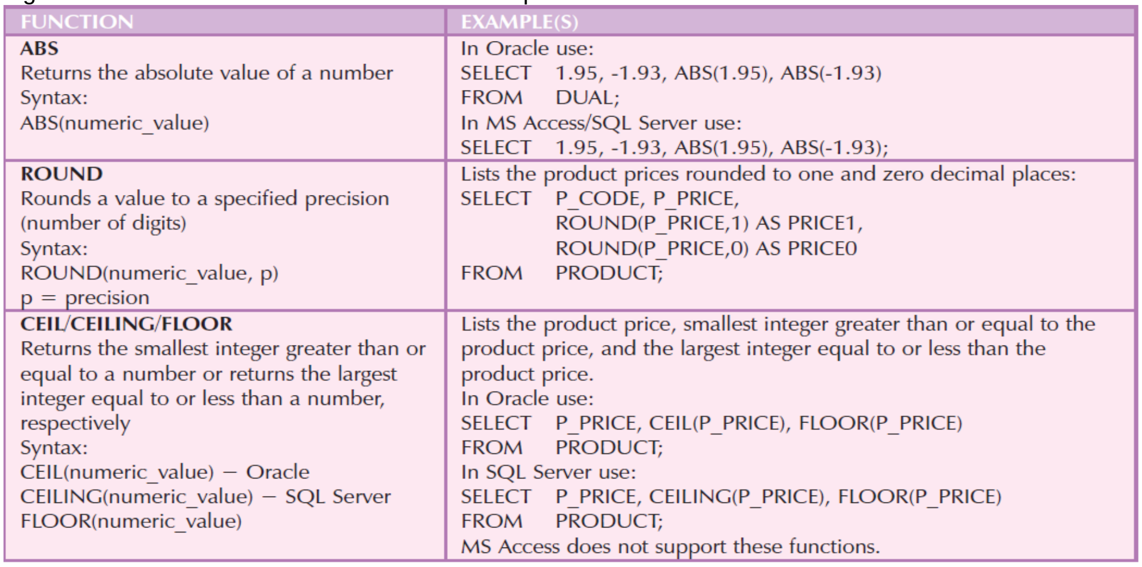
Q1) Explain various numeric and string functions used in oracle with syntax and example.

Ans.

Numeric Functions:

Numeric functions can be grouped in many different ways, such as algebraic, trigonometric, and logarithmic. Numeric functions take one numeric parameter and return one value.



FUNCTION	EXAMPLE(S)

