Oracle - RDBMS

SQL commands

DDL	DML	TCL	DCL
• create.	• Insert	• Commit	• Grant
• Alter.	• update	• Rollback	• Revoke.
• Truncate.	• delete	•	
• Drop.	• select		

Naming convention

- begin with an alphabet.
- A to Z or a to z
- '$', '__', '#' allowed as special chars
- no reserve words.
- names are case insensitive.

Constraints

-> primary key (unique value)
not null

-> foreign key.

* number of attributes defines the degree of relation in DBMS

oracle data types & ~~exer~~ CREATE ~~Statement~~.

Data types.

- datatype of each column must be specified.
- each data type has a specific format of storage, constraints and a range of values.

Datatype	Description	Type of Data.
Varchar2(size)	• memory occupied is based on number of characters in the input.	'John', 'Tom123' 'Alex@#'.
Char(size)	• fixed size of ~~memory~~ occupied.	'John', 'Tom123' 'Alex@ #'
Number(p)	• Numeric data • P-precision - number of digits	123 4567 8
Number(p, s)	- Floating point number. • P-precision - total number of digits reserved including decimal. • S-scale - total number of digits after decimal.	7.89 789.453 3.14
Date	• Date values are in the format 'DD-Mon-YY'	17-Oct-20

CREATE Statement :-

- All DB objects are created using 'CREATE' statement of DDL

Syntax

CREATE ObjectType ObjectName (Object Definition) ;

Description

CREATE -> keyword.
ObjectType -> Table, View, Sequence, Index....
ObjectName -> User defined Name.

ObjectDefinition -> Detailed structure of the object.
within ()
; SQL statement terminator.

CREATE TABLE - Syntax

CREATE TABLE TableName
(ColumnName1 Datatype [[CONSTRAINT constraintName] constraintType];
ColumnName2 ,
,
,ColumnName3 , , ,

)

CREATE, TABLE, CONSTRAINT -> keywords

DataType -> NUMBER, VARCHAR2, DATE, ...

ConstraintType -> Primary key, Not Null, Unique, Foreign key, Check, Default

'[]' -> contains optional part

',' -> column seperators

';' SQL statement terminator..

Example.

```
CREATE TABLE Employees
(
  EmployeeNo NUMBER(5) CONSTRAINT PK_Emp Primary key,
  Name VARCHAR2(15) Not Null,
  BirthDate DATE,
  Salary NUMBER(8,2) Check(Salary>=0),
  Email VARCHAR2(30) Unique,
  DeptNo REFERENCES Department(DepartmentNo)
);

CREATE TABLE Department
(
  DepartmentNo Number(3) CONSTRAINT PK_Dept
  Primary key,
  Name VARCHAR2(15) Not Null,
  Location VARCHAR2(15)
);
```

- Table should only have one Primary key, but it consists of more than one attribute.

```
CREATE TABLE Enrollment
(
  EmployeeNo NUMBER(5) REFERENCES Employees (Emp
        -loyeeNO)
  ProjectNo VARCHAR(4) REFERENCES Project (ProjectNO),
  Start_Date DATE,
  End_Date DATE,

  PRIMARY KEY(EmployeeNo, ProjectNO),   } Table level constrain
  Check Start_Date < End_Date).
);
```

Alter, Truncate & drop statements.

Alter statement is used to modify structure of the table which is already in the database.

ADD
For adding a new column with or without a constraint and CONSTRAINT to an existing column.

DROP
Remove an existing column or a constraint from a column.

MODIFY
change a data type or resize the column.

RENAME
Modify the name of the column or table.

ADD - Syntax
ALTER TABLE TableName ADD
(Column1 DataType [[constraint Constraint1] constraintType],
[[CONSTRAINT Constraint2] constraintType (Cname1[, Cname2, ...])
...
ColumnN Datatype...
);

```
Alter Table Employees ADD
( phNo Number(8) constraint Uk-PhNO UNIQUE,
  DeptNo Number(3) REFERENCES Department
   (DepartmentNO),
  CONSTRAINT Pk-Emp Primary key (EmployeeNO),
   check(Salary >0),
   unique(Email)
);
```

Resize a Column

Syntax

```
ALTER TABLE TableName
    MODIFY ColumnName Newolddatatype (Newoldsize)
        [NOT NULL];
```

SQL statement

```
ALTER TABLE Department MODIFY NAME VARCHAR(20);
```

DROP Column - syntax.

