CH-1 INTERACTIVE SQL

Table Fundamentals

A table is database object that holds user data. Each column of the table will have a specific data type bound to it. Oracle ensures that only data , which is identical to the data type of the column ,will be stored within the column.

Data types in Oracle

Oracle supports the following data types.

1. Char
2. Varchar /Varchar2
3. Number
4. Date
5. Long

1. CHAR : The CHAR datatype stores fixed-length character strings. When you create a table with a CHAR column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes for the CHAR column width. The default is 1 byte. Oracle then guarantees that: When you insert or update a row in the table, the value for the CHAR column has the fixed length.

- If you give a shorter value, then the value is blank-padded to the fixed length.
- If a value is too large, Oracle Database returns an error.

2. VARCHAR2: The VARCHAR2 data type stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column. For each row, Oracle Database stores each value in the column as a variable-length field unless a value exceeds the column's maximum length, in which case Oracle Database returns an error. Using VARCHAR2 and VARCHAR saves on space used by the table.

For example, assume you declare a column VARCHAR2 with a maximum size of 50 characters. In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece stores only the 10 characters.

3. NUMBER(P,S): The NUMBER datatype stores fixed and floating-point numbers. They are upto 38 digits of precision. For numeric columns, you can specify the column as: colu**mn_name NUMBER**

Optionally, number can also specify a precision (total number of digits) and scale (number of digits to the right of the decimal point): **column_name NUMBER (precision, scale)**

If a precision is not specified, the column stores values as given. If no scale is specified, the scale is zero.

4. DATE: The DATE datatype stores point-in-time values (dates and times) in a table. The DATE datatype stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight). For input and output of dates, the standard Oracle date format is DD-MON-YY e.g. 13-AUG-89

5. LONG: This data type is used to store variable length character strings containing upto 2 GB.LONG data can be used to store arrays of binary data in ASCII format.

CREATE TABLE Command

The **CREATE TABLE** statement is used to create a new table in the database. This command defines each column of the table uniquely . Each column has a minimum of three attributes a name, datatype and size. Each column has a specific data type which specifies how data is stored in the column. Each column definition is separated from the other by a comma and the SQL statement is terminated with a semicolon.

The general format of CREATE command is

Syntax:

CREATE TABLE tablename

(column1 data type(size),

column2 data type(size),

column3 data type(size),

....

column-n data type(size));

Example:

a. Create a table called student that contains roll number, name and marks in three subjects

SQL> **create table student**

(rollno number(2),

name varchar2(20),

m1 number(2),

m2 number(2),

m3 number(2));

Table created.

Inserting data into tables

INSERT: The **INSERT s**tatement is used to insert records into a table. This statement loads the table with data to be manipulated later. When inserting a single row into the table the insert operation does the following:

- Creates a new row in the database table
- Loads the values passed into the columns specified

Note: Character data type values must be enclosed in single quotes(').

Syntax:

INSERT INTO <table name> (column1, column2,...., column n)
VALUES(<expression1>, <expression2>,.....<expression n>);

Example:

Method 1

SQL> **insert into student(rollno,name,m1,m2,m3) values (11,'Rama',65,75,50);**

1 row created.

Method 2

SQL> **insert into student values(&rollno,'&name',&m1,&m2,&m3);**

Enter value for rollno: 12

Enter value for name: Raju

Enter value for m1: 70

Enter value for m2: 80

Enter value for m3: 65

old 1: insert into student values(&rollno,'&name',&m1,&m2,&m3)

new 1: insert into student values(12,'Raju',70,80,65)

1 row created.

Method 3

insert into student values (11,'Rama',65,75,50);

1 row created

Viewing data in the table

SELECT : Once data is inserted into the tables, the next logical operation would be to view the data that has been inserted. The SELECT verb in SQL is used to achieve this. The SELECT command is used to retrieve rows from one or more table**.**

Syntax:

a. To select all rows and all columns

SELECT * FROM <table name>;

b**.** To retrieve selected columns and all rows

SELECT column1, column2 FROM <table name>;

c. To retrieve selected rows and all columns

SELECT * FROM <table name> WHERE <condition>;

d. To retrieve selected columns and selected rows

SELECT column1, column2 FROM <table name> WHERE <condition>;

e**.** To eliminate duplicate rows when using select statement –

The **DISTINCT** clause in the select statement allows removing of duplicate rows from the result set. The DISTINCT clause can only be used with select statements. It scans through the values in the columns specified and displays only unique values amongst them.

