GENERAL INSTRUCTIONS FOR LABORATORY CLASSES

DO'S

Without Prior permission do not enter into the Laboratory.

While entering into the LAB students should wear their ID cards. The Students should come with proper dress code.

Students should sign in the LOGIN REGISTER before entering into the laboratory.

Students should come with observation, record note, class notes, and lab relevant book to the laboratory. Students should maintain silence inside the laboratory.

After completing the laboratory exercise, make sure to shutdown the system properly.

DONT'S

Students bringing the bags inside the laboratory..

Students wearing slippers insides the laboratory.

Students using the computers in an improper way.

Students scribbling on the desk and mishandling the chairs. Students using mobile phones inside the laboratory.

Students making noise inside the laboratory.

INTRODUCTION TO SQL

SQL:

Structured query language pronounced as (SEQUEL). This language is used to communicate to oracle database.

Database Management System (DBMS):

It is a software it helps to manage the database management system should able to perform the following activities very easily.

1. Inserting the new data.
2. Updating the exiting data.
3. Deleting unnecessary data.
4. Retrieving the require data.

A database along with the software which helps to manage. The database is called database management system (DBMS).

A DBMS which is based on relational theory is called as relational database management system.

Examples of RDBMS:

1. ORACLE
2. SQL SERVER
3. DB2
4. MYSQL
5. SYBASE
6. TERA DATA
7. MS ACCESS

SQL Commands

The SQL language is subdivided according to their functions as follows

DDL - Data Definition Language

DML - Data Manipulation Language

DRL/DQL - Data Retrieval Language / Data Query Language DCL) - Data Control Language

TCL) - Transaction Control

Data Definition Language (DDL):

Data Definition Language (DDL) or Schema Definition Language, statements are used to define the database structure or schema.

CREATE - to create objects in the database

ALTER - alters the structure of the database

DROP - delete objects from the database

TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed **COMMENT** - add comments to the data dictionary

RENAME - rename an object

Data Manipulation Language (DML):

Data Manipulation Language (DML) statements are used for managing data within schema objects.

INSERT - insert data into a table

UPDATE - updates existing data within a table

DELETE - deletes all records from a table, the space for the records remain **MERGE - UPSERT** operation (insert or update)

CALL - call a PL/SQL or Java subprogram

EXPLAIN PLAN - explain access path to data **LOCK TABLE** - control concurrency

Data Retrieval Language / Data Query Language (DRL/DQL):

SELECT - retrieve data from the a database

Data Control Language (DCL):

Data Control Language (DCL) statements. Some examples:

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command

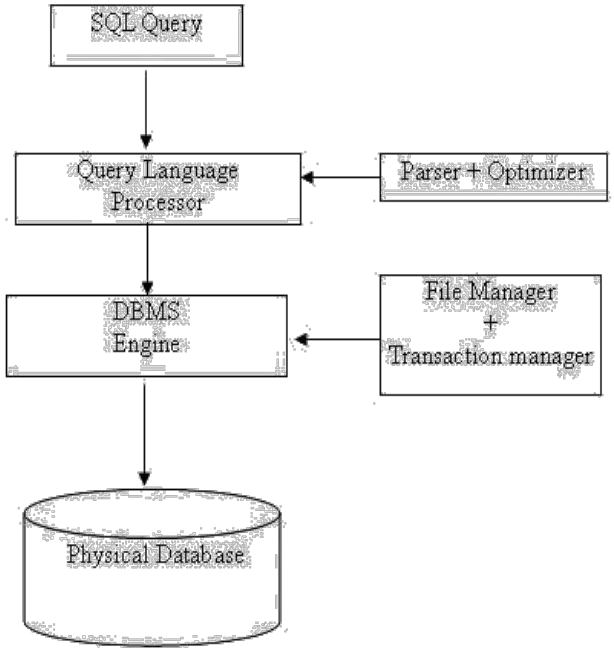
Transaction Control (TCL):

Transaction Control (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

COMMIT - save work done

SAVEPOINT - identify a point in a transaction to which you can later roll back **ROLLBACK** - restore database to original since the last COMMIT

SET TRANSACTION - Change transaction options like isolation level and what rollback segment to use



Fiq: SQL Architecture:

Rules:

1. Oracle reserved words cannot be used.
2. Underscore, numerals, letters are allowed but not blank space.
3. Maximum length for the table name is 30 characters.
4. Two different tables should not have same name.
5. We should specify a unique column name.
6. We should specify proper data type along with width.
7. We can include “not null” condition when needed. By default it is ‘null’.
8. It is a command based language.
9. It is not case sensitive.
10. Every command should end with ‘;’.
11. Every command starts with “verb”.
12. It is similar to English. This language is developed in the year 1972. Mr.CODD, by IBM developed by “IBM”.

WORDS TO REMEMBER

This appendix lists words that have a special meaning to Oracle. Each word plays a specific role in the context in which it appears. For example, in an INSERT statement, the reserved word INTO introduces the tables to which rows will be added. But, in a FETCH or SELECT statement, the reserved word INTO introduces the output host variables to which column values will be assigned.

Oracle Reserved Words

The following words are reserved by Oracle. That is, they have a special meaning to Oracle and so cannot be redefined. For this reason, you cannot use them to name database objects such as columns, tables, or indexes.

ACCESS	COMMENT	FLOAT	IS	OPTION	START	UNION
ADD	COMPRESS	FOR	LEVEL	OR	SELECT	UNIQUE
ALL	CONNECT	FROM	LIKE	ORDER	SESSION	UPDATE
ALTER	CREATE	GRANT	LOCK	PCTFREE	SET	USER
AND	CURRENT	GROUP	LONG	PRIOR	SHARE	VALIDATE
ANY	DATE	HAVING	MAXEXTENTS	PRIVILEGES	SIZE	VALUES
ARRAYLEN	DECIMAL	IDENTIFIED	MINUS	PUBLIC	SMALLINT	VARCHAR
AS	DEFAULT	IMMEDIATE	MODE	RAW	SQLBUF	VARCHAR2
ASC	DELETE	IN	NOTFOUND	RENAME	SUCCESSFUL	VIEW
AUDIT	DESC	INCREMENT	NOWAIT	RESOURCE	SYNONYM	WHENEVER
BETWEEN	DISTINCT	INDEX	NULL	REVOKE	SYSDATE	WHERE
BY	DROP	INITIAL	NUMBER	ROW	TABLE	WITH
CHAR	ELSE	INSERT	OF	ROWID	THEN	
CHECK	EXCLUSIVE	INTEGER	OFFLINE	ROWLABEL	TO	
CLUSTER	EXISTS	INTERSECT	ON	ROWNUM	TRIGGER	
COLUMN	FILE	INTO	ONLINE	ROWS	UID	

Oracle Keywords

The following words also have a special meaning to Oracle but are not reserved words and so can be redefined. However, some might eventually become reserved words.

ADMIN	COUNT	FOREIGN	MINEXTENTS	PRECISION	SQLERROR
AFTER	CURSOR	FORTRAN	MINVALUE	PRIMARY	SQLSTATE
ALLOCATE	CYCLE	FOUND	MODULE	PRIVATE	STATEMENT_ID
ANALYZE	DATABASE	FUNCTION	MOUNT	PROCEDURE	STATISTICS
ARCHIVE	DATAFILE	GO	NEXT	PROFILE	STOP
ARCHIVELOG	DBA	GOTO	NEW	QUOTA	STORAGE
AUTHORIZATION	DEC	GROUPS	NOARCHIVELOG	READ	SUM
AVG	DECLARE	INCLUDING	NOCACHE	REAL	SWITCH
BACKUP	DISABLE	INDICATOR	NOCYCLE	RECOVER	SYSTEM
BEGIN	DISMOUNT	INITRANS	NOMAXVALUE	REFERENCES	TABLES
BECOME	DOUBLE	INSTANCE	NOMINVALUE	REFERENCING	TABLESPACE
BEFORE	DUMP	INT	NONE	RESETLOGS	TEMPORARY
BLOCK	EACH	KEY	NOORDER	RESTRICTED	THREAD
BODY	ENABLE	LANGUAGE	NORESETLOGS	REUSE	TIME
CACHE	END	LAYER	NORMAL	ROLE	TRACING
CANCEL	ESCAPE	LINK	NOSORT	ROLES	TRANSACTION
CASCADE	EVENTS	LISTS	NUMERIC	ROLLBACK	TRIGGERS
CHANGE	EXCEPT	LOGFILE	OFF	SAVEPOINT	TRUNCATE
CHARACTER	EXCEPTIONS	MANAGE	OLD	SCHEMA	UNDER
CHECKPOINT	EXEC	MANUAL	ONLY	SCN	UNLIMITED

CLOSE	EXPLAIN	MAX	OPEN	SECTION	UNTIL
COBOL	EXECUTE	MAXDATAFILES	OPTIMAL	SEGMENT	USE
COMMIT	EXTENT	MAXINSTANCES	OWN	SEQUENCE	USING
COMPILE	EXTERNALLY	MAXLOGFILES	PACKAGE	SHARED	WHEN
CONSTRAINT	FETCH	MAXLOGHISTORY	PARALLEL	SNAPSHOT	WRITE
CONSTRAINTS	FLUSH	MAXLOGMEMBERS	PCTINCREASE	SOME	WORK
CONTENTS	FREELIST	MAXTRANS	PCTUSED	SORT	
CONTINUE	FREELISTS	MAXVALUE	PLAN	SQL	
CONTROLFILE	FORCE	MIN	PLI	SQLCODE	

PL/SQL Reserved Words

The following PL/SQL keywords may require special treatment when used in embedded SQL statements.

ABORT	CLUSTER	DELETE	HAVING	NULL	RESOURCE	TABLE
ACCEPT	CLUSTERS	DELTA	IDENTIFIED	NUMBER	RETURN	TABLES
ACCESS	COLAUTH	DESC	IF	NUMBER_BASE	REVERSE	TASK
ADD	COLUMNS	DIGITS	IN	OF	REVOKE	TERMINATE
ALL	COMMIT	DISPOSE	INDEX	ON	ROLLBACK	THEN
ALTER	COMPRESS	DISTINCT	INDEXES	OPEN	ROWID	TO
AND	CONNECT	DO	INDICATOR	OPTION	ROWLABEL	TRUE
ANY	CONSTANT	DROP	INSERT	OR	ROWNUM	TYPE
ARRAY	COUNT	ELSE	INTEGER	ORDER	ROWTYPE	UNION
ARRAYLEN	CRASH	ELSIF	INTERSECT	OTHERS	RUN	UNIQUE
AS	CREATE	END	INTO	OUT	SAVEPOINT	UPDATE
ASC	CURRENT	ENTRY	IS	PACKAGE	SCHEMA	USE
ASSERT	CURRVAL	EXCEPTION	LEVEL	PARTITION	SELECT	VALUES
ASSIGN	CURSOR	EXCEPTION_INIT	LIKE	PCTFREE	SEPARATE	VARCHAR
AT	DATABASE	EXISTS	LIMITED	POSITIVE	SET	VARCHAR2
AUTHORIZATION	DATA_BASE	EXIT	LOOP	PRAGMA	SIZE	VARIANCE
AVG	DATE	FALSE	MAX	PRIOR	SMALLINT	VIEW
BASE_TABLE	DBA	FETCH	MIN	PRIVATE	SPACE	VIEWS
BEGIN	DEBUGOFF	FLOAT	MINUS	PROCEDURE	SQL	WHEN
BETWEEN	DEBUGON	FOR	MLSLABEL	PUBLIC	SQLCODE	WHERE
BINARY_INTEGER	DECLARE	FORM	MOD	RAISE	SQLERRM	WHILE
BODY	DECIMAL	FROM	MODE	RANGE	START	WITH
BOOLEAN	DEFAULT	FUNCTION	NATURAL	REAL	STATEMENT	WORK
BY	CHECK	GENERIC	NEW	RECORD	STDDEV	XOR
CASE	CLOSE	GOTO	NEXTVAL	RELEASE	SUBTYPE	
CHAR	DEFINITION	GRANT	NOCOMPRESS	REMR	SUM	
CHAR_BASE	DELAY	GROUP	NOT	RENAME	TABAUTH	

Oracle Reserved Namespaces

Contains a list of namespaces that are reserved by Oracle. The initial characters of function names in Oracle libraries are restricted to the character strings in this list. Because of potential name conflicts, use function names that do not begin with these characters.

For example, the SQL*Net Transparent Network Service functions all begin with the characters "NS," so you need to avoid naming functions that begin with "NS."

Namespace	Library
O	OCI functions
S	function names from SQLLIB and system-dependent libraries
XA	external functions for XA applications only
GEN KP L NA NC ND NL NM NR NS NT NZ TTC UPI	Internal functions

SQL Operators

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

Arithmetic operators

Comparison operators

Logical operators

Operators used to negate condition.

Arithmetic operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

Comparison operators

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

Logical operators

Operator	Description

ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, et
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

AVG() - Returns the average value

COUNT() - Returns the number of rows FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value SUM() - Returns the sum

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

UCASE() - Converts a field to upper case

LCASE() - Converts a field to lower case

MID() - Extract characters from a text field

LEN() - Returns the length of a text field

ROUND() - Rounds a numeric field to the number of decimals specified NOW() - Returns the current system date and time

FORMAT() - Formats how a field is to be displayed

SQL String Functions

SQL string functions are used primarily for string manipulation. The following table details the important string functions:

Name	Description
ASCII()	Returns numeric value of left-most character
BIN()	Returns a string representation of the argument
BIT_LENGTH()	Returns length of argument in bits
CHAR_LENGTH()	Returns number of characters in argument
CHAR()	Returns the character for each integer passed
CHARACTER_LENGTH()	A synonym for CHAR_LENGTH()
CONCAT_WS()	Returns concatenate with separator
CONCAT()	Returns concatenated string
CONV()	Converts numbers between different number bases
ELT()	Returns string at index number
EXPORT_SET()	Returns a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Returns the index (position) of the first argument in the subsequent arguments
FIND_IN_SET()	Returns the index position of the first argument within the second argument
FORMAT()	Returns a number formatted to specified number of decimal places
HEX()	Returns a string representation of a hex value
INSERT()	Inserts a substring at the specified position up to the specified number of characters
INSTR()	Returns the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Returns the leftmost number of characters as specified
LENGTH()	Returns the length of a string in bytes
LOAD_FILE()	Loads the named file
LOCATE()	Returns the position of the first occurrence of substring
LOWER()	Returns the argument in lowercase
LPAD()	Returns the string argument, left-padded with the specified string
LTRIM()	Removes leading spaces
MAKE_SET()	Returns a set of comma-separated strings that have the corresponding bit in bits set
MID()	Returns a substring starting from the specified position
OCT()	Returns a string representation of the octal argument
OCTET_LENGTH()	A synonym for LENGTH()
ORD()	If the leftmost character of the argument is a multi-byte character, returns the code for that character
POSITION()	A synonym for LOCATE()
QUOTE()	Escapes the argument for use in an SQL statement
REGEXP	Pattern matching using regular expressions
REPEAT()	Repeats a string the specified number of times
REPLACE()	Replaces occurrences of a specified string
REVERSE()	Reverses the characters in a string
RIGHT()	Returns the specified rightmost number of characters
RPAD()	Appends string the specified number of times
RTRIM()	Removes trailing spaces
SOUNDEX()	Returns a soundex string
SOUNDS LIKE	Compares sounds
SPACE()	Returns a string of the specified number of spaces
STRCMP()	Compares two strings
SUBSTRING_INDEX()	Returns a substring from a string before the specified number of occurrences of the delimiter
SUBSTRING(), SUBSTR()	Returns the substring as specified
TRIM()	Removes leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Converts each pair of hexadecimal digits to a character
UPPER()	Converts to uppercase

SQL Numeric Functions

SQL numeric functions are used primarily for numeric manipulation and/or mathematical calculations. The following table details the numeric functions:

Name	Description
ABS()	Returns the absolute value of numeric expression.
ACOS()	Returns the arccosine of numeric expression. Returns NULL if the value is not in the range -1 to 1.
ASIN()	Returns the arcsine of numeric expression. Returns NULL if value is not in the range -1 to 1
ATAN()	Returns the arctangent of numeric expression.
ATAN2()	Returns the arctangent of the two variables passed to it.
BIT_AND()	Returns the bitwise AND all the bits in expression.
BIT_COUNT()	Returns the string representation of the binary value passed to it.
BIT_OR()	Returns the bitwise OR of all the bits in the passed expression.
CEIL()	Returns the smallest integer value that is not less than passed numeric expression
CEILING()	Returns the smallest integer value that is not less than passed numeric expression
CONV()	Convert numeric expression from one base to another.
COS()	Returns the cosine of passed numeric expression. The numeric expression should be expressed in radians.
COT()	Returns the cotangent of passed numeric expression.
DEGREES()	Returns numeric expression converted from radians to degrees.
EXP()	Returns the base of the natural logarithm (e) raised to the power of passed numeric expression.
FLOOR()	Returns the largest integer value that is not greater than passed numeric expression.
FORMAT()	Returns a numeric expression rounded to a number of decimal places.
GREATEST()	Returns the largest value of the input expressions.
INTERVAL()	Takes multiple expressions exp1, exp2 and exp3 so on.. and returns 0 if exp1 is less than exp2, returns 1 if exp1 is less than exp3 and so on.
LEAST()	Returns the minimum-valued input when given two or more.
LOG()	Returns the natural logarithm of the passed numeric expression.
LOG10()	Returns the base-10 logarithm of the passed numeric expression.
MOD()	Returns the remainder of one expression by diving by another expression.
OCT()	Returns the string representation of the octal value of the passed numeric expression. Returns NULL if passed value is NULL.
PI()	Returns the value of pi
POW()	Returns the value of one expression raised to the power of another expression
POWER()	Returns the value of one expression raised to the power of another expression
RADIANS()	Returns the value of passed expression converted from degrees to radians.
ROUND()	Returns numeric expression rounded to an integer. Can be used to round an expression to a number of decimal points
SIN()	Returns the sine of numeric expression given in radians.
SQRT()	Returns the non-negative square root of numeric expression.
STD()	Returns the standard deviation of the numeric expression.
STDDEV()	Returns the standard deviation of the numeric expression.
TAN()	Returns the tangent of numeric expression expressed in radians.
TRUNCATE()	Returns numeric exp1 truncated to exp2 decimal places. If exp2 is 0, then the result will have no decimal point.

LIST OF EXPERIMENTS

1. Creation of a database and writing SQL queries to retrieve information from the database.

   1.1 Data Definition Language (DDL).

   a. CREATE
   b. ALTER
   c. DROP
   d. TRUNCATE
   e. RENAME
   f. COMMENT

   1.2 Data Manipulation Language (DML)

   a. INSERT
   b. UPDATE
   c. DELETE
   d. SELECT

2. Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

3. Creation of Views, Synonyms, Sequence, Indexes, Save point.

   3.1. Implementation of Views.

   3.2. Implementation of Synonyms

   3.3. Implementation of Sequence

   3.4. Implementation of Indexes

   3.5. Implementation of Save point.

4. Creating an Employee database to set various constraints.

   (a). Primary key,
   (b).Foreign Key,`
   (c). Check,
   (d). Unique,
   (e).Null,
   (f). Not null,
   (g) . Default,
   (h). Enable Constraints,
   (i). Disable Constraints
   (j). Drop Constraints

5. Creating relationship between the databases.

   5.1 Implementation of set operations

   5.2. Implementation of Nested Queries / Subqueries

   5.2. Implementation the Join Operations

6. Study of PL/SQL block.
7. Write a PL/SQL block to satisfy some conditions by accepting input from the user.
8. Write a PL/SQL block that handles all types of exceptions.
9. Creation of Procedures.
10. Creation of database triggers and functions

    10.1 Implementation of Triggers and its application

    10.2 Implementation of Functions.

11. Mini project (Application Development using Oracle/ Mysql)

    a) Inventory Control System.

    b) Material Requirement Processing.

    c) Hospital Management System.

    d) Railway Reservation System.

    e) Personal Information System.

    f) Web Based User Identification System.

    g) Timetable Management System.

    h) Hotel Management System

Creation of a database and writing SQL queries to retrieve information from the database.

Ex: No: 01(1.1) **DATA DEFINITION LANGUAGE(DDL)**

__ : __ : __

AIM:

To execute the various Data Definition Language commands in RDBMS.

OBJECTIVE:

After completing the exercise the students can able to Understand how to create a table with list of fields, Modify a row using where clause, Drop a table, Delete the unwanted rows in a table.

DATA DEFINITION LANGUAGE

It is used to communicate with database. DDL is used to:

Create an object

Alter the structure of an object

To drop the object created.

ALGORITHM:

Step 1: Start the program

Step 2: Go to SQL.

Step 3: Enter the user name and password.

Step 4: Connect to the database.

Step 5: Type the commands for creating tables and perform various operations on the tables.

Step 6: The output is displayed.

Step 7: Stop the program

DDL COMMAND:

CREATE

ALTER

DROP

TRUNCATE

COMMENT

RENAME

CREATE TABLE

Constraints are condition for the data item to be stored into a database. There are two types of Constraints viz., Column Constraints and Table Constraints.

Tables

In relational database systems (DBS) data are represented using tables (relations). A query issued against the DBS also results in a table. A table has the following structure:

Tuple or record 1 ⟶

Tuple or record 2 ⟶

.

.

.

⟶

Tuple or record n

Column 1	Column 2		Columnn

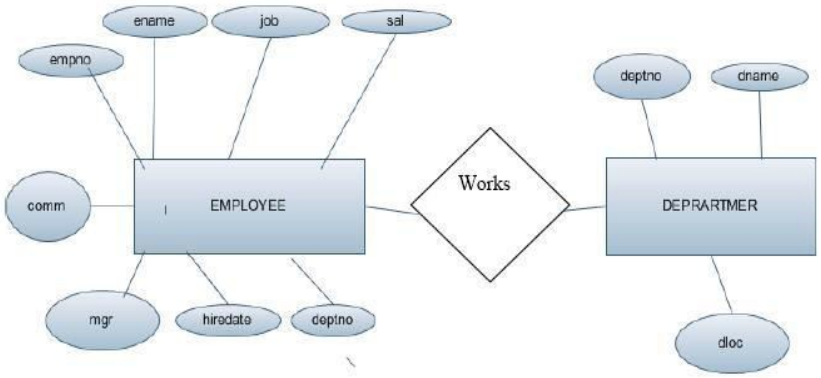
A table is uniquely identified by its name and consists of rows that contain the stored information, each row containing exactly one tuple (or record). A table can have one or more columns. A column is made up of a column name and a data type, and it describes an attribute of the tuples. The structure of a table, also called relation schema, thus is defined by its attributes. The type of information to be stored in a table is defined by the data types of the attributes at table creation time. Oracle offers the following basic data types:

Sl.No	Data Type	Description
1	**Char(n)**	Character String . n is the size of variable. Maximum size is 255 characters. The default size is 1
2	**Varchar2(n)**	Character string . n is the size of the variable
3	**Number**	Defines Numeric data type with space for 40 digit and space for sign and decimalpoint
4	**Number (n)**	Numeric variable.n is the size of the variable
5	**Number(n,d)**	Numeric variable.n is the size of variable and d is the size if the decimal point
6	**Raw(size)**	Raw Binary data of length size bytes .Maximum size is 32767 bytes.
7	**Integer**	It is same as number data type but values will be whole numbers. Columns defined with this format not accept decimal values.
8	**Integer(n)**	Specifies an integer data type of length n.
9	**Long**	Defines a character data type upto 32760bytes. One one long column may be efined for table. This type of column

		may not be used in sub queries, Where clauses or indexes.
10	**Long Raw**	Same as LONG except it contains binary data or byte strings and not interpreted by PL/SQL
11	**LOB Type**	LOB variables can used interchangeably with LONG and LONG RAW variables. It consists of BFILE,BLOB,CLOB and NLOB
12	**BFILE**	It is used to store large binary objects in operating system files outside the database
13	**BLOB**	The BLOB datat ype to store large binary objects in the database, in-line or out-of-line. Every BLOB variable stores a locator, which points to a large binary object. The size of a BLOB cannot exceed four gigabytes
14	**CLOB**	The CLOB datatype to store large blocks of character data in the database, in-line or out-of-line. Both fixed-width and variable-width character sets are supported. Every CLOB variable stores a locator, which points to a large block of character data. The size of a CLOB cannot exceed four gigabytes
15	**NCLOB**	The NCLOB datatype to store large blocks of NCHAR data in the database, in-line or out-of-line. Both fixed-width and variable-width character sets are supported. Every NCLOB variable stores a locator, which points to a large block of NCHAR data. The size of an NCLOB cannot exceed four gigabytes.
16	**DATE**	The DATE datatype to store fixed-length datetimes, which include the time of day in seconds since midnight. The date portion defaults to the first day of the current month; the time portion defaults to midnight. The date function SYSDATE returns the current date and time

SQL uses the terms table, row, and column for relation, tuple, and attribute, respectively. A table can have up to 254 columns which may have different or same data types and sets of values (domains), respectively. Possible domains are alphanumeric data (strings), numbers and date formats.

Sample Databases used for illustration of SQL Commands is given below with ER Diagram and corresponding Relational Model with suitable data entered in the tables.



Relation between Employee and Department is Works is many –to-one .

DATABASE for EMPLOYEE and DEPARTMENTEntities

EMP Table given with sample Data

EMPNOENAME	JOB	MGRHIREDATE	SALCOMM		DEPTNO
7369 SMITH	CLERK	790217-DEC-80	800		20
7499 ALLEN	SALESMAN	769820-FEB-81	1600	300	30
7521 WARD	SALESMAN	769822-FEB-81	1250	500	30
7566JONES	MANAGER	783902-APR-81	2975		20
7654 MARTIN	SALESMAN	769828-SEP-81	1250	1400	30

DEPT Table given with Sample Data

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

QUERY: 01

Q1: Write a query to create a table employee with empno, ename, designation, and salary.

Syntax: It is used to create a table

SQL: CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE),

COLUMN NAME.2 <DATATYPE> (SIZE) .....);

Command:

SQL>**CREATE TABLE** EMP (EMPNO **NUMBER (4),**ENAME **VARCHAR2 (10),** DESIGNATIN **VARCHAR2 (10),**SALARY **NUMBER (8, 2));**

Table created.

Constraints with Table Creation:

Constraints are condition for the data item to be stored into a database. There are two types of Constraints viz., Column Constraints and Table Constraints.

Syntax:

[CONSTRAINT constraint name]

{[NOT] NULL / UNIQUE / PRIMARY

KEY}(Column[,column]..) FOREIGN KEY (column [, colum]...)

REFERENCES table

[ON DELETE CASCADE]

[CHECK (condition)]

TABLE DESCRIPTION

It is used to view the table structure to confirm whether the table was created correctly.

QUERY: 02

Q2: Write a query to display the column name and data type of the table employee.

Syntax: This is used to view the structure of the table.

SQL: DESC <TABLE NAME>;

Command:

SQL> DESC EMP;

Name Null?	**Type**
---	---
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
DESIGNATIN	VARCHAR2(10)
SALARY	NUMBER(8,2)

QUERY: 03

Q3: Write a query for create a from an existing table with all the fields

Syntax: syntax for create a table from an existing table with all fields.

SQL> CREATE TABLE <TRAGET TABLE NAME> SELECT * FROM<SOURCE TABLE NAME>;

Command:

SQL> CREATE TABLE **EMP1** AS SELECT * FROM **EMP**;

Table created.

Command:

SQL> DESC EMP1

Name Null?	Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
DESIGNATIN	VARCHAR2(10)
SALARY	NUMBER(8,2)

QUERY: 04

Q4: Write a query for create a from an existing table with selected fields

Syntax: Syntax for create a from an existing table with selected fields.

SQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT EMPNO, ENAMEFROM <SOURCE TABLE NAME>;

Command:

SQL> CREATE TABLE **EMP2** AS SELECT **EMPNO**, **ENAME** FROM **EMP**;

Table created.

Command:

SQL> DESC EMP2

Name Null?	Type
EMPNO	NUMBER (4)
ENAME	VARCHAR2 (10)

QUERY: 05

Q5: Write a query for create a new table from an existing table without any record:

Syntax: The syntax for create a new table from an existing table without any record.

SQL> CREATE TABLE <TRAGET TABLE NAME> AS SELECT * FROM<SOURCE TABLE NAME>

WHERE <FALSE CONDITION>;

Command:

SQL> CREATE TABLE EMP3 AS SELECT * FROM EMP WHERE1>2;

Table created.

Command:

SQL> DESC EMP3;

Name Null?	Type
EMPNO	NUMBER(4)
ENAME	VARCHAR2(10)
DESIGNATIN	VARCHAR2(10)
SALARY	NUMBER(8,2);

ALTER & MODIFICATION ON TABLE

To modify structure of an already existing table to add one more columns and also modify the existing columns.

Alter command is used to:

1. Add a new column.
2. Modify the existing column definition.
3. To include or drop integrity constraint.

QUERY: 06

Q6: Write a Query to Alter the column EMPNO NUMBER (4) TO EMPNO NUMBER (6).

Syntax: The syntax for alter & modify on a single column.

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME><DATATYPE>(SIZE);

Command:

SQL>ALTER TABLE **EMP** MODIFY **EMPNO** NUMBER (6);

Table altered.

Command:

SQL> DESC EMP;

Name Null?	Type
EMPNO	NUMBER(6)
ENAME	VARCHAR2(10)
DESIGNATIN	VARCHAR2(10)
SALARY	NUMBER(8,2)

QUERY: 07

Q7. Write a Query to Alter the table employee with multiple columns (EMPNO,ENAME.)

Syntax: To alter table with multiple column.

SQL > ALTER <TABLE NAME> MODIFY <COLUMN NAME1><DATATYPE>(SIZE),
MODIFY <COLUMN NAME2><DATATYPE>(SIZE).....;

Command:

SQL>ALTER TABLE EMP MODIFY (EMPNO NUMBER (7),
ENAMEVARCHAR2(12)); Table altered.

Command:

```
SQL> DESC EMP;
Name Null?              Type
----------------  ------------------------------
EMPNO                   NUMBER(7)
ENAME                        VARCHAR2(12)
DESIGNATIN              VARCHAR2(10)
SALARY                  NUMBER(8,2);
```

QUERY: 08

Q8. Write a query to add a new column in to employee

Syntax: To add a new column.

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME><DATATYPE><SIZE>);

Command:

SQL> ALTER TABLE EMP ADD QUALIFICATION VARCHAR2(6);
Table altered.

```
SQL> DESC EMP;
Name Null?              Type
----------------        ------------------------
EMPNO                   NUMBER(7)
ENAME                   VARCHAR2(12)
DESIGNATIN VARCHAR2(10)
SALARY                  NUMBER(8,2)
QUALIFICATION       VARCHAR2(6)
```

QUERY: 09

Q9: Write a query to add multiple columns in to employee **Syntax:** Syntax for add a new column.

