Contents

Introduction

Components of SQL statements

Select	Data Retrieval
Insert Update Delete Merge	DML
Create Alter Drop Rename Truncate	DDL
Commit Rollback Save Point	Transaction Control
Grant Revoke	DCL

Select Statement: Select statement is used to retrieve the information from database using select statement you can do the following

- Projection
- Selection
- Joining

Projection: It is used to choose columns in a table that you want returned by the query.

Selection: It is used to choose rows in a table that you want returned by your query.

Joining: You can choose the join capability in SQL to bring together data that is stored in different tables by creating a link between them.

Note: Selection and projection often considered as horizantal and vertical partinioning.

Syntax: Select *| { [distinct] column| Expression [alias],….} From table

- Select identifies what columns
- From identifies which table
- Always write keywords in uppercase

Selecting all columns:

Select * from employees;

We can also display all columns of alll rows by lisitng column names after the seelct keyword

Ex: Select department_id, department_name, manager_id, location_id from deparrtments;

Selecting specific columns of all rows:

Ex: Select department_id, location_id from departments;

In this case we can specify the column names in the order in which you want them to appear on the output.

Note: We can also select from psuedo columns a psuedo column behaves like a table column but it is not actually stored in the table you cannot insert or delete values in the psuedo column.Some available psuedo columns are CURRVAL,NEXTVAL, LEVEL, ROWID, ROWNUM.

Arithematic Expressions: Create expressions with number and date data by using arithematic operators

Operation	Description
+	Add
-	Substraction
*	Multiply
/	Divide

Ex: Select last_name, salary, salary+300 from employees;

Operator Precedence: *, / , +,-

If the operators within a expression are of same priority then evaluation is done from left to right.

Ex: Select last_name, salary, 12*salary+100 from employees;

Note: use parentheses to reinforce the standard order of precedence and to improve clarity.

Ex: (12*salary)+100 with no change in above result

Note: parentheses are used to override the rules of precedence also

Ex: Select last_name, salary, 12*(salary+100) from employees;

Defining a null value:

A null value is a value that is unavailable, unassigned, unknown or in applicable..
Null is not same as zero or blank space.

Ex: Select last_name,job_id,salary,commission_pct from employees;

Arithematic expressions containing a null value evaluate to null

Ex: Select last_name, 12*salary*commission_pct from employees;

Defining column aliases:

A column alias
- Renames column heading
- It is useful for calculations
- Immediately followed by the column name,there can also be optional keyword AS keyword betweeen the column name and alias.
- Enclose alias name in double quotations if it contains a special characters such as # or $ or is case sensitive.
- Column aliases can be used in both select clause and the order by clause you cannot use column aliases int eh where clause.

Ex: Select last_name AS name, commossion_pct AS comm from employees;

Ex: Select last_name "name", salary*12 "Annual Salary" from employees;

Using concatenation operator:

- concatenates character string or columns to other columns
- It is represented by two vertical bars ||
- Creates resultant column that is a character expression.

Ex: Select last_name || job_id AS "Employees" from employees;

Literal characters strings:

A literal is a character, a number or a date included in the select list.

Date and character literal values must be enclosed within single quotation marks.

Each character string is output once for each row returned

Ex: Select last_name || 'is a' || job_id AS "Employee Details" from employees;

Duplicate Rows:

The default display of queries is all rows including duplicate rows.

Ex:Select distinct department_id from employees;

Distinct keyword should be used immediately after the seelct keyword.

Distinct can also be used with multiple columns and it affects all the columns selected

Ex: Select distinct department_id, job_id from employees;

Dispalying table structure:

Ex: Desc[ribe] <table name>;

Restricting and sorting data

Limiting the rows selected:

We restrict the rows returned by using the WHERE clause.

```
Select *|[distinct] column/expression [alias],---------}

        From table

    [where condition [s]];
```

where restricts the query to rows that meets the condition

Condition is composed of column names ,expressions constants ,and a comparison operator.

Where consists of three elements.

1. column name
2. comparison condition
3. column name, constant or list of values

Ex: select employee-id , last-name, jod-id, department-id from employees where
Department-id = 90;

Character strings and date values are enclosed in single quotation marks.

Character values are case sensitive ,and date values are format sensitive

The default date format is dd-mon-rr.

Ex: select last-name ,job-id ,department-id from employee where last-name='whalen'

Comparison conditions:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equl to
<	Less than
<=	Less than or equl to
<>,!=,^=	Not equl to

Ex: where hire-date ='01-jan-95'.

Using Comparison Conditions:

Ex:Select last-name ,salary from employees where salary <=3000.

Other Comparison Operators:

Operator	Meaning
Between ----- AND-----	Between two values (inclusive)
In (set)	Match any of a list a values

Like	Match a character pattern
Is null	Is a null value

Between: select last_name , salary from employees where salary between 2500 and 3500; while specifying between we must specify the lower limit first.

Between and AND are actually translated by the oracle server to a pair of and conditions (a>=lower limit) AND (a<= higher limit).

Using between and AND has no performance benefits ,and it is used logical simplicity.

IN: It is used to test the values in a list. In condition is also known as member ship condition

Ex: Select employee-id,last-name ,salary,manager-id from employees where manager-id In(100,101,201)

Ex: select employee-id ,manager-id ,department-id from employees where last-name IN ('hartstein','vargas').

Note: If characters or dates are used in a list they must be enclosed in a single quotation marks .

In is actually translated by a oracle server to a set of OR conditions a=value1 or a= value2 or a=value3.

Using IN has no performance benefits ,and is used for logical simplicity.

LIKE: it is used for performing wildcard searches of valid search string values.

Search conditions can contain either literal characters or numbers .

% denotes zero or many characters .
- denotes one character or any single character.

Ex: select first – name from employees where first –name like ‘s%’;

The above query returns all the rows whose first letter is uppercase s and it ignores lower case s.

Ex: select last-name , hire-date from employees where hire-date like’%95’;

Displays all the employees who joined between January 95 and December 95.

you can combine pattern – matching characters

Ex: Select last-name from employees where last-name like ‘-0%’

This query displays all the names of the employees whose last name have an 0 as the second character .

You can use the ESCAPE identifier to search for the actual % and – symbols.

Ex: Select employee-id ,last-name,job-id from employees where job-id like ‘%sa\-%’ escape’\’;

The escape option identifies the backslash(\) as the escape character .in the pattern ,the escape character precedes the underscope(-).this causes the oracle server to interpret the underscore literally.

Null condition:

Select last-name ,manager-id from employees where manager-id is null.

The null conditions include ISNULL condition and ISNOTNULL condition.

Logical Conditions:

Operator	Meaning
AND	Returns true if both component conditions are true
OR	Returns true if either component condition is true
NOT	Returns true if either condition is false

AND: It requires both conditions to be true.

Ex:Select employee-id ,last-name ,job-id salary from employees where salary>=10000 and Job-id like '%man%'.

AND Truth Table:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR: it requires either condition to be true.

Ex: Select employee-id , last-name ,job-id ,salary from employees where salary>=10000 OR job-id like '%man%'.

OR Truth Table:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE

FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT Operator:

Ex: Select last-name,job-id from employees where job – id NOTin ('it-prog','st-clerk','sa-rep');

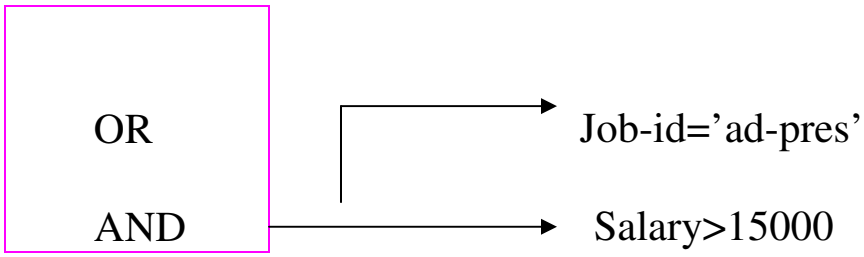
Truth Table:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Rules Of Precedence:

Order evaluated	Operator
1	Arithmetic Operators
2	Concatenation Operators
3	Comparison Operators
4	Is [not] null, like, [not] in
5	[Not] between
6	Not logical condition
7	And logical condition
8	Or logical condition

Ex: select last-name , job-id ,salary from employees where job-id ='sa-rep'



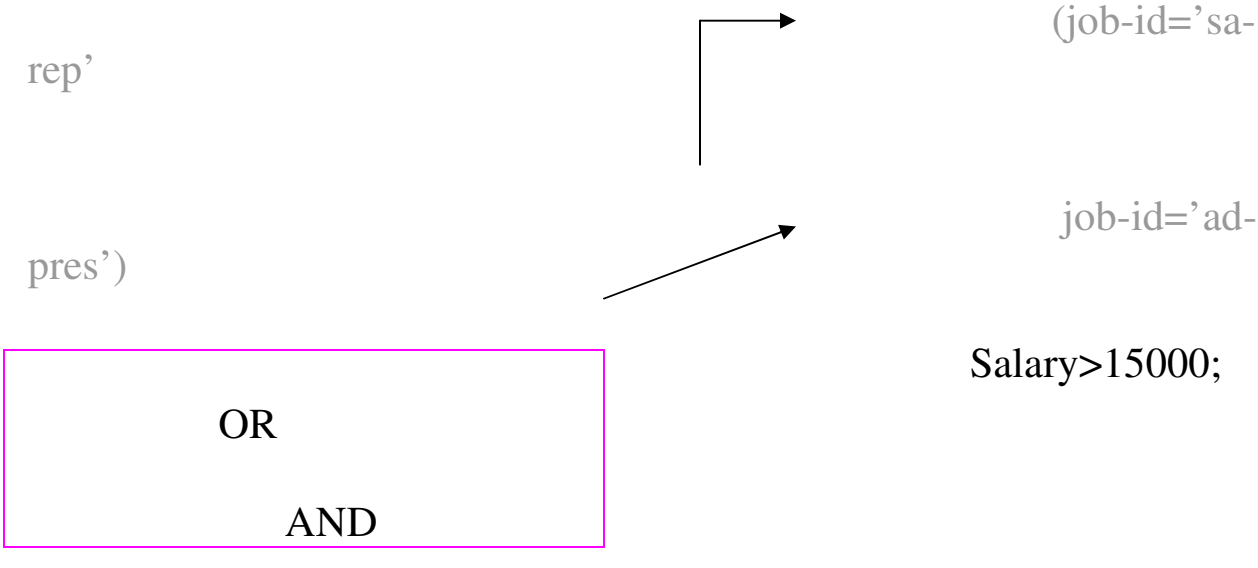
The first condition is that the job-id is ad-pres and the salary is greater than 15000.

The second condition is that the job id is a sa-rep.

Therefore select statement reads as follows “select the row if an employee is a president and carns more than 15000,or if the employee is a sales representative.

We use parentheses to force priority.

Ex: select last-name , job-id ,salary from employees where



Select the row it an employee is a president or a sales representative and it the employee carns more than 15,000.

Order By Clause:

We sort rows by using order by clause .