SELECT DISTINCT column1, column2 FROM <table name>;

Example:

a. Display the details of all the students

SQL> select * from student;

ROLLNO	NAME	M1	M2	M3
11	Raja	50	60	70
12	Rama	70	80	75
13	John	45	55	65
14	Gita	40	50	45
15	Tom	30	35	25

5 rows selected

b. Display the roll number and names of all the students

SQL> select rollno,name from student;

ROLLNO	NAME

11 Raja

12 Rama

13 John

14 Gita

15 Tom

5 rows selected

c. Display the details of students who got more than 60 in subject 2

SQL> select * from student where m2>=60;

ROLLNO	NAME	M1	M2	M3
11	Raja	50	60	70
12	Rama	70	80	75

2 rows selected

d. Display names of students who secured more than 50 either in subject1 or subject 3

SQL> select name from student where m1>=50 or m3>=50;

NAME

Raja

Rama

John

3 rows selected

e. Display the different courses available

SQL> select distinct course from student;

COURSE

BCA

BBM

MCA

3 rows selected

Sorting data in a table

The rows retrieved from the table will be sorted in ascending 0r descending order on the condition specified .

Syntax:

SELECT * FROM <table name> ORDER BY <ColumnName1>,< ColumnName2><[sort order]>;

Example:

a. Display the details of all the students according to the RollNo

SQL> select * from student order by RollNo;

ROLLNO	NAME	M1	M2	M3
11	Raja	50	60	70
12	Rama	70	80	75
13	John	45	55	65
14	Gita	40	50	45
15	Tom	30	35	25

5 rows selected

Creating a table from a table

We can create a table from another table in SQL. The syntax is as follows

CREATE TABLE <new table> (<column1>, <column2>)

AS SELECT <column1>, <column2> FROM <existing table>;

Example:

a. Create a table named sample from the existing table student that contains only rollno and names of students

SQL> create table sample(rollno, name) as select rollno,name from student;

Table created.

b. Create a table named sample1 that contains the same structure as student but with no records

SQL> create table sample1 as select * from student where 1=2;

Table created.

SQL> select * from sample1;

no rows selected

Inserting data into a table from another table

Syntax:

INSERT INTO <TableName> SELECT <columnname1>,<columnnameN >FROM <TableName>;

Insertion of a data set into a table from another table

Syntax:

INSERT INTO <TableName> SELECT <columnname1>,<columnnameN >FROM <TableName> WHERE <condition>;

DELETE: The DELETE command is used to delete rows from a table that satisfies the condition and returns the number of records that were deleted. If the DELETE command is executed without a WHERE clause then, all rows are deleted.

Syntax:

a. DELETE FROM <table name>; ---- all rows

b. DELETE FROM <table name WHERE <condition>; ---- specific rows

Example:

a. Remove all the records from sample table.

SQL> select * from sample;

ROLLNO NAME

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

11 Raja

12 Rama

13 John

14 Gita

15 Tom

5 rows selected

SQL> delete from sample;

5 rows deleted.

b. Remove the details of student 'Raja'

SQL> delete from student where name='Raja';

1 row deleted.

UPDATING THE CONTENTS OF A TABLE

UPDATE: The UPDATE command is used to change or modify data values in a table. The UPDATE in SQL is used to either update all the rows or selected rows in a table. The UPDATE statement updates columns in the existing table's rows with new values. The SET clause indicates which column data should be modified and the new values they should hold. The WHERE clause, if given, specifies which rows should be updated. Otherwise all the rows are updated.

Syntax:

a. **UPDATE <table name> SET <column 1>=<expression>,<column n>= <expression>;**

b. **UPDATE <table name>SET <column 1>=<expression>, <column n>= <expression>WHERE <condition>;**

Example:

a. Increment the marks of all students by 10 in subject2

   SQL> update student set m2=m2+10;

   4 rows updated.

b. Update the total of all the students

   SQL> update student set total=m1+m2+m3;

   4 rows updated.

c. Update the student details by changing the name of Rama to Sriram

   SQL> update student set name='Sriram' where name='Rama';

   1 row updated.

Modifying the structure of tables

ALTER: The structure of a table can be modified by using the ALTER TABLE command. ALTER TABLE allows the user to change the structure of the existing table. With ALTER TABLE command it is possible to

- Add or delete the columns
- Create or destroy indexes
- Change the data type of existing columns
- Rename columns or table itself

ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, then original table is deleted and the new one is modified.

Syntax:

a. Adding new tables

   ALTER TABLE <table name>

ADD(<new column> <data type>(size),

:

<new column> <data type>(size));

b. Modifying existing columns

ALTER TABLE <table name>

MODIFY(<column> <new datatype>(new-size));

c. Dropping a column from a table

ALTER TABLE <table name>DROP COLUMN(<ColumnName>;

Example:

a. Add a new column Total with data type number and size 2

SQL> alter table student add (total number(2));

Table altered.

b. Change the size of total to 3

SQL> alter table student modify (total number(3));

Table altered.