SQL> ALTER TABLE <TABLE NAME> ADD (<COLUMN NAME1><DATATYPE><SIZE>,
(<COLUMN NAME2><DATA TYPE><SIZE>…);

Command:

SQL>ALTER TABLE EMP ADD (DOB DATE, DOJ DATE);
Table altered.

```
SQL> DESC EMP;
Name Null?              Type
----------------        ------------------------------
EMPNO                   NUMBER(7)
ENAME                   VARCHAR2(12)
DESIGNATIN VARCHAR2(10)
SALARY                  NUMBER(8,2)
QUALIFICATION       VARCHAR2(6)
DOB                     DATE
DOJ                     DATE
```

REMOVE / DROP

It will delete the table structure provided the table should be empty.

QUERY: 10

Q10. Write a query to drop a column from an existing table employee

Syntax: syntax for add a new column.

SQL> ALTER TABLE <TABLE NAME> DROP COLUMN <COLUMN NAME>;

Command:

```
SQL> ALTER TABLE EMP DROP COLUMN DOJ;
Table altered.

SQL> DESC EMP;
Name Null?                    Type
----------------  ------------------------------
EMPNO                         NUMBER(7)
ENAME                         VARCHAR2(12)
DESIGNATIN                    VARCHAR2(10)
SALARY                        NUMBER(8,2)
QUALIFICATION                 VARCHAR2(6)
DOB                           DATE
```

QUERY: 11

Q10. Write a query to drop multiple columns from employee

Syntax: The Syntax for add a new column.

SQL> ALTER TABLE <TABLE NAME> DROP <COLUMNNAME1>,<COLUMN NAME2>,..... ;

Command:

```
SQL> ALTER TABLE EMP DROP (DOB, QUALIFICATION);
Table altered.

SQL> DESC EMP;
Name Null?                    Type
----------------          ---------------------------
EMPNO                         NUMBER(7)
ENAME                         VARCHAR2(12)
DESIGNATIN                    VARCHAR2(10)
SALARY                        NUMBER(8,2)
```

RENAME

QUERY: 12

Q10. Write a query to rename table emp to employee

Syntax: The Syntax for add a new column.

SQL> ALTER TABLE RENAME <OLD NAME> TO <NEW NAME>

Command:

SQL> ALTER TABLE RENAME EMP TO EMPLOYEE;

SQL> DESC EMPLOYEE;

Name Null?	Type
EMPNO	NUMBER(7)
ENAME	VARCHAR2(12)
DESIGNATIN	VARCHAR2(10)
SALARY	NUMBER(8,2)

TRUNCATE TABLE

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

Syntax:

TRUNCATE TABLE <TABLE NAME>;

Example:

Truncate table EMP;

DROP:

To remove a table along with its structure and data.

Syntax: The Syntax for add a new column.

SQL> Drop table<table name>;

Command:

SQL> drop table employee;

RESULT:

Thus the SQL commands for DDL commands in RDBMS has been verified and executed successfully.

DATA MANIPULATION LANGUAGE (DML) COMMANDS IN RDBMS

Ex: No: 1.2

__ : __ : __

AIM:

To execute and verify the DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects.

DML (DATA MANIPULATION LANGUAGE)

SELECT

INSERT

DELETE

UPDATE

ALGORITHM:

STEP 1: Start the DBMS.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert the record into table

STEP 4: Update the existing records into the table

STEP 5: Delete the records in to the table

STEP 6: use save point if any changes occur in any portion of the record to undo its original state.

STEP 7: use rollback for completely undo the records

STEP 8: use commit for permanently save the records

INSERT

The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

Insert a record from an existing table:

QUERY: 01

Q1. Write a query to insert the records in to employee.

Syntax: syntax for insert records in to a table

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....);

Command:

SQL>INSERT INTO EMP VALUES (101,'NAGARAJAN','LECTURER',15000);

1 row created.

Insert A Record Using Substitution Method:

QUERY: 03

Q3. Write a query to insert the records in to employee using substitution method.

Syntax: syntax for insert records into the table.

SQL :> INSERT INTO <TABLE NAME> VALUES< '&column name', '&column name 2', ...);

Command:

SQL> INSERT INTO EMP
VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY'); Enter value for empno: 102
Enter value for ename: SARAVANAN
Enter value for designatin: LECTURER
Enter value for salary: 15000
1 row created.
old 1: INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new 1: INSERT INTO EMP VALUES(102,'SARAVANAN','LECTURER','15000')
SQL> /

Enter value for empno: 103

Enter value for ename: PANNERSELVAM

Enter value for designatin: ASST. PROF

Enter value for salary: 20000

1 row created.

old 1: INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')

new 1: INSERT INTO EMP VALUES(103,'PANNERSELVAM','ASST.PROF','20000')

SQL> /

Enter value for empno: 104
Enter value for ename: CHINNI

Enter value for designatin: HOD,
PROF Enter value for salary: 45000

1 row created.

old 1: INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&DESIGNATIN','&SALARY')
new 1: INSERT INTO EMP VALUES(104,'CHINNI','HOD, PROF','45000')

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	15000
102	SARAVANAN	LECTURER	15000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI	HOD, PROF	45000

SELECT

SELECT Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Display the EMP table:

QUERY: 02

Q3. Write a query to display the records from employee.

Syntax: Syntax for select Records from the table.

SQL> SELECT * FROM <TABLE NAME>;

Command:

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	15000

UPDATE

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use **WHERE** clause with **UPDATE** query to update selected rows, otherwise all the rows would be affected.

QUERY: 04

Q1. Write a query to update the records from employee.

Syntax: syntax for update records from the table.

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

Command:

SQL> UPDATE EMP SET SALARY=16000 WHERE EMPNO=101;

1 row updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	LECTURER	15000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI	HOD,PROF	45000

Update Multiple Columns:

QUERY: 05

Q5. Write a query to update multiple records from employee.

Syntax: syntax for update multiple records from the table.

SQL> UPDATE <<TABLE NAME> SET <COLUMNANE>=<VALUE> WHERE <COLUMN NAME=<VALUE>;

Command:

SQL>UPDATE EMP SET SALARY = 16000, DESIGNATIN='ASST. PROF' WHERE EMPNO=102;

1 row updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
103	PANNERSELVAM	ASST. PROF	20000
104	CHINNI	HOD, PROF	45000

DELETE

The **SQL DELETE** Query is used to delete the existing records from a table. You can use **WHERE** clause with **DELETE** query to delete selected rows, otherwise all the records would be deleted.

QUERY: 06

Q5. Write a query to delete records from employee.

Syntax: Syntax for delete Records from the table:

SQL> DELETE <TABLE NAME> WHERE <COLUMN NAME>=<VALUE>;

Command:

SQL> DELETE EMP WHERE EMPNO=103;

1 row deleted.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	DESIGNATIN	SALARY
101	NAGARAJAN	LECTURER	16000
102	SARAVANAN	ASST. PROF	16000
104	CHINNI	HOD, PROF	45000

RESULT:

Thus the SQL commands for DML has been verified and executed successfully.

Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions.

Ex: No: 02

__ : __ : __

AIM:

To performing insertion, deletion, modifying, altering, updating and viewing records based on conditions.

ALGORITHM:

STEP 1: Start the DBMS

STEP 2: Connect to the database (DB)

STEP 3: Create the table with its essential attributes.

STEP 4: Insert the record into table based on some condition using WHERE CLAUSE

STEP 5: Update the existing records into the table based on some condition

STEP 6: Delete the records in to the table based on some condition

STEP 7: Use commit for permanently save the records

STEP 8: Stop the program

DRL-DATA RETRIEVAL IMPLEMENTING ON SELECT COMMANDS

Command:

```
SQL> CREATE TABLE EMP(
                        EMPNO      NUMBER (4),
                        ENAME      VARCHAR2 (10),
                        JOB        VARCHAR2(20),
                        MGR        NUMBER(4),
                        HIREDATE   DATE,
                        SAL        NUMBER(8,2),
                        DEPTNO     NUMBER(3)
                );
Table created.
```

SQL> INSERT INTO EMP VALUES(7369,'SMITH','CLERK',5001,'17-DEC-80','8000',200); 1 row created.

SQL> INSERT INTO EMP VALUES(7499,'ALLEN','SALESMAN',5002,'20-FEB-80','3000',300); 1 row created.

SQL> INSERT INTO EMP VALUES(7521,'WARD','SALESMAN',5003,'22-FEB-80','5000',500); 1 row created.

SQL> INSERT INTO EMP VALUES(7566,'JONES','MANAGER',5002,'02-APR-85','75000',200);
1 row created.

SQL> INSERT INTO EMP VALUES(7566,'RAJA','OWNER',5000,'30-APR-75',NULL,100);
1 row created.

SQL> INSERT INTO EMP VALUES(7566,'KUMAR','COE',5002,'12-JAN-87','55000',300);
1 row created.

SQL> INSERT INTO EMP VALUES(7499,'RAM KUMAR','SR.SALESMAN',5003,'22-JAN-87','12000.55',200);
1 row created.

SQL> INSERT INTO EMP VALUES(7521,'SAM KUMAR','SR.SALESMAN',5003,'22-JAN-75','22000',300); 1 row created.

THE SELECT STATEMENT SYNTAX WITH ADDITIONAL CLAUSES:

```
Select [ Distinct / Unique ] ( *columnname [ As alias}, ....]
From tablename
[ where  condition ]
[ Group BY group _by_expression ]
[Having group_condition ]
[ORDER BY {col(s)|expr|numeric_pos} [ASC|DESC] [NULLS FIRST|LAST]];
```

SQL> SELECT * FROM EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
7369	SMITH	CLERK	5001	17-DEC-80	8000	200
7499	ALLEN	SALESMAN	5002	20-FEB-80	3000	300
7521	WARD	SALESMAN	5003	22-FEB-80	5000	500
7566	JONES	MANAGER	5002	02-APR-85	75000	200
7566	RAJA	OWNER	5000	30-APR-75		100
7566	KUMAR	COE	5002	12-JAN-87	55000	300
7499	RAM KUMAR	SR.SALESMAN	5003	22-JAN-87	12000.55	200
7521	SAM KUMAR	SR.SALESMAN	5003	22-JAN-75	22000	300

8 rows selected.

BY USING SELECTED COLUNMS

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP;

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000
7499	ALLEN	SALESMAN	3000
7521	WARD	SALESMAN	5000
7566	JONES	MANAGER	75000
7566	RAJA	OWNER	
7566	KUMAR	COE	55000
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

8 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL=5000;

EMPNO	ENAME	JOB	SAL
7521	WARD	SALESMAN	5000

BY USING BETWEEN / NOT / IN / NULL / LIKE

BETWEEN Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* BETWEEN *value1* AND *value2;*

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **BETWEEN** 10000 AND 30000;

EMPNO	ENAME	JOB	SAL
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **NOT BETWEEN** 10000 **AND** 30000;

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000
7499	ALLEN	SALESMAN	3000
7521	WARD	SALESMAN	5000
7566	JONES	MANAGER	75000
7566	KUMAR	COE	55000

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...);
```

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **IN** (1000.5,75000);

EMPNO	ENAME	JOB	SAL
7566	JONES	MANAGER	75000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **NOT IN** (1000.5,75000);

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000
7499	ALLEN	SALESMAN	3000
7521	WARD	SALESMAN	5000
7566	KUMAR	COE	55000
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

6 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **IS NULL**;

EMPNO	ENAME	JOB	SAL
7566	RAJA	OWNER	

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL **IS NOT NULL**;

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000
7499	ALLEN	SALESMAN	3000
7521	WARD	SALESMAN	5000
7566	JONES	MANAGER	75000
7566	KUMAR	COE	55000
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

7 rows selected.

LIKE Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* LIKE *pattern*;

```
SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL LIKE 55000;
EMPNO       ENAME       JOB              SAL
----------  ----------  ---------------  ----------
7566        KUMAR       COE                55000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME LIKE 'S%';
EMPNO       ENAME              JOB              SAL
----------  ----------  --------------------  ----------
7369        SMITH       CLERK                 8000
7521        SAM KUMARSR.SALESMAN              22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME LIKE '%R';
EMPNO       ENAME       JOB                SAL
---------   ----------  ----------         --------------
7566        KUMAR       COE                55000
7499        RAM KUMAR SR.SALESMAN          12000.55
7521        SAM KUMARSR.SALESMAN           22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME    LIKE '%U%';
EMPNO       ENAME              JOB              SAL
----------  --------- -        ---------------  ----------
7566        KUMAR              COE              55000
7499        RAM KUMAR          SR.SALESMAN      12000.55
7521        SAM KUMAR          SR.SALESMAN      22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME    LIKE '%A%';
EMPNO       ENAME       JOB              SAL
----------  ----------  -------------    ----------
7499        ALLEN       SALESMAN         3000
7521        WARD        SALESMAN          5000
7566        RAJA        OWNER
7566        KUMAR       COE              55000
7499    RAM KUMAR    SR.SALESMAN      12000.55
7521    SAM KUMAR    SR.SALESMAN      22000
6 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME LIKE '%LL%';
EMPNO               ENAME       JOB              SAL
----------          ----------  ---------------  ------------
7499                ALLEN       SALESMAN          3000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME    LIKE '%E%';
EMPNO               ENAME       JOB              SAL
---------- --       -------- ----  -------------    ----------
7499                ALLEN     SALESMAN             3000
7566                JONES     MANAGER             75000
```

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '%U%A%';**

EMPNO	ENAME	JOB	SAL
7566	KUMAR	COE	55000
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'R___';** //**3_**

EMPNO	ENAME	JOB	SAL
7566	RAJA	OWNER	

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'R_J_';**

EMPNO	ENAME	JOB	SAL
7566	RAJAOWNER		

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '_M%';**

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '_M';**

no rows selected

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE '____R';** // **4_**

EMPNO	ENAME	JOB	SAL
7566	KUMAR	COE	55000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME **LIKE 'K___R';** // **3_**

EMPNO	ENAME	JOB	SAL
7566	KUMAR	COE	55000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE ENAME NOT **LIKE 'R_J_';**

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000
7499	ALLEN	SALESMAN	3000
7521	WARD	SALESMAN	5000
7566	JONES	MANAGER	75000
7566	KUMAR	COE	55000
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

7 rows selected.

RELATIONAL OPERATOR

```
SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL=55000;
EMPNO      ENAME      JOB                SAL
---------- ---------- ------------------ ----------
7566       KUMAR      COE                55000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL!=55000;
EMPNO      ENAME      JOB                SAL
---------- ---------- ------------------ ----------
7369       SMITH      CLERK              8000
7499       ALLEN      SALESMAN           3000
7521       WARD       SALESMAN           5000
7566       JONES      MANAGER            75000
7499       RAM KUMAR  SR.SALESMAN        12000.55
7521       SAM KUMAR  SR.SALESMAN        22000
6 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL<>55000;
EMPNO ENAMEJOB                SAL
---------- ---------- ------------------ ----------
7369 SMITH     CLERK              8000
7499 ALLEN     SALESMAN           3000
7521 WARD      SALESMAN           5000
7566 JONES     MANAGER            75000
7499 RAM KUMARSR.SALESMAN12000.55
7521 SAM KUMARSR.SALESMAN       22000
6 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL>55000;
EMPNO      ENAME      JOB                SAL
---------- ---------- ------------------ ----------
7566       JONES      MANAGER            75000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL<55000;
EMPNO      ENAME      JOB                SAL
---------- ---------- ------------------ ----------
7369       SMITH      CLERK              8000
7499       ALLEN      SALESMAN           3000
7521       WARD       SALESMAN           5000
7499       RAM KUMAR SR.SALESMAN         12000.55
7521       SAM KUMARSR.SALESMAN          22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL<=55000;
EMPNO      ENAME      JOB                SAL
---------- ---------- ------------------ ----------
7369       SMITH      CLERK              8000
7499       ALLEN      SALESMAN           3000
7521       WARD       SALESMAN           5000
7566       KUMAR      COE                55000
7499       RAM KUMAR SR.SALESMAN         12000.55
7521       SAM KUMARSR.SALESMAN          22000
6 rows selected.
SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE SAL>=55000;
EMPNO      ENAME      JOB            SAL
---------- ---------- ---------------- ---------
7566       JONES      MANAGER        75000
7566       KUMAR      COE            55000
```

AND / OR

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE JOB='SR.SALESMAN' **AND** SAL=22000;

EMPNO	ENAME	JOB	SAL
7521	SAM KUMAR	SR.SALESMAN	22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE JOB='SR.SALESMAN' **OR** SAL=22000;

EMPNO	ENAME	JOB	SAL
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP

WHERE SAL=5000 **AND** (JOB='SR.SALESMAN' **OR** JOB='SALESMAN');

EMPNO	ENAME	JOB	SAL
7521	WARD	SALESMAN	5000

ORDER BY

Syntax:

SELECT *column_name,column_name*
FROM *table_name*
ORDER BY *column_name,column_name* ASC|DESC;

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP **ORDER BY ENAME**;

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	3000
7566	JONES	MANAGER	75000
7566	KUMAR	COE	55000
7566	RAJA	OWNER	
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000
7369	SMITH	CLERK	8000
7521	WARD	SALESMAN	5000

8 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP ORDER BY ENAME **DESC;**

EMPNO	ENAME	JOB	SAL
7521	WARD	SALESMAN	5000
7369	SMITH	CLERK	8000
7521	SAM KUMAR	SR.SALESMAN	22000
7499	RAM KUMAR	SR.SALESMAN	12000.55
7566	RAJA	OWNER	
7566	KUMAR	COE	55000
7566	JONES	MANAGER	75000
7499	ALLEN	SALESMAN	3000

8 rows selected.

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP ORDER BY ENAME **ASC;**

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	3000
7566	JONES	MANAGER	75000
7566	KUMAR	COE	55000
7566	RAJA	OWNER	
7499	RAM KUMAR	SR.SALESMAN	12000.55
7521	SAM KUMAR	SR.SALESMAN	22000
7369	SMITH	CLERK	8000
7521	WARD	SALESMAN	5000

8 rows selected.

TOP

// **TOP** clause is not in oracle instead of that **ROWNUM**

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE **ROWNUM <4;**

EMPNO	ENAME	JOB	SAL
7369	SMITH	CLERK	8000
7499	ALLEN	SALESMAN	3000
7521	WARD	SALESMAN	5000

SQL> SELECT EMPNO,ENAME,JOB,SAL FROM EMP WHERE **ROWNUM <4 ORDER BY ENAME;**

EMPNO	ENAME	JOB	SAL
7499	ALLEN	SALESMAN	3000
7369	SMITH	CLERK	8000
7521	WARD	SALESMAN	5000

DISTINCT

Syntax:

```
SELECT DISTINCT column_name,column_name
FROM table_name;
```

Ex:

SQL> SELECT **DISTINCT** JOB FROM EMP;

JOB

CLERK
SALESMAN
SR.SALESMAN
MANAGER
COE
OWNER

6 rows selected.

USING ALTER

This can be used to add or remove columns and to modify the precision of the datatype.

a) ADDING COLUMN

Syntax:
alter table <*table_name*> add <*col datatype*>;

Ex:

```
SQL> DESC EMP;
 Name                      Null?    Type
 ------------------------- -------- ------------------------
 EMPNO                              NUMBER(4)
 ENAME                              VARCHAR2(10)
 JOB                                VARCHAR2(20)
 MGR                                NUMBER(4)
 HIREDATE                           DATE
 SAL                                NUMBER(8,2)
 DEPTNO                             NUMBER(3)
SQL> alter table EMP add TAX number;
Table altered.

SQL> DESC EMP;
 Name                      Null?    Type
 ----------------------------------------- -----------------------
 EMPNO                              NUMBER(4)
 ENAME                              VARCHAR2(10)
 JOB                                VARCHAR2(20)
 MGR                                NUMBER(4)
 HIREDATE                           DATE
 SAL                                NUMBER(8,2)
 DEPTNO                             NUMBER(3)
 TAX                                NUMBER
```

b) REMOVING COLUMN

Syntax:
alter table <*table_name*> drop <*col datatype*>;

Ex:

```
SQL> alter table EMP drop column TAX;
Table altered.

SQL> DESC EMP;
 Name                      Null?    Type
 ------------------------- -------- ----------------------------
 EMPNO                              NUMBER(4)
 ENAME                              VARCHAR2(10)
 JOB                                VARCHAR2(20)
 MGR                                NUMBER(4)
 HIREDATE                           DATE
 SAL                                NUMBER(8,2)
 DEPTNO                             NUMBER(3)
```

c) INCREASING OR DECREASING PRECISION OF A COLUMN

Syntax:

alter table <*table_name*> modify <*col datatype*>;

Ex:

SQL> alter table EMP modify DEPTNO number(5);
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(20)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(8,2)
DEPTNO		NUMBER(5)

* To decrease precision the column should be empty.

d) MAKING COLUMN UNUSED

Syntax:
alter table <*table_name*> set unused column <*col*>;

Ex:
SQL> alter table EMP set unused column DEPTNO;
Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(20)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(8,2)

SQL> SELECT * FROM EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
7369	SMITH	CLERK	5001	17-DEC-80	8000
7499	ALLEN	SALESMAN	5002	20-FEB-80	3000
7521	WARD	SALESMAN	5003	22-FEB-80	5000

9 rows selected.

Even though the column is unused still it will occupy memory.

d) DROPPING UNUSED
COLUMNS Syntax:
alter table <*table_name*> drop unused columns;

Ex:
SQL> alter table EMP drop unused columns;
Table altered.

* You can not drop individual unused columns of a table.

e) RENAMING
COLUMN Syntax:
alter table <*table_name*> rename column <*old_col_name*> to <*new_col_name*>;
Ex:
SQL> alter table EMP rename column SAL to SALARY;

Table altered.

SQL> DESC EMP;

Name	Null?	Type
EMPNO		NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(20)
MGR		NUMBER(4)
HIREDATE		DATE
SALARY		NUMBER(8,2)

INSERT

Method 1

GENERAL INSERT COMMAND:

```
SQL> INSERT INTO EMP(EMPNO,ENAME,JOB,MGR,HIREDATE,SALARY)
        VALUES(1111,'RAMU','SALESMAN',5063,'12-JAN-87','5643.55');
1 row created.
```

Method 2

WITHOUT SPECIFY THE COLUMNS DETAILS

SQL> INSERT INTO Emp VALUES(1111,'RAMU','SALESMAN',5063,'12-JAN-87','5643.55'); 1 row created.

Method 3

INSERTING DATA INTO SPECIFIED COLUMNS

SQL> INSERT INTO EMP(EMPNO,ENAME,JOB)
VALUES(1111,'RAMU','SALESMAN'); 1 row created.

Method 4

BY CHANGE THE ORDER OF COLUMNS

SQL> INSERT INTO EMP(salary,EMPNO,ENAME,JOB)
VALUES(35000,1111,'RAMU','SALESMAN'); 1 row created.

SQL> select * from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SALARY
7369	SMITH	CLERK	5001	17-DEC-80	8000
7499	ALLEN	SALESMAN	5002	20-FEB-80	3000
7521	WARD	SALESMAN	5003	22-FEB-80	5000
7566	JONES	MANAGER	5002	02-APR-85	75000
7566	RAJA	OWNER	5000	30-APR-75	
7566	KUMAR	COE	5002	12-JAN-87	55000
7499	RAM KUMAR	SR.SALESMAN	5003	22-JAN-87	12000.55
7521	SAM KUMAR	SR.SALESMAN	5003	22-JAN-75	22000
7521	SAM KUMAR	SR.SALESMAN	5003	22-JAN-75	22000
1111	RAMU	SALESMAN	5063	12-JAN-87	5643.55
1111	RAMU	SALESMAN	5063	12-JAN-87	5643.55
1111	RAMU	SALESMAN			
1111	RAMU	SALESMAN			35000

13 rows selected.

Method 5

INSERT WITH SELECT

Using this we can insert existing table data to another table in a single trip. But the table structure should be same.

Syntax:

Insert into <*table1*> select * from <*table2*>;

Ex:

SQL> DESC EMP

Name	Null?	Type
EMPNO		NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(20)
MGR		NUMBER(4)
HIREDATE		DATE
SALARY		NUMBER(8,2)

SQL> create table EMPLOYEE(EMP_NO,EMP_NAME,EMP_JOB,HR,HIREDATE,SALARY) as select * from EMP where 1 = 2;
Table created.

SQL> DESC EMPLOYEE

Name	Null?	Type
EMP_NO		NUMBER(4)
EMP_NAME		VARCHAR2(10)
EMP_JOB		VARCHAR2(20)
HR		NUMBER(4)
HIREDATE		DATE
SALARY		NUMBER(8,2)

SQL> SELECT * FROM
EMPLOYEE; no rows selected

SQL> insert into EMPLOYEE select * from
EMP; 13 rows created.

SQL> SELECT * FROM EMPLOYEE;

EMP_NO	EMP_NAME	EMP_JOB	HR	HIREDATE	SALARY
7369	SMITH	CLERK	5001	17-DEC-80	8000
7499	ALLEN	SALESMAN	5002	20-FEB-80	3000
7521	WARD	SALESMAN	5003	22-FEB-80	5000
7566	JONES	MANAGER	5002	02-APR-85	75000
7566	RAJA	OWNER	5000	30-APR-75	
7566	KUMAR	COE	5002	12-JAN-87	55000
7499	RAM KUMAR	SR.SALESMAN	5003	22-JAN-87	12000.55
7521	SAM KUMARSR	SALESMAN	5003	22-JAN-75	22000
7521	SAM KUMARSR	SALESMAN	5003	22-JAN-75	22000
1111	RAMU	SALESMAN	5063	12-JAN-87	5643.55
1111	RAMU	SALESMAN	5063	12-JAN-87	5643.55
1111	RAMU	SALESMAN			
1111	RAMU	SALESMAN			35000

13 rows selected.

Method 6

MULTIBLE INSERTS

We have table called DEPT with the following columns and data

SQL> select * from DEPT;

DEPTNO	DNAME	LOC
10	accounting	new york
20	research	dallas
30	sales	Chicago
40	operations	boston

CREATE STUDENT TABLE

SQL> Create table student(no number(2),name varchar(2),marks number(3));

b) MULTI INSERT WITH ALL FIELDS

```
SQL>  Insert all
      Into student values(1,'a',100)
      Into student values(2,'b',200)
      Into student values(3,'c',300)
      Select *from dept where deptno=10;
```

3 rows created.

SQL> Select * from student;

NO	NAME	MARKS
1	a	100
2	b	200
3	c	300

c) MULTI INSERT WITH SPECIFIED FIELDS

```
SQL>  insert all
        Into student (no,name) values(4,'d')
        Into student(name,marks) values('e',400)
        Into student values(3,'c',300)
        Select *from dept where
deptno=10; 3 rows created.

SQL> Select * from student;
        NO      NAME MARKS
        ------  -------- ---------
        1       a        100
        2       b        200
        3       c        300
        4       d
                e        400
        3       c        300
6 rows selected.
```

d) MULTI INSERT WITH DUPLICATE ROWS

```
SQL> insert all
        Into  student  values(1,'a',100)
        Into  student  values(2,'b',200)
        Into student values(3,'c',300)
        Select *from dept where deptno >
10; 9 rows created.

-- This inserts 9 rows because in the select statement retrieves 3 records (3 inserts for each row
retrieved) SQL> Select * from student;
NO      NAME MARKS
------  ---- --          ---------
1       a                100
2       b                200
3       c                300
4       d
        e                400
3       c                300
1       a                100
1       a                100
1       a                100
2       b                200
2       b                200
2       b                200
3       c                300
3       c                300
3       c                300
15 rows selected.
```

e) MULTI INSERT WITH CONDITIONS BASED

SQL> create table mytbl1(name varchar2(20),no number(10));
Table created.

SQL> insert into mytbl1 values('ram',111);
1 row created.

SQL> insert into mytbl1 values('sam',222);
1 row created.

SQL> insert into mytbl1 values('tam',333);
1 row created.

SQL> select * from mytbl1;

NAME	NO
ram	111
sam	222
tam	333

SQL> create table yourtbl1(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10));
Table created.

SQL> select * from
yourtbl1; **no rows selected**

SQL> select * from
yourtbl2; **no rows selected**

SQL> select * from
yourtbl3; **no rows selected**

```
SQL> insert all
        when no > 111 then
        into yourtbl1 values('ramu',1111)
        when name = 'sam' then
        into yourtbl2 values('samu',2222)
        when name = 'tam' then
        into yourtbl3 values('tamu',3333) select
        * from mytbl1 where no > 111;
```

4 rows created.
SQL> select * from mytbl1;

NAME	NO
ram	111
sam	222
tam	333

SQL> select * from yourtbl1;

NAME	NO
ramu	1111
ramu	1111

SQL> select * from yourtbl2;

NAME	NO
samu	2222

SQL> select * from yourtbl3;

NAME	NO
tamu	3333

-- This inserts 4 rows because the first condition satisfied 3 times, second condition satisfied once and the last none.

f) MULTI INSERT WITH CONDITIONS BASED AND ELSE

```
SQL> create table mytbl1(name varchar2(20),no
number(10)); Table created.

SQL> insert into mytbl1
values('ram',111); 1 row created.

SQL> insert into mytbl1
values('sam',222); 1 row created.

SQL> insert into mytbl1
values('tam',333); 1 row created.

SQL> select * from mytbl1;
NAME                 NO
-------------------- ----------
ram                  111
sam                  222
tam                  333

SQL> create table yourtbl1(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10));
Table created.

SQL> insert all
        when no > 111 then
        into yourtbl1 values('ramu',1111)
        when name = 'sam' then
        into yourtbl2 values('samu',2222)
        when name = 'tam' then
        into yourtbl3
        values('tamu',3333) else
        into yourtbl4 values('chotta',4444)
        select * from mytbl1 where no > 111;
4 rows created.

SQL> select * from yourtbl1;
NAME                 NO
-------------------- ----------
ramu                 1111
ramu                 1111

SQL> select * from yourtbl2;
NAME                 NO
-------------------- ----------
samu                 2222

SQL> select * from yourtbl3;
NAME                 NO
-------------------- ----------
tamu                 3333
```

g) MULTI INSERT WITH CONDITIONS BASED AND FIRST

```
SQL> create table mytbl1(name varchar2(20),no number(10));
Table created.

SQL> insert into mytbl1
values('ram',111); 1 row created.

SQL> insert into mytbl1
values('sam',222); 1 row created.

SQL> insert into mytbl1
values('tam',333); 1 row created.

SQL> create table yourtbl1(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10));
Table created.

SQL> select * from mytbl1;
NAME                 NO
-------------------- ----------
ram                  111
sam                  222
tam                  333

SQL> insert first
        when no=111 then
        into yourtbl1 values('ramu',1111)
        when name = 'sam' then
        into yourtbl2 values('samu',2222)
        when name = 'tam' then
        into yourtbl3 values('tamu',3333)
        select * from mytbl1 where
name='ram'; 1 row created.

SQL> select * from yourtbl1;
NAME                 NO
-------------------- ----------
ramu                 1111
-- This inserts 1 record because the first clause avoid to check the remaining conditions once the condition is satisfied.
```

h) MULTI INSERT WITH CONDITIONS BASED, FIRST AND ELSE

```
SQL> create table mytbl1(name varchar2(20),no number(10));
Table created.

SQL> insert into mytbl1
values('ram',111); 1 row created.