ASC: ascending order , default

DSC: descending order.

The order by clause comes last in the select statement.

Syntax: select expr from table

[where condition(s)]

[order by { column,expr}[asc/dsc]];

order by clause is executed last in the query execution .it is placed last unless the for update clause is used.

Default sorting is ascending

Numeric values are displayed with lowest values first ex:1-999

Date values are displayed with earliest value first ex:01-jan-92 before 01-jan-95.

Character values are displayed in alphabetic order ex:A-Z.

Null values are displayed last for ascending sequences and first for descending sequences.

Ex:select last_name , job_id ,department_id,hire –date from employees order by hired_date desc.

We can also sort by a column number in the select list.

Ex: select last-name , salary from employees order by 2 desc.

We can also sort by using column aliases .

Ex: select employee-id ,last – name ,salary*12 annsal from employees order by annsal.

The order of execution for a select statement is as follows :

1. from clause
2. where clause
3. select clause
4. orderby clause

we can also sort by using multiple columns.

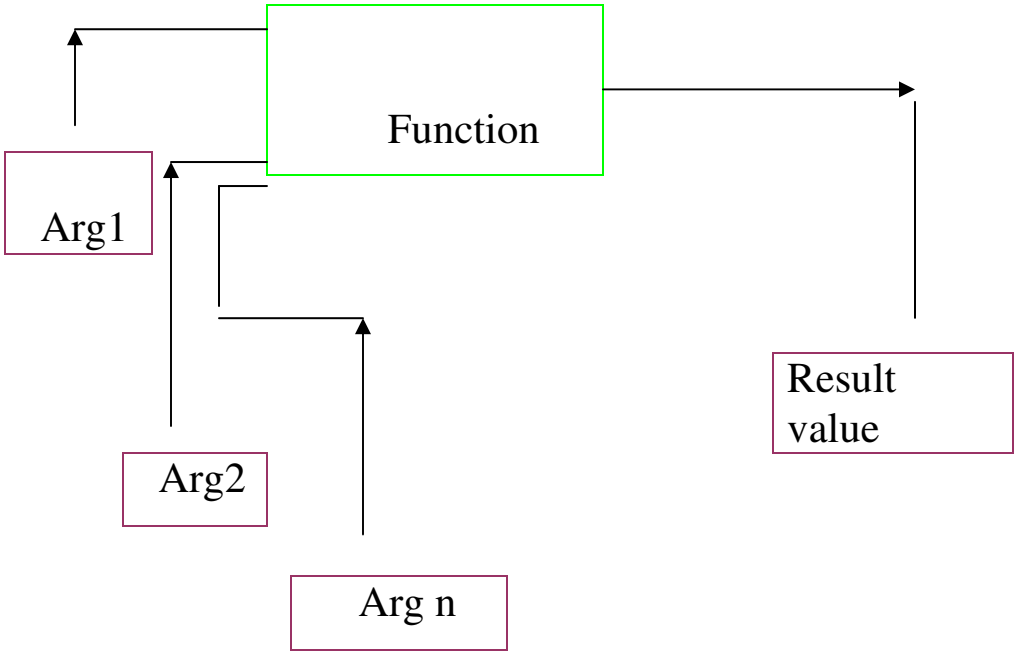
Ex: select last-name,department-id ,salary from employees order by department-id salary Desc.

You can also order by columns that are not included in the select clause.

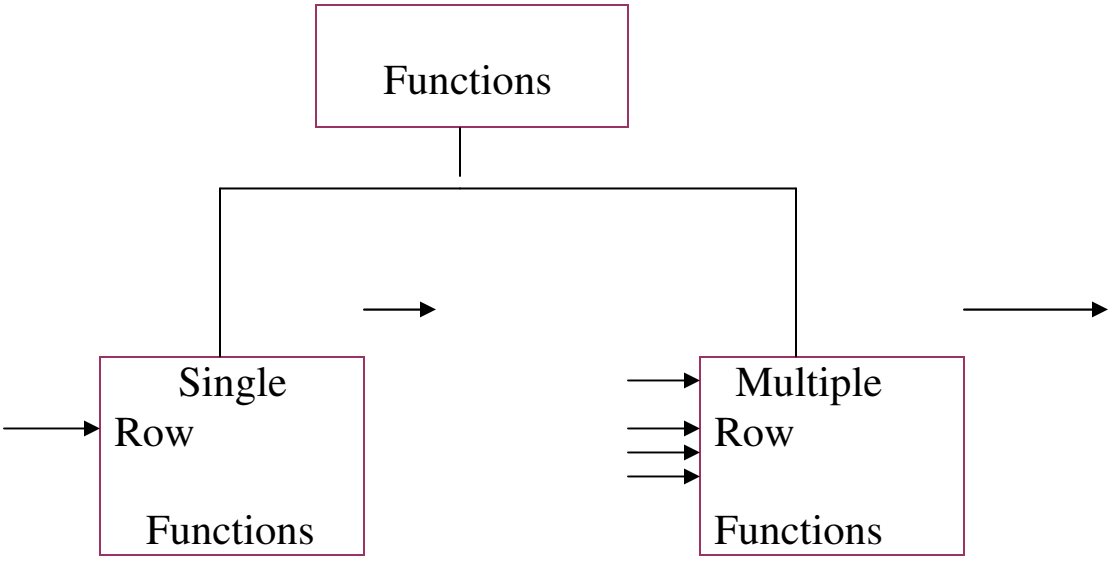
Ex: select last-name ,salary from employees order by department-id salary desc.
In the above query department –id column is sorted in ascending order and the salary column in descending order.

Single Row Functions

Sql Functions:



Two Types Of Sql Functions:



Single row functions operate on single rows only and return one result per row.

There are different types of single row functions

1.Character

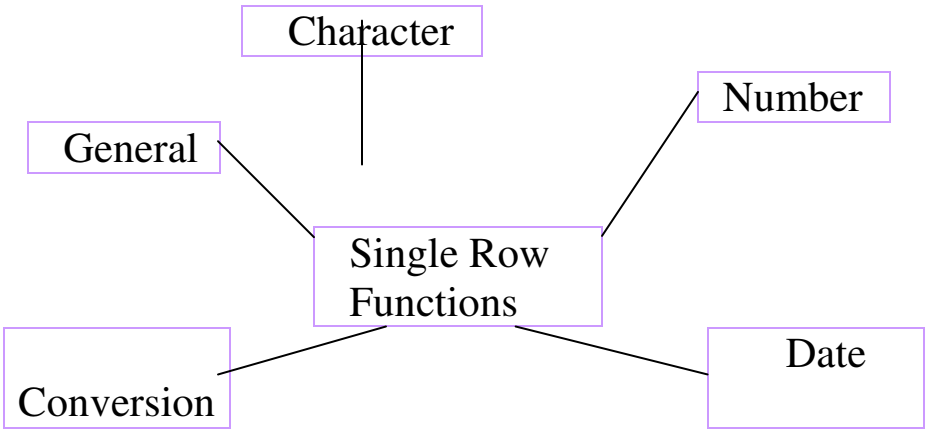
2.Number

3.Date

4.Conversion

Multiple row functions: functions can manipulate groups of rows to give one result per group of rows .these functions are known as Group functions.

Single Row Functions



Single row functions accepts one or more Arguments and return one value for each row returned by the query.an argument can be

1.User- Supplied Constant

2.Variable Name

3.Column Name

4.Expression

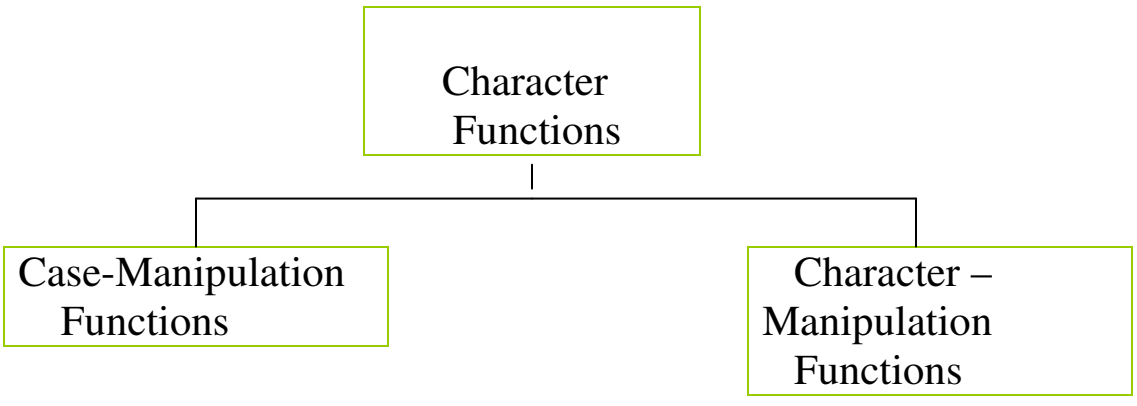
Syntax: function – name -----is name of function.
Arg1 ,arg2 ----------is any argument.

Note: Character functions can return both characters and number values.
Note: Number functions accept numeric input and return number values.
Note: Date functions operate on date data type (all date functions return a value of date data type except the months between function , which returns a number)
Note: Conversion functions convert a value from one data type to another .

General Functions :

- NVL
- NVL2
- NULLIF
- COALSECE
- CASE
- DECODE

Character Functions:



LOWER	CONCAT
UPPER	SUBSTR
INITCAP	LENGTH
	INSTR
	LPAD/RPAD
TRIM	REPLACE

LOWER: lower(column/expression) converts alpha character values to lower case.

UPPER: upper(column/expression) converts upper character values to upper case.

INITCAP: initcap (column/expression) converts alpha character values to upper case for the first letter of each word ,all other letters in lower case.

CONCAT: concat(col1/exp1, col2/exp2) concatenates the first character value to the
Second character value .

SUBSTR: (column/ expression,m[,n]) returns specified characters from character value
String at character position m,n character long.

LENGTH: (column/expression) returns no.of characters in the expression.

INSTR: instr(column/expression,g',[,m],[n]) returns the numeric position of a named string .Optionally you can provide a position m to start searching ,and the occurance n of the string . m and n default to 1,

meaning start the search at the beginning of the search and report the first occurance.

LPAD: lpad(column/expr,'string') pads the character value right – justified to a total width of n character positions.

RPAD: rpad(column/expr , n , 'string') pads the character value left – justified to a total width of n character positions.

TRIM: trim(leading/trailing/both, trim – character FROM trim – source) enables you to trim heading or trailing characters (or both) from a character string if trim –character or trim – source is a character literal ,you must enclose it in single quotes.

REPLACE: replace (text,search-string,replacement-string) searches a text expression for a character string and , if found ,replaces it with a specified replacement string.