Renaming tables

RENAME

RENAME command is used to rename or give another name to the existing table.

Syntax:

RENAME <old table name> TO <new table name>;

Example:

SQL> rename student to student1;

Table renamed.

DESTROYING TABLES: The DROP TABLE statement is used to destroy a specific table. If a table is dropped all the records held within it are lost can cannot be recovered.

Syntax:

DROP TABLE <table name>;

Example:

SQL> drop table student1;

Table dropped.

Displaying table structure

DESCRIBE: The DESCRIBE command is used to display the structure of a table. This command displays the column names, the data types and the special attributes that are connected to the table.

Syntax:

DESCRIBE <table name>;

Example:

SQL> describe student;

Name	Null?	Type
ROLLNO		NUMBER(2)
NAME		VARCHAR2(20)
M1		NUMBER(2)
M2		NUMBER(2)
M3		NUMBER(2)
TOTAL		NUMBER(3)

CH-2 DATA CONSTRAINTS

There are two types of data constraints that can be applied to data being inserted into an oracle table. One type of constraint is the **I/O constraint** that determines the speed at which data can be inserted or extracted from a table. The other type of constraint is called the **business rule constraint.**

1) I/O constraints: The various types of I/O constraints are

i) Primary key constraint – A primary key is one or more columns in a table use to uniquely identify rows in a table. None of the fields that are a part of primary key can contain null values. A table can have only one primary key. The data in that column should be unique i.e. no duplicate values are allowed. The column is a mandatory column i.e. it cannot be left blank and NOT NULL attribute A Single column key is called a simple key. A multicolumn key is called a composite primary key

Syntax :

At column Level: **<ColumnName><Datatype>(<size>) PRIMARY KEY**

At table level: **PRIMARY KEY(<ColumnName>,(<ColumnName>)**

ii) Foreign key constraint – Foreign keys represent relationship between tables. A foreign key is a column whose values are derived from primary key of some other tables. The table in which the foreign key is defined is called **foreign table**. The table in which the primary key is defined is called the **primary table or master table**. The master table can be referenced in the foreign key definition by using the **references** clause.

At column Level:

Syntax :

<ColumnName><Datatype>(<size>)

references<TableName>[(<ColumnName>)][on DELETE CASCADE]

At table level:

FOREIGN KEY (<ColumnName>,[(<ColumnName>]) references<TableName>

[(<ColumnName>),<ColumnName>)

Oracle displays an error message when the record in the master table is deleted and corresponding records exists in the foreign table. It prevents the delete operation to be performed. For this reason, Oracle provides **ON DELETE CASCADE** option. If the **ON DELETE CASCADE** option is set, a delete operation in the master table will trigger a DELETE operation for all the corresponding records in all the foreign tables.

Assigning User Defined Names to constraints

A constraint can be given a user-defined name by preceding the constraint defined with the reserved word CONSTRAINT and user-defined name

Syntax : CONSTRAINT<Constriant Name><Constraint Definition>

User Constraints defined at column level

Syntax:

<**ColumnName><Datatype>(<size)UNIQUE**

User Constraints defined at Table level

CREATE TABLE tablename (column1 data type(size), column2 data type(size),UNIQUE (<ColumnName1>,<ColumnName2>));

2) Business rule constraints

Oracle allows programmers to define constraints at table level and column level. If the constraints are defined as an attribute of a column definition when creating or altering the table structure, they are column level constraints. If data constraints are defined after defining all the table column attributes when creating or altering a table structure, then it is table level constraints. The various business rule constraints are

i. **NOT NULL** – A NULL is different from zero or blank. A NULL value can be inserted into columns of any data type. The NOT NULL constraint ensures that a table column cannot

be left empty. When a column is defined as NOT NULL, then that column becomes mandatory. It implies that a value must be entered into the column if that record is to be accepted for storage in the table**.**

Syntax: <column name> <data type(size)> NOT NULL

ii. **CHECK constraint –** This constraint must be specified as a logical expression that evaluates either to a TRUE or FALSE. E.g. Data values being inserted to begin with a particular letter, gender to accept only M or F, data values to be accepted only in uppercase.

Syntax:

a. Check constraint at Column level:

<column name> <data type(size)> check(logical expression)

b. Check constraint at Column level**:**

check(logical expression)

Example:

Create a table EMPLOYEE using SQL command to store details of employees such as EMPNO, NAME, DESIGNATION, DEPARTMENT, GENDER and SALARY. Specify Primary Key and NOT NULL constraints on the table and allow only 'M' or 'F' for the column GENDER. Write the following SQL queries :

```
SQL> create table EMPLOYEE
 (empno varchar2(4) PRIMARY KEY,
 name varchar2(20) NOT NULL,
 designation varchar2(20) NOT NULL,
 dept varchar2(20),
 gender char(1),
 salary number(6) NOT NULL,
```

check (gender in('M','F')));

Table created.