Syntax ->

```
ALTER TABLE TableName DROP COLUMN ColumnName;
```

SQL statement ->

```
ALTER TABLE Department DROP COLUMN Location;
```

DROP CONSTRAINT

```
ALTER TABLE TableName DROP CONSTRAINT ConstraintName;
```

.

```
ALTER TABLE Employees DROP CONSTRAINT Uk_PhNo;
```

- Oracle provides a name to each constraint if its not defined by the users.
- Oracle stores a collection of predefined tables to store meta data about database objects created by the users.

- meta data is called data dictionary which has information like:
  - Table Name.
  - Owner of the table.
  - Column Name.
  - Data type.
  - Constraint type
  - Constraint name.

- User has to access these details to know the predefined name of the constraints given by oracle.

♣ Disabling a constraint

Syntax →

```
ALTER TABLE TableName DISABLE CONSTRAINT
  ConstraintName;
```

SQL statement →

```
ALTER TABLE Employees DISABLE CONSTRAINT PK_EMP;
```

♣ ♣ ♣ ~~ALTER~~ Enabling a constraint.

```
ALTER TABLE TableName ENABLE CONSTRAINT
  constraintname;
```

```
ALTER TABLE Employees ENABLE CONSTRAINT PK_EMP;
```

RENAMING a column -

ALTER TABLE TableName RENAME COLUMN oldcolname TO newcolname.

ALTER TABLE Employee RENAME COLUMN PhNo to PhoneNo;

Renaming a table

RENAME oldtablename to newtablename.

TRUNCATE TABLE TableName;

DROP objectType ObjectName;
DROP TABLE tablename
DROP TABLE tablename cascade constraints
// for referential integrity

18/01/2020
Saturday.

Data manipulation:-
INSERT, UPDATE, DELETE, SELECT

- Insert Records
- Create sequence
- Alter a sequence
- Drop a sequence.

INSERT SYNTAX

Column names in the INSERT statement must be specified when inserting data if:

- partial attributes have to be stored, rather than all the attributes.
- the order of data is different than the order of the attribute in a table.

```
→ INSERT INTO
→     TableName[(col1, col2, colN)]
→     VALUES (Data1, Data2, ..., DataN);
```

e.g:-

Department No	Name	Location
101	Testing	Downtown
102	Development	
104	Admin Services	

```
INSERT INTO Department
    VALUES (101, 'Testing', 'Downtown');

INSERT INTO Department (DepartmentNo, Name)
    VALUES (104, 'Admin Services');
```

oracle gives ORA-02291 integrity constrai violated - parent key not found

```
INSERT INTO Department
    VALUES (102, 'Development', NULL);
```

SEQUENCE - intro

- one of the database objects
- generates seq of unique integers that can be used for primary key or unique value.
- Independent of users & tables
- one user/table can never acquire seq no. generated by another.
- Indepent of transaction whether it is committing or rolling back.

Syntax -

```
CREATE SEQUENCE SequenceName
[
 [INCREMENT BY integer]
 [START WITH integer]
 [MAXVALUE integer | NOMAXVALUE]
 [MINVALUE integer | NOMINVALUE]
 [CYCLE | NOCYCLE]
 [CACHE integer | NOCACHE]
 [ORDER | NOORDER] ];
```

- if none of the clauses are specified, then sequence object is created in an ascending order, that starts with 1 and is incremented by 1 with no upper limit.

CREATING a sequence:-

Seq name	Start	Increment by
FirstSeq	Default value	Default value
Sec seq	Default	2
Dept seq	101	Default

Create statements.

-> CREATE SEQUENCE FirstSeq;

-> CREATE SEQUENCE SecondSeq INCREMENT BY 2;

-> CREATE SEQUENCE DeptSeq STARTWITH 101;

CURRVAL - returns the current value of the seq

NEXTVAL - returns value after increment.