Case Manipulation Functions:

Function	Result
Lower ('SQL Course')	Sql course
Upper ('SQL Course')	Sql course
INITCAP('SQL Course')	Sql course

Ex: select 'the job id for ' || upper(last-name)|| 'is' || lower(job-id) as employee details from employees;

Ex: select employee-id , last-name, department-id from employees where lower(last_ name='higgins');

Character Manipulation Functions:

CONCAT('Hello' , 'World')	HelloWorld
SUBSTR (' Helloworld' , 1 , 5)	Hello
LENGTH(' Helloworld')	10
INSTR('Helloworld' ,'w')	6
LPAD(salary , 10 , '*')	*****24000
RPAD(salary , 10, '*')	24000*****
TRIM('H' FROM 'Helloworld')	Elloworld.

Ex: To display the data for those employees whose last names end with n..

Select employee-id ,concat (firste , last-name) name,length(last-name), INSTR(last-name ,' a ') " contains 'a' ? " from employees where SUBSTR (last-name,-1,1)='n';

Number Functions:

ROUND: rounds value to specified decimal round(45.926,2)-----45.93.

TRUNC: truncates value to specified decimal trunc(45.926,2)-----45.92

MOD: returns remainder of division mod(1600,300)----------100.

ROUND: (column/expression,n)
Rounds the column, expression or value to n decimal places or if n is omitted ,
No decimal places (if n is negative , numbers to the left of the decimal point are rounded)

TRUNC: (column/expr)
Truncates the column , expression or value to n decimal places or if n is omitted ,then n defaults to zero.

MOD(min): returns the remainder of m divided by n .

Ex: select round(45.923,2), round(45.923,0) round(45.923,-1) from dual;

Note: dual is a dummy table you can use to view results from functions and calculations.

MOD: select last-name,salary, mod(salary,5000) from employees where job-id='sa=rep'.

DATE: you can change the default date display setting for a user session by executing a command

Alter session set NLS-DATE-FORMAT='date format model';

Oracle database stores dates in an internal numeric format century, year ,month,day,hours ,minutes,seconds.

Default date format is DD-MM-RR.

Valid oracle dates are between January 1, 4712 b.c and December 31,9999 a.d.

07-jun-94 is internally stored as june 7th ,1994 5:10:43 p.m

Century	Year	Month	Day	Hour	Minute	Second
19	94	06	07	5	10	43

Oracle server is 2000 complaint. When a record with a date column is inserted into a table the century information is picked up from sysdate function .

Arithmetic With Dates:

Add or subtract a number to or from a date for a resultant date value.

Subtract two dates to find the the number of days between those dates

Add hours to a date by dividing the no. of hours by 24.

Ex: select last-name ,(sysdate – hire _ date)/7 as weeks from employees where department-id=90;

Note: SYSDATE is a sql function that returns the current date and time .

Date Functions:

Functions	Description
Months_Between	No.of months between two dates
Add_Months	Add calender months to date
Next_Day	Next day of the date specified
Round	Round date
Trunc	Truncate date

Note: all date functions return a value of DATE datatype except months – between which returns a numeric value.

Months_between: (date1,date2)

Find no.of months between date1 and date2.

Add months : (date,n)

Add n number of calendar months to date . the value of n must be a integer and can be negative.

Next day: (date,'char')

Find the date of the next specified day of the week (‘char’) following date the value of char may be a number representing a day or a calendar string

Ex: select next-day (’19-apr-07’,’monday’) from dual;

Last date: (date)

Find the date of the last day of month that contains date.

Round: (date[,’fmt’]);

Returns date rounded to the unit specified by the format model fmt. If the format model fmt is omitted is rounded to nearest day .

Trunc: (date[,’fmt]);

Returns date with the position of the day truncated to the unit specified by the
Format model fmt is omitted ,date is truncated to the nearest day.

Ex:Months_between (’01-sep-95’,’11-jan-94’)-------19.6774194.

Ex:Add_months(’11-jan-94’,6)-----------------------11-jul-94.

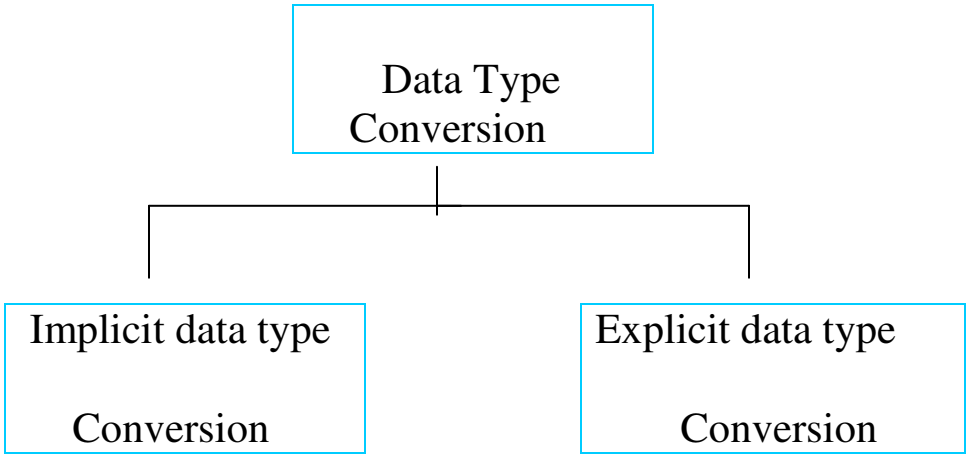
Ex:Next_day01-feb-95’,’friday’)----------------------’08-sep-95’

Ex:Last_day (’01-feb-95’)-----------------------------’28-feb-95’

Ex:compare the hire dates for all employees who started in 1997. dis play the employee number ,hire date and start month using the round and trunc function .

Select employee_ id ,hire_date ,round(hire_date,’month’),trunc (hire_date ,’month’) from employees where hire_date like ‘%97’;

Conversion Functions:



Implicit Data Type Conversion:

For assignments , the oracle server can automatically convert the following.

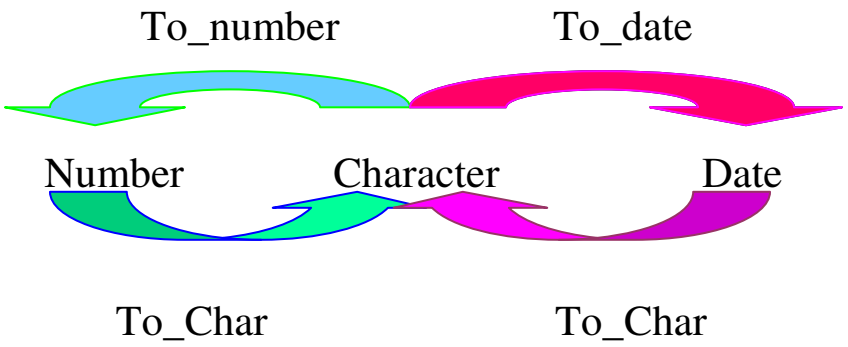
From	To
Varchar2 to Char	Number
Varchar2 or Char	Date
Number	Varchar2
Date	Varchar2

For express evaluation oracle server can automatically convert the following

Varchar2 or Char	Number
Varchar2 or Char	Date

Note: char to number conversions succeeds only if the character string represents a valid Number.

Explicit Data Type Conversion:



To_Char(number/date,[fmt],[nls params]):

Converts a number or date value to a varchar2 character string with format Model fmt.

Number Conversion: the nls params parameter specifies the following characters which are returned by number format elements .

1. decimal character
2. group separater
3. local currency symbol
4. international currency symbol

if nls params or any other parameter is omitted ,this function uses the default parameter values for this session.

To_Number : (char ,[fmt],[nlsparams])

Converts a character string digits to a number in the format specified by the optional format model fmt.

To_Date: (char,[fmt],[nls params])

Converts a character string representing a date to a date value according to the fmt specified. If fmt is omitted , the format is DD-MON-YY.

Using To-Char Function With Dates:

To-Char(date,'format-date')
Format Model:

Must be enclosed in a single quotation marks and is case sensitive . Can include any valid date format element

Has an fmt element to remove padded blank or suppress leading zero's. It separated from the date value by a column .

Ex: select employee-id , To-Char(hire-date,'MM/YY') month – hired from employees Where last-name = 'higgins';

Elements Of Date Format Model:

YYYY	Full years in numbers
YEAR	Year spelled out
MM	Digit value for month
MONTH	Full name of the month
MON	Three letter abbreviation of month
DY	Three letter abbreviation of

	day of week
DAY	Full name of day of the week
DD	Numeric day of the month

The elements format the time portion of the date.

HH24:MI:SS → AM 15:42:32 PM

Add character strings by enclosing them in double quotation marks.

DD "of" month → 12 of October

Number soffixes spell out numbers .

Ddspth → fourteenth.

Date Format Elements – Time Formats:

AM or PM , A.M or P.M

HH or HH12 or HH24

MI(minutes 0-59), SS(seconds 0-59) SSSS(seconds past midnight(0-86399))

Specifying Soffixes to Influence no.Display:

TH → ordinal number(ex: DDTH for 4^{th})

SP → spelled number (ex: DDSP for Four)

SPTH or THSP → spelled out ordinal numbers(ex: DDSPTH for fourth)

Ex: select last-name ,to-char (hire-date, 'FMDD ,MONTH,YYYY') As HIREDATE
From employees:

Other formats:

/., → punctuation is reproduced in the result.

"for the " → quoted string is reproduced in the result.

Ex: Select last-name to-char (hire-date , 'fm ddspth " of" month yyyy fm HH:MI:SS AM') HIREDATE from employees;

Using the to_char function with numbers:

To_char (number,'format_model')

There are some of the format element you can use with the to – char function to display

A number value as a character .

9---------- represents a number .

0----------forces 0 to be displayed

$----------places a floating dollar sign

L---------- floating local currency symbol

. ---------- prints decimal point

, ---------- prints a thousand indicator

MI----------minus signs to right

PR----------parenthesize negative numbers.

EEEE----------scientific notation (format must specify from E's)

V ----------multiply by n times(n=no.of 9's active v)

Ex:9999V99.

B----------display zero values as blank , not 0.

Ex:select to-char (salary,'$99,999.00') Salary from employees where last-name ='ernst';

Using the to_number and to_date functions:

Convert a character string to a number format using the to-number function

To_ number (char[,'format-model'])

Convert a character string to a date format using to-date function .

To_ date (char[,'format – model'])

These functions have an tx modifier .this modifier specifies the exact matching for

The character argument and date format model of a to-date function.

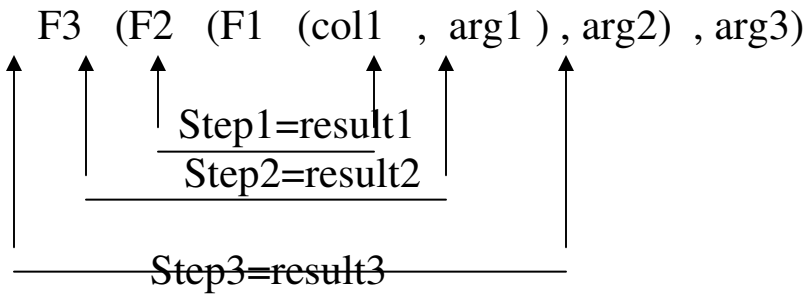
Ex: select last_ name ,hire_date from employees where hire_date =to_date('may24,1999',tx month DD,YYYY');

tx is used an exact match is required and the spaces after the word 'may' are not recognized.