User Constraints Table

A table can be created with multiple constraints attached to its columns. The user can view the structure along with its constraints using the DESCRIBE command. This command only displays the column names, data type, size and not null constraints. The information about other constraints like primary key, foreign key is not available using the describe command. Oracle stores such information in a table called the USER CONSTRAINTS.

Some of the columns available in USER CONSTRAINTS table is listed below.

Column Name	Description
OWNER	The owner of the constraint
CONSTRAINT_NAME	The name of the constraint
TABLE_NAME	The name of the table associated with the constraint
CONSTRAINT_TYPE	The type of constraint P:Primary key R:Foreign key U:Unique key C:Check constraint
SEARCH_CONDITION	Search condition used in check constraint
R_OWNER	The owner of table referenced by foreign key
R_CONSTRAINT_NAME	The name of the constraint referenced by foreign key

Defining and dropping integrity constraints in ALTER table:

Integrity constraints can be applied to tables using the ALTER table command.

Example:

a. Make Rollno field as primary key in student table

SQL> alter table student add primary key(rollno);

Table altered.

b. Show that the above constraint has been applied using USER CONSTRAINTS table

SQL>select owner, table_name, constraint_type from user_constraint where table_name='STUDENT';

OWNER TABLE_NAME C

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

SCOTT STUDENT P

Integrity constraints can be dropped if the constraint is no longer needed. This can be achieved by the ALTER table using the DROP clause.

Example:

a. Drop primary key constraint from student

SQL> alter table student drop primary key;

Table altered.

Default value Concepts:

Default values can be assigned to columns when we create a table. When a record is loaded into the table and a column is left empty, oracle engine will automatically load this column with the default value specified. The data type of the default value must match with the data type specified for the column.

Syntax:

<column name> <data type(size)> default <value>;

COMMISION NUMBER(10) DEFAULT 0;

CH-3 COMPUTATIONS ON TABLE DATA

Arithmetic Operators

Oracle allows arithmetic operators to be used while viewing records from a table or while performing data manipulation operations. The various arithmetic operators are as follows

Operator	Meaning
+	Addition
-	Subtraction
/	Division
*	Multiplication
**	Exponentiation

Comparison Operators

Comparison operators are used to compare the column data with specific values in a condition. Comparison Operators are also used along with the SELECT statement to filter data based on specific conditions.

The below table describes each comparison operator

Operators	Description
=	equal to
<>, !=	is not equal to
<	Less than
>	Greater than
<=	less than or equal to
>=	greater than or equal to

Examples:

a. List the roll number and names student whose total is more than 200

SQL> select rollno,name from student where total>200;

ROLLNO	NAME
12	Sriram
17	Shyam

2 rows selected

b. List the names of students who secured less than 50 in subject

SQL> select name from student where m1<50;

NAME
John
Gita
Tom

3 rows selected

c. List the names of students who got more than or equal to 70 in subject3

SQL> select name from student where m3>=70;

NAME
Sriram
Shyam

2 rows selected

Logical operators : There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. Logical operators are used in the WHERE clause and allows you to combine more than one condition.

Logical Operators	Description
OR	For the row to be selected at least one of the conditions must be true.
AND	For a row to be selected all the specified conditions must be true.
NOT	For a row to be selected the specified condition must be false.

Example:

a. List the details of students who secured more than 60 in subject 1 and subject 2

SQL> select * from student where m1>60 and m2>60;

ROLLNO	NAME	M1	M2	M3	TOTAL	COURSE
12	Sriram	70	90	75	235	bca

1 row selected

b. List the names of students who secured less than 50 in any subject

SQL> select name from student where m1<50 or m2<50 or m3<50;

NAME
John
Gita
Tom

3 rows selected

Range Searching

In order to select data that is within a range of values, the BETWEEN operator is used. The BETWEEN operator allows the selection of rows that contain values within a specified lower and upper limit. The two values in between the range must be linked with a keyword AND. The BETWEEN operator can be used with both character and numeric data types.

Example:

a. List the names of students whose total is in the range 180 and 250

SQL> select * from student where total between 180 and 250;

ROLLNO	NAME	M1	M2	M3	TOTAL	COURSE
12	Sriram	70	90	75	235	BCA
17	Shyam	80	60	70	210	BBM

2 rows selected

Pattern Matching

LIKE Operator predicates

The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose we use a the following wildcard character

- % allows to match any string of any length
- _ allows to match on a single character.