-> SELECT FirstSeq.CURRVAL FROM DUAL;

-> SELECT FirstSeq.NEXTVAL FROM DUAL;

DUAL is dummy table in ORACLE has a single col VARCHAR type

Use of seq in INSERT

```
INSERT INTO Dept
    VALUES(DeptSeq.NEXTVAL,'Testing','Poduna');

INSERT INTO Dept
    VALUES(DeptSeq.NEXTVAL,'Development',NULL);
```

% ALTER Statement is used to change all para of a seq but starting number. to change it drop the seq and the recreate it.

-> DROP SEQUENCE FirstSeq;

- Update Records
- Delete Records
- Create Foreign keys with Cascade Delete.
- Differentiate b/w truncate & Delete statement.

UPDATE - Syntax.

→ The DML UPDATE statement is used to change the existing values in a table or in the base table of a view or the master table of a materialized view.

→ By default, the update statement changes the existing values of a column to all records in a table.

→ A WHERE clause is used to limit the updated rows.

→ More than one condition can be specified in a WHERE clause using LOGICAL operator.

Syntax

```
UPDATE TableName
SET
[ col1 = NewValue1, col2 = NewValue2, ...,
  colN = newvalueN ]
[ WHERE condition ];
```

```
UPDATE Department
SET
Location = 'Down Town'
WHERE
```

% It changes every row on col location

```
UPDATE Department
SET
Location = 'DownTown'
WHERE DepartmentNo = 103;
```

% This codes tells which row to change. by reference to primary key

Delete - Syntax

- The Delete statement is used to remove record(s) from a table, a base table of a view and the master table of an updatable materialized view.
- By default, the delete statement removes all the existing records from a table.
- A WHERE clause is used to limit the deleted records.
- More than one condition can be specified in the WHERE clause using LOGICAL operators.

Syntax

```
DELETE FROM tablename
[WHERE CONDITION];
```

? Records will be deleted only if none of the records are referred from the child table.

eg:-

to avoid above violation.

3 methods

-> RESTRICT - rejects the delete or update operation of the parent table

-> Cascade - Deletes the row from the parent table, and automatically deletes the corresponding rows in the child table.

-> SET NULL - Deletes the row from the parent table, and sets the foreign key column in the child table to NULL.

These options can be specified while creating foreign keys in the child table by using ON DELETE clause in the CREATE statement.

```
CREATE TABLE TableName
( Col No1 DataType [ [ CONSTRAINT ConstraintName ]
   ConstraintType ],
  ForeignKeyName datatype [ CONSTRAINT constraintName ]
   [ FOREIGN KEY ] REFERENCES parentTable Name
   ( ColumnName FromParent )
   [ ON DELETE reference-option ],
```

TRUNCATE vs DELETE

• Removes all records	• Deletes all rows of a specific row from the table
• cannot be reverted	• can be reverted.
• will not work if the truncated table is referenced.	• Rows that are referred in the child table cannot be removed.

Merge statement.

- The MERGE statement is used to update or insert records which is selected from one or more sources.
- Conditions are specified to determine whether to update or insert records into the target be table or view.
- It is a convinient way to combine multiple operations and is used to avoid multiple INSERT & UPDATES.
- It cannot update the same row of the target table multiple times in the same MERGE statement.

Syntax:-

```
MERGE INTO table_name table_alias
    USING (table | view | sub_query) alias
    ON (join condition)
WHEN MATCHED THEN
    UPDATE SET
        col1 = col_val1 , col2 = col2_val
WHEN NOT MATCHED THEN
    INSERT (column_list)
      VALUES(column values);
```

Employees_OLD

EMPLOYEENO	name	salary
10001	Boot	8716
10002	Evania	8163
10003	Bruce	7397

Employees_UPDATES

to		
10002	Evania	4000
10004	Rufus	13605
10005	Lise	3700

Employees_OLD as

```
MERGE INTO Employees_OLD EO
USING Employees_UPDATES EU
ON ( EO.EMPLOYEENO = EU.EMPLOYEENO)
WHEN MATCHED THEN
        UPDATE SET EO.SALARY = EU.SALARY
WHEN NOT MATCHED THEN
        INSERT (EMPLOYEENO, NAME, SALARY)
VALUES (EU.EMPLOYEENO, EU.NAME, EU.SALARY);
```

Database transaction

- A transaction is a logical unit of work.
- Oracle ensures data consistency based on transactions.
- Transaction does consistent data change.
- Transaction begins when DML statement is executed.
- Transaction should end with either Commit or Rollback.
- DDL commands are by default auto commit.