Nesting Functions:

Single row functions can be nested to any level.

Nested functions are evaluated from deepest level to the least deep level.



Single row functions can be nested to any depth.

Ex: select last-name ,NVL (to-char (manager_id), 'nomanager') from employees where Manager_id is NULL.

NVL is used to replace null with a string.

Ex: display the date of the next Friday that is six months from the hire date.

Select to_ char (next_ day (add_months (hire_ date, 6) , ' Friday') ,'fmday,monthDDth,YYYY')"next 6 month review" from employees order by hire date .

General Functions:

These functions work with any data type and pertain to using nulls.

NVL(expr1 , expr2)

NVL2(expr1 , expr2 , expr3)

NULLIF(expr1 , expr2)

COALESCE(expr1,expr2 ----------exprn).

NVL: converts a null value to an actual value

NVL2: if expr1 is not null, NVL2 returns expr2. if expr1 is null NVL2 returns expression3 the argument expr1 can have any data type .

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

COALESCE: returns first not – null expression in the expression list

NVL Functions:

Converts a null to an actual value.

Data types that can be used are date , character ,and number

Data types must match

Ex1: NVL(commission – pct,0)

Ex2: NVL(hire-date,'01-jan-97')

Ex3: NVL(job-id,'NO job yet')

Syntax: NVL(expr1 , expr2)

Expr1: is the source value or expression that may contain null.

Expr2: is the target value for converting the null.

You can use NVL value to convert any data type ,but the return value is always the same as the data type of expr1.

NVL Conversion For Various Data Types:

Data Type	Conversion example
Number	NVL(number-column,9)
Date	NVL(date-column,'01-jan-91')
Char or Varchar2	NVL(char-column,'unavailable')

Ex: select last_name ,salary ,NVL(commission_pct,0),(salary*12) + (salary*12*NVL(Commission_pct,0)) AN_SAL from employees;

Using The NVL2 Function:

Ex: Select last-name, salary, commission-pct, NVL2 (commission-pct ,'sal+com','sal') In come from employees where department-id in (50,80).

NVL2 mines first expression .if the first expression is not null , then the NVL2 returns second expression. If the first expression is null then NVL2 returns third expression.

Syntax: NVL2(expr1,expr2,exor3)

Expr1: is the source value or expression that may contain null.

Expr2: is the value returned if expr1 is not null.

Expr3: is the value returned if expr2 is null

The argument expr1 have any data type.

The arguments expr2 and expr3 can have any data type except long .

If data types of expr2 and expr3 are diff.the oracle server converts expr3 to the Data type of expr2 before comparing them unless expr2 is a null constant in that Case data type conversion is not necessary.

Data type of return value is always data type of expr2 ,unless expr2 is char in Which case the return values data type is varchar2.

Using The NULLIF Function:

Ex: select first-name ,length(first – name) " expr1",last-name,length(last-name) "expr2", NULLIF(length (first-name),length(last-name)) result from employees;

NULLIF function compares two expressions .if they are equal the function

Returns null.if they are not equal the function returns the first expression .

Syntax: NULLIF (expr1 ,expr2)

Expr1: is the source value compared to expr2.

Expr2: is the source value compared with expr1.

Using the Coalesce Function:

Ex: Select last-name COALESCE(commission-pct, salary,10) comm. From Employees order by commission-pct;

The advantage of coalesce function over the nvl function is that the coalesce function Can take multiple alternative values.

If the first expression is not null it returns that expression ,other wise it does coalesce Of the remaining expressions.

Syntax: COALESCE(expr1,expr2----------exprn)

Expr1:returns this expression if it is not null

Expr2:returns this expression if the first expression is null and this expression is not null.

Exprn:returns this expression if the preceding expressions are null.

Conditional Expression:

Provides the use of IF-THEN-ELSE logic with a sql statement.

Use two methods.

1. case expression
2. decode function

Case expression:

```
Case expr when comparison---expr1 then returns expr1
            [when comparison----expr2 then returns expr2
             when comparison---expr3 then returns expr3

                    -


                    -
    when comparison-exprn then returns exprn

else
      else-expr

 end
```

Case expressions let you use if-then –else logic in sql statements without having to Invoke procedures.

In simple case oracle searches for the first when then pair for which the expression

Is equl to the comparison expr and returns return-expr if name of the when then paris

Meets the condition and an else clause exists then oracle returns else –expr other wise Oracle returns null.

You cannot specify null value for all the return –expr and else-expr.

Ex: Select last-name ,job-id ,salary, case job-id when 'it-prog' then 1.10*salary When 'st-clerk' then 1.20*salary Else salary END "revised salary" From employees;

Using The decode function:

Facilitates conditional inquires by doing the work of a case or if-then – else statement.

Decode(col/expression,search1,result1 [,search2,result2,------,]
[,default])

The decode function decodes an expression in a way similar to the if-then-else logic

The decode function decodes expressions after comparing it to each value.if

The expression is the same as search ,result is returned.

If the default value is omitted ,a null value is returned where a search value does not match any of the result values.

Ex: Select last-name,job-id , salary,decode(job-id,'it-prog',1.10*salary, 'st-clerk',1.20*salary salary) revised_salary.
From employees;

Displaying Data From Multiple Tables

- A Cartesian Product formed when
- A join condition is omitted.
- A join condition is invalid.
- All rows in the first table are joined to all rows in the second table.
- To avoid a Cartesian product always include a valid join condition in a where clause.

Ex: SELECT LAST_NAME, DEPARTMENT_NAME dept_name FROM EMPLOYEES, DEPARTMENTS;

Oracle Proprietary	**SQL:1999**
Joins(8i and Prior)	Complaint joins
Equi join	Cross joins
Non-Equi join	Natural joins
Outer join	Using Clause
Self join	Full or two sided Outer joins
	Arbitrary join conditions for Outer joins

Equi join: Equi joins are also called Simple joins or Inner joins.

Ex: For example, if you want to display the name and department number of all the employees who are in the same department as goyal: you can start by making the following decision free .

Columns to Display	Originating Table	Condition
Last_name	Employees	Last_name='goyal'
Department_name	Departments	Employees.department_id=departments.dep

Retrieving records with Equi join:

SELECT EMPLOYEES.EMPLOYEE_ID, EMPLOYEE, LAST_NAME, EMPLOYEES.DEPARTMENT_ID, DEPARTMENTS.DEPARTMENT_ID, DEPARTMENTS .LOCATION_ID FROM EMPLOYEES, DEPARTMENTS WHERE EMPLOYEES. DEPARTMENT_ID= DEPARTMENTS .DEPARTMENT_ID AND LAST_NAME = 'MATOS'

-We can use AND operator for additional search.
-Use table prefines to quality column names that are in multiple tables.
-Improve performance by using table prefixes.
-Distinguish columns that have identical names but reside in different tables by using column aliases.

- To join n tables together , you need a minimum of n-1 join conditions.

Ex: SELECT E.LAST_NAME,D.DEPARTMENT_NAME, L.CITY FROM EMPLOYEES E,DEPARTMENTS D,LOCATIONS L WHEREE E.DEPARTMENT_ID=D.DEPARTMENT_ID AND D.LOCATION_ID=L.LOCATION_ID.

Non-Equijoins:

- A non-equi joins is a join condition containing something other than an equality operator.

Ex: SELECT E.LAST_NAME,E_SALARY,J.GRADE_LEVEL FROM EMPLOYEES E,JOB_GRADES J WHERE E.SALARY BETWEEN J.LOWEST_SAL AND J.HIGHEST_SAL.

NOTE: other conditions , such as <= and >= can be used , but BETWEEN is the simplest.

- Between is always translated by the oracle server to a pair of AND conditions .

- IN is translated by the oracle server to a set of OR conditions.

Outer joins:

- We use an outer join to see rows that do not meet the join condition .
- The outer join operator is the PLUS sign(+).

```
SELECT TABLE1.COLUMN,TABLE2.COLUMN FROM
TABLE1,TABLE2 WHERE
TABLE1.COLUMN(+)=TABLE2.COLUMN

SELECT TABLE1.COLUMN,TABLE2.COLUMN FROM
TABLE1,TABLE2 WHERE
TABLE1.COLUMN=TABLE2.COLUMN(+).
```

The missing rows can be returned if an outer join operator is used in the join condition .

- The operator is a plus sign enclosed in parentheses(+), and it is placed on the side of the "join" that is deficient in intermation.

- The condition involving outer join cannot use the IN operator or be linked to another condition by the OR operator.

- The outer join operator can appear on only one side of the expression the side that has information missing.

Ex: SELECT E.LAST_NAME, E.DEPARTMENT_ID, D.DEPARTMENT_NAME FROM EMPLOYEES E.DEPARTMENTS D WHERE E.DEPARTMENT_ID(+)=D.DEPARTMENT_ID.

Self join: Some times you need to join a table to itself.

Ex: SELECT WORKER .LAST_NAME||'WORKS FOR'||

MANAGER.LAST_NAME FROM EMPLOYEES WORKER, EMPLOYEES MANAGER WHERE WORKER. MANAGER_ID =MANAGER.EMPLYEE_ID.

Joining Tables using SQL:1999 SYNTAX

Use a join to query data from more than one table.

SELECT TABLE1.COLUMN,TABLE2.COLUMN FROM TABLE1

[Cross join table2]|
[Natural join table2]|
[JOIN table2 USING (column_name)]|
[JOIN table2 ON (table1.column_name=table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2

ON(table1.column_name=table2.column_name)];

Cross join :

- The cross join clause produces the cross product of two tables.

Ex: SELECT LAST_NAME,DEPARTMENT_NAME FROM EMPLOYEES CROSS JOIN DEPARTMENTS;

Natural join:

- The natural join clause is based on all columns in the two tables that have the same name.
- It selects rows from the two tables that have equal value from all matched columns.
- If the columns having the same names have different datatypes , an error is returned.

Ex: Select department_id,department_name,location_id,city from departments natural join location.

- If several columns have the same names but the data types do not match, then
  Natural join clause can be modified with the USING clause to specify the
  Column that should be used for an equi join

- use the using clause to match only one column when more than one column matches
- do not use a table name or alias in the referenced columns.
- The natural join and using clauses are mutually exclusive.

Ex: SELECT L.CITY,D.DEPARTMENT_NAME FROM LOCATIONS L JOIN DEPARTMENTS D USING(LOCATION_ID)WHERE LOCATION_ID=1400;

Creating joins with ON clause:

- the join condition for the natural join is basically an equi join of all columns with the same name.
- To specify arbitrary conditions or specify columns to join , the ON clause is used
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

Ex: SELECT E.EMPLOYEE_ID, E.LAST_NAME, E.DEPARTMENT_ID,D.DEPARTMENT_ID,D.LOCATION_ID FROM EMPLOYEES E JOIN DEPARTMENTS D ON (E.DEPARTMENT_ID=D.DEPARTMENT_ID);

- The ON clause can be used as follows to join columns that have different names

Ex: SELECT E.LAST_NAME EMP,M.LAST_NAME MQR FROM EMPLOYEES E JOIN EMPLOYEES M ON (E.MANAGER_ID=M.EMPLOYEE_ID).