SQL> insert into mytbl1
values('sam',222); 1 row created.

SQL> insert into mytbl1
values('tam',333); 1 row created.

SQL> create table yourtbl1(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl4(name varchar2(20),no number(10));
Table created.

SQL> select * from mytbl1;
NAME                 NO
-------------------- ----------
ram                  111
sam                  222
tam                  333

SQL> insert first
          when no=111 then
          into yourtbl1 values('ramu',1111)
          when name = 'bam' then
          into yourtbl2 values('samu',2222)
          when name = 'tam' then
          into yourtbl3
          values('tamu',3333) else
          into yourtbl4 values('kamu',4444) select
          * from mytbl1 where name='ram';
1 row created.

SQL> select * from yourtbl1;
NAME                 NO
-------------------- ----------
ramu                 1111

SQL> select * from
yourtbl2; no rows selected

SQL> select * from
yourtbl3; no rows selected

SQL> select * from
yourtbl4; no rows selected
```

i) MULTI INSERT WITH MULTIBLE TABLES

```
SQL> create table mytbl1(name varchar2(20),no number(10));
Table created.

SQL> insert into mytbl1
values('ram',111); 1 row created.

SQL> insert into mytbl1
values('sam',222); 1 row created.

SQL> insert into mytbl1
values('tam',333); 1 row created.

SQL> select * from mytbl1;
NAME                 NO
-------------------- ----------
ram                  111
sam                  222
tam                  333

SQL> create table yourtbl1(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl2(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl3(name varchar2(20),no number(10));
Table created.

SQL> create table yourtbl4(name varchar2(20),no
number(10)); Table created.

SQL> insert all
        into  yourtbl1  values('ramu',11111)
        into  yourtbl2  values('samu',22222)
        into  yourtbl3  values('tamu',33333)
        into yourtbl4 values('kamu',44444)
        select * from mytbl1 where
name='ram'; 4 rows created.

SQL> select * from yourtbl1;
NAME                 NO
-------------------- ----------
ramu                 11111

SQL> select * from yourtbl2;
NAME                 NO
-------------------- ----------
samu                 22222

SQL> select * from yourtbl3;
NAME                 NO
-------------------- ----------
tamu                 33333

SQL> select * from yourtbl4;
NAME                 NO
-------------------- ----------
kamu                 44444
```

** You can use multi tables with specified fields, with duplicate rows, with conditions, with first and else clauses.

GROUP BY

Using group by, we can create groups of related information. Columns used in select must be used with group by; otherwise it was not a group by expression.

Ex:

```
SQL> select * from emp;
EMPNO      ENAME          JOB            MGR  HIREDATE     SAL  DEPTNO
---------- -------------- ------------------ ------- ---------- ---------- ----------
7369       SMITH          CLERK              5001 7-DEC-80         8000  200
7499       ALLEN          SALESMAN           5002 0-FEB-80         3000  300
7521       WARD           SALESMAN           5003 22-FEB-80        5000  500
7499  RAM KUMARSR.SALESMAN          5003 22-JAN-87        12000.55200
7566       JONES          MANAGER            5002 02-APR-85        75000 200
7521  SAM KUMARSR.SALESMAN          5003 22-JAN-75        22000 300
6 rows selected.

SQL> select job from EMP group by job;
JOB
--------------------
CLERK
SALESMAN
SR.SALESMAN
MANAGER

SQL> select job,SUM(SAL) from EMP group by job;
JOB                  SUM(SAL)
-----------------    ----------
CLERK                8000
SALESMAN    8000
SR.SALESMAN          34000.55
MANAGER     75000
```

HAVING

This will work as where clause which can be used only with group by because of absence of where clause in group by.

```
SQL> select deptno,job,sum(sal) Total_Salary_Of_Each_Dept
from emp group by deptno,job having sum(sal) > 3000;

   DEPTNO      JOB                TOTAL_SALARY_OF_EACH_DEPT
---------- ----  -----------------  -------------------------
      200      MANAGER             75000
      200      SR.SALESMAN        12000.55
      200      CLERK              8000
      500      SALESMAN            5000
      300      SR.SALESMAN        22000

SQL> select deptno,job,sum(sal) Total_Salary_of_Each_Dept from emp
                                        group by deptno,job
                                        having sum(sal) > 3000
                                        order by job;
DEPTNO      JOB                TOTAL_SALARY_OF_EACH_DEPT
----------  -----------------  -------------------------
200         CLERK               8000
200         MANAGER            75000
500         SALESMAN           5000
200         SR.SALESMAN        12000.55
300         SR.SALESMAN        22000
```

USING DELETE

```
SQL> select * from EMP;
EMPNO      ENAME      JOB              MGR    HIREDATE       SAL      DEPTNO
---------- ---------- ---------------- -------- ---------------- -------- ---------------
1001       RAM        CLERK            5001   17-DEC-84      8000     301
1002       SAM        MANAGER          5001   11-JAN-81      85000    301
1003       SAMU       SALESMAN         5003   09-FEB-82      8000     302
1004       RAMU       SR.SALESMAN      5002    22-JUN-85      45000    303

SQL> DELETE EMP WHERE ENAME='SAM';
1 row deleted.

SQL> select * from EMP;
EMPNO      ENAME      JOB          MGR    HIREDATE         SAL      DEPTNO
---------- ---------- ------------ -------- ---------------- ---------- ----------
1001       RAM        CLERK        5001   17-DEC-84      8000     301
1003       SAMU       SALESMAN     5003   09-FEB-82      8000     302
1004       RAMU       SR.SALESMAN  5002    22-JUN-85      45000    303

SQL> DELETE EMP WHERE ENAME LIKE 'R__';
1 row deleted.

SQL> select * from EMP;
EMPNO      ENAME      JOB          MGR    HIREDATE       SAL      DEPTNO
---------- ---------- ------------ -------- ---------------- --------- ----------
1003       SAMU       SALESMAN     5003   09-FEB-82      8000     302
1004       RAMU       SR.SALESMAN  5002    22-JUN-85      45000    303

SQL> DELETE FROM EMP WHERE
ENAME='SAMU'; 1 row deleted.
```

TO DELETE ALL RECORDS

```
SQL> DELETE FROM
EMP; 1 row deleted.
```

DELETE DUPLICATE ROWS

```
SQL> SELECT * FROM myTBL;
NAME  MARK
---------  ----------
RAM       101
RAM       101
SAM       102
SAM       102
RAMU
RAMU
SAMU     103
SAMU     103
SAMU     103
TAM
RAJA      555
KAJA      123

12 rows selected.

SQL> delete from myTBL t1
where t1.rowid > (select min(t2.rowID) from myTBL
t2 where t1.name = t2.name and t1.mark = t2.mark);
4 rows deleted.
```

SQL> SELECT * FROM myTBL;

NAME	MARK
RAM	101
SAM	102
RAMU	
SAMU	103
TAM	
RAJA	555
KAJA	123

8 rows selected.

Using UPDATE

SQL> select * from EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
1001	RAM	CLERK	5001	17-DEC-84	8000	301
1002	SAM	MANAGER	5001	11-JAN-81	85000	301
1003	SAMU	SALESMAN	5003	09-FEB-82	8000	302

SQL> UPDATE EMP SET SAL = 55555,JOB = 'SR.MANAGER' WHERE ENAME LIKE 'R__'; 1 row updated.

SQL> select * from EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
1001	RAM	SR.MANAGER	5001	17-DEC-84	55555	301
1002	SAM	MANAGER	5001	11-JAN-81	85000	301
1003	SAMU	SALESMAN	5003	09-FEB-82	8000	302

SQL> UPDATE EMP SET SAL = 55555,JOB = 'SR.MANAGER';
3 rows updated.

SQL> select * from EMP;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	DEPTNO
1001	RAM	SR.MANAGER	5001	17-DEC-84	55555	301
1002	SAM	SR.MANAGER	5001	11-JAN-81	55555	301
1003	SAMU	SR.MANAGER	5003	09-FEB-82	55555	302

RESULT:

Thus the SQL commands for Performing Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions has been verified and executed successfully.

Creation of Views, Synonyms, Sequence, Indexes, Save point.

Ex: No: 03 (3.1) **VIEWS**

__ : __ : __

AIM:

To create the view, execute and verify the various operations on views.

OBJECTIVE:

Views Helps to encapsulate complex query and make it reusable.

Provides user security on each view - it depends on your data policy security.

Using view to convert units - if you have a financial data in US currency, you can create view to convert them into Euro for viewing in Euro currency.

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following:

Structure data in a way that users or classes of users find natural or intuitive.

Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.

ALGORITHM:

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database(DB)

STEP 3: Create the table with its essential attributes.

STEP 4: Insert record values into the table.

STEP 5: Create the view from the above created table.

STEP 6: Display the data presented on the VIEW.

STEP 7: Insert the records into the VIEW,

STEP 8: Check the database object table and view the inserted values presented

STEP 9: Execute different Commands and extract information from the View.

STEP 10: Stop the DBMS.

COMMANDS EXECUTION

CREATION OF TABLE:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

```
SQL> CREATE TABLE EMPLOYEE (
                                    EMPLOYEE_NAME   VARCHAR2(10),
                                    EMPLOYEE_NO     NUMBER(8),
                                    DEPT_NAME       VARCHAR2(10),
                                    DEPT_NO         NUMBER (5),
                                    DATE_OF_JOIN    DATE
                                );
Table created.
```

TABLE DESCRIPTION:

```
SQL> DESC EMPLOYEE;
        NAME NULL?                  TYPE
        --------------------------- ----------------------
        EMPLOYEE_NAME               VARCHAR2(10)
        EMPLOYEE_NO                 NUMBER(8)
        DEPT_NAME                   VARCHAR2(10)
        DEPT_NO                     NUMBER(5)
        DATE_OF_JOIN                DATE
```

CREATE VIEW

SUNTAX FOR CREATION OF VIEW

```
CREATE [OR REPLACE] [FORCE ] VIEW viewname [(column-name, column-name)]
AS Query [with check option];
```

CREATION OF VIEW

```
SQL> CREATE VIEW EMPVIEW AS SELECT EMPLOYEE_NAME,
EMPLOYEE_NO,
DEPT_NAME,
DEPT_NO,
DATE_OF_JOIN FROM  EMPLOYEE;
View Created.
```

DESCRIPTION OF VIEW

```
SQL> DESC EMPVIEW;

NAME NULL?                  TYPE
----------------------      ----------------------------
EMPLOYEE_NAME               VARCHAR2(10)
EMPLOYEE_NO                 NUMBER(8)
DEPT_NAME                   VARCHAR2(10)
DEPT_NO                     NUMBER(5)
```

DISPLAY VIEW:

```
SQL> SELECT * FROM EMPVIEW;
EMPLOYEE_N          EMPLOYEE_NO          DEPT_NAME          DEPT_NO
-------------------- -------------------- ------------------ --------------
RAVI                124                  ECE                89
VIJAY               345                  CSE                21
RAJ                 98                   IT                 22
GIRI                100                  CSE                67
```

INSERTION OF VALUES INTO VIEW

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command. Here, we can not insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

INSERT STATEMENT SYNTAX:

```
SQL> INSERT INTO <VIEW_NAME> (COLUMN NAME1, ...) VALUES(VALUE1,....);
```

COMMAND:

```
SQL> INSERT INTO EMPVIEW VALUES ('SRI', 120,'CSE', 67,'16-NOV-1981');
        1 ROW CREATED.

SQL> SELECT * FROM EMPVIEW;
        EMPLOYEE_N          EMPLOYEE_NO          DEPT_NAMEDEPT_NO
        -------------------- -------------------- ------------------ --------------
        RAVI                124                  ECE                89
        VIJAY               345                  CSE                21
        RAJ                 98                   IT                 22
        GIRI                100                  CSE                67
        SRI                 120                  CSE                67

SQL> SELECT * FROM EMPLOYEE;
        EMPLOYEE_N          EMPLOYEE_NO          DEPT_NAMEDEPT_NO   DATE_OF_J
        -------------------- -------------------- ------------------ -------------- -----------------
        RAVI                124                  ECE                89             15-JUN-05
        VIJAY               345                  CSE                21             21-JUN-06
        RAJ                 98                   IT                 22             30-SEP-06
        GIRI                100                  CSE                67             14-NOV-81
        SRI                 120                  CSE                67             16-NOV-81
```

DELETION OF VIEW:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

DELETE STATEMENT SYNTAX:

SQL> DELETE <VIEW_NMAE>WHERE <COLUMN NMAE> ='VALUE';

Command:

SQL> DELETE FROM EMPVIEW WHERE
EMPLOYEE_NAME='SRI'; 1 row deleted.

SQL> SELECT * FROM EMPVIEW;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_NO
RAVI	124	ECE	89
VIJAY	345	CSE	21
RAJ	98	IT	22
GIRI	100	CSE	67

UPDATE STATEMENT:

A view can be updated under certain conditions:

The SELECT clause may not contain the keyword DISTINCT.
The SELECT clause may not contain summary functions.

The SELECT clause may not contain set functions.
The SELECT clause may not contain set operators.

The SELECT clause may not contain an ORDER BY clause. The FROM clause may not contain multiple tables.

The WHERE clause may not contain subqueries.

The query may not contain GROUP BY or HAVING. Calculated columns may not be updated.

All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

SYNTAX:

SQL>UPDATE <VIEW_NAME> SET< COLUMN NAME> = <COLUMN NAME> +<VIEW>
WHERE <COLUMN NAME>=VALUE;

Command:

SQL> UPDATE EMPKAVIVIEW SET EMPLOYEE_NAME='KAVI' WHERE
EMPLOYEE_NAME='RAVI'; 1 row updated.

SQL> SELECT * FROM EMPKAVIVIEW;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_NO
KAVI	124	ECE	89
VIJAY	345	CSE	21
RAJ	98	IT	22
GIRI	100	CSE	67

DROP A VIEW:

Obviously, where you have a view, you need a way to drop the view if it is no longer needed.

SYNTAX:

SQL> DROP VIEW <VIEW_NAME>

EXAMPLE

SQL>DROP VIEW
EMPVIEW; view droped

CREATE A VIEW WITH SELECTED FIELDS:

SYNTAX:

SQL>CREATE [OR REPLACE] VIEW <VIEW NAME>AS SELECT <COLUMN NAME1>.....FROM <TABLE ANME>;

EXAMPLE-2:

SQL> CREATE OR REPLACE VIEW EMPL_VIEW1 AS SELECT EMPNO, ENAME,SALARY FROM EMPL; SQL> SELECT * FROM EMPL_VIEW1;

EXAMPLE-3:

SQL> CREATE OR REPLACE VIEW EMPL_VIEW2 AS SELECT * FROM EMPL WHERE DEPTNO=10; SQL> SELECT * FROM EMPL_VIEW2;

Note:

Replace is the keyboard to avoid the error “ora_0095:name is already used by an existing abj

CHANGING THE COLUMN(S) NAME M THE VIEW DURING AS SELECT STATEMENT:

TYPE-1:

SQL> CREATE OR REPLACE VIEW EMP_TOTSAL(EID,NAME,SAL) AS SELECT EMPNO, ENAME,SALARY FROM EMPL;
View created.

EMPNO	ENAME S	ALARY
7369	SMITH	1000
7499	MARK 1050	
7565	WILL	1500
7678	JOHN	1800
7578	TOM	1500
7548	TURNER	1500

6 rows selected.
View created.

```
SQL> SELECT * FROM EMP_TOTSAL;
EMPNO       ENAME       SALARY      MGRNO       DEPTNO
----------  ----------  ----------  ----------  ----------
7578        TOM         1500        7298        10
7548        TURNER      1500        7298        10
View created.
```

TYPE-2:

```
SQL> CREATE OR REPLACE VIEW EMP_TOTSAL AS SELECT EMPNO "EID",
ENAME "NAME", SALARY "SAL" FROM EMPL;

SQL> SELECT * FROM EMP_TOTSAL;
```

EXAMPLE FOR JOIN VIEW:

TYPE-3:

```
SQL> CREATE OR REPLACE VIEW DEPT_EMP AS SELECT A.EMPNO "EID",
A.ENAME "EMPNAME",
A.DEPTNO "DNO",
B.DNAME "D_NAME",
B.LOC "D_LOC"
FROM EMPL A,DEPMT B WHERE  A.DEPTNO=B.DEPTNO;

SQL> SELECT * FROM DEPT_EMP;
EID      NAME SAL
-------- ---------- ---------- ----------
7369     SMITH            1000
7499     MARK 1050
7565     WILL             1500
7678     JOHN             1800
7578     TOM              1500
7548     TURNER           1500
6 rows selected.
View created.

EID              NAME SAL
---------        ----------         ----------
7369             SMITH               1000
7499             MARK 1050
7565             WILL               1500
7678             JOHN               1800
7578             TOM                1500
7548             TURNER             1500
6 rows selected.

View created.
```

EID	EMPNAME	DNO	D_NAME	D_LOC
7578	TOM	10	ACCOUNT	NEW YORK
7548	TURNER	10	ACCOUNT	NEW YORK
7369	SMITH	20	SALES	CHICAGO
7678	JOHN	20	SALES	CHICAGO
7499	MARK 30	RESEARCHZURICH		
7565	WILL	30	RESEARCH	ZURICH

VIEW READ ONLY AND CHECK OPTION:

READ ONLY CLAUSE:

You can create a view with read only option which enable other to only query .no DML operation can be performed to this type of a view.

EXAMPLE-4:

SQL>CREATE OR REPLACE VIEW EMP_NO_DML AS SELECT * FROM EMPL WITH READ ONLY;

WITH CHECK OPTION CLAUSE:

EXAMPLE-4:

SQL> CREATE OR REPLACE VIEW EMP_CK_OPTION AS SELECT EMPNO, ENAME, SALARY, DEPTNO FROM EMPL WHERE DEPTNO=10 WITH CHECK OPTION;

SQL> SELECT * FROM EMP_CK_OPTION;

JOIN VIEW:

EXAMPLE-5:

```
SQL> CREATE OR REPLACE VIEW DEPT_EMP_VIEW AS SELECT A.EMPNO,
                                                    A.ENAME,
                                                    A.DEPTNO,
                                                    B.DNAME,
                                                    B.LOC
                                                    FROM EMPL A, DEPMT B
                                                    WHERE A.DEPTNO=B.DEPTNO;
```

SQL> SELECT * FROM DEPT_EMP_VIEW;

View created.

EMPNO	ENAME	SALARY	DEPTNO
7578	TOM	1500	10
7548	TURNER	1500	10

View created.

EMPNO	ENAME	DEPTNO	DNAME	LOC	
7578	TOM	10	ACCOUNT	NEW YORK	
7548	TURNER	10	ACCOUNT	NEW YORK	
7369	SMITH	20	SALES	CHICAGO	
7678	JOHN	20	SALES	CHICAGO	
7499	MARK	30	RESEARCH	ZURICH	
7565	WILL	30	RESEARCH	ZURICH	

6 rows selected.

FORCE VIEW:

EXAMPLE-6:

SQL> CREATE OR REPLACE FORCE VIEW MYVIEW AS SELECT * FROM XYZ;

SQL> SELECT * FROM MYVIEW;

SQL> CREATE TABLE XYZ AS SELECT EMPNO,ENAME,SALARY,DEPTNO FROM EMPL;

SQL> SELECT * FROM XYZ;

SQL> CREATE OR REPLACE FORCE VIEW MYVIEW AS SELECT * FROM XYZ;

SQL> SELECT * FROM MYVIEW;

Warning: View created with compilation errors.

SELECT * FROM MYVIEW

*

ERROR at line 1:

ORA-04063: view "4039.MYVIEW" has errors

Table created.

EMPNO	ENAME	SALARY	DEPTNO
7369	SMITH	1000	20
7499	MARK 1050	30	
7565	WILL	1500	30
7678	JOHN	1800	20
7578	TOM	1500	10
7548	TURNER	1500	10

6 rows selected.

View created.

EMPNO	ENAME	SALARY	DEPTNO
7369	SMITH	1000	20
7499	MARK 1050	30	
7565	WILL	1500	30
7678	JOHN	1800	20
7578	TOM	1500	10
7548	TURNER	1500	10

6 rows selected

COMPILING A VIEW:

SYNTAX:

ALTER VIEW <VIEW_NAME> COMPILE;

EXAMPLE:

SQL> ALTER VIEW MYVIEW COMPILE;

RESULT:

Thus the SQL commands for View has been verified and executed successfully.

Synonyms

Ex: No: 03 (3.2)

__ : __ : __

AIM:

To create the Synonyms and verify the various operations on Synonyms

OBJECTIVE:

A **synonym** is an alias for any table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms are often used for security and convenience. For example, they can do the following:

Mask the name and owner of an object

Provide location transparency for remote objects of a distributed database Simplify SQL statements for database users

Enable restricted access similar to specialized views when exercising fine-grained access control

You can create both public and private synonyms. A **public** synonym is owned by the special user group named PUBLIC and every user in a database can access it. A **private** synonym is in the schema of a specific user who has control over its availability to others.

ALGORITHM:

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database(DB)

STEP 3: Create the table with its essential attributes.

STEP 4: Insert record values into the table.

STEP 5: Create the synonyms from the above created table or any data object.

STEP 6: Display the data presented on the synonyms.

STEP 7: Insert the records into the synonyms,

STEP 8: Check the database object table and view the inserted values presented

STEP 9: Stop the DBMS.

Example:

Syntax:

SQL>CREATE SYNONYM synonymName **FOR** object;

OR

SQL>CREATE SYNONYM **tt** for **test1;**

Dependent Oject - **tt** (SYNONYM NAME)

Referenced Object - **test1** (TABLE NAME)

USAGE:

Using emp you can perform DML operation as you have create sysnonm for table object

If employees table is dropped then status of emp will be invalid.

Local Dependencies are automatically managed by oracle server.

COMMANDS:

CREATE THE TABLE

SQL> CREATE TABLE student_table(Reg_No number(5),NAME varchar2(5),MARK number(3));

Table created.

INSERT THE VALUES INTO THE TABLE

SQL> insert into student_table
values(10001,'ram',85); 1 row created.

SQL> insert into student_table
values(10002,'sam',75); 1 row created.

SQL> insert into student_table
values(10003,'samu',95); 1 row created.

SQL> select * from STUDENT_TABLE;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95

CREATE THE SYNONYM FROM TABLE

SQL> CREATE SYNONYM STUDENT_SYNONYM FOR STUDENT_TABLE;

Synonym created.

DISPLAY THE VALUES OF TABLE BY USING THE SYSNONYM

SQL> select * from STUDENT_SYNONYM;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95

INSERT THE VALUES TO THE SYNONYM

SQL> insert into student_SYNONYM values(10006,'RAJA',80);

1 row created.

DISPLAY THE VALUES IN BOTH TABLE AND SYNONYM

SQL> select * from STUDENT_TABLE;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95
10006	RAJA	80

SQL> select * from STUDENT_SYNONYM;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95
10006	RAJA	80

YOU CAN UPDATE THE TABLE BY USING SYNONYM

SQL> UPDATE STUDENT_SYNONYM SET MARK=100 WHERE REG_NO=10006;

1 row updated.

SQL> select * from STUDENT_SYNONYM;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95
10006	RAJA	100

SQL> select * from STUDENT_TABLE;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95
10006	RAJA	100

TO DROP SYSNONYM

SQL> DROP SYNONYM STUDENT_SYNONYM;

Synonym dropped.

BUT WE CAN USE THE TABLE

SQL> select * from STUDENT_TABLE;

REG_NO	NAME	MARK
10001	ram	85
10002	sam	75
10003	samu	95
10006	RAJA	100

RESULT:

Thus the SQL commands for creation and various operation on Synonyms has been verified and executed successfully.

Sequence

Ex: No: 03 (3.3)

__ : __ : __

AIM:

To create the Sequence and verify the various operations on Sequence to get the incremented number.

OBJECTIVE:

The sequence generator provides a sequential series of numbers. The sequence generator is especially useful in multiuser environments for generating unique sequential numbers without the overhead of disk I/O or transaction locking

Sequence numbers are integers of up to 38 digits defined in the database. A sequence definition indicates general information, such as the following:

The name of the sequence

Whether the sequence ascends or descends The interval between numbers

Whether Oracle Database should cache sets of generated sequence numbers in memory

ALGORITHM:

Step 1: Start the DMBS.

Step 2: Connect to the existing database (DB)

Step 3: Create the sequence with its essential optional parameter.

Step 4: Display the data presented on the sequence by using pseudo column.

Step 5: Alter the sequence with different optional parameter.

Step 6: Drop the sequence

Step 7: Stop the DBMS.

Creating a Sequence

You create a sequence using the CREATE SEQUENCE statement, which has the following.

SYNTAX:

SQL>CREATE SEQUENCE sequence_name

[**START WITH** start_num]

[**INCREMENT BY** increment_num]

[{ **MAXVALUE** maximum_num | **NOMAXVALUE** }] [
{ **MINVALUE** minimum_num | **NOMINVALUE** }]

[{ **CYCLE** | **NOCYCLE** }]

[{ **CACHE** cache_num | **NOCACHE** }] [
{ **ORDER** | **NOORDER** }];

Where

sequence_name is the name of the sequence.

start_num is the integer to start the sequence. The default start number is 1.

increment_num is the integer to increment the sequence by. The default increment number is 1. The absolute value of *increment_num* must be less than the difference between *maximum_num* and *minimum_num*.

maximum_num is the maximum integer of the sequence; *maximum_num* must be greater than or equal to *start_num*, and *maximum_num* must be greater than *minimum_num*.

NOMAXVALUE specifies the maximum is 1027 for an ascending sequence or –1 for a descending sequence. NOMAXVALUE is the default.

minimum_num is the minimum integer of the sequence; *minimum_num* must be less than or equal to *start_num*, and *minimum_num* must be less than *maximum_num*.

NOMINVALUE specifies the minimum is 1 for an ascending sequence or –1026 for a descending sequence. NOMINVALUE is the default.

CYCLE means the sequence generates integers even after reaching its maximum or minimum value. When an ascending sequence reaches its maximum value, the next value generated is the minimum. When a descending sequence reaches its minimum value, the next value generated is the maximum.

NOCYCLE means the sequence cannot generate any more integers after reaching its maximum or minimum value. NOCYCLE is the default.

cache_num is the number of integers to keep in memory. The default number of integers to cache is 20. The minimum number of integers that may be cached is 2. The maximum integers that may be cached is determined by the formula CEIL(*maximum_num - minimum_num*)/ABS(*increment_num*).

NOCACHE means no caching. This stops the database from pre-allocating values for the sequence, which prevents numeric gaps in the sequence but reduces performance. Gaps occur because cached values are lost when the database is shut down. If you omit CACHE and NOCACHE, the database caches 20 sequence numbers by default.

ORDER guarantees the integers are generated in the order of the request. You typically use ORDER when using Real Application Clusters, which are set up and managed by database administrators.

NOORDER doesn't guarantee the integers are generated in the order of the request. NOORDER is the default.

Example: 1

Command:

```
SQL> CREATE SEQUENCE seq1
INCREMENT BY 1
START with 1
MAXVALUE 5
MINVALUE 0;
```

Sequence created.

TO DISPLAY THE VALUES OF SEQUENCES

After creating sequence use nextval as nextval is used to generate sequence values SQL> select seq1.nextval from dual;

```
NEXTVAL
----------
       1
```

SQL> select seq1.nextval from dual;

```
NEXTVAL
----------
       2
```

SQL> select seq1.nextval from dual;

```
NEXTVAL
----------
       3
```

SQL> select seq1.currval from dual;

```
CURRVAL
----------
       3
```

The following is the list of available pseudo columns in Oracle.

Pseudo Column		Meaning
CURRVAL	-	Returns the current value of a sequence.
NEXTVAL	-	Returns the next value of a sequence.
NULL	-	Return a null value.
ROWID	-	Returns the ROWID of a row. See ROWID section below.
ROWNUM	-	Returns the number indicating in which order Oracle selects rows. First row selected will be ROWNUM of 1 and second row ROWNUM of 2 and so on.
SYSDATE	-	Returns current date and time.
USER	-	Returns the name of the current user.
UID	-	Returns the unique number assigned to the current user.

TO ALTER THE SEQUENCES

```
alter SEQUENCE
seq1 maxvalue 25
INCREMENT BY
2 cycle
cache 2
drop SEQUENCE seq1;
```

EXAMPLE: 2

```
CREATE SEQUENCE seq2
INCREMENT BY
1 start with 1
maxvalue 5
minvalue 0
cycle
CACHE 4;
```

EXAMPLE: 3

CREATE SEQUENCE seq3
INCREMENT BY -
1 start with 2
maxvalue 5
minvalue 0;

EXAMPLE: 4

CREATE SEQUENCE seq3
INCREMENT BY -
1 start with 2
maxvalue 5
minvalue 0
cycle
cache 4;

EXAMPLE: 5

CREATE SEQUENCE seq1
INCREMENT BY
1 start with 1
maxvalue 10
minvalue 0;

EXAMPLE: 6

create table test1(a number primary key);

TO INSERT THE VALUES FROM SEQUENCES TO TABLE:

insert into test1 values(seq1.nextval)

TO DROP SEQUENCES

drop sequence sequenceNAme

RESULT:

Thus the SQL commands for creation and various operations on Sequence has been verified and executed successfully.

Indexes

Ex: No: 03 (3.4)

__ : __ : __

AIM:

To create the Index for the table data and verify the various operations on Index.

ALGORITHM:

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database (DB)

STEP 3: Create the table with its essential attributes.

STEP 4: Insert record values into the table.

STEP 5: Create the Index from the above created table or any data object.

STEP 6: Display the data presented on the Index.

STEP 7: Stop the DBMS.

Index

The indexes are special objects which built on top of tables. The indexes can do operation like SELECT , DELETE and UPDATE statement faster to manipulate a large amount of data. An INDEX can also be called a table and it has a data structure. An INDEX is created on columns of a table. One table may contain one or more INDEX tables

The SQL INDEX does the following:

INDEXES can locate information within a database very fast.

An INDEX makes a catalog of rows of a database table as row can be pointed within a fraction of time with a minimum effort.

A table INDEX is a database structure which arranges the values of one or more columns in a specific order.

The performance of an INDEX can not be recognized much when dealing with relatively small tables.

INDEX can work properly and quickly for the columns that have many different values.

It takes a long time to find an information for one or combination of columns from a table when there are thousands of records in the table. In that case if indexes are created on that columns, which are accessed frequently, the information can be retrieved quickly.

The INDEX first sorts the data and then it assigns an identification for each row.

The INDEX table having only two columns, one is a rowid and another is indexed-column (ordered). When data is retrieved from a database table based on the indexed-column, the index pointer

searches the rowid and quickly locates that position.in the actual table and display the rows sought for.

How it differ from view

An INDEX is also a table. So it has a data structure.

INDEXES are pointers that represents the physical address of a data. An INDEX is created on columns of a table.

An INDEX makes a catalog based on one or more columns of a table. One table may contain one or more INDEX tables.

An INDEX can be created on a single column or combination of columns of a database table.

Types of Indexes:

1. Simple Index
2. Composite Index

Command

SAMPLE TABLE:

SQL> SELECT * FROM STUDENT_TBL;

SL_NO	REG_NO	NAME	SEX	DOB	TOTAL_PERCENTAGE	MOBILE_NO	ADDRESS	BLOOD
1	10001	RAM	M	11-DEC-85	75	9756435789	PLOT.No:30/6 ABC	B+
2	10002	RAJA	M	16-JAN-74	87.5	9456435458	ABC Nager	O+
3	10003	NIRMALA	F	22-FEB-87	95.5	9461135411	SAKTHI Nager	A+
4	10004	Anitha	F	05-MAR-87	64.3	9461135555	ANNA Nager	AB+

Simple Index:

When index is created on one column it is called as simple index.

Syntax:

CREATE INDEX <INDEX_NAME> ON <TABLE_NAME> (COL_NAME);

Ex:

SQL> create index myIndex on student_tbl(name);
Index created.

***notes**

Index should be created on columns which we regularly use in the where clause.

When a index is created a separate structure is created with first column is ROWID and second column as indexed column.

The Rows in the index will be arranged in the ascending order of indexed column.

Composite Index:

when Index is created multiple columns it is called composite index.

Ex:

SQL> create index myCompIndex on

student_tbl(DOB,ADDRESS); Index created.

The above index **myCompIndex** is used only when both the columns are used in the where clause.

Disadvantages of index:

Index will consume memory.

The performance of DML command will be decreased.

Index can also be categorized two types:

1. Unique index
2. Non-unique index

Unique Index:

If the indexed column contains unique value it is called unique index.

A unique index is automatically created. When we create a table with primary key constraint or unique constraint.

Cmd

SQL> create unique index myIndex on student_tbl(name);

Non-unique index:

If an index column contains duplicated values they are called as non-unique index.

See to index tables:

SQL> Select index_name from user_indexes;