COMMIT - ends the current transaction by making all pending data changes permanent.

SAVEPOINT savepoint_name - Marks a save point within the current transaction.

ROLLBACK - ROLLBACK ends the current transaction by discarding all pending data changes.

ROLLBACK TO savepoint_name.

- Rollback to savepoint rolls back the current transaction to the speified savepoint, thereby discarding any changes or savepoints created after the savepoint to which you are rolling back.

Locking

- Prevent destructive interaction b/w concurrent transactions when accessing shared resources.
- Locking is performed automatically and requires no user action.
- Uses the lowest level of restrictiveness.
- locks are held for a duration of a transaction.
- Types
  - explicit locking
  - implicit "

Objective

- Select statement
- where clause
- Types of operators
- order by clause.
- Distinct.

SELECT statement.

- The SELECT statement of SQL is used to retrieve data from one or more tables or views.
- The result returned is stored in a temporary result table, called the result-set.

it can be used for retrieving

- all rows from a table.
- specific rows from a table.
- specific values from a table.

- SQL statements are not case sensitive
- data stored is case sensitive.

Syntax:-

```
SELECT [distinct][col1name, col2name...]|*
  FROM table_name[alias]{, table_name[alias]}...
  [WHERE conditions]
  [GROUP BY group [HAVING group-conditions]]
  [ORDER BY sort_columns [ASC|DESC]];
```

To Retrieve all the records.

Query:

```
SELECT * FROM Customer;
```

CID	CName	PhoneN . . .
-	-	-
-	-	-
-	-	-
-	-	-

To retrieve specific columns.

```
SELECT cname, address FROM Customer;
```

o/p

cname	address
-	-
-	-
-	-

e.g:- Performing Arithmetic operations. alias name

```
SELECT pid, MinAmt +(MinAmt * 0.1) PolicyAmt FROM
       policy;
```

PID	POLICYAMOUNT
-	-
-	-

Concatenation Operator (||)

```
SELECT cname ||'lives in'|| address as CustomerAddress
FROM customers;
```

Eliminating Duplicate rows while retrieving.

- `SELECT DISTINCT address FROM customers;`

WHERE clause

- Restrict the rows returned by using the WHERE clause. The 'WHERE' clause follows the 'FROM' clause.

```
SELECT *|{[DISTINCT] column|expression [alias],...}
    FROM table [WHERE condition(s)];
```

- Value in the condition is compared using Relational operators.
- logical operators are used to specify more than one condition in WHERE clause.

```
SELECT cname, emailid, address FROM customer WHERE
  address = 'chennai';
```

Comparision operators

=	equal to
>	greater than
<	less than
>=	greater than equal to
<=	less than or equal to
!= , <>	Not equal to
Between ... and	Range of values.
In(Set)	Match any of the list values.
Like	Match a character pattern.
Isnull	is a null value.

* A NULL is not the same as zero or a space, zero is a number and a space is a character.

SELECT * FROM Policy WHERE MinAmt <= 1500.;

Like operator

To select rows that match a character pattern by using the LIKE condition. Search conditions can contain either literal characters or numbers:

'%' denotes zero or many characters.

'_' denotes one character.

'A%' // - first digit '%Z' -> last digit.

SELECT cid, name, emailid FROM customer WHERE emailid LIKE '%yahoo%';

operators	Meaning	Example	Description
Between X and Y	Range of values between X and Y (X, Y is inclusive)	.. WHERE Salary Between 5000 and 8000	
IN (set)	Match any of the list values	WHERE Salary IN (5000, 6000, 7000)	
IS NULL	Is a null value	WHERE Salary IS NULL	

- 'NOT' can be used with BETWEEN, IN, LIKE and IS NULL operators.