Creating three way joins with the ON clause:

Ex: SELECT EMPLOYEE_ID,CITY,DEPARTMENT_NAME FROM EMPLOYEES E JOIN DEPARTMENTS D ON D.DEPARTMENT_ID=E.DEPARTMENT_ID JOIN LOCATIONS L ON D.LOCATION_ID=L.LOCATION_ID;

- the first join condition can reference columns in employees and departments but cannot reference columns in locations
- the second join condition can reference columns from all three tables .

INNER VERSUS OUTER JOINS:

- The join between two tables that returns the results of the inner join as well as unmatched rows left(or right) tables is a left (or right) outer join
- A join between the two tables that returns the results of an inner join as well as the results of the left and right join is a full outer join

LEFT OUTER JOIN:

Ex: SELECT E.LAST_NAME, E.DEPARTMENT_ID, D.DEPARTMENT_NAME FROM EMPLOYEES E LEFT OUTER JOIN DEPARTMENTS D ON (E.DEPARTMENT_ID=D.DEPARTMENT_ID).

Joins Comparing SQL:1999 to Oracle Syantax:

Oracle	Sql:1999
Equi join	Natural/Inner join
Outer join	Left Outer join
Self join	Join ON
Non-equi join	Join Using
Cartesian Product	Cross join

Right outer join :

Ex: SELECT E.LAST_NAME, E.DEPARTMENT_ID, D.DEPARTMENT_NAME FROM EMPLOYEE E RIGHT OUTER JOIN DEPARTMENTS D ON (E.DEPARTMENT_ID=D.DEPARTMENT_ID)

Full outer join*:*

Ex: SELECT E.LAST_NAME, E.DEPARTMENT_ID, D.DEPARTMENT_NAME FROM EMPLOYEES E FULL OUTER JOIN DEPARTMENTS D ON (E.DEPARTMENT_ID=D.DEPARTMENT_ID);

Aggregating Data Using Group Functions

- Group functions operate on sets of rows to give one result for group.
- Unlike single-row functions , group functions operate on sets of rows to give one result per group.

Types of Group Functions

Avg , Count , Max , Min , Stddev , sum , Variance.

- AVG([Distinct/All]n)
-
- COUNT({*|[Distinct/All]Expr})
-
- MAX([Distinct/All] Expr)
-
- MIN([Distinct/All] Expr)
-
- STDDEV([Distinct/All]x)
-
- SUM([Distinct/All]n)
-
- VARIANCE([Distinct/All]x).

Group Functions Syntax:

Select [column,] group function(column,---------- from table

[where condition]

[Group by column]

[Order by column];

Guidelines For Using Group Functions:

- distinct makes the function consider only non duplicate values ; ALL makes it consider every value including duplicates. The default is ALL and therefore does not need to be specified.
- The data type for the functions with an expr argument may be char,varchar2,number or date.
- All group functions ignores null values . to substitute a value for null values ,use the NVL,NVL2 or COALESCE functions.
- The oracle server implicity sorts the result set in ascending order when using a group by clause .to overrde this default ordering ,DESC can be used in an order by clause.

Using The AVG and SUM Functions:

- You can use AVG and SUM for numeric data.

Ex: Select avg(salary) , max(salary) , min(salary), sum(salary) from employees Where jod_id like '% rep%';

- We can use AVG,SUM,MAX,MIN functions against columns that can store numeric data.

Ex: Select min(hire_date),max(hire_date) from employees;

- We can use MIN and MAX for any data type.

Ex: Select min(last_name),max(last_name) from employees;

Note: AVG,SUM,VARIANCE and STDDEV functions can be used only with Numeric data types.

Using The COUNT Function:

- Count(*) returns the number of rows in a Table .

Ex: Select count(*) from employees where department_id=50;

- The count function has three formats.

  Count(*)

  Count(expr)

  Count(Distinct expr)

- count(*) returns the number of rows in a table that satisfy the criteria of the select statement including duplicate rows and rows countaining null values in any of the columns .
- in contrast ,count(expr) returns the number of non null values in the column identified by expr.
- Count(distinct expr) returns the number of unique , non values in the column identified by expr.

Ex: Select count(commission_pd from employees where department_id=80;

Ex: Select count(distinct department_id) from employees;

Group Functions and Null values:

- Group functions ignores null values in the column . we use NVL functions to include null values.

Ex: Select avg(commission_pd) from employees;

Ex: Select avg(NVL(commission_pd,0)) from employees;

Creating Groups of Data:

Syntax: select column, group_function(column) from table [where condition]
[group by enpression]
[order by column];

- we use group by clause to divide the rows in a table into groups.

Guidelines: If you include a group function in a select statement ,you cannot select individual results as well ,unless the individual column appears in the *GROUP BY* clause.

- using where clause you can include rows before dividing them into groups.
- you must include the columns in the group by clause.
- You cannot use a column alias in the group by clause.
- By default it ,rows are sorted by ascending order of the columns included in the group by list.you can override this by using order by clause.

Ex: Select department_id,avg(salary) from employees group by department_id;

- The above query is used to compute average salary for each department.

Note: Group results are stored implicitly ,on the grouping column .you can use order by to specify a different sort order ,remembering to use only group functions or the grouping column.

Note: The group by column does not have to be in the select list .
Ex: Select avg(salary) from employees group by department _id;
Grouping By More Than One Column:

Ex: Select department_id dept_id,job_id,sum(salary) from employees group by Department_id,job_id;

Note: You cannot use where clause to restrict groups .

Note: You cannot use group functions in the where clause

Ex: Select department_id,avg(salary) from employees where avg(salary)>8000 Group by department_id;

Having Clause:

- We use the having clause to restrict groups

    1. rows are grouped
    2. the group function is applicol
    3. groups matching the having clause are displayed.

Select column , group function from table

[where condition]

[group by group_by expression]

[having group_by condition]

[order by column];

- Oracle server performs the following steps when we use the having clause

    1. rows are grouped .
    2. the group function is applied to the group.
    3. the group that match the criteria in the having clause are displayed.

Note: the having clause can precede the group by clause but it is recommended that you place the group by clause first because it is more logical.

Ex: Select department_id, avg(salary)from employees group by department_id Having max(salary)>10000;

Ex: Select job_id , sum(salary) payroll from employees where job_id not like '%rep' group by job_id having sum(salary)>13000 order by sum(salary);

Nesting Group Functions:

- Group functions can be nested to a depth of two.

Ex: Select max(avg(salary)) from employees group by department_id;

Sub Queries

Syntax: Select select_list from table where expr operator (select select_list from Table);

- The sub query executes once before the main query.
- The result of the sub query is used by the main query (outer query)
- You can place the sub query in a number of sql statements including .

    1.the where clause

    2.the having clause

    3.the from clause

- operators include a comparision condition such as >,=,or IN.

Note: comparision condition fall into two classes: single row operators (>,=,>=,<,<>,<=)and multiple row operator (IN,ANY, ALL).

Ex: Select last_name from employees where salary>(select salary from employees Where last_name ='abel');

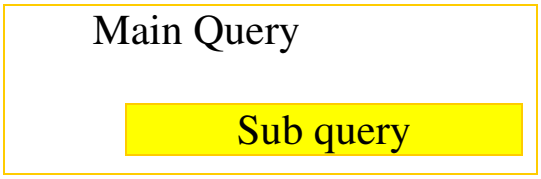
Guidelines For Using Sub Queries:

- A sub query must be enclosed in parentheses.
- Place a sub query on the right side of the comparision condition for readability.
- Two classes of comparisios conditions are used in sub queries :single row operators and multiple row operators

Note: The oracle server imposes no limit on the number of sub queries the limit is releated to the buffer size that the query uses.

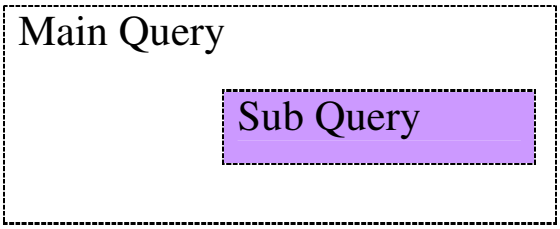
Types Of Subqueries

Single – row subquery



Returns St-clerk.

- **Multiple – row sub query**:



Returns St-clerk,St-man

Note: There are also multiple – column subqueries.

Single Row SubQueries:

- Return only one row
- Use single – row comparision operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>	Not Equal to

Ex: Select last_name , job-id from employees where job-id=(select job-id from Employees where employee-id = 141);

Ex: Select last-name ,job-id ,salary from employees where job-id = (select job-id From employees where employee-id =141) and salary >(select salary from Employees where employee id=143);

Note: The outer and inner queries can get data from different tables.

- We can use group functions in a sub query

Ex: Select last-name ,job-id ,salary from employees where salary =(select min (salary) from employees)

Having Clause With SubQueries:

- The oracle server executes sub queries first.

- The oracle server returns results into the having clause of the main query.

Ex: Select department-id ,min(salary) from employees group by department-id Having min(salary)>(selected min(salary) from employees where department-id=50);

Errors with SubQueries:

- More than one row returned by a single row sub query.
- No rows returned by the sub query.

Multiple Row SubQueries:

- Returns more than one row.
- Use multiple row comparision operator.

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each values returned by the subquery
ALL	Compare value to every value retterned by the subquery

Ex: Select last-name ,salary,department-id from employees where salary in (select min (salary) from employees group by department-id);

Using ANY Operator:

- The ANY operater (and its synonym some) compares a value to each value returned by a subquery.

Note: < ANY means lessthan the maximum.
Note: >ANY means more than the minimum.
Note: =ANY is equivalent to IN.

Ex: Select employee-id,last-name,job-id, salary from employees where salary< ANY(select salary from employees where job-id='it-prog') and job-id<>'it-prog';

Note: When using some or ANY you often use the DISTINCT keyword to prevent rows from being selected several times.

Using The ALL Operater:

- The all operator compares a value to every value returned by a subquery.

Note: >ALL means more than the maximum.
Note: <ALL means less than the minimum
Note: NOT operater can be used with IN , ANY,ALL operaters.

Ex: Select employee-id, last-name , job-id, salary from employees where salary <ALL(select salary from employees where job-id='it-prog') and job-id<>'it-prog'

Null Values In a Sub query:

Ex:Select emp.last-name from employees emp where emp.employee-id not in (select mgr.manager –id from employees mgr);

Output: No rows selected

Returning nulls in the resulting set of a subquery:

The sql statement mentioned above attempts to display all the employees who do not have any subordinates .logically ,the sql statement should have returned 12 rows.however,the sql statement does not return any rows .one of the values returned by the inner is a null value and hence entire query returns no rows .the reason is that all conditions that compare a null value result in a null . so whatever null values are likely to be part of the results set of a subquery , do not use the NOTIN operater . the NOTIN operater is equivalent to <> ALL.