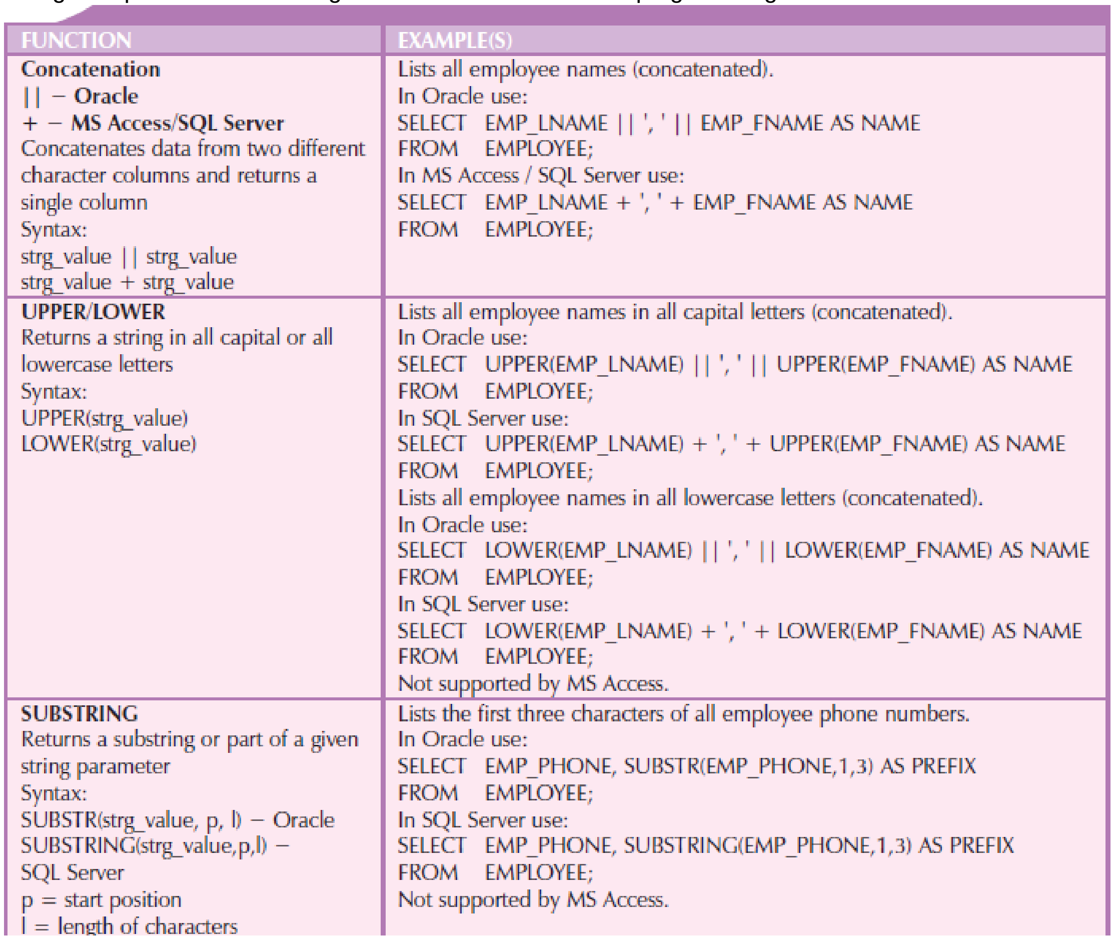
ABS
Returns the absolute value of a number
Syntax:
ABS(numeric_value)
	In Oracle use:
	SELECT 1.95, -1.93, ABS(1.95), ABS(-1.93)
	FROM DUAL;
	In MS Access/SQL Server use:
	SELECT 1.95, -1.93, ABS(1.95), ABS(-1.93);

ROUND Rounds a value to a specified precision (number of digits) Syntax: ROUND(numeric_value, p) p = precision	Lists the product prices rounded to one and zero decimal places: SELECT P_CODE, P_PRICE, ROUND(P_PRICE,1) AS PRICE1, ROUND(P_PRICE,0) AS PRICE0 FROM PRODUCT;

CEIL/CEILING/FLOOR
Returns the smallest integer greater than or equal to a number or returns the largest integer equal to or less than a number, respectively
Syntax:
CEIL(numeric_value) – Oracle
CEILING(numeric_value) – SQL Server
FLOOR(numeric_value)
	Lists the product price, smallest integer greater than or equal to the product price, and the largest integer equal to or less than the product price.
	In Oracle use:
	SELECT P_PRICE, CEIL(P_PRICE), FLOOR(P_PRICE)
	FROM PRODUCT;
	In SQL Server use:
	SELECT P_PRICE, CEILING(P_PRICE), FLOOR(P_PRICE)
	FROM PRODUCT;
	MS Access does not support these functions.

String Functions:

String manipulations are among the most-used functions in programming.

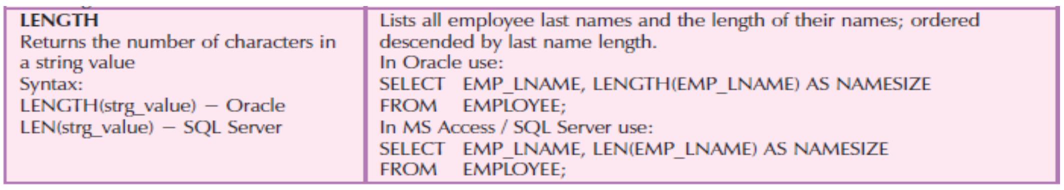


FUNCTION	EXAMPLE(S)

Concatenation
|| – Oracle
+ – MS Access/SQL Server
Concatenates data from two different character columns and returns a single column
Syntax:
strg_value || strg_value
strg_value + strg_value
	Lists all employee names (concatenated).
	In Oracle use:
	SELECT EMP_LNAME || ', ' || EMP_FNAME AS NAME
	FROM EMPLOYEE;
	In MS Access / SQL Server use:
	SELECT EMP_LNAME + ', ' + EMP_FNAME AS NAME
	FROM EMPLOYEE;