```
INDEX_NAME
----------
MYCOMPINDEX
MYINDEX
SYS_C0011164
```

Query to see list of all the indexes.

SQL> Select INDEX_NAME,TABLE_NAME from user_indexes;

```
INDEX_NAME                     TABLE_NAME
------------------------------ ------------------------------
SYS_C0011164                   TBL_PKEY
MYCOMPINDEX                    STUDENT_TBL
MYINDEX                        STUDENT_TBL
```

Query to see list of all the indexes along with column name.

SQL> Select index_name, table_name, column_name from user_ind_columns;

```
INDEX_NAME                     TABLE_NAME                     COLUMN_NAME
------------------------------ ------------------------------ ------------------------
MYCOMPINDEX                    STUDENT_TBL                    ADDRESS
MYCOMPINDEX                    STUDENT_TBL                    DOB
MYINDEX                        STUDENT_TBL                    NAME
```

.

SQL> Desc user_indexes;

SQL> Desc user_ind_columns;

Function based index:

When index is created by using functions it is called as function based index.

Ex:

SQL> CREATE INDEX myFuncIndex ON STUDENT_TBL(lower(name));

Index created.

To drop on index:

Ex:

SQL> drop index index_of_student_tbl;

RESULT:

Thus the SQL commands for creation and various operations on Indexes has been verified and executed successfully.

SAVE POINT

Ex: No: 03 (3.5)

__ : __ : __

AIM:

To create the SAVE POINT for the transaction and verify the various operations of TCL commands.

OBJECTIVE:

The **SAVEPOINT** statement names and marks the current point in the processing of a transaction. With the **ROLLBACK TO** statement, savepoints undo parts of a transaction instead of the whole transaction.

An implicit savepoint is marked before executing an **INSERT, UPDATE, or DELETE** statement. If the statement fails, a rollback to the implicit savepoint is done. Normally, just the failed SQL statement is rolled back, not the whole transaction; if the statement raises an unhandled exception, the host environment

ALGORITHM:

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database (DB)

STEP 3: Create the table with its essential attributes.

STEP 4: Insert record values into the table or perform any kind of DML operation.

STEP 5: Create the **SAVE POINT**s for some set of statement on the transaction of database object.

STEP 6: Use the **COMMIT** command to save the effect of the previous command operation except DDL command

STEP 7: Use the **ROLLBACK TO SP_LABLE / ROLLBACK** command for restore the database status up to the save point

STEP 8: Check the status of the database.

STEP 9: Stop the DBMS.

Syntax:

SAVEPOINT<SAVEPOINT_NAME>;

Ex:

```
SQL> create table ORDER_PROCESSING(
                                    Order_ID number(3),
                                    Product_ID varchar2(10),
                                    Quantity number(3,2),
                                    Price number(4,2)
                                    );
```

Table created.

SQL> insert into ORDER_PROCESSING values(101,'RICE-22','6.5','30.50');

1 row created.

SQL> insert into ORDER_PROCESSING values(102,'OIL','2.0','90.50');

1 row created.

SQL> SELECT * FROM ORDER_PROCESSING;

ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
101	RICE-22	6.5	30.5
102	OIL	2	90.5

SQL> **COMMIT;**

Commit complete.

SQL> insert into ORDER_PROCESSING values(103,'BAGS','2','95');

1 row created.

SQL> insert into ORDER_PROCESSING values(104,'WATER BOTS','2','20');

1 row created.

SQL> **SAVEPOINT A;**

Savepoint created.

SQL> insert into ORDER_PROCESSING values(105,'EGG','8','40.50');

1 row created.

SQL> insert into ORDER_PROCESSING values(106,'SHAMPOO','1','75.50');

1 row created.

SQL> **SAVEPOINT B;**

Savepoint created.

SQL> insert into ORDER_PROCESSING values(107,'BAR SOAP','1','45.50');

1 row created.

SQL> insert into ORDER_PROCESSING values(108,'TONER','1','75.50');

1 row created.

SQL> **SAVEPOINT C;**

Savepoint created.

SQL> insert into ORDER_PROCESSING values(109,'SUGAR','2.0','60.50');

1 row created.

SQL> SELECT * FROM ORDER_PROCESSING;

ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
101	RICE-22	6.5	30.5
102	OIL	2	90.5
103	BAGS	2	95
104	WATER BOTS	2	20
105	EGG	8	40.5
106	SHAMPOO	1	75.5
107	BAR SOAP	1	45.5
108	TONER	1	75.5
109	SUGAR	2	60.5

9 rows selected.

SQL> ROLLBACK TO B;

Rollback complete.

SQL> SELECT * FROM ORDER_PROCESSING;

ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
101	RICE-22	6.5	30.5
102	OIL	2	90.5
103	BAGS	2	95
104	WATER BOTS	2	20
105	EGG	8	40.5
106	SHAMPOO	1	75.5

6 rows selected.

SQL> ROLLBACK TO A;

Rollback complete.

SQL> SELECT * FROM ORDER_PROCESSING;

ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
101	RICE-22	6.5	30.5
102	OIL	2	90.5
103	BAGS	2	95
104	WATER BOTS	2	20

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT * FROM ORDER_PROCESSING;

ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
101	RICE-22	6.5	30.5
102	OIL	2	90.5

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT * FROM ORDER_PROCESSING;

ORDER_ID	PRODUCT_ID	QUANTITY	PRICE
101	RICE-22	6.5	30.5
102	OIL	2	90.5

RESULT:

Thus the SQL commands for creation and various operations on transaction (TCL COMMAND) save point has been verified and executed successfully

Creating an Employee database to set various constraints in RDBMS

Ex: No: 04

__ : __ : __

AIM:

At the end of this exercise students are able

To differentiate between self referential constraints and foreign key constraint.

To refer a field of a given table or another table by using foreign key.

To apply check constraint & default constraint in an effective manner.

ALGORITHM:

STEP 1: Start the DMBS.

STEP 2: Connect to the existing database (DB)

STEP 3: Create the table with its essential constraint.

STEP 4: Insert record values into the table and then check the constraint.

STEP 5: disable the constraints and insert the values into the table.

STEP 6: if you want to re-enable the constraint then enable you can do.

STEP 7: Stop the DBMS.

CONSTRAINTS

Constraints are part of the table definition that limits and restriction on the value entered into its columns.

INTEGRITY CONSTRAINT

An integrity constraint is a mechanism used by oracle to prevent invalid data entry into the table. It has enforcing the rules for the columns in a table.

The types of the integrity constraints are:

a) Domain Integrity

b) Entity Integrity

c) Referential Integrity

TYPES OF CONSTRAINTS:

1) Primary key

2) Foreign key/references

3) Check

4) Unique

5) Not null

6) Null

7) Default

CONSTRAINTS CAN BE CREATED IN THREE WAYS:

1) Column level constraints

2) Table level constraints

3) Using DDL statements-alter table command

OPERATION ON CONSTRAINT:

i) ENABLE

ii) DISABLE

iii) DROP

PRIMARY KEY CONSTRAINTS

A primary key avoids duplication of rows and does not allow null values. It can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null. A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

Column level constraints using primary key:

QUERY: 13

Q13. Write a query to create primary constraints with column level **Syntax:** Column level constraints using primary key.

SQL> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE)<TYPE OF CONSTRAINTS>, COLUMN NAME.1 <DATATYPE> (SIZE) .....);

Command:

```
SQL> CREATE TABLE TBL_PKEY(
                                    RegNo NUMBER(5) PRIMARY
                                    KEY, Name VARCHAR2(20),
                                    ANY_SUB_MARK NUMBER(3)
                                  );
```

Table created.

```
SQL> insert into result values(10001,'raju',75);
1 row created.
SQL> insert into result values(10002,'KAMAL;',100);
1 row created.
SQL> insert into result values(0,'RAVI;',75);
1 row created.
SQL> insert into result values(NULL,'KAVI',65);
1 row created.
SQL> insert into TBL_PKEY values(10001,'raju',75);
1 row created.
SQL> insert into TBL_PKEY values(10002,'raj',85);
1 row created.
SQL> insert into TBL_PKEY values(0,'Kaj',22);
1 row created.
SQL> insert into TBL_PKEY values(NULL,'Kaj',22);
insert into TBL_PKEY values(NULL,'Kaj',22)
                    *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SENTHIL"."TBL_PKEY"."REGNO")
SQL> insert into TBL_PKEY values(10002,'RAJAN',95);
insert into TBL_PKEY values(10002,'RAJAN',95)
*
ERROR at line 1:
ORA-00001: unique constraint (SENTHIL.SYS_C0011650) violated
SQL> insert into TBL_PKEY values(10003,'RAJA',85);
1 row created.

SQL> select * FROM TBL_PKEY;
REGNO          NAME            ANY_SUB_MARK
-----------    -------------   ----------------------------
10001          raju            75
10002          raj             85
0              Kaj              22
10003          RAJA             85
```

Column level constraints using primary key with naming convention:

QUERY: 14

Q14. Write a query to create primary constraints with column level with naming convention

Syntax: syntax for column level constraints using primary key.

SQL >CREATE <OBJ.TYPE><OBJ.NAME> (
COL NAME.1 <DATATYPE> (SIZE)CONSTRAINTS <NAME OF CONSTRAINTS><TYPE OF CONSTRAINTS>, COL NAME.2 <DATATYPE> (SIZE)……);

Command:

SQL>CREATE TABLE EMPLOYEE (
EMPNO NUMBER (4) **CONSTRAINT EMP_EMPNO_PK PRIMARY KEY,**
ENAMEVARCHAR2 (10),JOB VARCHAR2 (6),SAL NUMBER (5),DEPTNO NUMBER (7));

Table level primary key constraints:

QUERY: 15

Q15. Write a query to create primary constraints with table level with naming convention

Syntax: The syntax for table level constraints using primary key

SQL: >CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>
(SIZE), COLUMN NAME.1 <DATATYPE> (SIZE),

CONSTRAINTS <NAME OF THE CONSTRAINTS><TYPE OF THE CONSTRAINTS>);

Command:

SQL>CREATE TABLE EMPLOYEE (EMPNO NUMBER(6),ENAME VARCHAR2(20),JOB
VARCHAR2(6), SAL NUMBER(7), DEPTNO NUMBER(5),

CONSTRAINT EMP_EMPNO_PK PRIMARY KEY(EMPNO));

Table level constraint with alter command (primary key):

QUERY: 16

Q16. Write a query to create primary constraints with alter command

Syntax: The syntax for column level constraints using primary key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),
COLUMN NAME.1 <DATATYPE> (SIZE));

[OR]

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINTS <NAME OF THECONSTRAINTS> <TYPE OF

THE CONSTRAINTS><COLUMN NAME>);

Command:

SQL>CREATE TABLE EMPLOYEE(EMPNO NUMBER(5),ENAME VARCHAR2(6),JOB VARCHAR2(6),
SAL NUMBER(6),DEPTNO NUMBER(6));

SQL>ALTER TABLE EMP3 ADD CONSTRAINT **EMP3_EMPNO_PK PRIMARYKEY (EMPNO);**

REFERENCE /FOREIGN KEY CONSTRAINT

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

Foreign key

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

Referenced key

It is a unique or primary key upon which is defined on a column belonging to the parent table.

Column level foreign key constraint

QUERY: 17

Q.17. Write a query to create foreign key constraints with column level

Parent Table:

Syntax: Syntax for Column level constraints Using Primary key

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (
COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS> ,
COLUMN NAME.1 <DATATYPE> (SIZE).....);

Command:

```
SQL> CREATE TABLE DEPT(
DEPTNO NUMBER(3) PRIMARY KEY,
DNAME VARCHAR2(20),LOCATION VARCHAR2(15));
```

Table created.

```
SQL> desc DEPT;
 Name            Null?           Type
 --------------- --------------- --------------------
 DEPTNO          NOT NULL        NUMBER(3)
 DNAME                           VARCHAR2(20)
 LOCATION                        VARCHAR2(15)
```

```
SQL> select * from DEPT;
DEPTNO      DNAME           LOCATION
----------- --------------- ---------------
101         kamal           chennai
102         rajini          madurai
103         Ajith           kovai
```

Child Table:

Syntax: The syntax for column level constraints using foreign key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),
COLUMN NAME2 <DATATYPE> (SIZE) REFERENCES <TABLE NAME>(COLUMN NAME> ....);

Command:

```
SQL> CREATE TABLE EMPL(EMPNO NUMBER(4),
DEPTNO NUMBER(3) REFERENCES DEPT(DEPTNO),
DESIGN VARCHAR2(10));
```

Table created.

```
SQL> desc EMPL;
Name                      Null?          Type
------------------------- -------------  --------------------------
EMPNO                                    NUMBER(4)
DEPTNO                                   NUMBER(3)
DESIGN                                   VARCHAR2(10)

SQL> insert into EMPL
values(5001,101,'RAJA'); 1 row created.

SQL> insert into EMPL
values(5003,103,'KAJA'); 1 row created.

SQL> insert into EMPL values(5006,104,'RAMYA');
insert into EMPL values(5006,104,'RAMYA')
*
ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.SYS_C0011294) violated - parent key
not found

SQL> select * from EMPL;
EMPNO       DEPTNO       DESIGN
----------  -----------  -------------
5001        101          RAJA
5003        103          KAJA
```

Column level foreign key constraint with naming conversions

QUERY: 18

Q.18. Write a query to create foreign key constraints with column level

Parent Table:

Syntax: The syntax for column level constraints using primary key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS>,COLUMN NAME.1 <DATATYPE> (SIZE)…);

Child Table:

Syntax: syntax for column level constraints using foreign key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) , COLUMN NAME2 <DATATYPE> (SIZE) **CONSTRAINT <CONST.NAME>**REFERENCES <TABLE NAME> (COLUMN NAME>…);

Command:

SQL>CREATE TABLE DEPT (DEPTNO NUMBER (2) **PRIMARYKEY**,

DNAME VARCHAR2 (20), LOCATION VARCHAR2 (15));

SQL>CREATE TABLE EMP4A (EMPNO NUMBER (3),

DEPTNO NUMBER (2) **CONSTRAINT EMP4A_DEPTNO_FK REFERENCES DEPT (DEPTNO),**

DESIGN VARCHAR2 (10));

Table level foreign key constraints:

QUERY: 19

Q.19. Write a query to create foreign key constraints with Table level.

Parent Table:

Syntax:

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE) <TYPE OF CONSTRAINTS>, COLUMN NAME.1 <DATATYPE> (SIZE)…);

Child Table:

Syntax: The syntax for table level constraints using foreign key.

SQL :> CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1
<DATATYPE>(SIZE), COLUMN NAME2 <DATATYPE> (SIZE),
CONSTRAINT <CONST.NAME>REFERENCES <TABLE NAME> (COLUMN NAME>);

Command:

SQL>CREATE TABLE DEPT(DEPTNO NUMBER(2) **PRIMARY KEY**,
DNAME VARCHAR2(20),LOCATION VARCHAR2(15));

SQL>CREATE TABLE EMP5(EMPNO NUMBER(3),DEPTNO NUMBER(2),
DESIGN VARCHAR2(10) **CONSTRAINT ENP2_DEPTNO_FK FOREIGNKEY(DEPT NO) REFERENCES DEPT(DEPTNO));**

Table level foreign key constraints with alter command:

QUERY:20

Q.20. Write a query to create foreign key constraints with Table level with altercommand.

Parent Table:

Syntax:

SQL :>CREATE<OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)<TYPE OF CONSTRAINTS> , COLUMN NAME.1 <DATATYPE> (SIZE) ……);

Child Table:

Syntax: The syntax for table level constraints using foreign key.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE),
COLUMN NAME2 <DATATYPE> (SIZE));

Syntax:

SQL> ALTER TABLE <TABLE NAME> ADD CONSTRAINT <CONST. NAME>REFERENCES <TABLE NAME> (COLUMN NAME>);

Command:

SQL>CREATE TABLE DEPT(DEPTNO NUMBER(2) PRIMARY KEY, DNAME VARCHAR2(20),
LOCATION VARCHAR2 (15));

SQL>CREATE TABLE EMP5 (EMPNO NUMBER(3), DEPTNO NUMBER (2), DESIGN VARCHAR2 (10));

SQL>ALTER TABLE EMP6 ADD CONSTRAINT EMP6_DEPTNO_FK
FOREIGNKEY(DEPTNO)REFERENCES DEPT(DEPTNO);

CHECK CONSTRAINT

Check constraint can be defined to allow only a particular range of values .when the manipulation violates this constraint, the record will be rejected. Check condition cannot contain sub queries.

Column level checks constraint:

QUERY: 21

Q.21. Write a query to create Check constraints with column level

Syntax: syntax for column level constraints using check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1 <DATATYPE>(SIZE)

CONSTRAINT <CONSTRAINTS NAME><TYPE OF CONSTRAINTS>(CONSTRAITNS CRITERIA)
, COLUMN NAME2 <DATATYPE> (SIZE));

Command:

SQL>CREATE TABLE EMP7(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15),

SAL NUMBER(5)**CONSTRAINT** EMP7_SAL_CK CHECK(SAL>500 ANDSAL<10001),

DEPTNO NUMBER(2));

Table Level Check Constraint:

QUERY: 22

Q.22. Write a query to create Check constraints with table level

Syntax: Syntax for Table level constraints using Check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1
<DATATYPE>(SIZE), (COLUMN NAME2 <DATATYPE> (SIZE),
CONSTRAINT<CONSTRAINTS NAME><TYPE OF CONSTRAINTS> (CONSTRAITNSCRITERIA));

Command:

SQL>CREATE TABLE EMP8(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN
VARCHAR2(15), SAL NUMBER(5),DEPTNO NUMBER(2),
CONSTRAINTS EMP8_SAL_CK CHECK(SAL>500 **AND**SAL<10001));

Check Constraint with Alter Command:

QUERY:23

Q.23. Write a query to create Check constraints with table level using alter command.

Syntax: Syntax for Table level constraints using Check.

SQL:>CREATE <OBJ.TYPE><OBJ.NAME> (COLUMN NAME.1
<DATATYPE>(SIZE), (COLUMN NAME2 <DATATYPE> (SIZE),
CONSTRAINT<CONSTRAINTS NAME><TYPE OF CONSTRAINTS> (CONSTRAITNSCRITERIA)) ;

Command:

SQL>CREATE TABLE EMP9(EMPNO NUMBER,ENAME VARCHAR2(20),DESIGN
VARCHAR2(15), SAL NUMBER(5));
SQL>ALTER TABLE EMP9 ADD **CONSTRAINTS** EMP9_SAL_CK**CHECK**(SAL>500 AND SAL<10001);

UNIQUE CONSTRAINT

It is used to ensure that information in the column for each record is unique, as with telephone or drivers license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

Column Level Constraint:

QUERY:24

Q.24. Write a query to create unique constraints with column level

Syntax: syntax for column level constraints with unique.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1> <DATATYPE> (SIZE) CONSTRAINT <NAME OF CONSTRAINTS><CONSTRAINT TYPE>, (COLUMN NAME2 <DATATYPE> (SIZE));

Command:

SQL>CREATE TABLE EMP10(EMPNO NUMBER(3),ENAME VARCHAR2(20), DESGIN VARCHAR2 (15)**CONSTRAINT** EMP10_DESIGN_UK UNIQUE, SAL NUMBER (5));

Table Level Constraint:

QUERY: 25

Q.25. Write a query to create unique constraints with table level **Syntax:** syntax for table level constraints with unique.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1><DATATYPE> (SIZE), (COLUMN NAME2 <DATATYPE> (SIZE), CONSTRAINT<NAME OF CONSTRAINTS><CONSTRAINT TYPE>(COLUMN NAME);) ;

Command:

SQL>CREATE TABLE EMP11(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5), CONSTRAINT EMP11_DESIGN_UK UNIGUE(DESIGN));

Table Level Constraint Alter Command:

QUERY:26

Q.26. Write a query to create unique constraints with table level

Syntax: syntax for table level constraints with check using alter.

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (<COLUMN NAME.1><DATATYPE> (SIZE), (<COLUMN NAME.2><DATATYPE> (SIZE)) ;

SQL> ALTER TABLE ADD <CONSTRAINTS><CONSTRAINTS NAME><CONSTRAINTS TYPE> (COLUMN NAME);

Command:

SQL>CREATE TABLE EMP12(EMPNO NUMBER(3),ENAME VARCHAR2(20),DESIGN VARCHAR2(15), SAL NUMBER(5));
SQL>ALTER TABLE EMP12 ADD CONSTRAINT EMP12_DESIGN_UKUNIQUE(DESING);

NOT NULL CONSTRAINTS

While creating tables, by default the rows can have null value .the enforcement of not null constraint in a table ensure that the table contains values.

Column Level Constraint:

QUERY: 27

Q.27. Write a query to create Not Null constraints with column level
Syntax: syntax for column level constraints with not null

SQL :> CREATE <OBJ.TYPE><OBJ.NAME>(<COLUMN NAME.1><DATATYPE> (SIZE) CONSTRAINT <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>,
(COLUMN NAME2 <DATATYPE> (SIZE)) ;

Command:

SQL>CREATE TABLE EMP13(EMPNO NUMBER(4),
ENAME VARCHAR2(20) CONSTRAINT EMP13_ENAME_NN NOT NULL,
DESIGN VARCHAR2(20),SAL NUMBER(3));

NULL CONSTRAINTS

Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.

A null value is not equivalent to a value of zero.
A null value will always evaluate to null in any expression.

When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.

Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

Column Level Constraint:

QUERY:28

Q.28. Write a query to create Null constraints with column level
Syntax: syntax for column level constraints with null

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (
<COLUMN NAME.1><DATATYPE> (SIZE) **CONSTRAINT** <NAME OF CONSTRAINTS> <CONSTRAINT TYPE>,(COLUMN NAME2 <DATATYPE> (SIZE)) ;

Command:

SQL>CREATE TABLE EMP13(EMPNO NUMBER(4),
ENAME VARCHAR2(20) CONSTRAINT EMP13_ENAME_NN NULL,
DESIGN VARCHAR2(20),SAL NUMBER(3));

DEFAULT CONSTRAINTS

Default constraints assign the default values if the values is not passed at the time of inserting the values to the table

QUERY:28

Q.28. Write a query to create default constraints with column level
Syntax: syntax for column level constraints with default

SQL :> CREATE <OBJ.TYPE><OBJ.NAME> (
<COLUMN NAME.1><DATATYPE> (SIZE) ,
<COLUMN NAME.2 <DATATYPE> (SIZE) **Default** <**default value**>) ;

Command:

```
SQL> CREATE TABLE DF(
                        REGNO NUMBER(5),
                        NAME VARCHAR2(20),
                        MARKS NUMBER(3) DEFAULT 55
                    );
```

Table created.

```
SQL> INSERT INTO DF VALUES(1001,'ARJUN',NULL);
```

1 row created.

```
SQL> INSERT INTO DF(REGNO) VALUES(1005);
```

1 row created.

```
SQL> INSERT INTO DF VALUES(1001,'RAJ',78);
```

1 row created.

```
SQL> SELECT * FROM DF;

REGNO       NAME        MARKS
----------  ----------  ----------
1001        ARJUN
1005                    55
1001         RAJ        78
```

DISABLING AND DROPPING CONSTRAINTS

USING DISABLE/ENABLE

Whenever a constraint is created for a column(s), every time an entry is made to the column , it must be evaluated to determine whether the value is allowed in that column(it checks doesn‘t Violate the c large block of data is being added to a ta le the validation process can severely slow down the ora processing speed. You are certain that the data being added to a table adheres to the constraints then disable the constraints while adding that particular block of data of data to the table.

To DISABLE a constraint, issue an ALTER TABLE command and change the status of the constraint to DISABLE. At a later time can reissue the ALTER TABLE command and change the status of the constraint back to ENABLE Sometimes, temporarily disable or drop a constraint.

Constraint Disable

QUERY:29

Q.29. Write a query to disable the constraints

Syntax: Syntax for disabling a single constraint in a table.

SQL>ALTER TABLE <TABLE-NAME> DISABLE CONSTRAINT <CONSTRAINTNAME>

SQL>ALTER TABLE EMP13 DISABLE CONSTRAINT EMP13_ENAME_NNNULL;

Constraint Enable

QUERY: 30

Q.30. Write a query to enable the constraints

Syntax: Syntax for disabling a single constraint in a table:

SQL>ALTER TABLE <TABLE-NAME> DISABLE CONSTRAINT <CONSTRAINTNAME>

Command:

SQL>ALTER TABLE EMP13 ENABLE CONSTRAINT EMP13_ENAME_NN NULL;

TO LIST ALL THE CONSTRAINTS:

SQL>SELECT * FROM USER_CONSTRAINTS;

SQL> SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE,STATUS FROM USER_CONSTRAINTS;

```
CONSTRAINT_NAME              C STATUS
------------------------------ - --------
SYS_C0011652                 P ENABLED
SYS_C0011655                 U ENABLED
SYS_C0011658                 P DISABLED
14 rows selected.
```

SQL>select constraint_name,constraint_type,status FROM USER_CONSTRAINTS WHERE TABLE_NAME=ABC

RESULT:

Thus the SQL commands for Creating an Employee database to set various constraints has been verified and executed successfully.

CREATING RELATIONSHIP BETWEEN THE DATABASES IN RDBMS

Ex: No: 05 (5.1) **TO IMPLEMENTATION OF SET OPERATORS**

__ : __ : __

AIM:

To Execute and verify The SQL Commands For Set Operators Implementation In Relational Model .

OBJECTIVE:

Set operators are used to retrieve the data from two or multiple tables.
They are different types.

Union

Union all
Intersect
Minus

ALGORITHM:

STEP 1: Start

STEP 2: Create two different tables with its essential attributes.

STEP 3: Insert attribute values into the tables.

STEP 4: Create the result for the various set operation.

STEP 5: Execute Command and extract information from the tables.

STEP 6: Stop

SAMPLE TABLES

SQL> select * from STUDENT_IT;

REG_NO	NAME	BRANC	SUBJECT
10001	ram	IT	DATA STRUCTURE
10002	Sam	IT	DATABASE SYSTEM
10003	Tam	IT	WEB TECHNOLOGY
10004	RAJ	IT	DSP
10005	TAJ	IT	DIP
10006	khan	IT	WEB TECHNOLOGY
30005	RAJA	ECE	CIRCUIT DESIGN

7 rows selected.

SQL> select * from STUDENT_ECE;

REG_NO	NAME	BRANC	SUBJECT
30001	RAMU	ECE	DIP
30002	SAMU	ECE	DSP
30003	TAMU	ECE	CIRCUIT DESIGN
30004	RAJU	ECE	ELECTRO MECHANICS
30005	RAJA	ECE	CIRCUIT DESIGN
30005	RAJA	ECE	CIRCUIT DESIGN

UNION

This will combine the records of multiple tables having the same structure.

Ex:

```
SQL> select * from student_IT union select * from student_ECE;
REG_NO      NAME BRANC      SUBJECT
----------  -------- ----------  -----------------
10001       ram      IT         DATA STRUCTURE
10002       Sam      IT         DATABASE SYSTEM
10003       Tam      IT         WEB TECHNOLOGY
10004       RAJ      IT         DSP
10005       TAJ      IT         DIP
10006       khan     IT         WEB TECHNOLOGY
30001       RAMUECE             DIP
30002       SAMU   ECE          DSP
30003       TAMU   ECE          CIRCUIT DESIGN
30004       RAJU     ECE        ELECTRO MECHANICS
30005       RAJA     ECE        CIRCUIT DESIGN
11 rows selected.
```

UNION ALL

This will combine the records of multiple tables having the same structure but including duplicates.

Ex:

```
SQL> select * from student_IT union all select * from student_ECE;
REG_NO      NAME       BRANC      SUBJECT
----------  ---------- ---------- --------------------
10001       ram        IT         DATA STRUCTURE
10002       Sam        IT         DATABASE SYSTEM
10003       Tam        IT         WEB TECHNOLOGY
10004       RAJ        IT         DSP
10005       TAJ        IT         DIP
10006       khan       IT         WEB TECHNOLOGY
30005       RAJA       ECE        CIRCUIT DESIGN
30001       RAMU       ECE        DIP
30002       SAMU       ECE        DSP
30003       TAMU       ECE        CIRCUIT DESIGN
30004       RAJU       ECE        ELECTRO MECHANICS
30005       RAJA       ECE        CIRCUIT DESIGN
30005       RAJA       ECE        CIRCUIT DESIGN
13 rows selected.
```

INTERSECT

This will give the common records of multiple tables having the same structure.

Ex:

