRAPHICS | 95 | 6 | 6 | 5 |

# Regular Expression in MySql-- REGEXP

**Case 1** − If you want only those rows which have exactly 10 digits and all must be only digit, use the below regular expression.

SELECT \* FROM yourTableName WHERE yourColumnName **REGEXP** '^[0-9]{10}$';

**Case 2 –** only integer value of **any length**

SELECT \*FROM yourTableName WHERE yourColumnName REGEXP '**^**[0-9]\*$';

**Case3-** mobile number start with 8 or 9 and should be of 10 digit REGEXP '[89][0-9]{9}';

**Case4**- password contains First capital letter, followed by small letter and one integer in between

[A-B][a-z]\*[0-9][a-z]\*

**Case5:** EmailId like [khalid0142759@gmail.com](mailto:khalid0142759@gmail.com) and contains \_ . –

[a-zA-Z0-9\\_\.\-]\*[@][a-z]\*[\.][a-z]{2,3}

**Case6**  search .(dot) in a string or %

Regexp ‘[\.]’

Regexp ‘[\%]’

**Case6**  search \(backsalsh) in a string.. need to use thrice because two \\ will be escaped.

Regexp ‘\\\’

|  |  |  |
| --- | --- | --- |
| **Pattern** | **What the Pattern matches** | **Example** |
| ^ | caret(^) matches **Beginning** of string | '^man' == Like 'man%' ;beginning with this word '^[0-9]\*$' == first character will be from 0 to 9 and remaing could be any character of any length |
| $ | **End** of string | 'khan$' == Like '%khan' '[ab]$' == Like '%a' or '%b' |
| [abc] | Any character listed between the square brackets | '[abc]'== string contains either a or b or c like '%a%' or '%b%' or '%c%' '**[ab]c'** ==Like '%ac%' or '%bc%' ; all combination '^[ab] == Like 'a%' or 'b%' |
| [^abc] | Any character **not listed** between the square brackets | string does not contains either a or b or c |
| [A-Z] | match any upper case letter. | range A,B,C,…Z [a-zA-Z] ==[abc…zABC…Z] |
| [a-z] | match any lower case letter | range a,b,c,…z |
| [0-9] | match any digit from 0 through to 9 | range 0,1,2,…9 |
| | | multiple pattern; 'ram|khan|sri' | 'ram|khan|sri' ;string containing **ram** **Or** **Khan** **Or** **sri** '^ram|khan$|sri' ;string begin with ram Or ends with Khan or contains sri |
| \* | **Quantifier:** Matches zero or more times | 'man' == Like '%man% |
| + | Previous character must occur **at least once** or more times. |  |
| . | Any **single** character |  |
| ? | preceding character is optional. Match 0 or 1 time |  |
| {n} | Matches exactly n occurrences of the preceding character. |  |
| {n,} | Matches n or more occurrences of the preceding character. |  |
| {n,m} | Matches between n and m occurrences of the preceding element |  |

## Quantifiers

In regular expressions, quantifiers are metacharacters that specify how many times the previous character or group should be matched. Above green colors are quantifiers.

E.g [a-z]\* [a-z]+ [a-z]?