Logical operators:

AND
OR
NOT

SELECT cid, ~~cn~~name FROM customers WHERE cid=1
AND address = 'chennai';

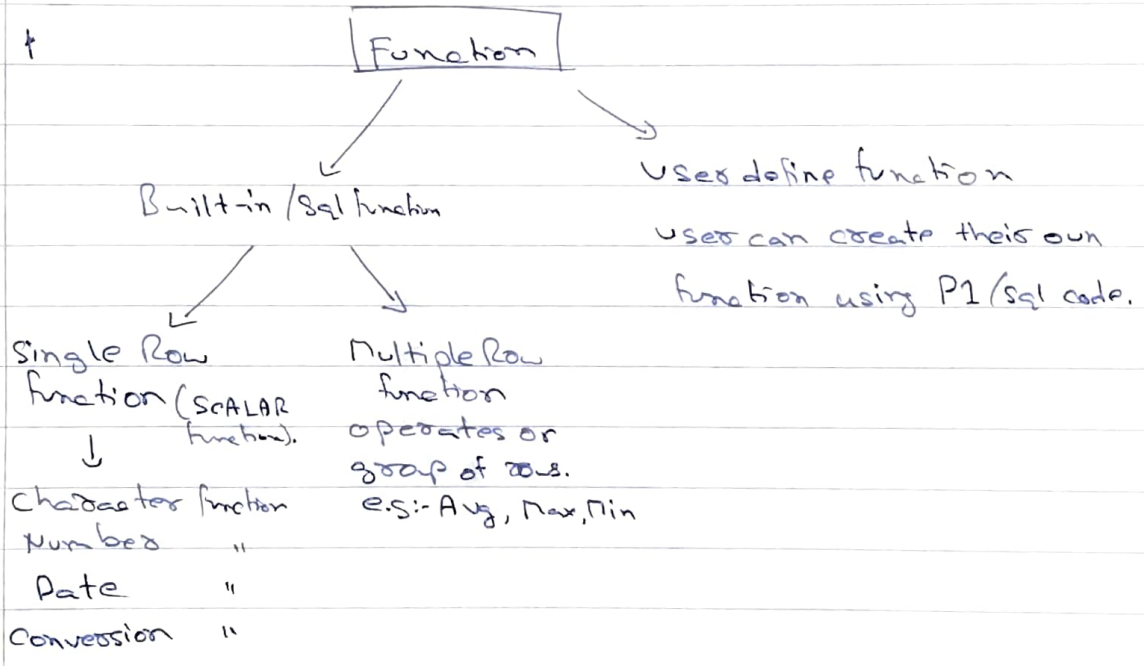
* to retrieve names of the customers in ascending order.

-> SELECT cname FROM customers ORDER BY cname;
desc;
For descending order

Functions

Sing row / SCALAR function multi-Row / Group / aggregate ...

- Character functions
- numbers "
- Date "
- Conversion "
- nesting "
- general "
- group "
- Group by clause
- having clause.



A function is a block of code that sometimes take arguments and always returns a value functions are a very powerful feature of SQL and can be used to do the following:

- perform calculations on data.
- modify individual data items.
- manipulate items for groups of rows.
- convert column data types.

Single row functions

- Manipulates data items.
- Accepts arguments and returns one value.
- Acts on each row returned.
- Returns one result per row.
- May modify the data type.
- Can be nested.
- Accepts arguments which can be a column or an expression.
- function_name [(arg1, arg2, ...)].

Character function

Function	
lower (col | exp)	- converts alpha character values to lower case.
upper (col | exp)	- Converts alpha character values to upper case.
Initcap (col | exp)	- converts alpha character values to upper case for the first letter of each word.
Concat (col1 | exp1, col2 | exp2)	- Concatenates two col or expression values.
Substr (col | exp, m, [n])	- Returns specified characters from the character value starting at character position m, n characters long.
length (col | exp)	- Returns the number of characters in the expression.

Number functions

abs(col/exp) - returns the absolute value of n.

ceil(col/exp) - returns the smallest integer values that is greater than or equal to a number.