```
SQL> select * from student_IT INTERSECT select * from student_ECE;
REG_NO      NAME       BRANC      SUBJECT
----------  ---------  ---------- -----------------
30005       RAJA       ECE        CIRCUIT DESIGN
```

MINUS

This will give the records of a table whose records are not in other tables having the same structure.

Ex:

SQL> select * from student_IT MINUS select * from student_ECE;

REG_NO	NAME	BRANC	SUBJECT
10001	ram	IT	DATA STRUCTURE
10002	Sam	IT	DATABASE SYSTEM
10003	Tam	IT	WEB TECHNOLOGY
10004	RAJ	IT	DSP
10005	TAJ	IT	DIP
10006	khan	IT	WEB TECHNOLOGY

6 rows selected.

RESULT:

Thus the SQL commands for SET operators has been verified and executed successfully.

CREATING RELATIONSHIP BETWEEN THE DATABASES IN RDBMS

Ex: No: 05 (5.2) **To implementation of Nested Queries**

__ : __ : __

AIM:

To execute and verify the SQL commands for Nested Queries.

OBJECTIVE:

Nested Query can have more than one level of nesting in one single query. A SQL nested query is a SELECT query that is nested inside a SELECT, UPDATE, INSERT, or DELETE SQL query.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create two different tables with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Create the Nested query from the above created table.

STEP 5: Execute Command and extract information from the tables.

STEP 6: Stop the program.

Table -1

Syntax: syntax for creating a table**.**

SQL: CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE),
COLUMN NAME.1 <DATATYPE> (SIZE) …);

Command:

SQL> CREATE TABLE EMP2(EMPNO NUMBER(5),ENAME VARCHAR2(20),JOB VARCHAR2(20), SAL NUMBER(6),MGRNO NUMBER(4),DEPTNO NUMBER(3));

Syntax: syntax for insert records in to a table.

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....);

Command:

SQL> SELECT *FROM EMP2;

EMPNO	ENAME	JOB	SAL	MGRNO	DPTNO
1001	MAHESH	PROGRAMMER	15000	1560	200
1002	MANOJ	TESTER	12000	1560	200
1003	KARTHIK	PROGRAMMER	13000	1400	201
1004	NARESH	CLERK	1400	1400	201
1005	MANI	TESTER	13000	1400	200
1006	VIKI	DESIGNER	12500	1560	201
1007	MOHAN	DESIGNER	14000	1560	201
1008	NAVEEN	CREATION	20000	1400	201
1009	PRASAD	DIR	20000	1560	202
1010	AGNESH	DIR	15000	1400	200

TABLE- 2

Syntax: syntax for creating a table.

SQL: CREATE <OBJ.TYPE> <OBJ.NAME> (COLUMN NAME.1 <DATATYPE> (SIZE),
COLUMN NAME.1 <DATATYPE> (SIZE) ……;

Command:

SQL> CREATE TABLE DEPT2(DEPTNO NUMBER(3),DEPTNAME VARCHAR2(10), LOCATION VARCHAR2(15));

Table created.

Syntax: syntax for insert records in to a table.

SQL :> INSERT INTO <TABLE NAME> VALUES< VAL1, 'VAL2',.....);

SQL> SELECT *FROM DEPT2;

DEPTNO	DEPTNAME	LOCATION
107	DEVELOP	ADYAR
201	DEBUG	UK
200	TEST	US
201	TEST	USSR
108	DEBUG	ADYAR
109	BUILDPOTHERI	

6 rows selected.

Syntax:

GENERAL SYNTAX FOR NESTED QUERY:

SQL> SELECT "COLUMN_NAME1"

FROM "TABLE_NAME1"

WHERE "COLUMN_NAME2"

[COMPARISON OPERATOR] (SELECT "COLUMN_NAME3" FROM "TABLE_NAME2"

WHERE [CONDITION])

Syntax: syntax nested query statement.

SQL> SELECT <COLUMN_NAME>

FROM FRORM <TABLE _1>

WHERE <COLUMN_NAME> <RELATIONAL _OPERATION> 'VALUE'

(SELECT (AGGRECATE FUNCTION) FROM <TABLE_1> WHERE <COLUMN NAME> = 'VALUE'
(SELECT <COLUMN_NAME> FROM <TABLE_2> WHERE <COLUMN_NAME= 'VALUE'));

NESTED QUERY STATEMENT:

Command:

SQL> SELECT ENAME FROM EMP2

WHERE SAL>(SELECT MIN(SAL)

FROM EMP2 WHERE DPTNO=(SELECT DEPTNO FROM DEPT2 WHERE LOCATION='UK'));

Nested Query Output:

ENAME

MAHESH
MANOJ
KARTHIK
MANI
VIKI
MOHAN
NAVEEN
PRASAD
AGNESH

RESULT:

Thus the SQL commands for to implementation of nested queries has been verified and executed successfully.

CREATING RELATIONSHIP BETWEEN THE DATABASES IN RDBMS

Ex: No: 05 (5.3) **To implementation the Join Operations**

__ : __ : __

AIM:

To execute and verify the SQL commands for various join operation.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create two different tables with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Create the table object for easy reference.

STEP 5: Join two tables by using JOIN operator.

STEP 6: Display the result of the result table.

STEP 7: Stop the program.

JOINS:

Joins are used to retrieve the data from multiple tables.

Types of Joins:

1. EQUI_JOIN
2. NON EQUI_JOIN
3. SELF JOIN
4. OUTER JOIN
   - 4.1 Right outer join
   - 4.2 Left outer join
   - 4.3 Full outer join

1. **EQUI_JOIN**:

When tables are joined basing on a common column it is called EQUI_JOIN.

Ex:

select empno, ename, dname from emp, dept where emp.deptno = dept.deptno;

EMPNO	ENAME	DNAME
7369	SMITH	RESEARCH
7499	ALLEN	SALES
7521	WARD	SALES

Note:

We need to mention join conditions in the where clause.

In EQUI_JOINS we along use to equal to operator in join condition.

Ex:

SQL>Selete empno, ename, sal, job, dname, loc

from emp, dept

where emp.deptno = dept.deptno;

SQL>Selete empno, ename, sal, deptno, dname,

loc from emp, dept

where emp.deptno = dept.deptno;// error

SQL>Selete empno, ename, sal, emp.deptno, dname,

loc from emp, dept

where emp.deptno = dept.deptno; //valid

Note:

we need to mention table name dot column(emp.deptno) name for the common column to resolve the any table.

The common column can be retrieved from any of the table.

We can filter the data from the result of join.

Ex:

SQL>Select empno, ename, sal, emp.deptno, dname,

loc from emp, dept

where emp.deptno = dept.deptno AND sal > 2000;

To improve the performance of the join we need mention table name dot column name for all the columns.

Ex:

SQL>Select emp.empno, emp.ename, emp.sal,emp.deptno, dept.dname,

dept.loc from emp,dept

where emp.deptno = dept.deptno AND sal > 2000;

Table alias:

Table alias is an alternate name given to a table.

By using a table alias length of the table reduces and at the same time performance is maintains.

Table alias are create in same clause can be used in select clause as well as where clause.

Table alias is temporary once the query is executed the table alias are losed.

Ex:

SQL>Select E.Empno, E.Ename, E.sal, E.deptno, D.Dname,

D.loc from emp E, Dept D

where E.deptno = D.deptno;

Join the multiple tables(3 tables):

Select * from Areas;

City	**State**
Newyork	AP
Dallas	Mh

Ex:

SQL>Select E.empno, E.ename,

E.sal,D.dname,A.state from emp E, dept D, Areas A

where E.deptno = D.deptno AND D.loc = A.city;

Note: To join 'n' tables we need n-1 conditions.

NON EQUI JOIN:

When we do not use NON EQUI JOIN to operator in the join condition is NON EQUI JOIN.

Ex:

SQL>Select * from SALGRADE;

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

SQL>Select e.empno, e.ename, e.sal,

s.grade from emp e, salgrade s

where e.sal BETWEEN s.losal AND hisal;

EMPNO	ENAME	GRADE
7369	SMITH	1
7876	ADAMS	1
7900	JAMES	2

SQL>Select e.empno, e.ename,

s.grade from emp e, salgrade s

where e.sal BETWEEN s.losal AND s.hisal AND s.grade = 4;

SELF JOIN:

When a table is joining to it self it is called self join. In self joins we need to create two table aliases for the same table.

SQL>Select empno, ename, job, mgr, from emp;

SQL>Select e.empno, e.ename, e.job,

m.ename from emp e, emp m

where e.mgr = m.empno;

Empno	**Ename**	**Job**	**Ename**
7902	FORD	ANALYST	JONES
7869	SCOTT	CLERK	JONES
7900	JAMES	SALESMAN	BLAKE

CARTESIAN PRODUCT:

When tables are joined without any join condition it is called Cartesian product. In the result we get all possible combination.

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e, dept d; //14*4=56 rows are selected

ANSI JOINS:

They are the three types.

INNER JOINS:

It is same as Equi join.

Ex:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname,

d.loc from emp e INNER JOIN dept d ON(e.deptno =

d.deptno); 2.**NATURAL JOIN**:

It is same as Equi join.

Ex:

SQL>Select empno, ename, sal, deptno, dname,loc from NATURAL JOIN dept;

CROSS PRODUCT/CROSS JOIN:

It is same as Cartesian product.

Ex:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e CROSS JOIN dept d; //14*4 = 56 rows are displayed.

DEFAULT:

Ex:

SQL>Create table stu1(sno number(3),

Sname varchar2(10),

Marks number(3) default 100,

Doj Date DEFAULT sysdate);

SQL>Insert into stu1(sno, sname) values(101,'malli');

SQL>Insert into stu1 values(102,'ARUN',40,'11-JAN-09');

SQL>Insert into stu1 values (103,'KIRAN',NULL,'12-FEB-10');

SNO	SNAME	MARKS	DOJ
101	malli	100	26-JUN-12
102	ARUN	40	11-JAN-09
103	KIRAN		12-FEB-10

SUPER KEY:

Combination of columns which can be used unique key identify every row is called as super key. Table object

Column Attributes

Row Tuple/Record

OUTER JOINS:

It is extension of EQUI JOINS.

In outer joins we get match as well as non matching rows.

(+) This called as outer join operator.

1. RIGHT OUTER JOIN:

SQL Syntax:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e, dept d

where e.deptno(+) = d.deptno; //14 + 1 = 15 rows

empno	ename	sal	deptno	dname	loc
7900	james	950	30	sales	chicago
8963	adams	1400	20	clerk	newyork
6798	adams	2000	10	sales	india

ANSI SYNTAX OF RIGHT OUTER JOIN:

ANSI SYSTAX:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc

from emp e RIGHT OUTER JOIN dept d ON(e.deptno = d.deptno);

LEFT OUTER JOIN:

SQL Syntax:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname,
d.loc from emp e, dept d

where e.deptno = d.deptno(+); //14+3 = 17 row displayed

ANSI SYNTAX OF LEFT OUTER JOIN:

ANSI SYNTAX:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from
emp e LEFT OUTER JOIN dept d ON(e.deptno = d.deptno);

FULL OUTER JOIN:

ANSI SYNTAX:

SQL>Select e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc from
emp e FULL OUTER JOIN dept d ON(e.deptno = d.deptno);

//14 + 2 + 3 = 19 rows are displayed.

RESULT:

Thus the SQL commands **to** implementation the join operations has been verified and executed successfully.

Study of PL/SQL block.

Ex: No: 06

__ : __ : __

AIM:

To write a PL/SQL block using different control (if, if else, for loop, while loop,...) stater

OBJECTIVE:

PL/SQL Control Structure provides conditional tests, loops, flow control and branches that let to produce well-structured programs

PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL allows you to mix SQ statements with procedural statements like IF statement, Looping structures etc.

It is extension of SQL the following or advantages of PL/SQL.

1. We can use programming features like if statement loops etc.
2. PL/SQL helps in reducing network traffic.
3. We can have user defined error massages by using concept of exception handling.
4. We can perform related actions by using concept of Triggers.
5. We can save the source code permanently for repeated execution.

PL/SQL Block:

A PL/SQL programs called as PL/SQL block.

PL/SQL Block:

```
DECLARE
                    Declaration of variable
                    Declaration of cursor----------        (OPTIONAL)
                    Declaration of exception
BEGIN
                     Executable commands---------          (MANDATORY)
EXCEPTION
                     Exception handlers----------           (OPTIONAL)
END;
/                    To execute the program / command
```

Declare:

This section is used to declare local variables, cursors, Exceptions and etc. This section is optional.

Executable Section:

This section contains lines of code which is used to complete table. It is mandatory.

Exception Section:

This section contains lines of code which will be executed only when exception is raised. This section is optional.

Simplest PL/SQL Block:

```
Begin
--------
--------
--------
END;
```

SERVEROUTPUT

This will be used to display the output of the PL/SQL programs. By default this will be off.

Syntax:

Set serveroutput on | off

Ex:

SQL> set serveroutput on

BLOCK TYPES

Anonymous blocks
Named blocks
Labeled blocks
Subprograms
Triggers

ANONYMOUS BLOCKS

Anonymous blocks implies basic block structure.

Ex:

Q : program to display the string “”

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('My first program');
END;
/
```

LABELED BLOCKS

Labeled blocks are anonymous blocks with a label which gives a name to the block.

Ex:

```
<<my_bloock>>

BEGIN
    Dbms_output.put_line('My first program'):
END;
```

SUBPROGRAMS

Subprograms are procedures and functions. They can be stored in the database as stand-alone objects, as part of package or as methods of an object type.

TRIGGERS

Triggers consist of a PL/SQL block that is associated with an event that occurs in the database.

NESTED BLOCKS

A block can be nested within the executable or exception section of an outer block.

IDENTIFIERS

Identifiers are used to name PL/SQL objects, such as variables, cursors, types and subprograms.

Identifiers consists of a letter, optionally followed by any sequence of characters, including letters, numbers, dollar signs, underscores, and pound signs only. The maximum length for an identifier is 30 characters.

QUOTED IDENTIFIERS

If you want to make an identifier case sensitive, include characters such as spaces or use a reserved word, you can enclose the identifier in double quotation marks.

Ex:

```
DECLARE
    "a" number := 5;
    "A" number := 6;
BEGIN
    dbms_output.put_line('a = ' || a);
    dbms_output.put_line('A = ' || A);
END;
/
```

Output:

```
a = 6
A = 6
```

COMMENTS

Comments improve readability and make your program more understandable. They are ignored by the PL/SQL compiler.

There are two types of comments available.

Single line comments

Multiline comments

SINGLE LINE COMMENTS

A single-line comment can start any point on a line with two dashes and continues until the end of the line.

Ex:

```
BEGIN
    Dbms_output.put_line('hello');          -- sample program
END;
/
```

MULTILINE COMMENTS

Multiline comments start with the /* delimiter and ends with */ delimiter.

Ex:

```
BEGIN
    Dbms_output.put_line('hello');          /* sample program */
END;
/
```

VARIABLE DECLERATIONS

Variables can be declared in declarative section of the block;

Ex:

```
DECLARE
    a number;
    b number := 5;
    c number default 6;
```

CONSTANT DECLERATIONS

To declare a constant, you include the CONSTANT keyword, and you must supply a default value.

Ex:

DECLARE

b **constant** number := 5;

c **constant** number default 6;

NOT NULL CLAUSE

You can also specify that the variable must be not null.

Ex:

DECLARE

b constant number **not null**:=

5; c number not null **default 6**;

ANCHORED DECLERATIONS

PL/SQL offers two kinds of anchoring.

Scalar anchoring

Record anchoring

SCALAR ANCHORING

Use the %TYPE attribute to define your variable based on table's column o
PL/SQL scalar variable.

Ex:

DECLARE

dno **dept.deptno%type;**

Subtype t_number is number;

a t_number;

Subtype t_sno is **student.sno%type;**

V_sno t_sno;

RECORD ANCHORING

Use the %ROWTYPE attribute to define your record structure based on a table.

Ex:

DECLARE

V_dept **dept%rowtype;**

Benefits of Anchored Declarations

Synchronization with database columns.

Normalization of local variables.

PROGRAMMER-DEFINED TYPES

With the SUBTYPE statement, PL/SQL allows you to define your own subtypes or aliases of predefined datatypes, sometimes referred to as abstract datatypes.

There are two kinds of subtypes.

Constrained

Unconstrained

CONSTRAINED SUBTYPE

A subtype that restricts or constrains the values normally allowd by the datatype itself.

Ex:

Subtype positive is binary_integer range 1..2147483647;

In the above declaration a variable that is declared as positive can store only ingeger greater than zero even though binary_integer ranges from -2147483647..+2147483647.

UNCONSTRAINED SUBTYPE

A subtype that does not restrict the values of the original datatype in variables declared with the subtype.

Ex:

Subtype float is number;

DATATYPE CONVERSIONS

PL/SQL can handle conversions between different families among the datatypes.

Conversion can be done in two ways.

Explicit conversion

Implicit conversion

EXPLICIT CONVERSION

This can be done using the built-in functions available.

IMPLICIT CONVERSION

PL/SQL will automatically convert between datatype families when possible.

Ex:

```
DECLARE
    a varchar(10);
BEGIN
    select deptno into a from dept where dname='ACCOUNTING';
END;
```

In the above variable a is char type and deptno is number type even though, oracle will automatically converts the numeric data into char type assigns to the variable.

PL/SQL can automatically convert between

Characters and numbers

Characters and dates

VARIABLE SCOPE AND VISIBILITY

The scope of a variable is the portion of the program in which the variable can be accessed. For PL/SQL variables, this is from the variable declaration until the end of the block. When a variable goes out of scope, the PL/SQL engine will free the memory used to store the variable.

The visibility of a variable is the portion of the program where the variable can be accessed without having to qualify the reference. The visibility is always within the scope. If it is out of scope, it is not visible.

Ex1:

```
DECLARE
        a number; -- scope of a
BEGIN
--------
                DECLARE
                        b number; -- scope of b
                BEGIN
                -----
                END;
------
END;
```

Ex2:

```
        DECLARE
                a number;
                b number;
        BEGIN
                -- a , b available here
                DECLARE
                        b char(10);
                BEGIN
                        -- a and char type b is available here
                END;
        -----
        END;
```

Ex3:

```
<<my_block>>
DECLARE
        a number;
        b number;
BEGIN
        -- a , b available here
        DECLARE
                b char(10);
        BEGIN
        -- a and char type b is available here
        -- number type b is available using <<my_block>>.b
        END;
------
END;
```

PL/SQL CONTROL STRUCTURES

PL/SQL has a variety of control structures that allow you to control the behaviour of the block as it runs. These structures include conditional statements and loops.

If-then-
else Case

- Case with no else
- Labeled case
- Searched

case Simple loop

While loop
For loop
Goto and Labels

IF-THEN-ELSE

Syntax:

```
If <condition1> then
        Sequence of statements;
Elseif <condition1> then
        Sequence of statements;
        ……
Else
        Sequence of statements;
End if;
```

Ex:

```
DECLARE
    dno number(2);
BEGIN
    select deptno into dno from dept where dname =
        'ACCOUNTING'; if dno = 10 then
            dbms_output.put_line('Location is NEW
        YORK'); elseif dno = 20 then
            dbms_output.put_line('Location is
        DALLAS'); elseif dno = 30 then
            dbms_output.put_line('Location is CHICAGO');
        else
            dbms_output.put_line('Location is BOSTON');
        end if;
END;
```

Output:

Location is NEW YORK

CASE

Syntax:

Case ***test-variable***

When ***value1*** then *sequence of statements;*

When ***value2*** then *sequence of statements;*

.
.

When ***valuen*** then *sequence of statements;*

Else *sequence of statements;*

End case;

Ex:

```
DECLARE
    dno number(2);
BEGIN
    select deptno into dno from dept where dname =
    'ACCOUNTING'; case dno
        when 10 then
            dbms_output.put_line('Location is NEW
        YORK'); when 20 then
            dbms_output.put_line('Location is
        DALLAS'); when 30 then
            dbms_output.put_line('Location is CHICAGO');
        else
            dbms_output.put_line('Location is BOSTON');
    end case;
END;
```

Output:

Location is NEW YORK

CASE WITHOUT ELSE

Syntax:

Case ***test-variable***

When ***value-1*** then *sequence of statements;*

When ***value-2*** then *sequence of statements;*

……

When ***value-n*** then *sequence of statements;*

End case;

Ex:

```
DECLARE
    dno number(2);
BEGIN
    select deptno into dno from dept where dname =
    'ACCOUNTING'; case dno
        when 10 then
            dbms_output.put_line('Location is NEW
        YORK'); when 20 then
            dbms_output.put_line('Location is
        DALLAS'); when 30 then
            dbms_output.put_line('Location is
        CHICAGO'); when 40 then
            dbms_output.put_line('Location is BOSTON');
    end case;
END;
```

Output:

Location is NEW YORK

LABELED CASE

Syntax:

```
<<label>>
    Case test-variable
        When value1 then sequence of statements;
        When value2 then sequence of statements;
        ……
        When valuen then sequence of statements;
    End case;
```

Ex:

```
DECLARE
    dno number(2);
BEGIN
    select deptno into dno from dept where dname =
        'ACCOUNTING'; <<my_case>>
            case dno
                when 10 then
                    dbms_output.put_line('Location is NEW
                YORK'); when 20 then
                    dbms_output.put_line('Location is
                DALLAS'); when 30 then
                    dbms_output.put_line('Location is
                CHICAGO'); when 40 then
                    dbms_output.put_line('Location is BOSTON');
            end case
        my_case;
END;
```

Output:

Location is NEW YORK

SEARCHED CASE

Syntax:

Case

When <***condition-1***> then *sequence of statements;*

When <***condition-2***> then *sequence of statements;*

……

When <***condition-n***> then *sequence of statements;*

End case;

Ex:

```
DECLARE
    dno number(2);
BEGIN
    select deptno into dno from dept where dname = 'ACCOUNTING';
        case dno
            when dno = 10 then
                dbms_output.put_line('Location is NEW
            YORK'); when dno = 20 then
                dbms_output.put_line('Location is
            DALLAS'); when dno = 30 then
                dbms_output.put_line('Location is
            CHICAGO'); when dno = 40 then
                dbms_output.put_line('Location is BOSTON');
        end case;
END;
```

Output:

Location is NEW YORK

SIMPLE LOOP

Syntax:

Loop

Sequence of statements;

Exit when <*condition*>;

End loop;

In the syntax exit when <*condition*> is equivalent to

If <*condition*> then

Exit;

End if;

Ex:

```
DECLARE
    i number := 1;
BEGIN
    loop
        dbms_output.put_line('i = ' ||
        i); i := i + 1;
        exit when i > 5;
    end loop;
END;
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

WHILE LOOP

Syntax:

While <condition> loop

Sequence of statements;

End loop;

Ex:

```
DECLARE
    i number := 1;
BEGIN
    While i <= 5 loop
        dbms_output.put_line('i = ' || i);
        i := i + 1;
    end loop;
END;
```

Output:

i = 1

i = 2

i = 3

i = 4

i = 5

FOR LOOP

Syntax:

For <*loop_counter_variable*> in *low_bound..high_bound* loop

Sequence of statements;

End loop;

Ex1:

```
BEGIN
    For i in 1..5 loop
        dbms_output.put_line('i = ' || i);
    end loop;
END;
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

Ex2:

```
BEGIN
    For i in reverse 1..5 loop
        dbms_output.put_line('i = ' || i);
    end loop;
END;
```

Output:

```
i = 5
i = 4
i = 3
i = 2
i = 1
```

GOTO AND LABELS

Syntax:

Goto *label*;

Where *label* is a label defined in the PL/SQL block. Labels are enclosed in double angle brackets.

When a goto statement is evaluated, control immediately passes to the statement identified by the label.

Ex:

```
BEGIN
    For i in 1..5 loop
        dbms_output.put_line('i = ' || i);
        if i = 4 then
            goto exit_loop;
        end if;
    end loop;
    <<exit_loop>>
    Null;
END;
```

Output:

```
i = 1
i = 2
i = 3
i = 4
```

Restrictions on GOTO

It is illegal to branch into an inner block, loop.

At least one executable statement must follow.

It is illegal to branch into an if statement.

It is illegal to branch from one if statement to another if statement. It is illegal to branch from exception block to the current block.

RESULT:

Thus the Study of PL/SQL block has been implemented by various control structure are verified and executed successfully.

Write a PL/SQL block to satisfy some conditions by accepting input from the user.

Ex: No: 07

__ : __ : __

AIM:

To implement the PL/SQL block to satisfy some conditions by accepting input from the user.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Create the PL/SQL Block with necessary blocks.

STEP 5: Declare the necessary variable in the declaration section

STEP 6: write the main program logics in the begin block.

STEP 7: if you want to access the table use the SQL statement.

STEP 8: if you want to solve any exception, write the exception name with WHEN stetment

STEP 9: Execute the PL/SQL block.

STEP 10: Give the input values or validate the information from the tables.

STEP 11: Stop the program.

Q1: Write PL/SQL block which will calculate some of two numbers and display the output?

```
DECLARE
        A number(2);
        B number(2);
        C number(3);
BEGIN
        A := 10;
        B := 20;
        C := A + B;
                DBMS_OUTPUT.PUT_LINE(C);
                DBMS_OUTPUT.PUT_LINE( 'sum of two numbers' || C);
END;
/
```

Output:

```
 30
sum of two numbers 30
PL/SQL procedure successfully completed.
```

Q2: Write a PL/SQL block which accepts employee number and increment is salary by 1000?

```
DECLARE
        A number(4);
        A := &Empno;
                Update emp set sal = sal + 1000 where Empno = A;
END;
/
```

Q3: Write a PL/SQL block which empno and delete that row from the emp table?

```
DECLARE
        A number(4);
BEGIN
        A := &Empno;
        Delete from emp where Empno = A;
END;
/
```

Q4: PL/SQL for reversing the given string

Algorithm:

1. Get the input string.
2. Find the length of the string.
3. Extract the characters one by one from the end of the string.
4. Concatenate the extracted characters.
5. Display the concatenated reversed string.
6. Stop the program.

Program:

```
declare
        b varchar2(10) := '&b';
        c varchar2(10);
        l number(2);
        i number(2);
        g number(2);
        d varchar2(10);
begin

        l:=length(b);
        g:=l;
        for i in 1..l
        loop
                c:=substr(b,g,1);
                g := g - 1;
                d := d ||
        c; end loop;
        dbms_output.put_line('revised string
        is'); dbms_output.put_line(d);
end;
```

OUTPUT:

```
Enter value for b: ramu
old   2: b varchar2(10) := '&b';
new   2: b varchar2(10) := 'ramu';
revised string is
umar


PL/SQL procedure successfully completed.
```

Q5: PL/SQL for Fibonacci Series.

Algorithm:

1. Get the no.of terms N, in the fibonacci series to be generated.
2. If N is less than 2, then raise an exception and display the message.
3. Otherwise initialize the value of A as 0 and B as 1 and display them.
4. Repeat the steps 5&6 in N-3 times.
5. C:=A+B & Display C.
6. A:=B&B:=A
7. Stop the program.

Program:

```
declare
        a number(3);b number(3);c number(3);n
        number(3):=&n; negative exception;
begin
        if n < 2 then
                raise negative;
        end if;
        a := 0;b := 1;
        dbms_output.put_line('fibonacci series
        is'); dbms_output.put_line(a);
        dbms_output.put_line(b);
        for i in 3 ..
        n loop
                c := a + b;
                dbms_output.put_line(c);
                a := b;
                b :=
        c; end loop;
exception
        when negative then
                dbms_output.put_line('n should be greater than 1');
end;
```

SQL> /

```
Enter value for n:10
old  5:  n  number(3):=&n;
new  5:n  number(3):=10;
Fibonacci
0
1
1
```

2
3
5
8
13
21
34

PL/SQL procedure successfully completed

Q6: Program to check whether given number is Armstrong or not.

Algorithm:

Step 1: Declare the variable N, S, D and DUP.
Step 2: Store the value in var. N and var. DUP..
Step 3: check for the value of N, which is not equal to 0.
Step 4: divide value stored in N by 10 and store it var. D. (D=n%10).
Step 5: the reminder will be multiply 3 times and store it in Var. S.
Step 6: The coefficient will be calculated using FLOOR function. And store it in var.
N. Step 7: repeat the Steps 3, 4, 5, and 6 till loop will be terminated.
Step 8: Check whether the stored value and calculated values are same
Step 9: if both the values are same, then display "The given number is Armstrong"
Step 10: Otherwise display "it is not Armstrong" and terminate the loop.

```
Declare
N number;
S number;
D number;
Begin
        N:=&n;
        S:=0;
While(n!=0)
Loop
        D=n%10;
        S:=s+(D*D*D);
        N:=floor(n/10);
End loop;
If (DUP=S) then
        DBMS_output.put_line('number is armstrong');
Else
        DBMS_output.put_line('number is not armstrong');
End if;
End;
```

Test Valid Data Set:

Enter value of
n 153

Output:

number is Armstrong

Q7: Write a program to generate all prime numbers below 100.

AIM: to generate all prime numbers below 100.

```
Declare
        I number;
        J number;
        C number;
Begin
While(i<=100)
Loop
        C:=0;
        J:=1;
        While(j<=i)
        Loop
                If(floor(i%j)=0) then
                        C:= C+1;
                End if;
        J:=j+1;
        End loop;
If(c=2) then
        Dbms_output.put_line(i);
End if;
Endloop;
End;
```

Valid Test Data

OUTPUT:

2

3

5

7

11

.

.

99

Q8: Write a program to demonstrate %type and %rowtype attributes

AIM: to demonstrate %type and %rowtype attributes

```
Declare
        My_Empno emp.empno%type;
        My_Ename emp.ename%type;
        My_Emprow emp%rowtype; No
        number;
Begin
        No:=&no;
        Select empno,ename into my_empno,my_ename from emp where
        empno=no; If(SQl%rowcount=1) then
                Dbms_output.put_line('empno is' || my_empno || 'ename is ' || my_ename);
        Else
                Dbms_output.put_line( 'error');
        End if;
        Select * into my_emprow from emp where
        empno=no; If(SQl%rowcount=1) then
          Dbms_output.put_line('empno is' || my_emprow.empno || 'ename is ' ||
        my_emprow.ename); Else
                Dbms_output.put_line( 'error');
        End if;
End;
```

Valid Test Data

Enter the value for no:

7788

OUTPUT

empno is 7788 ename is vinay s.

empno is 7788 ename is vinay s.

RESULT:

Thus the PL/SQL block to satisfy some conditions by accepting input from the user has been verified and executed successfully.

Write a PL/SQL block that handles all types of exceptions.

Ex: No: 08

__ : __ : __

AIM:

To Write a PL/SQL block that handles all types of exceptions.

ALGORITHM:

STEP1: Start the program.

STEP2: Create a table with some valid data.

STEP3: write the PL/SQL program that to handle the exception on exception block

STEP4: Execute the PL/SQL program and give the input values or make the error on the table data.

STEP5: Display the PL/SQL program error message.

STEP6: Stop the program.

EXCEPTIONS:

In PL/SQL, errors and warnings are called as exceptions. Whenever a predefined error occurs in the program, PL/SQL raises an exception. For example, if you try to divide a number by zero then PL/SQL raises an exception called ZERO_DIVIDE and if SELECT can not find a record then PL/SQL raises exception NO_DATA_FOUND.

PL/SQL has a collection of predefined exceptions. Each exception has a name. These exceptions are automatically raised by PL/SQL whenever the corresponding error occurs.

In addition to PL/SQL predefined exceptions, user can also create his own exceptions to deal with errors in the applications.

They are three types of Exceptions.