UPPER/LOWER
Returns a string in all capital or all lowercase letters
Syntax:
UPPER(strg_value)
LOWER(strg_value)
	Lists all employee names in all capital letters (concatenated).
	In Oracle use:
	SELECT UPPER(EMP_LNAME) || ', ' || UPPER(EMP_FNAME) AS NAME
	FROM EMPLOYEE;
	In SQL Server use:
	SELECT UPPER(EMP_LNAME) + ', ' + UPPER(EMP_FNAME) AS NAME
	FROM EMPLOYEE;
	Lists all employee names in all lowercase letters (concatenated).
	In Oracle use:
	SELECT LOWER(EMP_LNAME) || ', ' || LOWER(EMP_FNAME) AS NAME
	FROM EMPLOYEE;
	In SQL Server use:
	SELECT LOWER(EMP_LNAME) + ', ' + LOWER(EMP_FNAME) AS NAME
	FROM EMPLOYEE;
	Not supported by MS Access.

SUBSTRING
Returns a substring or part of a given string parameter
Syntax:
SUBSTR(strg_value, p, l) – Oracle
SUBSTRING(strg_value,p,l) – SQL Server
p = start position
l = length of characters
	Lists the first three characters of all employee phone numbers.
	In Oracle use:
	SELECT EMP_PHONE, SUBSTR(EMP_PHONE,1,3) AS PREFIX
	FROM EMPLOYEE;
	In SQL Server use:
	SELECT EMP_PHONE, SUBSTRING(EMP_PHONE,1,3) AS PREFIX
	FROM EMPLOYEE;
	Not supported by MS Access.



LENGTH
Returns the number of characters in a string value
Syntax:
LENGTH(strg_value) – Oracle
LEN(strg_value) – SQL Server
	Lists all employee last names and the length of their names; ordered descended by last name length.
	In Oracle use:
	SELECT EMP_LNAME, LENGTH(EMP_LNAME) AS NAMESIZE
	FROM EMPLOYEE;
	In MS Access / SQL Server use:
	SELECT EMP_LNAME, LEN(EMP_LNAME) AS NAMESIZE
	FROM EMPLOYEE;

Q2) Describe oracle sequence. Explain sequence in Oracle with syntax and example.

Ans:

An Oracle Sequence is a database object, just like a table or view, that represents a sequence of integers that can be used by any table or view in the global database namespace. A Sequence's values can be accessed using the NEXTVAL, and CURRVAL pseudo-columns. Oracle does not support the AutoNumber data type or the Identity column property. Instead, you can use a "sequence" to assign values to a column on a table. But an Oracle sequence is very different from the Access AutoNumber data type and deserves close scrutiny:

- _ Oracle sequences are an independent object in the database. (Sequences are not a data type.)
- _ Oracle sequences have a name and can be used anywhere a value is expected.
- _ Oracle sequences are not tied to a table or a column.
- _ Oracle sequences generate a numeric value that can be assigned to any column in any table.
- _ The table attribute to which you assigned a value based on a sequence can be edited and modified.
- _ An Oracle sequence can be created and deleted anytime.

The basic syntax to create a sequence in Oracle is:

CREATE SEQUENCE *name* [START WITH *n*] [INCREMENT BY *n*] [CACHE | NOCACHE]

where:

- _ *name* is the name of the sequence.
- _ *n* is an integer value that can be positive or negative.
- _ *START WITH* specifies the initial sequence value.
- _ *INCREMENT BY* determines the value by which the sequence is incremented. (The default increment value
- is 1. The sequence increment can be positive or negative to enable you to create ascending or descending
- sequences.)
- _ The *CACHE* or *NOCACHE* clause indicates whether Oracle will preallocate sequence numbers in memory.
- (Oracle preallocates 20 values by default.)

For example, you could create a sequence to automatically assign values to the customer code each time a new customer is added and create another sequence to automatically assign values to the invoice number each time a new invoice is added. The SQL code to accomplish those tasks is:

CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;
CREATE SEQUENCE INV_NUMBER_SEQ START WITH 4010 NOCACHE;

To use sequences during data entry, you must use two special pseudo-columns: NEXTVAL and CURRVAL. NEXTVAL retrieves the next available value from a sequence, and CURRVAL retrieves the current value of a sequence.