Example:

a. List the names of students whose name begins with 'S'

SQL> select name from student where name like 'S%'

NAME

Sriram

Sita

Shyam

3 rows selected

b. List the name and roll number of students whose name ends with 'a'

SQL> select rollno,name from student where name like '%a';

ROLLNO NAME

14 Gita

16 Sita

2 rows selected

c. List the names of students who have 'i' as the second letter in their name.

SQL> select name from student where name like '_i%';

NAME

Gita

Sita

2 rows selected

d. List the course whose name ends with the letter 'm'

SQL>select course from student where course like '%m'

COURSE

bbm

bbm

2 rows selected

IN and NOT IN predicates:

The arithmetic operator (=) compares a single value to another single value. In case a value needs to be compared to a list of values then the IN predicate is used. It reduces the need to use multiple OR conditions in a query.

Example:

a. List the student names and course who are studying either MCA OR BCA

SQL> select name, course from student where course in ('MCA','BCA');

NAME	COURSE
Sriram	BCA
Gita	BCA
Sita	MCA
Tom	MCA

4 rows selected

b. List the student names along with their course who are not studying MCA

SQL> select name, course from student where course not in ('mca')

NAME	COURSE
Sriram	BCA
John	BBM
Gita	BCA
Shyam	BBM

4 rows selected

Oracle table – DUAL

Dual is a table owned by SYS. SYS owns the data dictionary and DUAL is a part of data dictionary. Dual is a small Oracle work table which consists of only one row and one column. It contains the value x in that column. It is used to perform small arithmetic calculations and also supports date retrieval and formatting.

SQL> **describe dual;**

Name Null? Type

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

DUMMY VARCHAR2(1)

SQL> select * from dual;

DUAL

X

Example:

SQL> select 3*5 from dual;

3*5

15

SYSDATE is a pseudo column that contains the current date and time. It requires no arguments when selected from the table DUAL and returns the current date.

Example:

SQL> select sysdate from dual;

SYSDATE

18-NOV-15

Oracle Functions: Oracle functions serve the purpose of manipulating data items and returning the result. Functions are also capable of accepting user-supplied variables or values and operating on them. Such variables are called arguments. The general format of a function is

Function_name(argument1, argument2,....)

Oracle functions can be classified depending on whether they operate on a single row or a group of rows. Oracle functions can be classified as

- Group Functions or Aggregate Functions
- Scalar functions or single row functions

Group Functions (Aggregate Functions)

Functions that act on a **set of values** are called group functions. A group function returns a single result row for a group of queried rows. A list of group functions with examples is given below.

Examples:

1. **AVG** – Returns the average of the values in a column

   Syntax: AVG([<DISTINCT>|<ALL>]<n>)

   E.g. Display the average total of all the students

   SQL> select avg (total) "Average" from student;

   Average

   171.666667

2. **MAX** – Returns the maximum value in a given column

   Syntax: MAX([<DISTINCT>|<ALL>]<expr>)

   E.g. Display the maximum total from student table

   SQL> select max(total) "Maximum Total" from student;

   Maximum Total

   235

3. **MIN** – Returns the minimum value in a given column

   Syntax: MIN([<DISTINCT>|<ALL>]<expr>)

   E.g. Display the minimum total from student table

   SQL> select min(total) "Minimum Total" from student;

   Minimum Total

   100

4. **COUNT** (*)– Returns the number of rows in a table, including duplicates & nulls

   Syntax: COUNT(*)

   E.g. Display the number of students in student table

   SQL> **select count(*) from student;**

   COUNT(*)

   6

5. COUNT (expr)– Returns the number of rows where expr is not null

   Syntax: COUNT ([<DISTINCT>|<ALL>]<expr>)

6. **SUM** – Returns the sum of values in a given column

   Syntax: SUM ([<DISTINCT>|<ALL>]<n>)

E.g. Display the sum total of marks in subject1 from student table

```
SQL> select sum(m1) "Marks in Subject1" from student;

Marks in Subject1
-----------------
295
```

Scalar Functions (Single row Functions)

Functions that act only on **one value at a time a**re called scalar functions. A single row function returns one result for every row of a queried table. They can further be grouped depending on the data type of value upon which they act. The classification of scalar functions is as follows:

- String functions – String data type
- Numeric functions – Numeric data type
- Date functions – Date data type
- Conversion functions – converting one data type to another

String Functions:

String functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Few of the character or text functions are as given below:

Function Name	**Return Value**
LOWER (string_value)	All the letters in 'string_value' is converted to lowercase.
UPPER (string_value)	All the letters in 'string_value' is converted to uppercase.
INITCAP (string_value)	All the letters in 'string_value' is converted to mixed case.
LTRIM (string_value, trim_text)	All occurrences of 'trim_text' is removed from the left of 'string_value'.
RTRIM (string_value, trim_text)	All occurrences of 'trim_text' is removed from the right of 'string_value' .