1. ORACLE Predefined Exception
2. ORACLE Non Predefined Exception
3. USER Defined Exception

SYNTAX OF EXCEPTION HANDLING

```
WHEN exception-1 [or exception -2] ...
        THEN statements;
[WHEN exception-3 [or exception-4] ... THEN
        statements; ] ...
[WHEN OTHERS THEN
        statements; ]
```

Q: The following example handles **NO_DATA_FOUND** exception.

```
declare
    …
    begin
        select
    … exception
        when no_data_found
            then statements;
end;
```

Output:

Q: The following exception handling part takes the same action when either **NO_DATA_FOUND** or **TOO_MANY_ROWS** exceptions occur.

```
declare
    ...
begin
    select ...
exception
    when no_data_found or too_many_rows
        then statements;
end;
```

Output:

Q: The following snippet handles these two exceptions in different ways.

```
declare
        ...
begin
        select ...
exception
        when no_data_found
                then statements;
        when too_many_rows
                then statements;
end;
```

Output:

WHEN OTHERS is used to execute statements when an exception other than what are mentioned in exception handler has occurred.

```
declare
        newccode varchar2(5) := null;
begin
        update courses set ccode = newccode where ccode =
'c'; exception
        when dup_val_on_index then
                dbms_output.put_line('Duplicate course code');
        when others then dbms_output.put_line(
                sqlerrm);
end;
```

Output:

Predefined exceptions

PL/SQL has defined certain common errors and given names to these errors, which are called as predefined exceptions.

Each exception has a corresponding Oracle error code. The following is the list of predefined exceptions and the corresponding Oracle error code.

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Table 1: Predefined Exceptions

NO_DATA_FOUND:

This Exception is Raised if a SELECT INTO statement returns no rows or if you reference an un-initialized row in a PL/SQL table.

Ex:

```
Declare
        L_sal emp.sal%type;
Begin
        DBMS_OUTPUT.PUT_LINE( 'WELCOME' );
        Select sal INTO L_sal from emp where empno =
        &empno; DBMS_OUTPUT.PUT_LINE(L_sal);
        DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
EXCEPTION
        when NO_DATA_FOUND then
                DBMS_OUTPUT.PUT_LINE( 'INVALID EMPNO');
END;
/
```

TOO_MANY_ROWS:

This Exception is Raised if a SELECT INTO statement returns more than one row.

Ex:

```
Declare
        L_sal emp.sal%type;
Begin
        DBMS_OUTPUT.PUT_LINE( 'WELCOME');
        Select sal INTO L_sal from emp where deptno =
        30; DBMS_OUTPUT.PUT_LINE(L_sal);
        DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
EXCEPTION
        when TOO_MANY_ROWS then
                DBMS_OUTPUT.PUT_LINE( 'MORE THEN ONE ROW RETURNED');
END;
/
```

ZERO_DIVIDE:

Raised when your program attempts to divide a number by zero.

Ex:

```
Declare
        A Number;
Begin
        A :=
5/0; Exception
        when ZERO_DIVIDE then
                DBMS_OUTPUT.PUT_LINE( 'DO NOT DIVIDE BY 0' );
END;
/
```

Note:

This Exception is raised when we try to divided by zero.

VALUE_ERROR:

This Exception is raised when there is miss match with the value and data type of local variable or size of local variables.

Ex 1:

```
Declare
        L_sal emp.sal%type;
Begin
        DBMS_OUTPUT.PUT_LINE( 'WELCOME' );
        Select ename INTO L_sal from emp where empno =
        7521; DBMS_OUTPUT.PUT_LINE(L_sal);
        DBMS_OUTPUT.PUT_LINE( 'THANK YOU' );
EXCEPTION
        when VALUE_ERROR then
                DBMS_OUTPUT.PUT_LINE( 'please check the local variables');
END;
/
```

Ex 2:

```
Declare
        A number(3);
Begin
        A :=
1234; Exception
        when VALUE_ERROR then
                DBMS_OUTPUT.PUT_LINE( 'PLEASE CHECK THE LOCAL VARIABLES' );
END;
/
```

DUP_VAL_ON_INDEX: (duplicate value on index)

This Exception is raised when we try to insert a duplicate value in primary key constraint.

Ex:

```
Begin
        DBMS_OUTPUT.PUT_LINE( 'welcome' );
        Insert into student values(104, 'ARUN',60);
        DBMS_OUTPUT.PUT_LINE( 'Thank you' );
Exception
        when DUP_VAL_ON_INDEX then
                DBMS_OUTPUT.PUT_LINE(' Do not insert duplicates' );
END;
/
```

The above program works on an assumption the table student for if having a primary key SNO column with value 104.

WHEN OTHERS:

When others are a universal Exception angular this can catch all the Exceptions.

```
Declare
        L_sal number(4);
        A number;
Begin
        DBMS_OUTPUT.PUT_LINE( 'Welcome' );
        Select sal INTO L_SAL from emp where deptno = &deptno;
        DBMS_OUTPUT.PUT_LINE('The sal is ....'||L_sal);
        A :=10/0;
        DBMS_OUTPUT.PUT_LINE( 'Thank you' );
Exception
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE( 'please check the code' );
END;
/
```

ERROR REPORTING FUNCTIONS:

They are two Error Reporting functions.

1. SQLCODE
2. SQLERRM

These error reporting functions are used in when others clause to identified the exception which is raised.

1. **SQLCODE**: It returns ERRORCODE

2. **SQLERRM**: It returns Exception number and Exception message.

Note: for NO_DATA_FOUND Exception SQLCODE will return 100.

```
Declare
        L_sal number(4);
        A number;
Begin
        DBMS_OUTPUT.PUT_LINE( 'Welcome' );
        Select sal INTO L_SAL from emp where deptno = &deptno;
        DBMS_OUTPUT.PUT_LINE('The sal is
        ....'||L_sal); A :=15/0;
        DBMS_OUTPUT.PUT_LINE( 'Thank you' );
Exception
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE( 'please check the code' );
                DBMS_OUTPUT.PUT_LINE(SQLCODE);
                DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

NESTED BLOCK:

```
Declare
        A number := 10;
Begin
        DBMS_OUTPUT.PUT_LINE('HELLO1');
                Declare
                        B number := 20;
                Begin
                        DBMS_OUTPUT.PUT_LINE('HELLO2');
                        DBMS_OUTPUT.PUT_LINE(B);
                        DBMS_OUTPUT.PUT_LINE(A);
                END;
        DBMS_OUTPUT.PUT_LINE('HELLO3');
        DBMS_OUTPUT.PUT_LINE(B); --ERROR
END;
/
```

Note:

outer block variables can be accessed in nested block nested block variables can not be accessed in outer block.

EXCEPTION PROPAGATION:

```
Begin
        DBMS_OUTPUT.PUT_LINE('HELLO1');
        L_SAL EMP.SAL%TYPE;
        Begin
                DBMS_OUTPUT.PUT_LINE('HELLO2');
                Select sal INTO L_SAL from emp where empno = 1111;
                DBMS_OUTPUT.PUT_LINE('HELLO3');
        END;
        DBMS_OUTPUT.PUT_LINE('HELLO4');
EXCEPTION
        WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('HELLO5');
END;
/
```

ORALE NON PREDEFINED EXCEPTIONS:

These Exceptions will have only Exception number. But does not have Exception name. Steps to handle non predefined exceptions.

Syntax:

Step1: Declare the Exception

<EXCEPTION_NAME> EXCEPTION;

Step2: Associate the Exception

PRAGMA EXCEPTION_INIT(<EXCEPTION_NAME>,<EXCEPTION NO>);

Step3: Catch the Exception

WHEN <EXCEPTION_NAME> THEN

END;

/

ORA -2292 is an example of non predefined exception.

This exception is raised when we delete row from a parent table. If the corresponding value existing the child table.

```
Declare
    MY_EX1 Exception; --step1
    PRAGMA EXCEPTION_INIT(MY_EX1, -2292); --step2
Begin
    DBMS_OUTPUT.PUT_LINE('Welcome');
    Select from student where eno = 102;
EXCEPTION
    WHEN MY_EX1 THEN --step3
    DBMS_OUTPUT.PUT_LINE('do not delete pargma table');
END;
/
```

Pragma Exception_init is a compiler directive which is used to associated an Exception name to the predefined number.

USER DEFINED EXCEPTION:

These Exceptions are defined and controlled by the user. These Exceptions neither have predefined name nor have predefined number. Steps to handle user defined Exceptions.

Step1: Declare the Exception

```
declare
        out_of_stock exception;
begin
        statements;
end;
```

Step2: Raised the Exception

```
if qty < 10 then
    raise out_of_stock;
end if;
```

Step3: Catch the Exception

```
exception
    when out_of_stock then
        -- handle the exception (that is reraised) in outer block
        ...
end;
```

Ex

```
Declare
    MY_EX1 EXCEPTION; --Step1
    L_SAL EMP.SAL%TYPE;
Begin
    DBMS_OUTPUT.PUT_LINE('welcome');
    Select SAL INTO L_SAL from emp where empno =
    &empno; IF L_SAL > 2000 THEN
        RAISE MY_EX1; --Step2
    ENDIF;
    DBMS_OUTPUT.PUT_LINE('The sal is ... '||L_sal);
    DBMS_OUTPUT.PUT_LINE('Thank you');
EXCEPTION
    WHEN MY_EX1 THEN --Step3
    DBMS_OUTPUT.PUT_LINE('Sal is two high');
END;
/
```

Note: When others should be the last handler of the exception section other wise we get a compiler ERROR.

RAISE_APPLICATION_ERROR:

RAISE_APPLICATION_ERROR is a procedure which is used to throw one error number and error message to the calling environment.

It internally performance rolls back.

ERROR number should be range of -20000 to -20999. ERROR message should be displayed less then or equal to 512 characters.

```
Declare
    L_sal emp.sal%TYPE;
Begin
    DBMS_OUTPUT.PUT_LINE('Welcome');
    Insert INTO dept values (08,'arun',70);
    Select sal INTO L_sal from emp where empno =
        7698; IF L_sal > 2000 THEN
            RAISE_APPLICATION_ERROR(-20150, 'SAL IS TOO HIGH');
        END IF;
    DBMS_OUTPUT.PUT_LINE('THE SAL IS...'||L_SAL);
END;
/
```

RESULT:

Thus the PL/SQL block that handles all types of exceptions has been verified and executed successfully.

Creation of Procedures.

Ex: No: 09

__ : __ : __

AIM:

To write a PL/SQL block to display the student name, marks whose average mark is above 60%.

ALGORITHM:

STEP1: Start the program.

STEP2: Create a table with table name stud_exam

STEP3: Insert the values into the table and Calculate total and average of each student

STEP4: Execute the procedure function the student who get above 60%.

STEP5: Display the total and average of student

STEP6: Terminate the application

SYNTAX

CREATE [OR REPLACE] PROCEDURE name [(parameter[,parameter, ...])]

{IS|AS}

[local declarations]

BEGIN

executable statements

[EXCEPTION

exception handlers]

END [name];

EXECUTION:

SQL> SET SERVEROUTPUT ON

I) PROGRAM:

PROCEDURE USING POSITIONAL PARAMETERS:

SETTING SERVEROUTPUT ON:

SQL> SET SERVEROUTPUT ON

SQL> CREATE OR REPLACE PROCEDURE PROC1 AS

BEGIN

DBMS_OUTPUT.PUT_LINE('Hello from procedure...');

END;

/

Output:

Procedure created.

SQL> EXECUTE PROC1

Hello from procedure...

PL/SQL procedure successfully completed.

II) PROGRAM:

Q: PROCEDURE USING NOTATIONAL PARAMETERS:

```
SQL> CREATE OR REPLACE PROCEDURE PROC2(N1 IN NUMBER,N2 IN NUMBER,TOT OUT NUMBER)
IS
        BEGIN
                TOT := N1 + N2;
        END;
        /
```

Output:

Procedure created.

```
SQL> VARIABLE T NUMBER

SQL> EXEC PROC2(33,66,:T)
PL/SQL procedure successfully completed.

SQL> PRINT T
        T
        ----------
        99
```

PROCEDURE FOR GCD NUMBERS

III) PROGRAM:

```
SQL> create or replace procedure
        pro is
                a number(3);
                b number(3);
                c number(3);
                d number(3);
        begin
                a:=&a;
                b:=&b;
                        if(a>b) then
                                c:=mod(a,b);
                                if(c=0) then
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(b);
                                else
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(c);
                                end if;
                        else
                                d:=mod(b,a);
                                if(d=0) then
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(a);
                                else
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(d);
                                end if;
                        end if;
        end;
/
```

Out put:

```
Enter value for a:
8 old 8: a:=&a;
new 8: a:=8;

Enter value for b:
16 old 9: b:=&b;
new 9: b:=16;

Procedure created.

SQL> set serveroutput on;

SQL> execute pro;
        GCD is
        8
```

RESULT:

Thus the implementation of PL/SQL procedure has been verified and executed successfully.

CREATION OF DATABASE TRIGGERS AND FUNCTIONS

10.1 Implementation of Triggers and its Application

Ex: No: 10

__ : __ : __

AIM:

To the Implementation of Triggers for the purpose of monitor the database object(table..etc) for any modification by the query user and/or the Application program.

ALGORITHM:

1. Start the program.
2. Create the table with its essential attributes.
3. Insert attribute values into the table.
4. Create the trigger for a particular table
5. Specify when the trigger is to be fired - before or after
6. Specify DML statement that invokes the trigger - UPDATE, DELETE, or INSERT
7. Specify the type of triger whether row-level trigger or not
8. Condition to filter rows.
9. PL/SQL block that is to be executed when trigger is fired.
10. Modify the specified table to fire the trigger.
11. Display the trigger message for the particular kind of modification
12. Stop the program.

DATABASE TRIGGERS

Triggers are similar to procedures or functions in that they are named PL/SQL blocks with declarative, executable, and exception handling sections. A trigger is executed implicitly whenever the triggering event happens. The act of executing a trigger is known as firing the trigger.

USE OF TRIGGERS

Maintaining complex integrity constraints not possible through declarative constraints enable at table creation.

Auditing information in a table by recording the changes made and who made them.

Automatically signaling other programs that action needs to take place when chages are made to a table.

Perform validation on changes being made to tables.

Automate maintenance of the database.

RESTRICTIONS ON TRIGGERES

Like packages, triggers must be stored as stand-alone objects in the database and cannot be local to a block or package.

A trigger does not accept arguments.

TYPES OF TRIGGERS

DML Triggers

Instead of Triggers

DDL Triggers

System Triggers

Suspend Triggers

CATEGORIES

Timing -- Before or After

Level -- Row or Statement

Row level trigger fires once for each row affected by the triggering statement. Row level trigger is identified by the FOR EACH ROW clause.

Statement level trigger fires once either before or after the statement.

TRIGGER SYNTAX

CREATE [OR REPLACE] TRIGGER
trigername {BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF columns]}
[OR {DELETE | INSERT |UPDATE [OF columns]}]...
ON table
[FOR EACH ROW [WHEN condition]]
[REFERENCING [OLD AS old] [NEW AS new]]

PL/SQL block

DML TRIGGERS

A DML trigger is fired on an INSERT, UPDATE, or DELETE operation on a database table. It can be fired either before or after the statement executes, and can be fired once per affected row, or once per statement.

The combination of these factors determines the types of the triggers. These are a total of 12 possible types (3 statements * 2 timing * 2 levels).

ORDER OF DML TRIGGER FIRING

Before statement level

Before row level

After row level

After statement level

Ex:

Suppose we have a follwing table.

SQL> select * from student;

NO	NAME	MARKS
1	a	100
2	b	200
3	c	300
4	d	400

Also we have triggering_firing_order table with firing_order as the field.

Ex:

```
CREATE OR REPLACE TRIGGER
TRIGGER1 before insert on student
        BEGIN
                insert into trigger_firing_order values('Before Statement Level');
END TRIGGER1;
```

Ex:

```
CREATE OR REPLACE TRIGGER
TRIGGER2 before insert on student
for each row
        BEGIN
                insert into trigger_firing_order values('Before Row Level');
END TRIGGER2;
```

Ex:

```
CREATE OR REPLACE TRIGGER
TRIGGER3 after insert on student
        BEGIN
                insert into trigger_firing_order values('After Statement Level');
END TRIGGER3;
```

Ex:

```
CREATE OR REPLACE TRIGGER
TRIGGER4 after insert on student
for each row
        BEGIN
                insert into trigger_firing_order values('After Row Level');
END TRIGGER4;
```

Output:

```
SQL> select * from
trigger_firing_order; no rows selected
SQL> insert into student
values(5,'e',500); 1 row created.
```

SQL> select * from trigger_firing_order;

FIRING_ORDER

Before Statement
Level Before Row
Level After Row Level
After Statement Level SQL>

select * from student;

NO	NAME	MARKS
1	a	100
2	b	200
3	c	300
4	d	400
5	e	500

Ex:

Suppose we have a table called marks with fields no, old_marks, new_marks.

```
CREATE OR REPLACE TRIGGER OLD_NEW
before insert or update or delete on student
for each row
        BEGIN
                insert into marks values(:old.no,:old.marks,:new.marks);
END OLD_NEW;
```

Output:

SQL> select * from student;

NO	NAME	MARKS
1	a	100
2	b	200
3	c	300
4	d	400
5	e	500

SQL> select * from
marks; no rows selected

SQL> insert into student
values(6,'f',600); 1 row created.

SQL> select * from student;

NO	NAME	MARKS
1	a	100
2	b	200
3	c	300
4	d	400
5	e	500
6	f	600

SQL> select * from marks;

NO	OLD_MARKS	NEW_MARKS
		600

SQL> update student set marks=555 where no=5; 1 row updated.

SQL> select * from student;

NO	NAME	MARKS
1	a	100
2	b	200
3	c	300
4	d	400
5	e	555
6	f	600

SQL> select * from marks;

NO	OLD_MARKS	NEW_MARKS
		600
5	500	555

SQL> delete student where no = 2; 1 row deleted.

SQL> select * from student;

NO	NAME	MARKS
1	a	100
3	c	300
4	d	400
5	e	555
6	f	600

SQL> select * from marks;

NO	OLD_MARKS	NEW_MARKS
		600
5	500	555
2	200	

REFERENCING CLAUSE

If desired, you can use the REFERENCING clause to specify a different name for :old ane :new. This clause is found after the triggering event, before the WHEN clause.

Syntax:

REFERENCING [old as old_name] [new as new_name]

Ex:

```
CREATE OR REPLACE TRIGGER REFERENCE_TRIGGER
before insert or update or delete on student
referencing old as old_student new as
new_student for each row
    BEGIN
        insert into
        marks values(:old_student.no,:old_student.marks,:new_student.marks);
END REFERENCE_TRIGGER;
```

WHEN CLAUSE

WHEN clause is valid for row-level triggers only. If present, the trigger body will be executed only for those rows that meet the condition specified by the WHEN clause.

Syntax:

WHEN *trigger_condition*;

Where *trigger_condition* is a Boolean expression. It will be evaluated for each row. The *:new* and *:old* records can be referenced inside *trigger_condition* as well, but like REFERENCING, the colon is not used there. The colon is only valid in the trigger body.

Ex:

```
CREATE OR REPLACE TRIGGER WHEN_TRIGGER
before insert or update or delete on student
referencing old as old_student new as
new_student for each row
    when (new_student.marks > 500)
    BEGIN
        insert into
        marks values(:old_student.no,:old_student.marks,:new_student.marks);
END WHEN_TRIGGER;
```

TRIGGER PREDICATES

There are three Boolean functions that you can use to determine what the operation is. The predicates are

INSERTING

UPDATING

DELETING

Ex:

```
CREATE OR REPLACE TRIGGER PREDICATE_TRIGGER
before insert or update or delete on student
        BEGIN
        if inserting then
                insert into predicates
        values('I'); elsif updating then
                insert into predicates
        values('U'); elsif deleting then
                insert into predicates values('D');
        end if;
END PREDICATE_TRIGGER;
```

Output:

```
SQL> delete student where
no=1; 1 row deleted.

SQL> select * from predicates;
MSG
---------------
D

SQL> insert into student
values(7,'g',700); 1 row created.

SQL> select * from predicates;
MSG
---------------
D
I
```

SQL> update student set marks = 777 where
no=7; 1 row updated.

SQL> select * from predicates;

MSG

D

I

U

INSTEAD-OF TRIGGERS

Instead-of triggers fire instead of a DML operation. Also, instead-of triggers can be defined only on views. Instead-of triggers are used in two cases:

To allow a view that would otherwise not be modifiable to be modified. To modify the columns of a nested table column in a view.

RESULT:

Thus the Implementation of Triggers and its applications to monitor the modification in database has been verified and executed successfully.

10.2 IMPLEMENTATION OF FUNCTIONS AND ITS APPLICATION

Ex: No: 10.2

__ : __ : __

AIM:

To write the PL/SQL block for the implementation of functions and its application.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create the table with its essential attributes.

STEP 3: Insert attribute values into the table.

STEP 4: Create the function with necessary arguments and return data types.

STEP 5: create the PL/SQL block to call / use the function.

STEP 6: Execute the PL/SQL program.

STEP 7: Give the input values

STEP 8: Extract/process the information from the function.

STEP 9: Stop the program.

STORED FUNCTION

A function is similar to procedure, except that it returns a value. The calling program should use the value returned by the function.

CREATE FUNCTION

The **create function** command is used to create a stored function.

SYNTAX:

CREATE [OR REPLACE] FUNCTION **name**

[(parameter[,parameter, ...])]

RETURN datatype

{IS | AS}

[local declarations]

BEGIN

executable statements

RETURN value;

[EXCEPTION

exception handlers]

END [name];

Note:

OR REPLACE is used to create a function even though a function with the same name already exists

RETURN datatype specifies the type of data to be returned by the function.

RETURN statement in the executable part returns the value. The value must be of the same type as the return type specified using RETURN option in the header.

User-defined PL/SQL functions can be used in SQL in the same manner as the standard functions such as ROUND and SUBSTR

Q 1: To write a PL/SQL block to implementation of factorial using function

I) PROGRAM:

```
SQL>create function fnfact(n
        number) return number is
        b number;
                begin
                        b:=1;
                        for i in 1..n
                                loop
                                        b:=b*i;
                                end loop;
                        return b;
                end;
/

SQL>Declare
                n number:=&n;
                y number;
        begin
                dbms_output.put_line(y);
        end;
/
```

Output:

Function created.

Enter value for n: 5

old 2: n

number:=&n; **new 2:**

n number:=5; 120

PL/SQL procedure successfully completed.

Q2:create a function which count total no.of employees having salary less than 6000.

/*function body*/

```
Create or replace function count_emp(esal number)return number as
        Cursor vin_cur as Select empno,sal from emp;
        Xno emp.empno%type;
        Xsal emp.sal%type;
        C number;
Begin
        Open vin_cur;
        C:=0;
        loop
                fetch vin_cur into xno,xsal;
                        if(xsal<esal) then
                                c:=c+1;
                        end if;
                exit when
        vin_cur%notfound; end loop;
        close vin_cur;
        return c;
end;
/
```

Function created.

/*function specification*/

```
Declare
        Ne number;
        Xsal number;
Begin
        Ne:=count_emp(xsal);
        Dbms_output.put_line(xsal);
        Dbma_output.put_line('there are '||ne||;employees');
End;
/
```

OUTPUT

There are 8 employees.

Q2: To write a PL/SQL function to search an address from the given database

II) PROGRAM

```
SQL> create table phonebook (phone_no number (6) primary
        key, username varchar2(30),
        doorno varchar2(10),
        street varchar2(30),
        place varchar2(30),
        pincode char(6));
      table created.

SQL> insert into phonebook values(20312,'vijay','120/5D','bharathi
street','NGOcolony','629002'); 1 row created.

SQL> insert into phonebook values(29467,'vasanth','39D4','RK bhavan','sarakkal
vilai','629002'); 1 row created.

SQL> select * from phonebook;
PHONE_NO   USERNAME        DOORNO      STREET       PLACE         PINCODE
---------- ---------------- ----------- ----------- ---------- ----------
20312      vijay           120/5D      bharathi street NGO colony 629002
29467      vasanth         39D4        RK bhavan    sarakkal vilai 629002

SQL> create or replace function findAddress(phone in number) return varchar2 as address
        varchar2(100); begin
            select username||','||doorno ||','||street ||','||place||','||pincode into address from
            phonebook where phone_no=phone;
                return address;
                exception
            when no_data_found then return 'address not found';
        end;
/

Function created.
```

```
declare
        address varchar2(100);
begin
        address:=findaddress(20312);
        dbms_output.put_line(address);
end;
/
```

OUTPUT 1:

Vijay,120/5D,bharathi street,NGO colony,629002

PL/SQL procedure successfully completed.