Floor(col/exp) - returns the largest integer value that is equal to or less than a number.

Mod(m,n) - returns the remainder of m divided by n.

Round(n,[m]) - returns a number rounded to a certain number of decimal places. m refers to decimal places.

Trunc(n[,m]) - returns a number truncated to a certain number of decimal places.

Date function

To get current date

SELECT SYSDATE from Dual;

arithmetic operation

- add days to date.
- Subtract days from date.
- Subtract two dates.

months_between - number of months b/w two dates

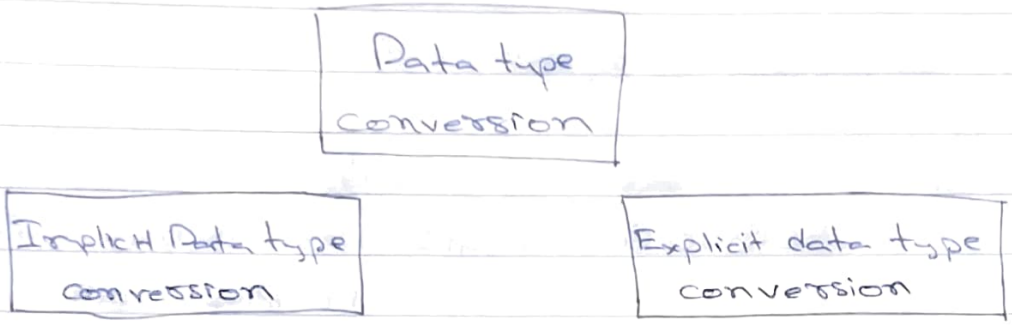
Add_months - Add calendar months to date

Next_date - Next date to the day specified.

Last_day - last day of the month.

Round - Round date.

Trunc - Truncate date.

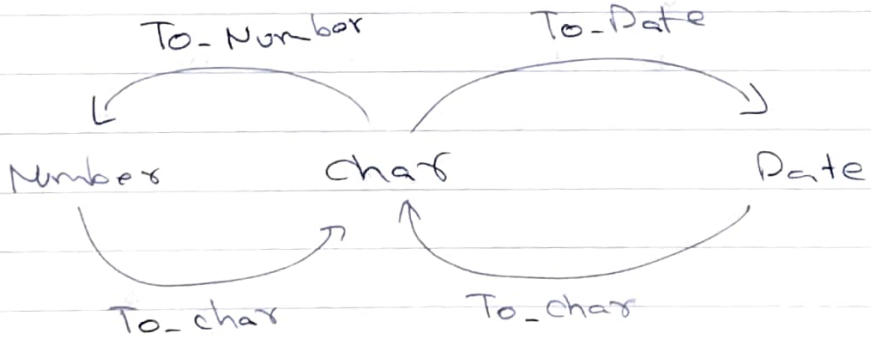


Conversion function

To_char (number | date, [fmt]) - converts a number to or date to string.

To_date (char, [fmt]) - Converts a string to a date.

To_number (char, [fmt]) - converts a string to a number.



elements of the date format model

Year - Year, spelled out.
YYYY - 4-digit year.
YY - 2-digit year.
MM - month number (1-12).
Month - abbreviated name of month
Mon - Full name of month.

WW - week of year.
W - week of month
D - Day of week
DAY - Name of day
DD - Day of month
DDD - Day of year

HH - Hours of day (0-12)
HH24 - Hours of day (0-23)
MI - minute
SS - second.

Conversion Function :-

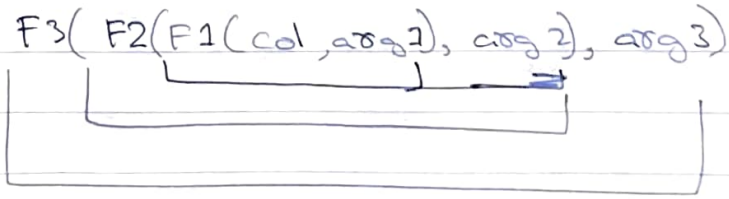
TO_CHAR (number | date, [fmt])
SELECT TO_CHAR(SYSDATE, 'dd month yyyy') FROM DUAL
18 APRIL 2019

TO_DATE(char, [fmt])
SELECT TO_DATE ('may-12-2019', 'mon-dd-yyyy') FROM DUAL
12-MAY-19

TO_NUMBER (char, [fmt])
SELECT TO_NUMBER('1234.67', 999.99) FROM DUAL
234.67.