TRIM (trim_text FROM string_value)	All occurrences of 'trim_text' from the left and right of 'string_value' , 'trim_text' can also be only one character long .
SUBSTR (string_value, m, n)	Returns 'n' number of characters from 'string_value' starting from the 'm' position.
LENGTH (string_value)	Number of characters in 'string_value' in returned.
LPAD (string_value, n,pad_value)	Returns 'string_value' left-padded with 'pad_value' . The length of the whole string will be of 'n' characters.
RPAD (string_value, n, pad_value)	Returns 'string_value' right-padded with 'pad_value' . The length of the whole string will be of 'n' characters.

Examples:

Function Name	**Examples**
LOWER(string)	SQL> select lower('Srinivas') from dual srinivas
UPPER(string)	SQL> select upper('Srinivas') from dual; SRINIVAS
INITCAP(string)	SQL> select initcap('Sri Nivas') from dual; sRI nIVAS
LTRIM(string, trim_text)	SQL> select ltrim('nivas','ni') from dual; Vas
RTRIM(string, trim_text)	SQL>select rtrim('sreenivas','nivas') from dual Sree
SUBSTR(string_value, start, number of characters)	SQL> select substr('Srinivas',4,5) from dual; Nivas
LENGTH (string_value)	SQL> select length('Srinivas') from dual; 8
LPAD (string_value, n, pad_value)	SQL> select lpad('Nivas',10,'*') from dual; *****Nivas

RPAD (string_value, n, pad_value)	SQL> select rpad('Nivas',10,'*') from dual Nivas*****

Numeric Functions:

Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	Return Value
ABS (x)	Absolute value of the number 'x'
POWER(m,n)	Returns m raised to n^{th} power. Here n must be an integer
ROUND (x, y)	Rounded off value of the number 'x' up to the number 'y' decimal places
CEIL (x)	Integer value that is Greater than or equal to the number 'x'
FLOOR (x)	Integer value that is Less than or equal to the number 'x'
TRUNC (x, y)	Truncates value of number 'x' up to 'y' decimal places

The following examples explains the usage of the above numeric functions

Function Name	Examples
ABS (x)	SQL> select abs(-26) from dual; ABS(-26) 26
CEIL (x)	SQL> select ceil(34.6) from dual; CEIL(34.6) 35
FLOOR (x)	SQL> select floor(34.6) from dual; FLOOR(34.6) 34
TRUNC (x, y)	SQL> select trunc(38.546,2) from dual; TRUNC(38.546,2)

	38.54
ROUND (x, y)	SQL> select round(38.546,2) from dual ROUND(38.546,2) 38.55
POWER(m,n)	SQL> select power(5,2) from dual; POWER(5,2) 25

DATE CONVESION FUNCTIONS

The DATE data type is used to store date & time information

The value in the column of a DATE data type is in default format

DD-MON-YY HH: MI: SS

If data from a date column has to viewed in any other format then

TO_DATE function is used

To_DATE: convert a character field to a date field

Syntax: TO_DATE(char[,fmt])

Date Functions:

These functions are used to manipulate and extract values from the date column of a table

Function Name	Return Value
ADD_MONTHS(date,n)	Returns the date after adding the number of months specified in the function
LAST_DAY(date)	Returns the last date of the month specified in the function
MONTHS_BETWEEN(d1,d2)	Returns the number of months between d1 and d2

Examples

Function Name	Example
ADD_MONTHS(date,n)	SQL>select add_months('25-sep-2013',3) from dual

	ADD_MONTH 25-DEC-13
LAST_DAY(date)	SQL> select last_day('25-sep-2013') from dual; LAST_DAY(30-SEP-13
MONTHS_BETWEEN(d1,d2)	SQL>select months_between('25-sep-2013','25-nov-2013') from dual; MONTHS_BETWEEN('25-SEP-2013','25-NOV-2013') -2 SQL>select months_between('25-sep-2013','25-apr-2013') from dual; MONTHS_BETWEEN('25-SEP-2013','25-APR-2013') 5

Conversion Functions:

These are functions that help us to convert a value in one form to another form. For example, a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by 'date_format'.
NVL (x, y)	If 'x' is NULL, replace it with 'y'. 'x' and 'y' must be of the same datatype.