```
declare
        address
        varchar2(100); begin
                address:=findaddress(23556);
                dbms_output.put_line(address);
        end;
/
```

OUTPUT2:

Address not found

PL/SQL procedure successfully completed.

Result:

Thus the implementation of functions and its applications has been executed successfully.

SUPPLEMENT - A

DBMS LAB VIVA QUESTIONS

1. What is database?

A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

2. What is DBMS?

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

3. What is a Database system?

The database and DBMS software together is called as Database system.

4. What are the advantages of DBMS?

1. Redundancy is controlled.
2. Unauthorised access is restricted.
3. Providing multiple user interfaces.
4. Enforcing integrity constraints.
5. Providing backup and recovery.

5. What is the disadvantage in File Processing System?

1. Data redundancy and inconsistency.
2. Difficult in accessing data.
3. Data isolation.
4. Data integrity.
5. Concurrent access is not possible.
6. Security Problems.

6. Describe the three levels of data abstraction?

The are hree levels of abstraction:

1. Physical level: The lowest level of abstraction describes how data are stored.

2. Logical level: The next higher level of abstraction, describes what data are stored in database and what relationship among those data.

3. View level: The highest level of abstraction describes only part of entire database.

7. Define the "integrity rules"?

There are two Integrity rules.

1. Entity Integrity: States that "Primary key cannot have NULL value"

2. Referential Integrity: States that "Foreign Key can be either a NULL value or should be Primary Key value of other relation.

8. What is extension and intension?

1. Extension: It is the number of tuples present in a table at any instance. This is time dependent.

2. Intension: It is a constant value that gives the name, structure of table and the constraints laid on it.

9. What is System R? What are its two major subsystems?

System R was designed and developed over a period of 1974-79 at IBM San Jose Research Center. It is a prototype and its purpose was to demonstrate that it is possible to build a Relational System that can be used in a real life environment to solve real life problems, with performance at least comparable to that of existing system.

Its two subsystems are

1. Research Storage

2. System Relational Data System.

10. How is the data structure of System R different from the relational structure? Unlike Relational systems in System R

1. Domains are not supported
2. Enforcement of candidate key uniqueness is optional
3. Enforcement of entity integrity is optional
4. Referential integrity is not enforced

11. What is Data Independence?

Data independence means that "the application is independent of the storage structure and access strategy of data". In other words, The ability to modify the schema definition in one level should not affect the schema definition in the next higher level.

Two types of Data Independence:

1. Physical Data Independence: Modification in physical level should not affect the logical level.
2. Logical Data Independence: Modification in logical level should affect the view level. NOTE: Logical Data Independence is more difficult to achieve

12. What is a view? How it is related to data independence?

A view may be thought of as a virtual table, that is, a table that does not really exist in its own right but is instead derived from one or more underlying base table. In other words, there is no stored file that direct represents the view instead a definition of view is stored in data dictionary.

Growth and restructuring of base tables is not reflected in views. Thus the view can insulate users from the effects of restructuring and growth in the database. Hence accounts for logical data independence.

13. What is Data Model?

A collection of conceptual tools for describing data, data relationships data semantics and constraints.

14. What is E-R model?

This data model is based on real world that consists of basic objects called entities and of relationship

among these objects. Entities are described in a database by a set of attributes.

15. What is Object Oriented model?

This model is based on collection of objects. An object contains values stored in instance variables with in the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain same types of values and the same methods are grouped together into classes.

16. What is an Entity?

It is a 'thing' in the real world with an independent existence.

17. What is an Entity type?

It is a collection (set) of entities that have same attributes.

18. What is an Entity set?

It is a collection of all entities of particular entity type in the database.

19. What is an Extension of entity type?

The collections of entities of a particular entity type are grouped together into an entity set.

20. What is Weak Entity set?

An entity set may not have sufficient attributes to form a primary key, and its primary key compromises of its partial key and primary key of its parent entity, then it is said to be Weak Entity set.

21. What is an attribute?

It is a particular property, which describes the entity.

22. What is a Relation Schema and a Relation?

A relation Schema denoted by R(A1, A2, ..., An) is made up of the relation name R and the list of attributes Ai that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples (t1, t2, t3, ..., tn). Each tuple is an ordered list of n-values t=(v1,v2, ..., vn).

23. What is degree of a Relation?

It is the number of attribute of its relation schema.

24. What is Relationship?

It is an association among two or more entities.

25. What is Relationship set?

The collection (or set) of similar relationships.

26. What is Relationship type?

Relationship type defines a set of associations or a relationship set among a given set of entity types.

27. What is degree of Relationship type?

It is the number of entity type participating.

28. What is DDL (Data Definition Language)?

A data base schema is specifies by a set of definitions expressed by a special language called DDL.

29. What is VDL (View Definition Language)?

It specifies user views and their mappings to the conceptual schema.

30. What is SDL (Storage Definition Language)?

This language is to specify the internal schema. This language may specify the mapping between two schemas.

31. What is Data Storage - Definition Language?

The storage structures and access methods used by database system are specified by a set of definition in a special type of DDL called data storage-definition language.

32. What is DML (Data Manipulation Language)?

This language that enable user to access or manipulate data as organised by appropriate data model.

1. Procedural DML or Low level: DML requires a user to specify what data are needed and how to get those data.

2. Non-Procedural DML or High level: DML requires a user to specify what data are needed without specifying how to get those data.

33. What is DML Compiler?

It translates DML statements in a query language into low-level instruction that the query evaluation engine can understand.

34. What is Query evaluation engine?

It executes low-level instruction generated by compiler.

35. What is DDL Interpreter?

It interprets DDL statements and record them in tables containing metadata.

36. What is Record-at-a-time?

The Low level or Procedural DML can specify and retrieve each record from a set of records. This retrieve of a record is said to be Record-at-a-time.

37. What is Set-at-a-time or Set-oriented?

The High level or Non-procedural DML can specify and retrieve many records in a single DML statement. This retrieve of a record is said to be Set-at-a-time or Set-oriented.

38. What is Relational Algebra?

It is procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation.

39. What is Relational Calculus?

It is an applied predicate calculus specifically tailored for relational databases proposed by E.F. Codd.

E.g. of languages based on it are DSL ALPHA, QUEL.

40. How does Tuple-oriented relational calculus differ from domain-oriented relational calculus?

1. The tuple-oriented calculus uses a tuple variables i.e., variable whose only permitted values are tuples of that relation. E.g. QUEL

2. The domain-oriented calculus has domain variables i.e., variables that range over the underlying domains instead of over relation. E.g. ILL, DEDUCE.

41. What is normalization?

It is a process of analysing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties

(1).Minimizing redundancy,

(2). Minimizing insertion, deletion and update anomalies.

42. What is Functional Dependency?

A Functional dependency is denoted by X Y between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuple that can form a relation state r of R. The constraint is for any two tuples t1 and t2 in r if t1[X] = t2[X] then they have t1[Y] = t2[Y]. This means the value of X component of a tuple uniquely determines the value of component Y.

43. What is Lossless join property?

It guarantees that the spurious tuple generation does not occur with respect to relation schemas after decomposition.

44. What is 1 NF (Normal Form)?

The domain of attribute must include only atomic (simple, indivisible) values.

45. What is Fully Functional dependency?

It is based on concept of full functional dependency. A functional dependency X Y is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

46. What is 2NF?

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

47. What is 3NF?

A relation schema R is in 3NF if it is in 2NF and for every FD X A either of the following is true

1. X is a Super-key of R.

2. A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

48. What is BCNF (Boyce-Codd Normal Form)?

A relation schema R is in BCNF if it is in 3NF and satisfies an additional constraint that for every FD X A, X must be a candidate key.

49. What is 4NF?

A relation schema R is said to be in 4NF if for every Multivalued dependency X Y that holds over R, one of following is true.

1.) X is subset or equal to (or) XY = R.

2.) X is a super key.

50. What is 5NF?

A Relation schema R is said to be 5NF if for every join dependency {R1, R2, ..., Rn} that holds R, one the following is true

1.) Ri = R for some i.

2.) The join dependency is implied by the set of FD, over R in which the left side is key of R.

51. What is Domain-Key Normal Form?

A relation is said to be in DKNF if all constraints and dependencies that should hold on the the constraint can be enforced by simply enforcing the domain constraint and key constraint on the relation.

52. What are partial, alternate,, artificial, compound and natural key?

1. Partial Key:

It is a set of attributes that can uniquely identify weak entities and that are related to same owner entity. It is sometime called as Discriminator.

2. Alternate Key:

All Candidate Keys excluding the Primary Key are known as Alternate Keys.

3. Artificial Key:

If no obvious key, either stand alone or compound is available, then the last resort is to simply create a key, by assigning a unique number to each record or occurrence. Then this is known as developing an artificial key.

4. Compound Key:

If no single data element uniquely identifies occurrences within a construct, then combining multiple elements to create a unique identifier for the construct is known as creating a compound key.

5. Natural Key:

When one of the data elements stored within a construct is utilized as the primary key, then it is called the natural key.

53. What is indexing and what are the different kinds of indexing?

Indexing is a technique for determining how quickly specific data can be found.

Types:

1. Binary search style indexing
2. B-Tree indexing
3. Inverted list indexing
4. Memory resident table
5. Table indexing

54. What is system catalog or catalog relation? How is better known as?

A RDBMS maintains a description of all the data that it contains, information about every relation and index that it contains. This information is stored in a collection of relations maintained by the system called metadata. It is also called data dictionary.

55. What is meant by query optimization?

The phase that identifies an efficient execution plan for evaluating a query that has the least estimated cost is referred to as query optimization.

56. What is durability in DBMS?

Once the DBMS informs the user that a transaction has successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called durability.

57. What do you mean by atomicity and aggregation?

1. Atomicity:

Either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions. DBMS ensures this by undoing the actions of incomplete transactions.

2. Aggregation:

A concept which is used to model a relationship between a collection of entities and relationships. It is used when we need to express a relationship among relationships.

58. What is a Phantom Deadlock?

In distributed deadlock detection, the delay in propagating local information might cause the deadlock detection algorithms to identify deadlocks that do not really exist. Such situations are called phantom deadlocks and they lead to unnecessary aborts.

59. What is a checkpoint and When does it occur?

A Checkpoint is like a snapshot of the DBMS state. By taking checkpoints, the DBMS can reduce the amount of work to be done during restart in the event of subsequent crashes.

60. What are the different phases of

transaction? Different phases are

1.) Analysis phase,

2.) Redo Phase,

3.) Undo phase.

61. What do you mean by flat file database?

It is a database in which there are no programs or user access languages. It has no cross-file capabilities but is user-friendly and provides user-interface management.

62. What is "transparent DBMS"?

It is one, which keeps its Physical Structure hidden from user.

63. What is a query?

A query with respect to DBMS relates to user commands that are used to interact with a data base. The query language can be classified into data definition language and data manipulation language.

64. What do you mean by Correlated subquery?

Subqueries, or nested queries, are used to bring back a set of rows to be used by the parent query. Depending on how the subquery is written, it can be executed once for the parent query or it can be executed once for each row returned by the parent query. If the subquery is executed for each row of the parent, this is called a correlated subquery.

A correlated subquery can be easily identified if it contains any references to the parent subquery columns in its WHERE clause. Columns from the subquery cannot be referenced anywhere else in the parent query. The following example demonstrates a non-correlated subquery.

Example:

Select * From CUST Where '10/03/1990' IN (Select ODATE From ORDER

Where CUST.CNUM = ORDER.CNUM)

65. What are the primitive operations common to all record management systems? Addition, deletion and modification.

66. Name the buffer in which all the commands that are typed in are stored?

'Edit' Buffer.

67. What are the unary operations in Relational Algebra? PROJECTION and SELECTION.

68. Are the resulting relations of PRODUCT and JOIN operation the same? No.

PRODUCT: Concatenation of every row in one relation with every row in another.

JOIN: Concatenation of rows from one relation and related rows from another.

69. What is RDBMS KERNEL?

Two important pieces of RDBMS architecture are the kernel, which is the software, and the data dictionary, which consists of the system-level data structures used by the kernel to manage the database You

might think of an RDBMS as an operating system (or set of subsystems), designed specifically for controlling data access; its primary functions are storing, retrieving, and securing data.

An RDBMS maintains its own list of authorized users and their associated privileges; manages memory caches and paging; controls locking for concurrent resource usage; dispatches and schedules user requests; and manages space usage within its table-space structures.

70. Name the sub-systems of a RDBMS. I/O, Security,

Language Processing,

Process Control,

Storage Management,

Logging and Recovery,

Distribution Control,

Transaction Control,

Memory Management,

Lock Management.

71. Which part of the RDBMS takes care of the data dictionary? How?

Data dictionary is a set of tables and database objects that is stored in a special area of the database and maintained exclusively by the kernel.

72. What is the job of the information stored in data-dictionary?

The information in the data dictionary validates the existence of the objects, provides access to them, and maps the actual physical storage location.

73. How do you communicate with an RDBMS?

You communicate with an RDBMS using Structured Query Language (SQL).

74. Define SQL and state the differences between SQL and other conventional programming Languages.

SQL is a nonprocedural language that is designed specifically for data access operations on normalized

relational database structures. The primary difference between SQL and other conventional programming languages is that SQL statements specify what data operations should be performed rather than how to perform them.

75. Name the three major set of files on disk that compose a database in Oracle.

There are three major sets of files on disk that compose a database. All the files are binary. These are

1.) Database files

2.) Control files

3.) Redo logs

The most important of these are the database files where the actual data resides. The control files and the redo logs support the functioning of the architecture itself. All three sets of files must be present, open, and available to Oracle for any data on the database to be useable. Without these files, you cannot access the database, and the database administrator might have to recover some or all of the database using a backup, if there is one.

76. What is database Trigger?

A database trigger is a PL/SQL block that can defined to automatically execute for insert, update, and delete statements against a table. The trigger can e defined to execute once for the entire statement or once for every row that is inserted, updated, or deleted. For any one table, there are twelve events for which you can define database triggers. A database trigger can call database procedures that are also written in PL/SQL.

77. What are stored-procedures? And what are the advantages of using them?

Stored procedures are database objects that perform a user defined operation. A stored procedure can have a set of compound SQL statements. A stored procedure executes the SQL commands and returns the result to the client. Stored procedures are used to reduce network traffic.

78. What is Storage Manager?

It is a program module that provides the interface between the low-level data stored in database, application programs and queries submitted to the system.

79. What is Buffer Manager?

It is a program module, which is responsible for fetching data from disk storage into main memory and deciding what data to be cache in memory.

80. What is Transaction Manager?

It is a program module, which ensures that database, remains in a consistent state despite system failures and concurrent transaction execution proceeds without conflicting.

81. What is File Manager?

It is a program module, which manages the allocation of space on disk storage and data structure used to represent information stored on a disk.

82. What is Authorization and Integrity manager?

It is the program module, which tests for the satisfaction of integrity constraint and checks the authority of user to access data.

83. What are stand-alone procedures?

Procedures that are not part of a package are known as stand-alone because they independently defined. A good example of a stand-alone procedure is one written in a SQL*Forms application. These types of procedures are not available for reference from other Oracle tools. Another limitation of stand-alone procedures is that they are compiled at run time, which slows execution.

84. What are cursors give different types of cursors?

PL/SQL uses cursors for all database information accesses statements. The language supports the use two types of cursors

1.) Implicit

2.) Explicit

85. What is cold backup and hot backup (in case of Oracle)?

1. Cold Backup: It is copying the three sets of files (database files, redo logs, and control file) when the instance is shut down. This is a straight file copy, usually from the disk directly to tape. You must shut down the instance to guarantee a consistent copy. If a cold backup is performed, the only option available in the event of data file loss is restoring all the files from the latest backup. All work performed on the database since the last backup is lost.

2. Hot Backup: Some sites (such as worldwide airline reservations systems) cannot shut down the database while making a backup copy of the files. The cold backup is not an available option.

86. What is meant by Proactive, Retroactive and Simultaneous Update.

1. Proactive Update: The updates that are applied to database before it becomes effective in real world.

2. Retroactive Update: The updates that are applied to database after it becomes effective in real world.

3. Simulatneous Update: The updates that are applied to database at the same time when it becomes effective in real world.

SUPPLEMENT - B

DATABASE APPLICATION

Employees Management Application

This application is used to keep track of information about employees of a company. It also stores the information about departments and leaves taken by employees. You are required to create tables (as shown below) and insert data into each of the table.

Apart from giving you an idea about how to create tables with constraints, it also enables you to understand how to create queries, pl/sql programs, stored procedures and functions and database triggers.

However, note, this sample collection of tables is only for learning purpose and they are hypothetical.

Required Tables

The following are the set of tables to be created to store the required information.

Table Name	Meaning
DEPT	Stores the details of departments of the company.
EMPLOYEE	Stores information about all the employees of the company.
LEAVES	Stores information about types of leaves available
EMP_LEAVES	Stores information about leaves taken by the employees.

Structure of Tables
The following is the structure of each of the required table.

DEPT Table
Stores information about all the departments of the company.

Column Name	Datatype	Meaning
DEPTNO	number(2)	Department Number
DEPTNAME	varchar2(20)	Department Name
HOD	varchar2(20)	Head of the department

Constraints

DEPTNO is primary key
DEPTNAME must be unique

EMPLOYEE table

Contains information about all the employees of the company.

Column Name	Datatype	Meaning
EMPNO	number(5)	Employee Number
EMPNAME	varchar2(20)	Employee Name
SAL	number(6)	Basic Salary
DEPTNO	number(2)	Department to which employee belongs
DJ	Date	Date of joining the company
DESG	varchar2(20)	Designation of the employee

Constraints

EMPNO is primary key
SAL must be >= 1000
DEPTNO is foreign key referencing DEPTNO of DEPT table

LEAVES Table

Contains information about the types of leaves available in the company.

Column Name	Datatype	Meaning
LEAVETYPE	char(1)	Code for the type of leave
LEAVENAME	varchar2(20)	Description of the type of the leave
NOLEAVES	number(2)	Number of leaves allotted to each employee for a leave type

Contraints

LEAVETYPE is primary key
NOLEAVES must be <= 20

EMP_LEAVES Table

Contains information about the leaves taken by employees.

Column Name	Datatype	Meaning
EMPNO	number(5)	Employee number of the employee who has taken leave
LEAVETYPE	char(1)	Type of the leave taken by the employee
STDATE	Date	Starting date of the leave
ENDDATE	Date	Ending date of the leave

Contraints

EMPNO + STDATE is primary key LEAVETYPE is not null
EMPNO is foreign key referencing EMPNO in EMPLOYEES table
LEAVETYPE is foreign key referencing LEAVETYPE in LEAVES table STDATE must be <= ENDDATE

Creating Tables

The following scirpt is used to create sample tables. Click here to download the script and run the script at SQL> promt using START command of SQL*PLUS>

```
REM   ****** script to create tables realted to EMPLOYEES application     ******
REM   ******                          AUTHOR : P.SRIKANTH                      ******
REM   ******                          DATE: 23-AUG-2001                           ******
REM   ************************************************************************
```

```
drop table emp_leaves cascade constraints;
drop table employee cascade constraints;
drop table dept cascade constraints;
drop table leaves cascade constraints;
```

```
CREATE TABLE DEPT
(
  DEPTNO NUMBER(2) CONSTRAINT DEPT_PK PRIMARY KEY,
  DEPTNAME VARCHAR2(20) CONSTRAINT DEPT_DEPTNAME_U UNIQUE,
  HOD VARCHAR2(20)
);

CREATE TABLE LEAVES
(
   LEAVETYPE CHAR(1) CONSTRAINT LEAVES_PK PRIMARY KEY,
   LEAVENAME VARCHAR2(20),
   NOLEAVES   NUMBER(2) CONSTRAINT LEAVES_NOLEAVES_CHK   CHECK ( NOLEAVES <= 20)
);


CREATE TABLE EMPLOYEE
(
   EMPNO NUMBER(5) CONSTRAINT EMPLOYEE_PK PRIMARY KEY,
   EMPNAME VARCHAR2(20),
   SAL    NUMBER(5)   CONSTRAINT EMPLOYEE_SAL_CHK  CHECK ( SAL >= 1000),
   DEPTNO NUMBER(2) CONSTRAINT EMPLOYEE_DEPTNO_FK REFERENCES DEPT(DEPTNO),
   DESG VARCHAR2(20),
   DJ     DATE,
);


CREATE TABLE EMP_LEAVES
(
   EMPNO NUMBER(5)  CONSTRAINT EMP_LEAVES_EMPNO_FK   REFERENCES EMPLOYEE(EMPNO),
   LEAVETYPE CHAR(1) CONSTRAINT EMP_LEAVES_LEAVETYPE_FK    REFERENCES LEAVES(LEAVETYPE)
                CONSTRAINT EMP_LEAVES_LEAVETYPE_NN  NOT NULL,
   STDATE DATE,
   ENDDATE DATE,
   CONSTRAINT EMP_LEAVES_PK  PRIMARY KEY (EMPNO,STDATE),
   CONSTRAINT EMP_LEAVES_DATES_CHK CHECK (STDATE <= ENDDATE)
);
```

Loading data into tables

The following script inserts a few rows into sample tables. Click here to download the script to create sample data or copy the script given below into notepad, save it with .SQL extension and run it at SQL> using START command.

```
REM   ****** script to insert sample data into EMPLOYEES application      ******
REM   ******                                  AUTHOR : P.SRIKANTH                                  ******
REM   ******                                  DATE: 23-AUG-2001                                        ******
REM   ************************************************************************


rem remove all existing rows first

DELETE FROM EMP_LEAVES;
DELETE FROM EMPLOYEE;
DELETE FROM DEPT;
DELETE FROM LEAVES;


INSERT INTO LEAVES VALUES('S','SICK',15);
INSERT INTO LEAVES VALUES('C','CASUAL',15);
INSERT INTO LEAVES VALUES('E','EARNING',5);
INSERT INTO LEAVES VALUES('O','OVERTIME',5);


INSERT INTO DEPT VALUES(1,'MAINFRAME','GEORGE');
INSERT INTO DEPT VALUES(2,'CLIENT/SERVER','BILL');
INSERT INTO DEPT VALUES(3,'SYSTEMS','GARRY');
INSERT INTO DEPT VALUES(4,'INTERNET','PAUL');
INSERT INTO DEPT VALUES(5,'ACCOUNTS','ANDY');


INSERT INTO EMPLOYEE VALUES(101,'GEORGE',12000,1,'12-JUL-2001','PM');
INSERT INTO EMPLOYEE VALUES(102,'BILL',12000,2,'14-JUL-2001','PM');
INSERT INTO EMPLOYEE VALUES(103,'GARRY',15000,3,'1-JUL-2001','PM');
INSERT INTO EMPLOYEE VALUES(104,'PAUL',11000,4,'2-JUL-2001','PL');
INSERT INTO EMPLOYEE VALUES(105,'ANDY',7000,5,'25-JUN-2001','AM');
INSERT INTO EMPLOYEE VALUES(106,'KEATS',10000,1,'17-JUL-2001','SA');
INSERT INTO EMPLOYEE VALUES(107,'JOEL',8000,2,'15-JUL-2001','SP');
INSERT INTO EMPLOYEE VALUES(108,'ROBERTS',7500,2,'15-JUL-2001','PRO');
INSERT INTO EMPLOYEE VALUES(109,'HERBERT',8000,4,'22-JUL-2001','SA');
INSERT INTO EMPLOYEE VALUES(110,'MICHEAL',6000,4,'15-JUL-2001','PRO');


INSERT INTO EMP_LEAVES VALUES(102,'S','23-JUL-2001','25-JUL-2001');
INSERT INTO EMP_LEAVES VALUES(104,'C','24-JUL-2001','25-JUL-2001');
INSERT INTO EMP_LEAVES VALUES(104,'S','28-JUL-2001','29-JUL-2001');
INSERT INTO EMP_LEAVES VALUES(101,'C','27-JUL-2001','28-JUL-2001');
INSERT INTO EMP_LEAVES VALUES(106,'O','28-JUL-2001','29-JUL-2001');
INSERT INTO EMP_LEAVES VALUES(109,'C','1-AUG-2001','2-AUG-2001');
INSERT INTO EMP_LEAVES VALUES(103,'C','2-AUG-2001','5-AUG-2001');
INSERT INTO EMP_LEAVES VALUES(105,'S','17-AUG-2001',NULL);
INSERT INTO EMP_LEAVES VALUES(108,'S','23-AUG-2001',NULL);


COMMIT;
```

Queries Related To Employees Management Application

Q: DISPLAY EMPLOYEES WHO HAVE JOINED IN THE LAST 15 DAYS

```
SELECT * FROM EMPLOYEE
WHERE SYSDATE - DJ <= 15;
```

Q: DISPLAY EMPLOYEES WHO HAVE JOINED TODAY

```
SELECT * FROM EMPLOYEE
WHERE TRUNC(SYSDATE) = TRUNC(DJ);
```

NOTE:

TRUNC FUNCTION IS REQUIRED IN ORDER TO IGNORE TIME DIFFERENCE BETWEEN TWO DATES.

Q: DISPLAY WHO HAVE JOINED IN THE LAST WEEK OF THE MONTH

```
SELECT * FROM EMPLOYEE
WHERE DJ >= LAST_DAY(DJ) - 7;
```

Q: DISPLAY WHEN EMPLOYEE 102 HAS TAKEN HIS FIRST SALARY

```
SELECT EMPNAME, LAST_DAY(DJ) + 1 "FIRST SALARY DATE"
FROM EMPLOYEE
WHERE EMPNO = 102;
```

Q: DELETE EMPLOYEES WHO HAVE JOINED THE CURRENT MONTH

```
DELETE FROM EMPLOYEE WHERE TO_CHAR(SYSDATE,'MMYYYY') = TO_CHAR(DJ,'MMYYYY')
```

Q: DETAILS OF EMPLOYEES WHOSE SALARY RANGE IS BETWEEN 12,000 TO 14,000

```
SELECT * FROM EMPLOYEE
WHERE SAL BETWEEN 12000 AND 14000;
```

Q: DETAILS OF EMPLOYEES WHO BELONG TO DEPARTMENT 1 OR 3

```
SELECT * FROM EMPLOYEE
WHERE DEPTNO IN (1,3);
```

Q: SELECT NAMES OF EMPLOYEES WHOSE NAMES START WITH 'M'

```
SELECT EMPNAME
FROM EMPLOYEE WHERE EMPNAME LIKE 'M%';
```

Q: DELETE THOSE EMPLOYEES WHERE NAME HAS THE CHARACTER 'A'

```
DELETE FROM EMPLOYEE
WHERE EMPNAME LIKE '%A%';
```

Q: SELECT EMPLOYEES WHERE SECOND CHARACTER IN NAME IS 'S'

```
SELECT * FROM EMPLOYEE

WHERE EMPNAME LIKE '_S%';
```

Q: SELECT THOSE EMPLOYEES WHOSE SALARY IS NOT KNOWN

```
SELECT * FROM EMPLOYEE

WHERE  SAL IS NULL;
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO HAVE JOINED IN THE LAST 20 DAYS

```
SELECT * FROM EMPLOYEE

WHERE  SYSDATE - DJ <= 20;
```

Q: DISPLAY THE DETAILS OF LEAVES IF THE NUMBER OF LEAVES IS MORE THAN 10

```
SELECT * FROM EMP_LEAVES

WHERE ENDDATE - STDATE > 10;
```

Q: DISPLAY EMPNO,EMPNAME,DATE OF JOINING,NUMBER OF MONTHS OF EXPERIENCE AND BASIC SALARY

```
SELECT EMPNO, EMPNAME, DJ, MONTHS_BETWEEN(SYSDATE,DJ) EXP, SAL

FROM EMPLOYEE;
```

Q: DISPLAY DETAILS OF EMPLOYEES WHO ARE DRAWING MORE THAN 10000 AND THE DESIGNATION IS CONTAINING MORE THAN 3 LETTERS

```
SELECT * FROM EMPLOYEE

WHERE  SAL > 10000 AND  LENGTH(DESG) > 3;
```

Q: DISPLAY DETAILS OF EMPLOYEES WHOSE NAME IS CONTAINING MORE THAN ONE SPACE

```
SELECT * FROM EMPLOYEE

WHERE  INSTR(EMPNAME, ' ' , 1, 2) <> 0;
```

Q: DISPLAY DETAILS OF LEAVES WHERE THE LEAVE STARTED IN THE PREVIOUS MONTH AND THE LEAVE IS NOT YET COMPLETED

```
SELECT * FROM EMP_LEAVES

WHERE STDATE BETWEEN LAST_DAY( ADD_MONTHS(STDATE,-2)) + 1 AND

                LAST_DAY( ADD_MONTHS(STDATE,-1)) + 1

AND  ENDDATE IS NULL;
```

Q: DISPLAY DETAILS OF EMPLOYEES WHERE BASIC SALARY IS MORE THAN 10000 OR DESIGNATION IS PL AND EXPERIENCE IS MORE THAN 3 YEARS

```
SELECT * FROM EMPLOYEE

WHERE SAL > 10000 OR DESG = 'PL'  AND  MONTHS_BETWEEN(SYSDATE,DJ) > 36;
```

Q: DISPLAY EMPNO,NAME AND FIRST NAME OF THE EMPLOYEE AND WHEN EMPLOYEE HAS TAKEN HIS FIRST

SALARY SELECT SUBSTR(EMPNAME,1,INSTR(EMPNAME,' ') -1) FNAME, LAST_DAY(DJ) + 1

FROM EMPLOYEE;

Q: FIND THE AVERAGE SALARY OF THE EMPLOYEE WHO JOINED IN THE CURRENT YEAR

```
SELECT AVG(SAL)

FROM EMPLOYEE

WHERE TO_CHAR(DJ,'YYYY')  = TO_CHAR(SYSDATE,'YYYY');
```

Q: FIND THE AVERAGE SALARY OF EACH DEPARTMENT BY TAKING EMPLOYEES WHO EARN MORE THAN 10000

```
SELECT DEPTNO, AVG(SAL)

FROM EMPLOYEE

WHERE SAL > 10000

GROUP BY DEPTNO;
```

Q: DISPLAY DETAILS OF EMPLOYEES ALONG WITH BONUS WHICH WILL BE 100% ON SALARY FOR EMPLOYEES OF DEPARTMENT 1 AND 75% FOR OTHERS

```
SELECT EMPNO, EMPNAME, DESG, DJ, SAL, SAL * DECODE(DEPTNO,1,1.0,0.75) BONUS

FROM EMPLOYEE;
```

Q: DISPLAY DETAILS OF LEAVES TAKEN BY EMPLOYEES WHERE TYPE OF LEAVE IS 'S' AND LEAVE STARTED ON MONDAY

```
SELECT * FROM EMP_LEAVES

WHERE  LEAVETYPE = 'S' AND  TO_CHAR(STDATE,'fmDAY') = 'MONDAY';
```

Q: DISPLAY EMPNO AND NO. OF LEAVES TAKEN BY EMPLOYEE

```
SELECT EMPNO, SUM(ENDDATE-STDATE) "NO LEAVES"

FROM EMP_LEAVES

GROUP BY  EMPNO;
```

Q: DISPLAY DESIGNATION AND TOTAL SALARY OF THE EMPLOYEES OF DESIGNATION

```
SELECT DESG,SUM(SAL)

FROM EMPLOYEE

GROUP BY DESG;
```

Q: FIND THE SUM OF SALARIES IN EACH DESIGNATION IN EACH DEPARTMENT

```
SELECT DEPTNO, DESG, SUM(SAL)

FROM EMPLOYEE

GROUP BY DEPTNO, DESG;
```

Q: FIND THE AVERAGE SALARY OF EACH DEPARTMENT AND SELECT ONLY THOSE EMPLOYEES HAVING SALARY MORE THAN 10000

```
SELECT DEPTNO, AVG(SAL)

FROM EMPLOYEE

WHERE SAL > 10000

GROUP BY DEPTNO;
```

Q: DISPLAY MAXIMUM SALARY

```
SELECT MAX(SAL) FROM EMPLOYEE;
```

Q: DISPLAY EMPNO ,TYPE OF LEAVE,TOTAL NO OF LEAVES TAKEN

```
SELECT EMPNO,LEAVETYPE, SUM( ENDDATE -STDATE) "NO. LEAVES"

FROM EMP_LEAVES

GROUP BY EMPNO, LEAVETYPE;
```

Q: DISPLAY DEPTNO,MIN SALARY,MAX SALARY ,DIFFERENCE BETWEEN MAX AND MIN SALARY FOR THE DEPARTMENTS
THAT HAVE MORE THAN 2 EMPLOYEES

```
SELECT DEPTNO, MIN(SAL), MAX(SAL), MAX(SAL) - MIN(SAL)

FROM  EMPLOYEE

GROUP BY DEPTNO

HAVING COUNT(*) > 2;
```

Q: DISPLAY LEAVETYPE AND HOW MANY TIMES EACH EMPLOYEE HAS TAKEN LEAVE

```
SELECT LEAVETYPE, EMPNO, COUNT(*)

FROM EMP_LEAVES

GROUP BY LEAVETYPE,EMPNO;
```

Q: DISPLAY EMPNO OF THE EMPLOYEE WHO HAS TAKEN MORE THAN 2 LEAVES IN THE CURRENT MONTH

```
SELECT EMPNO

FROM EMP_LEAVES

WHERE TO_CHAR(STDATE,'MMYYYY')   = TO_CHAR(SYSDATE,'MMYYYY')

GROUP BY EMPNO

HAVING SUM(ENDDATE -STDATE) > 2;
```

Q: DISPLAY DESIGNATION THAT CONTAIN EITHER MORE THAN 5 EMPLOYEES OR AVERAGE SALARY MORE THAN 12000

```
SELECT DESG

FROM EMPLOYEE

GROUP BY DESG

HAVING COUNT(*) > 5 OR AVG(SAL) > 12000;
```

Q: DISPLAY THE TYPE OF LEAVE THAT IS TAKEN BY MORE THAN 3 EMPLOYEES

```
SELECT LEAVETYPE
FROM EMP_LEAVES
GROUP BY LEAVETYPE
HAVING  COUNT (DISTINCT EMPNO) > 3;
```

Q: DISPLAY EMPNO,EMPNAME,DATE OF JOINING,DEPTNAME, SALARY AND HOD

```
SELECT  EMPNO, EMPNAME, DJ, DEPTNAME,SAL, HOD
FROM    EMPLOYEE E, DEPT D
WHERE    E.DEPTNO =D.DEPTNO;
```

Q: DISPLAY EMPNO,STDATE,ENDDATE,LEAVENAME FOR ALL THE COMPLETED LEAVES

```
SELECT EMPNO, STDATE, ENDDATE, LEAVENAME
FROM   EMP_LEAVES EL, LEAVES L
WHERE  EL.LEAVETYPE = L.LEAVETYPE AND ENDDATE IS NOT NULL;
```

Q: DISPLAY DEPTNO,DEPTNAME,EMPNAME,YEARS OF EXPERIENCE FOR ALL THE EMPLOYEES WITH DESIG

'PRO' SELECT E.DEPTNO, DEPTNAME, EMPNAME, TRUNC(MONTHS_BETWEEN(SYSDATE,DJ) / 12)

```
FROM   EMPLOYEE E, DEPT D
WHERE  E.DEPTNO = D.DEPTNO AND  DESG ='PRO';
```

Q: DISPLAY EMPNO,EMPNAME,DEPTNAME,LEAVENAME,STDATE AND MAX NO. OF LEAVES IN THE CATEGORY

```
SELECT EL.EMPNO, EMPNAME, DEPTNAME, LEAVENAME,STDATE, NOLEAVES
FROM   EMPLOYEE E, DEPT D, EMP_LEAVES EL, LEAVES L
WHERE E.DEPTNO = D.DEPTNO AND EL.LEAVETYPE = L.LEAVETYPE
      AND EL.EMPNO = E.EMPNO;
```

Q: DISPLAY THE DETAILS OF LEAVES TAKEN BY EMPLOYEES WHO ARE HAVING 'DUKE' AS THE HEAD OF THE DEPARTMENT.

```
SELECT EL.*
FROM EMP_LEAVES EL, EMPLOYEE E,DEPT D
WHERE  E.EMPNO = EL.EMPNO
      AND E.DEPTNO = D.DEPTNO AND  HOD = 'DUKE';
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO HAVE JOINED AFTER EMPLOYEE 'WILLY' HAS

JOINED. SELECT E1.*

```
FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E2.EMPNAME = 'WILLY'
     AND  E1.DJ > E2.DJ;
```

Q: SELECT THE EMPLOYEES WHO HAVE TAKEN LEAVE IN THE PRESENT MONTH

```
SELECT * FROM EMPLOYEE
WHERE EMPNO IN
  ( SELECT EMPNO FROM EMP_LEAVES
    WHERE TO_CHAR(SYSDATE,'MMYYYY') =
  TO_CHAR(STDATE,'MMYYYY') );
```

Q: DISPLAY THE DETAILS OF DEPARTMENTS WHICH HAVE MORE THAN 2 EMPLOYEES

```
SELECT * FROM DEPT
WHERE DEPTNO IN
 ( SELECT DEPTNO
  FROM EMPLOYEE
  GROUP BY DEPTNO
  HAVING COUNT(*) > 2);
```

Q: DISLAY THE DETAILS OF EMPLOYEES WHO HAVE TAKEN MORE THAN 10 LEAVES

```
SELECT * FROM EMPLOYEE
WHERE EMPNO IN
  ( SELECT EMPNO
    FROM EMP_LEAVES
    GROUP BY EMPNO
    HAVING SUM( ENDDATE - STDATE) >
  10 );
```

Q: DISPLAY THE DETAILS OF DEPARTMENTS WHICH HAVE MORE THAN 3 EMPLOYEES JOINED IN THE CURRENT YEAR

```
SELECT * FROM DEPT
WHERE DEPTNO IN
  ( SELECT DEPTNO
    FROM EMPLOYEE
    WHERE TO_CHAR(DJ,'YYYY') = TO_CHAR(SYSDATE,'YYYY')
    GROUP BY DEPTNO
    HAVING COUNT(*) > 3
  );
```

Q: DISPLAY THE NAME OF THE EMPLOYEE DRAWING THE MAX SALARY

```
SELECT EMPNAME FROM EMPLOYEE
WHERE SAL =
  ( SELECT MAX(SAL)
    FROM EMPLOYEE
  );
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO HAS TAKEN MORE THAN 10 SICKLEAVES OR MORE THAN 15 LEAVES

```
SELECT * FROM EMPLOYEE
WHERE EMPNO IN
  ( SELECT EMPNO
    FROM  EMP_LEAVES
    WHERE LEAVETYPE='S'
    GROUP BY EMPNO
    HAVING SUM(ENDDATE - STDATE ) > 10
  )
  OR EMPNO IN
  ( SELECT EMPNO
    FROM  EMP_LEAVES
    GROUP BY EMPNO
    HAVING SUM(ENDDATE - STDATE ) > 15
  );
```

Q: DISPLAY EMPNO,EMPNAME,DESIGNATION AND DEPTNAME OF EMPLOYEES WHO HAVE NOT TAKEN ANY LEAVES IN THE CURRENT YEAR

```
SELECT EMPNO,EMPNAME,DESG,DEPTNAME
FROM  EMPLOYEE E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
      AND EMPNO NOT IN
       ( SELECT EMPNO
         FROM EMP_LEAVES
         WHERE TO_CHAR(STDATE,'YYYY') = TO_CHAR(SYSDATE,'YYYY')
       );
```

Q: DISPLAY THE DETAILS OF HOD'S

```
SELECT * FROM EMPLOYEE
WHERE EMPNAME IN
  ( SELECT HOD
    FROM DEPT
  );
```

Q: DISPLAY THE DEPARTMENTS IN WHICH EMPLOYEES HAVE TAKEN MAX NO OF LEAVES

```
SELECT * FROM DEPT
WHERE DEPTNO IN
 (
  SELECT DEPTNO
  FROM   EMP_LEAVES EL, EMPLOYEE E
  WHERE  EL.EMPNO = E.EMPNO
  GROUP  BY DEPTNO
  HAVING SUM(ENDDATE-STDATE) =
                      (
                       SELECT MAX(SUM(ENDDATE-STDATE))
                       FROM   EMP_LEAVES EL, EMPLOYEE E
                       WHERE  EL.EMPNO = E.EMPNO
                       GROUP  BY DEPTNO
                      )
 );
```

Q: DISPLAY EMPNO,NOOFLEAVES FOR ALL EMPLOYEES WHO ARE HEADED BY 'STEVE'

```
SELECT EMPNO, SUM(ENDDATE-STDATE)
FROM EMP_LEAVES
WHERE EMPNO IN
(
 SELECT EMPNO FROM EMPLOYEE
 WHERE  DEPTNO IN
  (
      SELECT DEPTNO
      FROM   DEPT
```

```
      WHERE  HOD = 'STEVE'
  )
)
GROUP BY EMPNO;
```

Q: DISPLAY DETAILS OF EMPLOYEES DRAWING TOP 2 HIGHEST SALARIES

```
SELECT * FROM EMPLOYEE E
  WHERE  2 > ( SELECT COUNT(*)
            FROM  EMPLOYEE
            WHERE SAL > E.SAL);
```

Q: DROP AN UNWANTED COLUMN FROM ANY TABLE

THIS IS DONE IN THREE STEPS.