NESTING FUNCTIONS :-

- Single-row functions can be nested to any level.
- Nested functions are evaluated from deepest level to the least deep level.



Select cid, cname, round (months_between (sysdate, dob)/12)
age from customer where cid=1;

Conditional Expression

gives the use of IF-THEN-ELSE Logic within a SQL Statement.

Two methods
- CASE expression.
- DECODE expression.

Facilitates conditional inquires by doing the work of an IF-THEN-ELSE statement.

```
SELECT pid,pname,
 CASE   pname
WHEN 'MoneyBackPlan' THEN 'Money Savings'
WHEN 'Personal Protect' THEN 'Life Insurance'.
ELSE 'Normal Policy'
END "Category of Policies" FROM policy;
```

DECODE is a function in Oracle and is used to provide if-then-else type of logic to SQL. It is not available in MySQL or SQL Server.

```
SELECT pid, pname, DECODE(pname, 'MoneyBackPlan', 'MoneySave',
        'Personal Protect', 'life insurance', 'Normal
        Policy') "Category of Policies" FROM Policy;
```

General Function

These functions work with any datatype and pertain to using null value.

NVL(exp1, exp2) - Converts a null value to an actual value.

NVL2(exp1, exp2, exp3) - If exp1 is not null then return exp2.
If exp2 is null then return exp3.

NULLIF - compares two expressions and returns null if they are equal or the first expression if they are not equal.

COALESCE - Returns the first not null expression in the expression list

NVL Function

- converts a null to an actual value.
- Data types that can be used as date, character, and number.
- Data types must match.

```
SELECT cid, pid, duedate, paiddate, amount, NVL(penalty, 0)
        penalty FROM policyenrollment;
```

NVL2 function.

Lets you determine a value returned by a query based on whether a specified expression is null or not null, if exp1 is not null then NVL2 returns exp2 if exp1 is null the NVL returns exp3.

```
SELECT cid, pid, duedate, NVL2(Paiddate, 'Paid', 'Not Paid')
  status FROM policyenrollment;
```

Coalesce function

takes to a more compatible argument and returns the first argument that is not NULL the result is null only if all the arguments are null.

```
SELECT cid, cname, COALESCE(emailid, TO_CHAR(phoneno),
        address) contact FROM customer;
```

Aggregate or Group function :-

Group functions operate on the entire set of rows on the table and give one result for the entire set.
group functions can be nested to a depth of two.

Various group functions
- MAX
- AVG
- MIN
- COUNT
- SUM
- .

> Group functions ignore the null values.

```
SELECT MAX(penalty) as maximum, Min(penalty) as
        minimum FROM policyenrollment;
```

Group by Clause

- The Group by clause divides the rows in a table into groups.
- Divide row of a table into smaller groups by using the GROUP BY clause and with this GROUP BY a column function results in a single value for each group.
- If a group function is ~~inclined~~ included in a SELECT clause along with other non grouped columns, then these non grouped column's should appear in the

GROUP BY clause.

- SQL Compiler generates an error if the non grouped columns are not included in the GROUP BY clause.
- The GROUP BY column does not have to be in the SELECT list.
- A column alias cannot be used in the GROUP BY clause.

SELECT CID, MAX(penalty) as maximum, MIN(penalty) as minimum FROM policyenrollment GROUP BY CID;

HAVING CLAUSE

- The WHERE clause cannot be used to restrict groups. The group functions cannot be in the WHERE clause.
- Conditions for groups must be specified in the HAVING clause. HAVING clause of select groups satisfy certain conditions.
- HAVING can ~~be~~ specify any column function on any column in a table being queried. This column needs not be in the SELECT list.

SELECT CID, sum(amount) as maximum FROM policyenrollment GROUP BY CID HAVING sum(amount) > 2000;