The below table provides the examples for the above functions

Function Name	Examples
TO_CHAR ()	SQL> select to_char(sysdate,'dd-month-yyyy') from dual;

	TO_CHAR(SYSDATE,' 25-september-2013
TO_DATE ()	SQL> select to_date('15-aug-1947','dd-month-yy') from dual; TO_DATE(' 15-AUG-47

Grouping data from tables in sql

GROUP BY and HAVING Clause:

The GROUP BY clause tells Oracle to group rows based on distinct values that exists from specified columns. The group by clause creates a data set containing several sets of records grouped together based on a condition.

Syntax:

SELECT <column1>,<column2>....<column n>

AGGREGATE FUNCTION(Expression)

FROM table name WHERE <condition>

GROUP BY <column1>,<column2>....<column n>;

Example:

a. Display the number of students in each course

SQL> select course, count(*) "Number of Students" from student group by course;

COURSE Number of Students

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

BCA 2

BBM 2

MCA 2

3 rows selected

b. Display the maximum total from each course

SQL> select course, max(total) from student group by course;

COURSE MAX(TOTAL)

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

BCA 235

BBM 210

MCA 165

3 rows selected

Having clause

The HAVING clause is used in conjunction with the group by clause. It imposes a condition on the group by clause, which further filters the groups created by the group by clause. Each column specification specified in the having clause must occur in the list of columns mentioned in the group by clause.

Example:

a. List the course with more than 1 student.

SQL> select course from student group by course having count(*)>1;

COURSE

BCA

MCA

Determining whether values are unique

the HAVING clause can be used to find unique values in situations to which DISTINCT does not apply the DISTINCT clause eliminates duplicates,but does not show which values actually were duplicated in the original data the HAVING clause can identify which values were unique or non-unique

ORDER BY Clause: The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

Syntax:

SELECT column-list

FROM table_name [WHERE condition]

[ORDER BY column1 [, column2, .. columnN] [DESC]];

Example:

a. Display the student name and total marks in ascending order.

SQL> select name,total from student order by total;

NAME TOTAL

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

Tom 100

Gita 145

Sita 165

Shyam 210

Sriram 235

5 rows selected

b. Display the student names and marks in subject 1 in descending order

SQL> select name,m1 from student order by m1 desc;

NAME M1

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

Sriram 70

Shyam 60

Sita 50

Gita 40

Tom 30

5 rows selected

Sub Queries

A subquery is a form of SQL statement that appears inside another SQL statement. It is also called a nested query. The statement containing the subquery is called parent query. The parent statement uses the result set returned by the subquery. It can be used for

- Inserting records in the target table
- Create and insert records in a table
- Update records in target table
- To provide values for conditions in the WHERE, HAVING, IN clause used with SELECT, UPDATE and DELETE commands

The concept of using a subquery in the FROM clause of the SELECT statement is called **inline view**.

A **correlated subquery** is one where a subquery references a column from a table in the parent query. A correlated subquery is evaluated once for each row of the parent statement, which can be any of the SELECT, UPDATE or DELETE.

The **EXISTS** operator is usually used with correlated subqueries. This operator enables to test whether a value retrieved by the outer query exists in the result set of the values retrieved by the inner query. If the subquery returns at least one row, the operator returns **true**. If the value does not exist, it returns **false**. The **EXISTS** operator ensures that the search in the inner query terminates when at least one match is found. The **NOT EXISTS** operator enables to test whether a value retrieved by the outer query is not a part of the result set of the values retrieved by the inner query.

Examples:

Consider the information in the following tables

SQL> select * from employ;

EMPNO	ENAME	SAL	DEPTNO
E001	Raja	6000	D002
E002	Rani	7000	D001
E003	Jaya	5000	D003
E004	Ranbir	8000	D001
E005	Rajni	4000	D004
E006	Sita	5500	D003

6 rows selected.

SQL> select * from depart;

DNO	DNAME	LOC
D001	accounts	chennai
D002	research	mumbai
D003	HRD	bangalore
D004	EDP	hyderabad
D005	Inventory	Goa

4 rows selected

a. Display the names of employees who are working in Chennai

SQL> select ename from employ where deptno in (select dno from depart where loc='chennai');

ENAME

Rani

Ranbir

2 rows selected

b. Display the location where Sita is working

SQL> select loc from depart where dno=(select deptno from employ where ename='Sita');

LOC

Bangalore

c. List the employee number, employee name, salary and average salary of the department whose salary is more than the average salary in the department(inline view)

SQL> select a.empno,a.ename,a.sal,b.avgsal from employ a,(select deptno,avg(sal) avgsal from employ group by deptno) b where a.deptno=b.deptno and a.sal>b.avgsal