For example, you can use the following code to enter a new customer:
INSERT INTO CUSTOMER VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2008', 0.00);
The preceding SQL statement adds a new customer to the CUSTOMER table and assigns the value 20010 to the CUS_CODE attribute. Let's examine some important sequence characteristics:

- _ CUS_CODE_SEQ.NEXTVAL retrieves the next available value from the sequence.
- _ Each time you use NEXTVAL, the sequence is incremented.
- _ Once a sequence value is used (through NEXTVAL), it cannot be used again. If, for some reason, your SQL
- statement rolls back, *the sequence value does not roll back*. If you issue another SQL statement (with another
- NEXTVAL), the next available sequence value will be returned to the user—it will look as though the sequence
- skips a number.
- _ You can issue an INSERT statement without using the sequence.

CURRVAL retrieves the current value of a sequence—that is, the last sequence number used, which was generated with a NEXTVAL. You cannot use CURRVAL unless a NEXTVAL was issued previously in the same session. The main use for CURRVAL is to enter rows in dependent tables.

Q3) What is trigger? What is its purpose? Explain types of trigger?
Q4) What is trigger? Explain in detail, syntax to create trigger in oracle.
Ans:
DEF:

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

It is useful to remember that:

- _ A trigger is invoked before or after a data row is inserted, updated, or deleted.
- _ A trigger is associated with a database table.
- _ Each database table may have one or more triggers.
- _ A trigger is executed as part of the transaction that triggered it.

Triggers are critical to proper database operation and management. For example:

- _ Triggers can be used to enforce constraints that cannot be enforced at the DBMS design and implementation
- levels.
- _ Triggers add functionality by automating critical actions and providing appropriate warnings and suggestions
- for remedial action. In fact, one of the most common uses for triggers is to facilitate the enforcement of

- referential integrity.
- _ Triggers can be used to update table values, insert records in tables, and call other stored procedures.

Triggers play a critical role in making the database truly useful; they also add processing power to the RDBMS and to the database system as a whole. Oracle recommends triggers for:

- _ Auditing purposes (creating audit logs).
- _ Automatic generation of derived column values.
- _ Enforcement of business or security constraints.
- _ Creation of replica tables for backup purposes.

Syntax:

create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]

Syntax In Oracle:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER } triggering_event ON table_name
[FOR EACH ROW]
[FOLLOWS | PRECEDES another_trigger]
[ENABLE / DISABLE ]
[WHEN condition]
DECLARE
    declaration statements
BEGIN
    executable statements
EXCEPTION
    exception_handling statements
END;
```

Explanation of syntax:

- create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
- [before | after]: This specifies when the trigger will be executed.
- {insert | update | delete}: This specifies the DML operation.
- on [table_name]: This specifies the name of the table associated with the trigger.
- [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
- [trigger_body]: This provides the operation to be performed as trigger is fired.

SQL Trigger to problem statement.

```
create trigger stud_marks

before INSERT

on

Student

for each row

set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

Different Types of Triggers:

1. DML Triggers:
    - DML triggers is a special type of stored procedure that automatically takes effect when a data manipulation language (DML) event takes place that affects the table or view defined in the trigger. DML events include INSERT, UPDATE, or DELETE statements.
    - DML triggers can be used to enforce business rules and data integrity, query other tables, and include complex Transact-SQL statements. The trigger and the statement that fires it are treated as a single transaction, which can be rolled back from within the trigger. If a severe error is detected (for example, insufficient disk space), the entire transaction automatically rolls back.

    Types of DML Triggers:

    - AFTER trigger
      AFTER triggers are executed after the action of the INSERT, UPDATE, MERGE, or DELETE statement is performed. AFTER triggers are never executed if a constraint violation occurs; therefore, these triggers cannot be used for any processing that might prevent constraint violations. For every INSERT, UPDATE, or DELETE action specified in a MERGE statement, the corresponding trigger is fired for each DML operation.

    - INSTEAD OF trigger
      INSTEAD OF triggers override the standard actions of the triggering statement. Therefore, they can be used to perform error or value checking on one or more columns and perform additional actions before inserting, updating or deleting the row or rows.

2. DDL Triggers:
    - DDL triggers fire in response to a variety of Data Definition Language (DDL) events. These events primarily correspond to Transact-SQL statements that start with the keywords CREATE, ALTER, DROP, GRANT, DENY, REVOKE or UPDATE STATISTICS. Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.