Notice that the null value as part of the results set of a subquery is not a problem if you e IN operater .the IN operater is equivalent to=ANY, for example ,to display the employees who have subordinates ,use the following sql statement.

Ex: Select emp.last-name from employees emp where emp.employee-id IN (select mgr.manager-id from employee mgr).

Alternatively a where clause can be included in the subquery to display all employees who do not have any subordinates .

Ex: Select last-name from employees where employee-id NOTIN(select manager-id from employees where manager –id is not null);

Manipulating Data

Data manipulating language(DML) is a core part of SQL when u want to add, update or delete data in the database you execute a DML statement.

Insert Statement:

Insert into Table[(column[,column…..])] values(value[,value….]);

Only one row is inserted with this syntax.

Note: *Insert* a new row containing values for each column.

Note: List values in the default order of the columns in the table.

Note: Optionally,list the columns in the *insert* clause.

Ex: Insert into departments(department_id,department_name, manager_id, location_id) values(70,'public relations',100,1700);

Note:Enclose character and data values within single quotation marks.

Methods of inserting null values:

Implict: Omit the column from the column list

Explicit: Specify the *null* keyword in the values list,specify the empty string*('')* in the values list for character strings and dates.

Inserting Special Values:

-The *SYSDATE* function records the current date and time.

Ex: Insert into employees(employee_id,first_name,last_name, email, phone_number,hire_date, job_id, salary, commission_pct, manager_id, department_id) values (113,'louis','pop','lpop', '515.124.4567',SYSDATE, 'ac_account,6900, NULL, 205, 100);

Inserting Specific Date Values:

-Add a new employee

Ex: Insert into employees values(114,'den,raphealy','drapheal', '515.127.4761', To_Date('feb3,1999','MON DD YYYYY'), 'ac-account',11000,Null ,100,30);

Copying rows from another table:

-Write your *insert* statement with a subquery

Ex: Insert into sales_reps(id, name, salary, commission_pct) select employee_id, last_name, salary, commission_pct from employees where job_id like '%rep%';

-Do not use the *values* clause.

-Match the number of columns in the insert clause to those in the subquery.

-To create copy of the rows of a table, use select * in the subquery.
Ex: Insert into copy_emp select * from employees;

Changing Data In the Table:

Update Syntax: Modify existing rows with the update statement.

Update table set column=value[,column=value,…..] [where condition];

-*Update* more than one row if required.
Note: In general uwe the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated.

Updating rows in a table:
-Specific row or rows are modified if you specify the *where* clause.

Ex: update employees set department_id=70 where employee_id=113;

-All rows in the table are modified if you omit the *where* clause.

Updating two columns with a subquery:

Ex: Update employees set job_id=(select job_id from employees where employee_id=205), salary=(select slary from employees where employee_id=205) where employee_id=114;

Updating rows based on another table: Use subqueries in *update* statements to update rows in a table based on vlaues from another table.
Ex: Update copy_emp set department_id=(selct department_id from employees where employee_id=100) where job_id=(select job_id from employees where employee_id=200);

Note: If you attempt to update a record with a value that is tied to an integrity constraint, an error is returned.

Removing a row from a table: After all the rows has been eliminated with the *DELETE* statement only the data structure of the table remains.A more efficient method of emptying a table is with the *truncate* statement.you can use the *truncate* statement to quickly remove all rows from the table or cluster.Removing rows with the *truncate* statement is faster than removing rows with the *delete* staement for the follwing reasons

- The *truncate* statement is DDL statement and generates no *rollback* information.
- Truncating a table does not fire the *delete* triggers of the table
- If the table is the parent of a referential integrity constriant, you cannot *truncate* the table.

Delete [from] table [where condition];

Note: The *delete* statement does not ask for confirmation.How ever the *delete* operation is not made permanent until the date transaction is commited. Therefore youu can undo the operation with the *ROLLBACK* statement.

Ex: Delete from departments where department_name = 'Finance';
All the rows in the table are deleted if you omit the *where* clause.

- Use subqueries to delete statements to remove rows based on values from another table.

Ex: Delete from employees where department_id =(select department_id from departments where department_name like '%public%');

- If you attempt to *delete* a record with a value that is tied to any integrity constraint an error is returned.

Note: If referential integrtiy constraints are in use, you may receive an oracle server error message when you attempt to *delete* a row.However if the referential integrity contains the *ON DELETE CASCADE* option,then the selected row and its children are deleted from their respective tables.

Using a subquery in an insert statement:

Insert into(select employee_id,last_name,email,hire_date, job_id, salary, department_id from employees where department_id=50) values(99999,'Taylor','Dtaylor', To_Date('o7-jun-99','dd-mon-rr'),'St_clerk',5000,50);

Using the with check option keyword on DML statements:

- A subquery is used to identify the table and columns of the DML statement.
- The *WITH CHECK OPTION* keyword prohibits you from changing rows that are not in the subquery.

Ex: Insert into(select employee_id,last_name,email,hire_date, job_id,salary from employees where department_id=50 with check option) values(9998,'smith','jsmith',To_Date('07-jun-99','dd-mon-rr'),'st_clerk',5000);

Error at line 1

Explicit default feature:

- With the explicit default feature you can use the *DEFAULT* keyword as a column value where the column default is desired.
- This allows the user to control where and when the *default* values should be applied to data.
- Explicit defaults can be used in *insert* and *update* statements to identify a *default* column value.If no *default* value exists, a *null* values is used.

Ex: *Default* with *insert*

Insert into departments (department_id, department_name,manager_id) values(300,' engineering',default);

Ex: *Default* with *update*

Set manager_id=default where department_id=10;

The Merge statement:

SQL has been extended to include the *MERGE* statement.Using this statement you can update or insert a row conditionally into a table thus avoiding multiple *update* statements.The decision whether to *update* or *insert* into the targer table is based on a condition in the *ON* clause.

Since the *MERGE* command combines the *INSERT* and *UPDATE* commands, you need both *INSERT* and *UPDATE* privileges on the target table and the *select* privilege on the source table.

The *merge* statement is deterministic you cannot *update* the same row of the target table multiple times in the same *merge* statement.

An alternative approach is to use the PL/SQL loops and multiple DML statements,The *MERGE* statement however is to use and more simply expressed as a single SQL statement.

Ex: merge into table_name table_alias using(table|view| subquery) alias on(join condition) when matched then update set
col1=col_val1;
col2=col_val2
when not matched then insert(column_list)
values(column_values);

Ex: Merge into copy_emp c using employees e
on(c.employee_id=e.employee_id) when matched then update set
c.first_name=e.first_name, c.last_name=e.last_name,……
c.department_id=e.department_id when not matched then insert
values(e.employee_id ,e.first_name,e.last_name,e.email,
e.phone_number.e.hire_date, e.job_id,e.salary, e.commission_pct,
e.manager_id,e.department_id);

Database Transactions: A transaction begins with the first statement is encounterd and ends when one of the following occurs.

- A *commit* or *rollback* statement is issued.
- A DDL statement, such as *create* is issued.
- A DCL statement is issued.
- The user wxists isql*Plus
- The system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and therefore implicitky ends a transaction.

Controlling Transcations:

Explicit transcation control statements: You can control the logic of transactions by using the *commit, savepoint* and *rollback* statements.

Statement	Description
Commit	Ends the current transaction by making al pending data changes permanent
SavePoint	Marks the savepoint with in the current transaction
Rollback	Rollback ends the current transaction by discarding all pending data changes
Rollback to Savepoint	Rollback to savepoint roll back the current transaction to the specified savepoint, thereby discard any changes and or savepoints created after the savepoint to which you are

	rolling back.If you omit to savepoint clause the rollback statement rollbacks entire transcation as savepoints are logical there is no way to list the savepoint you have created.

Note: Savepoints are not schema objects and cannot be referenced in the data dictionary.

Rollback changes to a marker:

- Create a marker in a current transaction by using the *savepoint* statement.

- *Rollback* to that marker by using the *rollback to savepoint* statement.

Update……..
Savepoint update_done;
Savepoint created
Insert…..
Rollback to update_done;
Rollback complete.

Note: if you create a second *savepoint* with the same name as an earlier savepoint,the earlier *savepoint* is deleted.

Implicit transaction processing:

Status	Circumstances
Automatic Commit	DDL or DCL statement is issued or isql*plus exited normally or without explicitly issuing *commit* or *rollback* commands

Automatic Rollback	Abnormal termination of isql*plus or system failure.

State of the data before commit or rollback:

- The previuous state of the data can be recovered.
- The current user can review the results of the DML operations by using the *SELECT* statement.
- Other users cannot view the results of the DML satatements by the current user.
- The affected rows are locked other users cannot change the data within the affected rows.

Note: Data manipulation operations primarily affest the database buffer therefore the previous state can be recovered.

Note: With the oracle server data changes can actually be written to the database files before transactions are committed,but they are still only temporary.

Note: By default oracle server has row level locking.It is possible to alter the default locking mechanism.

Note: If numbers of users are making changes simultaneously to the same table, then each user sees only his or her cahnges until other users commit their changes.

State of the data after commit:

- Data changes are made permanent in the database.
- The previous state of the data is permanently lost
- All users can view the result
- Locks on the affected rows are released those rows are available for other users to manipulate.

- All save points are erased.

Ex: Delete from employees where employee_id=9999;
Insert into departmenrts values(290,'corporate tax',null,1700);
Commit;

Note: Commit ensures that two related operations occur togethere or not at all in this case commit prevents empty departments from being created.

State of the data after after rollback:

Discard all pending changes using the rollback statement

- Datachanges are undone
- Previous state of the date is restored
- Locks on the affected rows are released.

Ex: Delete from copy_emp ;
Rollback;

Statement level rollback:

Part of the transacrtion can be discarded by an implicit rollback if a statement execution error is detected.

If a single DML statement fails during execution of a transaction its effect is undone by a statement level rollback

Oracle issues an implicit commit before and after any data definition language(DDL) statement.So even if your DDL statement does not execute successfully you cannot roll back the previous statement because the server issued the commit.

Read Consistency:

Database users access the database in two ways

- Read operations(select statement)
- Write operations(Insert, update,delete statements)

You need read consistency so that the following occur

- The database reader and writer are ensured a consistent view of the data
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent way.
- Changes made by one writer do not disrupt or conflict with cahnges another writer is making.

Creating And Managing Tables

Database Objects:

Object	Description
Table	Basic unit of storage composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence Index	Numeric value generator
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

Create Table:

Tables can be created at any time even while users are using the database.

You donot specify size of any table.The size is ultimately defined by the amount of space allocated to the database as awhole.It is important however to estimate how much space a table will use over time.

Table structure can be modified online.

Create table syntax:

Create table[schema.]table (column datatype [default expression][,…..]);

A schema is a collection of objects schema objects are logical structures that directly refer to the data in a database.Schema objects include

tables,views,synonyms,sequences, stored procedures, indexes, clusteres and database links.

If a table does not belong to the user the owners name must be prefixed to the table.

Ex: select * from user_b.employees;

Tables in the oracle database:

Tables are categorized in two ways user tables and data dictionary.