EMPNO	ENAME	SAL	AVGSAL
E006	Sita	5500	5250
E004	Ranbir	8000	7500

d. List the employee names, salary who are drawing maximum salary in each department (correlated subquery)

SQL> select ename,deptno,sal from employ a where sal = (select max(sal) from employ where deptno=a.deptno);

ENAME	DEPT	SAL
Raja	D002	6000
Ranbir	D001	8000
Rajni	D004	4000
Sita	D003	5500

e. List the departments that do not have any employees

SQL> select dno,dname from depart a where not exists(select deptno from employ where deptno=a.dno);

DNO DNAME

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

D005 inventory

f. List the employee names who are working in accounts department

SQL> select ename from employ a where exists (select dname from depart where dno=a.deptno and dname='accounts');

ENAME

Ranbir

Rani

Joins

There are occasions where we need to retrieve data from multiple tables. This is achieved in SQL using joins. Tables are joined on columns that have same data type and data width in the tables. The tables are related with the help of primary key and foreign keys of the table. The different types of joins are

1) INNER JOIN
2) OUTER JOIN
3) CROSS JOIN
4) SELF JOIN

1) INNER JOIN – These joins are also called equi joins. They are known as equi joins because the where clause compares two columns using the = operator. It is the most commonly used operator. It returns all the rows from both the tables where there is a match.

Example:

a. Display the names of employees and the department in which they work

```
SQL> select ename,dname from employ, depart where employ.deptno=depart.dno;

ENAME        DNAME
---------- ----------
Raja         research
Rani         accounts
Jaya         HRD
Ranbir       accounts
Rajni        EDP
Sita         HRD
6 rows selected.
```

b. Display the names of employees working in HRD department

```
SQL> select ename from employ, depart where employ.deptno=depart.dno
  and dname='HRD';

ENAME
----------
Jaya
Sita
```

2) OUTER JOIN – This type of join can be used in selecting rows from both the tables regardless of whether the tables have common values or not. NULL values are appended in the result set where the data is missing. For left outer join use a (+) to the left condition and for right outer join use a (+) to the right condition.

Syntax for Left outer join

```
SELECT table1.column, table2.column
FROM table1t1, table2 t2
```

WHERE t1.column(+)=t2.column;

Syntax for Right outer join

SELECT table1.column, table2.column

FROM table1t1, table2 t2

WHERE t1.column=t2.column(+);

Example:

a. Display the employee name and department using left outer join

SQL>select ename, dname from employ e, depart d where e.deptno(+)=d.dno;

ENAME DNAME

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

Raja research

Rani accounts

Jaya HRD

Ranbir accounts

Rajni EDP

Sita HRD

6 rows selected.

b. Display the employee name and department using right outer join

SQL> select ename, dname from employ e, depart d where e.deptno=d.dno(+);

ENAME DNAME

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

Ranbir accounts

Rani accounts

Raja research

Sita HRD

Jaya HRD

Rajni EDP

6 rows selected.

3) CROSS JOIN – A cross join returns the Cartesian product. This means that the join combines every row from the left table with every row in the right table.

Example: SQL> select * from sam1;

ENO ENAME

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

E001 janani

E002 jahnavi

SQL> select * from sam2;

PNO PNAME

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

P001 soap

P002 shampoo

a. llustrate a cross join operation on sam1 and sam2

SQL> select eno,ename,pno,pname from sam1 cross join sam2;

ENO ENAME PNO PNAME

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

E001 janani P001 soap

E001 janani P002 shampoo

E002 jahnavi P001 soap

E002 jahnavi P002 shampoo

4) SELF JOIN – A self-join is one where the same table is involved in the join operation

Example:

a. Display the employee names and salary who earn more than Sita

SQL> select e1.ename,e1.sal from employ e1, employ e2* where (e1.sal>e2.sal) and (e2.ename='Sita');

ENAME SAL

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

Raja 6000

Rani 7000

Ranbir 8000

UNION, INTERSECT and MINUS

Union Clause:

Multiple queries can be put together and their output can be combined using the union clause. The union clause merges the output of two or more queries into a single set of rows. Here, the number of columns and the data type of the columns must the same in all the select statements used in the query.

Intersect Clause:

Multiple queries can be put together and their output can be combined using the intersect clause. It outputs only the rows produced by both the queries intersected. The output in the intersect clause will include only those rows that are retrieved common to both the queries. Here, the number of columns and the data type of the columns must the same in all the select statements used in the query.

Minus Clause:

Multiple queries can be put together and their output can be combined using the minus clause. The minus clause outputs the rows produced by the first query, after filtering the rows retrieved by the second query. Here, the number of columns and the data type of the columns must the same in all the select statements used in the query.