Use DDL triggers when you want to do the following:

- Prevent certain changes to your database schema.
- Have something occur in the database in response to a change in your database schema.
- Record changes or events in the database schema.

Types Of DDL Trigger:

- Transact-SQL DDL Trigger:
  A special type of Transact-SQL stored procedure that executes one or more Transact-SQL statements in response to a server-scoped or database-scoped event. For example, a DDL Trigger may fire if a statement such as ALTER SERVER CONFIGURATION is executed or if a table is deleted by using DROP TABLE.
- CLR DDL Trigger:
  Instead of executing a Transact-SQL stored procedure, a CLR trigger executes one or more methods written in managed code that are members of an assembly created in the .NET Framework and uploaded in SQL Server.

3. Logon Trigger:

- Logon triggers fire stored procedures in response to a LOGON event.
- This event is raised when a user session is established with an instance of SQL Server.
- Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established.
- Therefore, all messages originating inside the trigger that would typically reach the user, such as error messages and messages from the PRINT statement, are diverted to the SQL Server error log.
- Logon triggers do not fire if authentication fails.

Q5) What is stored procedure in PL/SQL? Give its advantages. Explain in detail, syntax to create stored procedure in PL/SQL.
Q6) Explain in detail, syntax to create stored procedure in PL/SQL. Write PL/SQL Procedure to find factorial of given number.

Ans:
A **stored procedure** is a named collection of procedural and SQL statements. Just like database triggers, stored procedures are stored in the database. One of the major advantages of stored procedures is that they can be used to encapsulate and represent business transactions.
For example, you can create a stored procedure to represent a
product sale, a credit update, or the addition of a new customer. By doing that, you can encapsulate SQL statements within a single stored procedure and execute them as a single transaction.

There are two clear advantages to the use of stored procedures:

- _ Stored procedures substantially reduce network traffic and increase performance. Because the procedure is stored at the server, there is no transmission of individual SQL statements over the network. The use of stored procedures improves system performance because all transactions are executed locally on the RDBMS, so each SQL statement does not have to travel over the network.
- _ Stored procedures help reduce code duplication by means of code isolation and code sharing (creating unique PL/SQL modules that are called by application programs), thereby minimizing the chance of errors and the cost of application development and maintenance.

To create a stored procedure, you use the following syntax:

```
CREATE OR REPLACE PROCEDURE procedure_name [(argument [IN/OUT] data-
type, _ )]
                             [IS/AS]
                             [variable_namedata type[:=initial_value]
                        ]
BEGIN
                 PL/SQL or SQL statements;
...
END;
```

To execute the stored procedure, you must use the following syntax:

```
EXEC procedure_name[(parameter_list)];
```

Note the following important points about stored procedures and their syntax:

- _ *argument* specifies the parameters that are passed to the stored procedure. A stored procedure could have
- zero or more arguments or parameters.
- _ *IN/OUT* indicates whether the parameter is for input, output, or both.
- _ *data-type* is one of the procedural SQL data types used in the RDBMS. The data types normally match those
- used in the RDBMS table-creation statement.
- _ Variables can be declared between the keywords IS and BEGIN. You must specify the variable name, its data
- type, and (optionally) an initial value.

PL/SQL Procedure to find factorial of given number:

```
create or replace procedure factorial(n in number)
is
v number :=1;
begin
for i in  1..n
loop
v :=v * i;
end loop;
dbms_output.put_line(v);
end;
```

Q7) Explain in detail, syntax to create function in PL/SQL. Write PL/SQL function to find factorial of given number.

Ans:

PL/SQL Function:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Here:

- Function_name: specifies the name of the function.
- [OR REPLACE] option allows modifying an existing function.
- The optional parameter list contains name, mode and types of the parameters.
- IN represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.
- The function must contain a return statement.
- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

PL/SQL Function Example

Let's see a simple example to create a function.

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
```

PL/SQL function to find factorial of given number

```
declare
-- it gives the final answer after computation--
fac number :=1;
```

```
-- given number n--
-- taking input from user--
n number := &1;

-- start block--
begin

-- start while loop--
while n > 0 loop

-- multiple with n and decrease n's value--
fac:=n*fac;
n:=n-1;
end loop;
-- end loop--

-- print result of fac--
dbms_output.put_line(fac);

-- end the begin block--
end;
```