User tables are created by user

Collection of views and tables in the oracle database are known as data dictionary.This collection is created and maintained by orcle server and contains information about database.

There are four categories of data dictionary views;each category has a distinct prefix that reflects its indexed use.

Prefix	Description
User_	These views contains information about objects owned by the user.
All_	These views contain information about all of the tables accessible to the user
Dba_	These views are restricted views which can be acccessed only by people who has been assigned DBA role
V$	These views are dynamic performance views, database server performance memory and locking

Quering the data dictionary:

You can query the data dictionary tables to view various database objects owned by you. The data dictionary tables frequently used are those

- User_tables
- User_objcets
- User_catalog

Ex: Select * from user_catalog;

Creating a table by using subquery syntax:

Ex: Create table dept80 as select employee_id, last_name, salary*12 ANNSAL, hire_date from employees where department_id=80;

TO create a table with the same structure as an existing table but with out the data from the exixting table use a subquery with a where clause that always evaluate as false.

Ex: create table copy_table as (select * from employees where 1=2);

The Alter Table Statement:

Alter Table table
ADD(column datatype [Default expr][,column datatype]…..);

Alter Table table
Modify(column datatype[default expr][;column datatype]….);

Alter Table table
Drop(column);

Ex: Alter table dept80 add(job_id varchar2(9));;

You can add or modify columns

You cannot specify where the column is to appear.The new column becomes last column

If a table already contains rows when a column is added then the new column is initially null for all the rows.

Ex: Alter table dept80 modify(last_name varchar2(80));

A cahnge to the default value affects only subsequent insertions to the table.

GuideLines:

- You can increase the width or precision of a numeric column
- You can increase the width of numeric or character columns
- You can decrease the width of a column only if the column contains only null values or if the table has no rows
- You can cahnge the datatype only if column contains null values
- You can convert a cahr column to the varchar2 datatype or convert a varchar2 column to the char datatype only if the column contains null values or if you donot change the size
- A cahnge to default value of a column affects only subsequent insertions to the table.

Drop:

Ex: Alter table dept80 drop column job_id;

Guidelines:

- The column may or may not contain data

- Using the alter table statement only one column can be dropped at a time
- The table must have atleast one column remaining in it after it is altered
- Once a column is dropped it cannot be recovered.

Note: When a column is dropped fronm a table any other columns in that table that are marked with setunused option are dropped too.

Set unused option:

Ex: Alter Table table set unused(column);

OR

Ex: Alter Table table set unused COLUMN column;

Ex: Alter Table table dept80 drop unused columns;

The set unused option marks one or more columns as unused so that they can be dropped when the demand on system resources is lower.Specifying this clause does not actually remove the target columns from each row in the table.Therefore the response time is faster than if you executed the drop clause. Unused columns are treated as if they were dropped even though their column data remains in the tables rows.After the column has been marked as unused you have no acess to that column. A select * query will not retreive the data from unused columns. In addition the names and types of columns marked unused will not be described using a describe and you can add to the table a new column with same name as an uuused column. Set unused information is stored in user_unused_col_tabs dictionary view.

The drop unused columns removes from the table all columns currently marked as unused.

Dropping a table:

Ex: Drop table dept80;

Guidelines:

- All data is deleted from the table
- Any views and synonyms remain but are invalid
- Any pending transactions are committed
- Only the creator of the table or a user with the drop any table privilege can remove a table
- **Drop** table statement once executed is irreversible
- *Drop* table is committed automatically.

Changing the name of the object:

Ex: Rename dept to detail_dept

Note: You must be a owner of the object.

Truncating the table:

Ex: Truncate Table detail_dept;

- It is a DDL statement it is used to remove all rows from a table and to release the storage space used by that table
- You cannot rollback row removal;
- You must be the owner of the table with delete table system privileges to truncate a table
- The *delete* statement can also remove all rows from a table but it does not release stroage space
- *Truncate* command is faster.

Adding comments to a table:

You can add comments to a table or column by using comment statement .

Comments can be viewed through the data dictionary views

ALL_COL_COMMENTS
USER_COL_COMMENTS
ALL_TAB_COMMENTS
USER_TAB_COMMENTS

Syntax: comment on Table table|column table.column is ‘text’

Ex: comment on table employee is ‘’;

Including Constraints

- Constraints enforce rules at table level
- The constriant must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables

Data Integrity Constraints:

Not Null: Specifies that the column cannot contain a null value.

Unique: Specifies a column or combination of columns whose values must be unique for all rows in a table.

Primary Key: Uniquely identifies each row of the table

Foreign Key: Establishes and enforces a foreign key relationship between the column and a column of the referenced table.

Check: Specifies the condition that must be true.

Guidelines: All constraints are stored in the data dictionary, constraints are easy to reference if you give them a meaningful name. IF you don't name your constraint the oracle server generates a name with the format Sys_Cn where n is an integer so that constraint name is unique.

You can view the constraint defined for a specific table by looking at the USER_CONSTRAINTS data dictionary table.

Ex: Create table[schema] table (column datatype [Default expr] [column constraint], ... [table-conastraint][,…]);

Ex: Create table employees(employee_id number(6), first_name varchar2(20), ……. Job_id varchar2(10) not null, constraint emp_emp_id_pk primary key(employee_id));

Note: Schema same as the owners name.

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also temporarily disabled.

Constarints can be defined at one of the two levels

Column Level: References a single column and is defines within a specification for the owning column; can define any type of integrity constraint.

Table Level: Reference one or more columns and is defined seperately from the definitions of the columns in a table;can define any constraints except *NOT NULL*.

Not Null Constraint:

Ensures that columns contain no null values.

It is specified at column level not at table level

Ex: Create table employees(employee_id number(6), last_name varchar2(25) *not null* salary number(9,2), hire_date date constraint emp_hire_nn *not null*

Unique Constriant:

It ensures that every value in a column or set of columns be unique i.e. no two rows of a table can have duplicatevalues in a specified column or set of columns.

If unique constraint contains group of columns it is called composite unique key.

Unique constraints allows the input of nulls unless you also define *not null* constraint on the same column. In fact any number of rows can include nulls for the columns with out *not null* constraints because nulls are not considered equal to anything. A null in a column always satisfy a unique constraint.

Note: Because of the search mechanism for unique constraints on more than one column you cannot have identical values in the non-null columns of a partially null composite unique key constraint.

It can be created at column level or table level.

Oracle server enforces the unique constraint by implicitly creating a unique index on the unique key column or columns.

Ex:
```
Create table employees( employee_id    number(6)
                        Last_name      varcahr2(25) not null,
                        Email          varchar2(25),
    ……
    Constraint emp_email_uk unique(email));
```

Primary Key Constraint:

It can be defined at column level or table level. A composite primary key is created by using the table level definition.

A table can have only one primary key constraint but can have several unique constraints.

A unique index is created for a primary key column.

Ex: Create table departments(department_id number(4), Department_name varchar2(30), constraint dept_name_nn not null Location_id number(4), constraint dept_id_pk primary key(department_id));

Foreign Key Constraint:

The foreign key or referential integrity constraint designates a column or combination of columns as a foreign key and establishes a relationship between a primary key or a unique key in the same table or a different table.

A foreign key value must match an existing value in the parent table or be null

Foreign keys are based on data values and are purely logical, not physical pointers.

You cannot create a foreign key ithout existing primary key values.

Foreign key constraints can be defined at the column or table constraint level. A composite foreign key must be created by using
Table level definition.

Ex: create table employees(employee_id number(6), last_name varchar2(20) not null, email varcahr2(25), salary number(8,2), commission_pct number(2,2), hire_date date not null, …… department_id number(4), constraint emp_dept_fk foreign key(department_id) references departments(department_id), constraint emp_email_uk unique(email));

The foreign key can also be defined at the column level provided the constraint is based on a single column. The syntax differs in that the keywords foreign keys donot appear.

Ex: Create table employees (…… department_id number(4) constraint emp_deptid_fk references departments(department_id) ….. ……..);

Foreign Key: Defines the column in the child table at the table constraint level.

References: Identifies the table and column in the parent table.

On Delete Cascade: Delete the dependent rows in the child table when a row in the parent table is deleted.

On Delete Set Null: Converts dependent foreign key values to null.

The default behavior is called the restrict rule, which disallows the update or deletion of referenced data.

Without the *on delete cascade* or the *on delete set null* options the row in the parent table cannot be deleted if it is referenced in the parent table.

Check Constraint:

It defines a condititon that each row must satisfy the condition can use the same constructs as query conditions with the following exceptions

References to the *CURRVAL, NEXTVAL, LEVEL* and *ROWNUM* psuedocolumns

Calls to Sysdate, Uid, User and UserNv functions.

Queries that refers to other values in other rows.

A single column can have multiple check constraints which refer to the column in its definition. There is no limit to the number of check constraints which you can define on a column.

Check constraints can be defined at the column level or table level.

Ex: Create table employees (… salary number(8,2) constraint emp_salary_ck check(salary>0),…..).

Adding a Constraint: You can add a constraint for existing tables by using the *Alter* table statement with *Add* clause

Ex: Alter Table table add[Constraint constraint] type (column);

In the Syntax:

Table is the name of the table
Constraint is the name of the constraint
Type is th constraint type
Column si the name of the column affected by the constraint.

Guidelines: you can add, enable or disable a constraint but you cannot modify its structure

Note: you can define Not Null column only if the table is empty or if the column has a value for every row.

Add aforeign key constraint to the employees table indicating that the manager must already exist as a valid employee in the employees table.

Alter table employees add constraint emp_manager_fk foreign key(manager_id) references employees(employee_id);

To add a not null constraint use the alter table modify syntax.

Alter table employees modify(salary constraint emp_salary_nn not null);

Dropping a Constraint: To drop a constraint you ca nidentify the constraint name from the user constraints and *user_cons_columns* data dictionary views. Then use the Alter table statement with the drop clause. The cascade option of the drop caluse causes any dependent constraints also to be dropped.

Syntax: Alter Table table drop primary key / unique (column) | Constraint constraint [cascade]);

Ex: Remove the manager constraint from the employees table

Alter table employees drop constriant emp_manager_fk;

Ex: Remove the primary key constraints on the departments table and drop the associated foreign key constraint on the employees department_id column
Alter table departments drop primary key cascade;

Disabling Constarints: You can disable a constrint with out dropping it or recreating it by using alter table statement with the disable clause.

Syntax: Alter Table table disable Constriant constraint [cascade];

You can use the disable clause in both the create table statement and the alter table statement.

The cascade clause disables dependent integrity constriant.

Disabling a unique or primary key constraint removes the unique index

Ex: Alter Table employees disable constraint emp_emp_id_pk cascade.

Enabling Constraints: We can enable a constraint without dropping ot or re creating it by using the alter table statement with the enable clause.

Syntax: Alter Table table enable Constraint constraint;

If you enable a constraint that constraitnt applies to all the data in the table. All the data in the table must fit the constraint.

If you enable a unique key or primary key constraint a unique or primary key index is created automatically.

You can use the enable clause in both the create table statement and the alter tablel statement.

Enabling a primary key constraint that was disabled with the cascade option does not enable any foreign keys that are dependent upon the primary key.

Cascading Constraints: The cascade constarints clause is used along with the drop column clause.

The cascade constraints clause drops all referential integrity constraints that refer to the primary key and unique keys defined on the dropped table.

The cascade constraints clause also drops all the muticolumn constraints defined on the dropped columns

Ex: create table test1(pk number primary key, fk number, col1 number, col2 number, constraint fk_constraint foreignkey(fk) references test1, constraint ck1 check(pk>0 and col1>0), constarint ck2 check(col2>0));

An error is returned for the following statements:

Alter table test1 drop(pk); (pk is parent key)

Alter table test1 drop(col1); (col1 is referenced by multicolumn constraint ck1)

To remove the errors:

Alter table test1 drop(pk) cascade constraints;

Alter table test1 drop(pk,fk,col1) cascade constarints

Viewing Constraints:

After creating a table you can confirm its existence by issuing a describe command. The only constraint that you can verify is the not null constraint. The view all the constraints on your table query the user_constraints table.

Note: Constraints that are not named by the table owner receive the system assigned constraitnt name. In constraint type C stands for check, P stands for primary key, R for referential integrity and U for unique key. Notice that the NOT NULL constraint is really a check constraint.

```
Select constraint_name, constraint_type, search_condition from
user_constraints where table_name='employees';
```

Viewing the columns associated with constraints:

View the columns associated with constraint names in the user_cons_columns view.

```
Select constraint_name,column_name from user_cons_columns where
table_name='employees';
```

Views

Database Objects:

Table: Basic unit of storage composed of rows and columns.

View: Logically represents subset of data from one or more tables

Sequence: Generally primary key values

Index: Improves the performance of some queries.

Synonym: Alternative name for an object

View: You can present logical subsets or combinations of data by creating views of tables.

A view is a logical table based on a table or another view.

We use views

- To restrict data access
- To make complex queries easy
- To provide data independence
- To provide different views of the same data

Simple view Vs complex view:

The basic difference is related to the DML(insert, update and delete) operations

A simple view is one that

- Derives data from only one table
- Contains no functions or groups of data
- Can perform DML operations through the view

A complex view is one that

- Derives data from many tables
- Contains functions or groups of data
- Does not always allow DML operations through the view

Creating a view:

Create [or replace] [force| no force] VIEW view [(alias [, alias] ……)] as subquery [with check option [CONSTRAINT constraint]] [WITH READ ONLY [CONSTRAINT constraint]].

With check option: Specifies that only rows accessible to the view can be inserted or updated.

Ex: Create view empvu80 as select employee_id, last_name, salary from employees where department_id=80;

Describe empvu80 is used to describe the view.

Guidelines for creating a view:

- The subquery that defines a view can contain complex select syntax, including joins, groups and sub queries.
- The subquery that defines the view cannot contain an *order by* clause . The order by clause is specified when you retrieve data from the view.
- If you donot specify a constraint name for a view created with the *with check option* the system assigns a default name in the format Sys_cn

- We can use the *or replace* option to change the definition of the view without dropping and re creating it or regranting object privileges previously granted on it.

Creating a view by using a column aliases in the subquery:

Create view salvu50 as select employee_id id_number, last_name name, salary*12 ann_salary from employees where department_id=50;

Retrieving data from the view:

Select * from salvu50

When you access data using a view the oracle server performs the following operations

- It retrieves the view definition from the data dictionary table user_views.
- It checks access privileges for the view base table
- It converts the view query into an equivalent operation on the underlying base table or tables.

Modifying a view:

With the *or replace* option a view can be created even if one exists with this name already thus replacing old version of the view for ots owner.

Note: When assigning column aliases in the create view clause remember that the aliases are listed in the same order as the columns in the subquery.

Ex: Create or replace view empvu80(id_number, name, sal, department_id) as select employee_id, first_name, salary, department_id from employees where department_id=80;

Rules for performing DML operations on view:

- You can perform DML operations on simple views
- You cannot remove a row if the view contains the following
  - Group functions
  - The distinct keyword
  - The psuedo column *ROWNUM* keyword
  - Columnns defined by expression(salary*12)
  - *Not Null* columns in the base table that are not selected by the view

Using the with check option clause:

You can ensure that DML operations performed on the view stay within the domain of the view by using the with check option.

Ex: create or replace view empvu20 as select * from employees where department_id=20 with check option constraint empvu20_ck.

Any attempt to change the department number for any row in the view fails because it violates the with check option constraint.

Denying DML operations:

You can ensure that no DML operations occur by adding the *with read only option* to your view definition.

Any attempt to perform a DML on any row in the view results in an oracle server error

Note: If the user does not supply a constraint name, the system assigns a name in the form Sys_cn where n is an integer.

Ex: Create or replace view empvu10 (employee_number, employee_name,job_title) as select employee_id, last_name, job_id from employees where department_id=10 with read only.

Removing a view:

Drop view empvu80;

Inline view:

An inline view is a subquery with an alias(or correlation name) that you can use within a SQL statement.

A named subquery in the from clause of the main query is an example of an inline view.

An inline view is not a schema object

Ex: Select a.last_name, a.salary, a.department_id, b.maxsal from employees a, (select department_id, max(salary) maxsal from employees group by department_id) b where a.department_id=b.department_id and a.salary<b.maxsal;

Top N analysis:

The high level structure of a Top-n analysis query is

Select [column_list], rownum from (select [column_list] from table order by top n column) where rownum<=n;

Ex: select rownum as rank,last_name salary from (select last_name, salary from employees order by desc) where rownum<=3;

Other DataBase Objects

1. Sequence
2. Index
3. Synonym

Sequence : Is a user created object that can be shared by multiple users to generate Unique integers.

A typical usage for sequence is to create a primary key value ,which must be Unique for each row .the sequence is generated and incremented by an Internal oracle routine.

Sequence numbers are stored and generated independently of tables . Therefore, the same sequence can be used for multiple tables

```
Create SEQUENCE sequence
        [increment by n]
        [start with n       ]
        [{max value n| no max value}]
    [{min value n| no min value}]
        [{cycle| no cycle}]
         [{ cache n | no cache}].
```

Cache n | no cache: specifies how many values the oracle server preallocats and keep In memory (by default ,the oracle server caches 20 values).

Ex: Create sequence dept_depid_sea Increment by 10 Start with 120 Max value 9999 No cache No cycle

Note: If sequence values are cached ,they will be lost if there is a system failure.

Confirming sequences:

Verify your sequence values in the user_ sequence data dictionary table.

Ex: Select sequence_name ,min_value ,max_value,increment_by, last_number from User_ sequences;

Note: The last number columns displays the next available sequence value. It returns a Unique value every time it is referenced ,even for a different users.

Currval obtains the current sequence value.

Nextval must be issued for that sequence before currval contains a value.

Rules for using nextval and currval:

You can use nextval and currval in the following contexts:

The select list of select statement that is not a part of the subquery.

The select list of a subquery in an insert statement.

The values clause of an insert statement.

The set clause of an update statement.

Using a Sequence:

Ex: Insert into departments (department_id,department_name , location _id) values (dept_dept_id_seq.nextval,'support',2500);

To view the currval for the dept_depid_seq sequence.

Ex: Select dept_deptid_seq . currval from dual;

Casching sequence values in memory gives faster access to those values

Caps in sequence values can occur when

A rollback occurs

The system crasches

A sequence is used in another table.

If the sequence was created with nocache view the next available value,by Querying the user sequence table.

Modifying a Sequence :

Alter sequence dept_depid_seq

Increment by 20

Max value 99999

No cache

No cycle;

Guidelines for modifying:

You must be the owner or the alter privilege for the sequence .

Only future sequence numbers are affected.

The sequence must be dropped and recreated to restart the sequence at a different Number.

Some validation is performed . for example,a new maxvalue that is less than the current sequence number cannot be imposed.

Removing a Sequence:

Remove a sequence from the data dictionary by using the drop sequence statement.

Once removed the sequence can no longer be referenced.

Ex:Drop sequence dept_deptid_seq.

Index

An oracle server index is a schema object that can speed up the retrieval of rows by using a pointer . indexes can be created explicitly or automatically .if you do not have an index on the column , then a full table scan occurs.

An index provides direct and fast access to rows in a table ,its purpose is to reduce the necessity of disk i/o by using an indexed path to locate data quickly. The index is used and maintained automatically by the oracle server .once the index is created ,nodirect activity is required by the user

Indexes are logically and physically independent of the table they index.this means that they can be created or dropped at any time and have no effect on the base tables other indexes.

Note: when you drop a table ,corresponding indexes are also dropped.

Indexes are created in two ways .

-Automatically a unique index is created automatically when you define a primary key or unique constraint.

- Manually uses can create non unique indexes on columns to speed up access to the rows.

Ex: create index index ON table(column[,col]---)

Ex: create index emp_last_name_idx ON employees(last_name);

The user indexes date dictionary view contains the name of the index and its uniqueness.

The user_ind_columns view contains the index,the table name ,and the column name.

Ex: Select ic.index_name,ic.column_name,ic.column_position col_pos,ix.uniqueness from user_indexes ix,user_ind_columns ic where ic.index-name = ix.indexes_name and ic.table_name='employees';

Function Based Indexes:

A function based index is an index based on expressions

The index expression is built from table columns ,constants sql functions and user defined functions .

Ex: Cerate index upper_dept_name_idx ON departments UPPER (department _name);

Ex: Select * from departments where upper(department_name)='sales';

Removing a Index:

Ex: Drop index index;

Ex: Drop index upper_last_name_idx;

You cannot modify a index ,to change an index you must drop it and recreate it.

SET OPERATORS

Union / Union ALL

Intersect

Minus

Union : All distinct rows selected by either query.

Union all: All rows selected by either query , including all duplicates.

Interest : All distinct rows selected by both queries.

Minus : All distinct rows that are selected by the first select statement and not selected in the second select staement.

All the set operators has equal precedence.

Union: The no.of columns and the data types of the columns being selected must be identical in all the select statements used in the query the names of the columns need not be identical.

Union operators over all of the columns being selected .

Null values are not ignored during duplicate checking.

The in operator has highest precedence that the union operator.

By default ,the output is stored in ascending order of the first column of the select clause.

Ex: Select employee_id ,job_id from employees

Union

Select employee_id,job_id from job_history;

Union all: Unlike union ,duplicate rows are not eliminated and the output is not sorted by default.

The distinct keyword cannot be used .

Note: With the exception of the above , the guidelines for union and union all same.

Intersect: The number of columns and the datatypes of the columns being selected by the select statements in the queries must be identical in all the select statements used in the query .the names of the columns need not be identical.

Reversing the order of the intersected tables does not alter the result.

Intersect does not ignore null values .

Ex: Select employee_id ,job_id from employees

Intersect

Select employee_id ,job_id from job_history;

Minus: The number of columns and the data types of the columns being select ed by the select statements in the queries must be identical in all the select statements used in the query the names of the columns need not be identical .

All of the columns in the where clause most be in the select clause for the minus operator to work.