1. CREATE TABLE NEWTABLE AS SELECTA,B,CFROMOLDTABLE;
2. DROP TABLE OLDTABLE
3. RENAME NEWTABLE TO OLDTABLE

Q: DISPLAY DETAILS OF DEPARTMENT IN WHICH ATLEAST ONE EMPLOYEE HAS TAKEN MORE NO. OF LEAVES THAN AVERAGE LEAVES OF ALL THE EMPLOYEES WHO JOINED IN THE CURRENT YEAR

```
SELECT * FROM DEPT
WHERE DEPTNO IN
   ( SELECT DEPTNO
     FROM  EMPLOYEE
     WHERE  EMPNO IN
            (
              SELECT EMPNO
              FROM  EMP_LEAVES
              GROUP  BY EMPNO
              HAVING SUM(ENDDATE - STDATE) >
              (
               SELECT  AVG(ENDDATE - STDATE)
               FROM    EMP_LEAVES
               WHERE   TO_CHAR(SYSDATE,'YYYY') = TO_CHAR(STDATE,'YYYY')
              )
            )
   );
```

Q: HOW MANY EMPLOYEES ARE EARNING MORE THAN THE AVERAGE SALARY OF

MANAGERS SELECT COUNT(*)

FROM EMPLOYEE

WHERE SAL > (SELECT AVG(SAL)

FROM EMPLOYEE

WHERE DESG = 'MANAGER');

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO BELONG TO DEPARTMENT 1 OR 3 AND DRAW MORE THAN 5000 SALARY

SELECT * FROM EMPLOYEE

WHERE DEPTNO IN (1,3) AND SAL > 5000;

Q: DISPLAY THE DETAILS OF LEAVES WHERE THE EMPNO IS IN THE RANGE 103 TO 110

SELECT * FROM EMP_LEAVES

WHERE EMPNO BETWEEN 103 AND 110;

Q: DISPLAY DETAILS OF EMPLOYEES WHERE THE NAME CONTAINS LETTER X OR Z.

SELECT * FROM EMPLOYEES

WHERE NAME LIKE '%X%' OR NAME LIKE '%Z%';

Q: DISPLAY DETAILS OF DEPARTMENT WHERE HEAD OF DEPARTMENT IS 'STEVE' AND THE DEPTNAME CONTAINS 'P' AS THE LAST CHARACTER.

SELECT * FROM DEPT

WHERE HOD = 'STEVE' AND DEPTNAME LIKE '%P';

Q: DISPLAY CONSTRAINTS OF EMP_LEAVES TABLE

SELECT * FROM USER_CONSTRAINTS

WHERE TABLE_NAME = 'EMP_LEAVES';

Q: DISPLAY EMPNO,EMPNAME,DATE OF JOINING & EXPEREINCE IN MONTHS

SELECT EMPNO, EMPNAME, DJ, MONTHS_BETWEEN(SYSDATE,DJ) "NO MONTHS"

FROM EMPLOYEE;

Q: DISPLAY EMPNO,LEAVETYPE,STDATE,NO OF DAYS BETWEEN SYSDATE & STDATE FOR LEAVES THAT ARE NOT COMPLETED.

SELECT EMPNO, LEAVETYPE, STDATE, ENDDATE - STDATE

FROM EMP_LEAVES

WHERE ENDDATE IS NULL;

Q: DISPLAY EMPNO,EMPNAME,DATE ON WHICH EMPLOYEE TOOK FIRST SALARY (ASSUMING ON 1ST OF EACH MONTH SALARY IS PAID).

```
SELECT EMPNO, EMPNAME,  LAST_DAY(DJ) + 1 "FIRST SAL DATE"

FROM EMPLOYEE;
```

Q: DISPLAY THE DETIALS OF EMPLOYEES WHO HAVE THE PATTERN 'TE' IN NAME AND NAME HAS MORE THAN 5 LETTERS.

```
SELECT * FROM EMPLOYEE

WHERE EMPNAME LIKE '%TE%'  AND LENGTH(EMPNAME) > 5;
```

Q: DISPLAY THE DETAILS OF LEAVES ALONG WITH THE DATE OF COMING SATURDAY AFTER STDATE AND NO. OF DAYS OF LEAVES FOR LEAVES THAT ARE COMPLETED.

```
SELECT EMPNO, LEAVETYPE,STDATE, NEXT_DAY(STDATE,'Saturday'), ENDDATE - STDATE

FROM EMP_LEAVES

WHERE ENDATE IS NOT NULL;
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO HAVE JOINED IN THE CURRENT YEAR

```
SELECT * FROM EMPLOYEE

WHERE TO_CHAR(DJ,'YYYY') = TO_CHAR(SYSDATE,'YYYY');
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHOSE NAME CONTAINS 'APP' IN 4TH,5TH,6TH POSITIONS.

```
SELECT * FROM EMPLOYEE WHERE

INSTR(EMPNAME,'APP') = 4;

  OR

SELECT * FROM EMPLOYEE

WHERE SUBSTR(EMPNAME,4,3) ='APP';
```

Q: DISPLAY EMPNO,NAME,HOLIDAY WEEK,WHICH DEPENDS ON THE DEPT AS FOLLOWS: DEPT1:MONDAY DEPT2:THURSDAY OTHERS:SUNDAY

```
SELECT EMPNO, EMPNAME,

      DECODE(DEPTNO, 1,'MONDAY',2,'THURSDAY','SUNDAY') HOLIDAY

FROM EMPLOYEE;
```

Q: DISPLAY THE EMPNO,LEAVETYPE,STDATE IN 'DD-MM' FORMAT AND ENDING DATE FOR ALL THE LEAVES THAT ARE TAKEN BY EMPLOYES WITH NUMBERS IN THE RANGE 103-107 AND IN THE CURRENT YEAR.

```
SELECT EMPNO, LEAVETYPE, TO_CHAR(STDATE,'DD-MM'), ENDDATE

FROM EMP_LEAVES

WHERE EMPNO BETWEEN 103 AND 107

      AND TO_CHAR(SYSDATE,'YYYY') = TO_CHAR(STDATE,'YYYY');
```

Q: TRUNCATE TIME PORTION IN STARTING DATE OF THE LEAVE.

```
UPDATE EMP_LEAVES
  SET  STDATE = TRUNC(STDATE);
```

Q: DISPLAY THE SUM OF SALARY OF EACH DEPT

```
SELECT DEPTNO, SUM(SAL)
FROM  EMPLOYEE
GROUP BY DEPTNOL;
```

Q: DISPLAY THE AVERAGE SALARY OF EACH DEPT BY TAKING EMPLOYEES WHO HAVE JOINED IN THE CURRENT YEAR.

```
SELECT  DEPTNO, AVG(SAL)
FROM EMPLOYEE
WHERE TO_CHAR(SYSDATE,'YYYY') = TO_CHAR(DJ,'YYYY')
GROUP BY DEPTNO;
```

Q: DISPLAY EMPNO,TOTAL NO.OF LEAVES TAKEN BY EMPLOYEE

```
SELECT EMPNO, SUM(ENDDATE - STDATE)
FROM EMP_LEAVES
GROUP BY EMPNO;
```

Q: DISPLAY THE TOTAL NO. OF LEAVES TAKEN FOR EACH LEAVETYPE

```
SELECT LEAVETYPE, SUM(ENDDATE - STDATE)
FROM EMP_LEAVES
GROUP BY LEAVETYPE;
```

Q: DISPLAY EMPNO WHERE EMPLOYEE HAS TAKEN MORE THAN 10 LEAVES.

```
SELECT EMPNO
FROM EMP_LEAVES
GROUP BY  EMPNO
HAVING  SUM(ENDDATE  - STDATE) > 10;
```

Q: DISPLAY THE YEAR IN WHICH MORE THAN 5 EMPLOYEES HAVE

```
JOINED SELECT TO_CHAR(DJ,'YYYY')
FROM   EMPLOYEE
GROUP BY TO_CHAR(DJ,'YYYY')
HAVING COUNT(*) > 5;
```

Q: DISPLAY EMPNO FOR EMPLOYEES WHO HAVE TAKEN MORE THAN 20 LEAVES IN THE CURRENT YEAR.

```
SELECT EMPNO

FROM EMP_LEAVES

WHERE TO_CHAR(SYSDATE,'YYYY') = TO_CHAR(STDATE,'YYYY')

GROUP BY EMPNO

HAVING  SUM(ENDDATE - STDATE) > 20;
```

Q: DISPLAY THE LEAVETYPE THAT HAS BEEN TAKEN FOR MORE THAN 10 TIMES

```
SELECT LEAVETYPE

FROM EMP_LEAVES

GROUP BY LEAVETYPE

HAVING  COUNT(*) > 10;
```

Q: DISPLAY DEPT,DESIGNATION,YEAR & NO. OF EMPLOYEES JOINED IN THAT YEAR IN THAT DEPARTMENT AND DESIGNATION.

```
SELECT  DEPTNO, DESG, TO_CHAR(DJ,'YYYY'), COUNT(*)

FROM  EMPLOYEE

GROUP BY DEPTNO, DESG, TO_CHAR(DJ,'YYYY');
```

Q: DISPLAY DEPT IN WHICH THE AVGERAGE SAL OF ANY SINGLE DESIGNATION IS MORE THAN 10000.

```
SELECT DISTINCT DEPT

FROM   EMPLOYE

GROUP  BY DEPT, DESG

HAVING AVG(SAL) > 1000;
```

Q: DISPLAY DEPTNO,DIFFERENCE BETWEEN MIN & MAX OF SALARY OF THE DEPT.

```
SELECT DEPTNO, MAX(SAL) - MIN(SAL)

FROM   EMPLOYEE

GROUP BY  DEPTNO;
```

Q: DISPLAY LEAVETYPE FOR WHICH MORE THAN 10 LEAVES ARE TAKEN IN THE CURRENT MONTH OR 20 LEAVES TAKEN SO FAR.

```
SELECT LEAVETYPE
FROM  EMP_LEAVES
WHERE TO_CHAR(STDATE,'MMYYYY') = TO_CHAR(SYSDATE,'MMYYYY')
GROUP BY LEAVETYPE
HAVING  SUM(ENDDATE - STDATE) > 10

UNION
SELECT LEAVETYPE
FROM  EMP_LEAVES
GROUP BY LEAVETYPE
HAVING  SUM(ENDDATE - STDATE) > 20
```

Q: DISPLAY TOTAL NO.OF LEAVES OF ALL EMPLOYES (CONSIDERING SYSDATE AS ENDING DATE IF ENDING DATE IS NOT AVAILABLE).

```
SELECT SUM( NVL(ENDDATE,SYSDATE) - STDATE)

FROM EMP_LEAVES;
```

Q: DISPLAY EMPNO,LEAVETYPE ,STDATE,NO. OF LEAVES & MAX NO. OF LEAVES FOR THAT CATEGORY.

```
SELECT EMPNO, EL.LEAVETYPE,STDATE, ENDDATE - STDATE, NOLEAVES

FROM  EMP_LEAVES EL, LEAVES L

WHERE EL.LEAVETYPE = L.LEAVETYPE;
```

Q: DISPLAY DEPTNO,DEPTNAME,EMPNAME,DESIGNATION FOR THAT DEPT WHERE STARTING LETTER IS 'A'.

```
SELECT E.DEPTNO, DEPTNAME,EMPNAME,DESG

FROM  EMPLOYEE E, DEPT D

WHERE  DETPNAME LIKE 'A%'  AND  E.DEPTNO = D.DEPTNO;
```

Q: DISPLAY EMPNO,NAME,DEPTNAME,HOD FOR THE EMPLOYEES WHO HAVE NOT TAKEN ANY LEAVE SO FAR.

```
SELECT EMPNO,EMPNAME, DEPTNAME,HOD

FROM  EMPLOYEE E, DEPT D

WHERE  EMPNO NOT IN

    (SELECT EMPNO FROM EMP_LEAVES)

    AND E.DEPTNO = D.DEPTNO;
```

Q: DISPLAY EMPNO,NAME,DEPTNAME,LEAVENAME,STDATE,ENDDATE

```
SELECT E.EMPNO,EMPNAME,DEPTNAME, LEAVENAME,STDATE,ENDDATE

FROM  EMPLOYEE E, DEPT D, EMP_LEAVES EL

WHERE  E.DEPTNO = D.DEPTNO AND  E.EMPNO = EL.EMPNO;
```

Q: DISPLAY DETAILS OF DEPT IN WHICH AT LEAST ONE EMPLOYEE HAS JOINED IN THE CURRENT MONTH.

```
SELECT * FROM DEPT

WHERE  DEPTNO IN

    ( SELECT DEPTNO FROM EMPLOYEE

      WHERE  TO_CHAR(SYSDATE,'MMYYYY') = TO_CHAR(DJ,'MMYYYY') );
```

Q: DISPLAY LEAVETYPE,LEAVENAME,EMPNO AND STDATE FOR ALL THE LEAVES INCLUDING LEAVETYPES THAT HAVE NOT BEEN USED BY ANY EMPLOYEE.

```
SELECT L.LEAVETYPE,LEAVENAME, EMPNO , STDATE

FROM  EMP_LEAVES EL, LEAVES L

WHERE  L.LEAVETYPE = EL.LEAVETYPE (+);
```

Q: DISPLAY THE DETAILS OF LEAVES WHERE THE NO.OF DAYS OF LEAVES IS MORE THAN THE NOOFDAYS OF LEAVE TAKEN BY 101 IN LEAVE THAT STARTED ON 5-MARCH-00.

```
SELECT E1.*
FROM  EMP_LEAVES E1, EMP_LEAVES E2
WHERE E2.STDATE= '05-MAR-2000' AND E2.EMPNO = 101
    AND  E1.ENDDATE - E1.STDATE >  E2.ENDDATE - E2.STDATE;
```

Q: DISPLAY DETAILS OF DEPT IN WHICH THE AVGERAGE SALARY IS > 10000.

```
SELECT * FROM DEPT
WHERE  DEPTNO IN
    ( SELECT  DEPTNO
      FROM EMPLOYEE
      GROUP BY DEPTNO
      HAVING AVG(SAL) > 10000);
```

Q: DISPLAY DETAILS OF DEPT WHERE THE DEPT HAS MORE THAN 3 EMPLOYEES DRAWING MORE THAN 5000

```
SELECT * FROM DEPT
WHERE DEPTNO IN
    ( SELECT  DEPTNO
      FROM EMPLOYEE
      WHERE SAL > 5000
      GROUP BY DEPTNO
      HAVING COUNT(*) > 3);
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO HAS NOT TAKEN SICK LEAVE IN CURRENT MONTH.

```
SELECT * FROM EMPLOYEE
WHERE EMPNO NOT IN
    ( SELECT  EMPNO
      FROM EMP_LEAVES
      WHERE LEAVETYPE = 'S'
          AND  TO_CHAR(STDATE,'MMYYYY') = TO_CHAR(SYSDATE,'MMYYYY')
    );
```

Q: DISPLAY DETAILS OF EMPLOYEES DRAWING THE MAXSAL.

```
SELECT * FROM EMPLOYEE

WHERE SAL = (SELECT

      MAX(SAL)

       FROM EMPLOYEE);
```

Q: DISPLAY DETAILS OF EMPLOYEES DRAWING MORE SALARY THAN THE AVERAGE SAL OF EMPLOYEES JOINED IN THE CURRENT YEAR.

```
SELECT * FROM EMPLOYEE

WHERE  SAL >

  ( SELECT  AVG(SAL)

    FROM  EMPLOYEE

    WHERE TO_CHAR(SYSDATE,'YYYY') = TO_CHAR(DJ,'YYYY')

  );
```

Q: DISPLAY DETAILS OF DEPT'S IN WHICH NO EMPLOYEE JOINED IN THE CURRENT YEAR.

```
SELECT * FROM DEPT

WHERE DEPTNO NOT IN

 ( SELECT DEPTNO FROM EMPLOYEE

   WHERE TO_CHAR(DJ,'YYYY') = TO_CHAR(SYSDATE,'YYYY')

  );
```

Q: DELETE DETAILS OF LEAVES TAKEN BY EMPLOYEE WHOSE EMPNO IS THE HIGHEST EMPNO

```
DELETE FROM  EMP_LEAVES

WHERE  EMPNO  = ( SELECT MAX(EMPNO) FROM EMPLOYEE);
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WHO ARE BELONGING TO PRODUCTION DEPT AND HAVE TAKEN MORE THAN 20 LEAVES SO FAR.

```
SELECT E.*

FROM EMPLOYEE E, DEPT D

WHERE  E.DEPTNO = D.DEPTNO AND DEPTNAME = 'PRODUCTION'

      AND  EMPNO IN

          ( SELECT EMPNO

            FROM EMP_LEAVES

            GROUP BY EMPNO

            HAVING SUM(ENDDATE-STDATE) > 20
```

```
);
```

Q: DISPLAY DETAILS OF LEAVES WHERE THE EMPLOYEE SAL IS MORE THAN 10000 AND JOINED IN THE LAST 6 MONTHS.

```
SELECT * FROM EMP_LEAVES
WHERE EMPNO  IN
   ( SELECT EMPNO
     FROM  EMPLOYEE
     WHERE  SAL > 10000 AND  MONTHS_BETWEEN(SYSDATE,DJ) <= 6
   );
```

Q: Display details of employees drawing top 5 salaries.

This is done using correlated subquery. Subquery is used to return

the number of employees whose salary is greater than the salary of the

current employee in main query. If that count is less than 5 that means

the employee is drawing one of the top five salraies.

```
select * from employee e
where 5 >
    ( select count(distinct sal)
      from employee
      where  sal > e.sal);
```

Q: UPDATE THE SALARY OF EMPLOYEE 102 WITH THE AVERAGE SALARY OF HIS DEPARTMENT.

```
UPDATE EMPLOYEE E SET SAL= ( SELECT AVG(SAL) FROM EMPLOYEE WHERE DEPTNO = E.DEPTNO)
WHERE EMPNO = 102;
```

Q: DISPLAY DETAILS OF DEPARTMENT IN WHICH THERE ARE HIGHEST NUMBER OF LEAVES TAKEN

```
SELECT * FROM DEPT
WHERE DEPTNO IN
(
 SELECT DEPTNO
 FROM  EMPLOYEE E,EMP_LEAVES EL
 WHERE  E.EMPNO=EL.EMPNO
 GROUP  BY DEPTNO
 HAVING SUM(ENDDATE-STDATE) =
   ( SELECT MAX(SUM(ENDDATE - STDATE))
     FROM   EMPLOYEE  E, EMP_LEAVES EL
     WHERE  E.EMPNO = EL.EMPNO
     GROUP  BY DEPTNO
   )
)
```

Q: RENAME COLUMN **DJ** TO **JOINDATE** OF EMPLOYEE TABLE.

RENAMING A COLUMN DONE IN THREE STEPS.

1. CREATE A NEW TABLE FROM EMPLOYEE TABLE. GIVE ALIAS JOINDATE TO DJ COLUMN IN QUERY.
2. *CREATE TABLE TEMPLOYEE*
3. *AS SELECT EMPNO,EMPNAME,SAL,DJ JOINDATE, DESG, DEPTNO*
4. *FROM EMPLOYEE;*

5. DROP EMPLOYEE TABLE.

6. *DROP TABLE EMPLOYEE;*
7. RENAME NEW TABLE TO EMPLOYEE TABLE.
8. *RENAME TEMPLOYEE TO EMPLOYEE;*

Q: SWAP THE SALARY OF 101 WITH SALARY OF 103.

```
DECLARE
  SAL_103 EMPLOYEE.SAL%TYPE;
BEGIN
-- GET SALARY OF 103
            SELECT  SAL INTO SAL_103
             FROM EMPLOYEE
            WHERE EMPNO = 103;
 -- UPDATE SALARY OF 103 WITH SALARY OF 101
            UPDATE EMPLOYEE
            SET SAL = (SELECT SAL FROM EMPLOYEE WHERE EMPNO = 101)
            WHERE EMPNO = 103;
-- UPDATE SALARY OF 101 WITH SALARY OF 103
            UPDATE EMPLOYEE
            SET SAL = SAL_103;
            WHERE EMPNO = 101;
  COMMIT;
END;
```

Q: CREATE A PROCEDURE TO TAKE EMPNO AND LEAVETYPE AND INSERT A ROW INTO EMP_LEAVES TABLE WITH THE FOLLOWING CONDITIONS.

CHECK WHETHER EMPNO AND LEAVETYPE ARE VALID

CHECK WHETHER EMPLOYEE IS ALREADY ON LEAVE

CHECK WHETHER EMPLOYEE HAS ALREADY USED ALL LEAVES IN THAT TYPE

```
-- PROCEDURE TO INSERT A NEW ROW INTO EMP_LEAVES TABLE

-- TAKES EMPLOYEE NUMBER AND LEAVETYPE

-- STDATE OF LEAVE IS SYSDATE AND ENDDATE IS NULL

CREATE OR REPLACE PROCEDURE NEWLEAVE(PEMPNO NUMBER, PLT CHAR)

IS

STATUS NUMBER(1):=0; -- INITALIZE VARIABLE

CNT NUMBER(3);

TNL NUMBER(2);

NL  NUMBER(2);

BEGIN

-- CHECK WHETHER EMPLOYEE NO. IS VALID

        SELECT 0 INTO CNT

        FROM  EMPLOYEE

        WHERE  EMPNO = PEMPNO;

-- CHECK WHETHER LEAVETYPE IS VALID

-- IF LEAVETYPE IS VALID THEN GET MAX NO. OF LEAVES

-- IN THAT LEAVETYPE


          STATUS := 1;


        SELECT NOLEAVES INTO TNL

        FROM  LEAVES

        WHERE LEAVETYPE = PLT;

 -- CHECK WHETHER EMPLOYEE HAS ALREADY IN A LEAVE

        SELECT COUNT(*) INTO CNT

        FROM EMP_LEAVES

         WHERE EMPNO = PEMPNO AND  ENDDATE IS NULL;
```

```
            IF  CNT <> 0 THEN  -- EMPLOYEE IS ALREADY ON LEAVE
            RAISE_APPLICATION_ERROR(-20120,'EMPLOYEE IS ALREADY ON LEAVE');
 END IF;
 -- CHECK WHETHER EMPLOYEE HAS CROSSED THE LIMIT
 -- CHECK NO. OF LEAVES OF THIS TYPE ALREADY CONSUMED
            SELECT SUM( ENDDATE-STDATE) INTO NL FROM EMP_LEAVES
            WHERE EMPNO = PEMPNO AND LEAVETYPE = PLT;

            IF  NL >= TNL THEN
            RAISE_APPLICATION_ERROR(-20130,'ALREADY CONSUMED TOTAL NUMBER OF LEAVES IN THIS LEAVETYPE');
            END IF;
 -- VALID ENTRY, INSERT ROW
            INSERT INTO EMP_LEAVES VALUES(PEMPNO,PLT,SYSDATE,NULL);
 COMMIT;
EXCEPTION
            WHEN NO_DATA_FOUND THEN
  -- IF STATUS IS 0 THEN FIRST SELECT RAISE EXCEPTION
            IF STATUS = 0 THEN
                        RAISE_APPLICATION_ERROR(-20110,'EMPLOYEE NUMBER IS NOT FOUND');
            ELSE -- SECOND SELECT RAISE EXCEPTION RAISE_APPLICATION_ERROR(-
                        20120,'LEAVETYPE IS NOT FOUND');
            END IF;
            WHEN OTHERS THEN
                        RAISE_APPLICATION_ERROR(SQLCODE,SQLERRM);
END;
```

Q: CREATE A FUNCTION TO RETURN NAMES OF THE EMPLOYEES DRAWING HIGEST SALARY. IF THERE IS MORE THAN ONE EMPLOYEE, EMPLOYEE NAMES ARE TO BE SEPARATED BY COMMA.

```
CREATE OR REPLACE FUNCTION GETHSEMPNAMES  RETURN VARCHAR2
IS
        CURSOR HSEMP_CURSOR IS
        SELECT EMPNAME FROM EMPLOYEE WHERE SAL = (SELECT MAX(SAL) FROM EMPLOYEE);
        ENAME VARCHAR2(20);
        ALLNAMES VARCHAR2(200);
BEGIN
        ALLNAMES := '';
                FOR REC IN HSEMP_CURSOR
                LOOP
    -- IF NOT FIRST NAME THEN ADD COMMA
                        IF LENGTH(ALLNAMES) != 0 THEN
                                ALLNAMES := ALLNAMES || ',';
                        END IF;
                ALLNAMES := ALLNAMES || REC.EMPNAME;
                END LOOP;

 RETURN  ALLNAMES;
END;
```

Q: GET THE DETAILS OF DEPT. WHICH IS HEADED BY PERSON WITH THE NAME THAT CONTAINS LETTER 'C' AND 'A'.

```
SELECT * FROM DEPT
WHERE HOD LIKE '%C%A%';
```

Q: DISPLAY THE DETAILS OF DEPT'S WHERE THE DEPTNO IS >10 AND DEPTNAME ENDS WITH 'A'.

```
SELECT * FROM DEPT
WHERE  DEPTNO > 10  AND DEPTNAME LIKE '%A';
```

Q: DISPLAY DETAILS OF EMPLOYEES WHO HAVE MORE THAN 10000 SALARY OR DESG. 'SA'.

```
SELECT * FROM EMPLOYEE WHERE

SAL > 10000 OR DESG = 'SA';
```

Q: DISPLAY EMPNO,EMPNAME,SALARY ROUNDED TO 100'S.DOJ AND NO.OF MONTHS BETWEEN TODAY AND DATE OF JOINING.

```
SELECT EMPNO, EMPNAME, ROUND(SAL,-2) "SAL" , DJ, MONTHS_BETWEEN(SYSDATE,DJ) "NO MONTHS"

FROM EMPLOYEE;
```

Q: DISPLAY DETAILS OF LEAVES THAT WERE TAKEN IN LAST 20DAYS.

```
SELECT * FROM EMP_LEAVES

  WHERE SYSDATE -STDATE <= 20;
```

Q: DISPLAY DETAILS OF LEAVES IN WHICH THE NO.OF DAYS OF LEAVES IS MORE THAN 5.

```
SELECT * FROM EMP_LEAVES

   WHERE ENDDATE - STDATE > 5;
```

Q: DISPLAY ALL SICK LEAVES BY EMPNO 104.

```
SELECT * FROM EMP_LEAVES WHERE EMPNO = 104 AND LEAVETYPE ='S';
```

Q: DISPLAY EMPNO,LEAVETYPE,NO.OF DAYS OF LEAVE FOR LEAVES THAT WERE COMPLETED.

```
SELECT EMPNO, LEAVETYPE, ENDDATE - STDATE "NO DAYS"

FROM EMP_LEAVES

WHERE ENDDATE IS NOT NULL;
```

Q: DISPLAY EMPLOYEE WHERE EMPNAME CONTAINS LETTERS 'M' AND 'J' IN ANY ORDER.

```
SELECT * FROM EMPLOYEE

WHERE EMPNAME LIKE '%M%' AND EMPNAME LIKE '%J%';
```

Q: DISPLAY THE ROWS OF EMPLOYEE TABLE WHERE EMPLOYEE JOINED IN THE LAST 6 MONTHS AND SAL>5000 AND DESG. IS NOT PROGRAMMER.

```
SELECT * FROM EMPLOYEE

WHERE SAL > 5000 AND  MONTHS_BETWEEN(SYSDATE,DJ) <= 6  AND DESG <> 'PRO';
```

Q: DISPLAY DETAILS OF LEAVES THAT WERE TAKEN IN THE CURRENT MONTH.

```
SELECT * FROM EMP_LEAVES

WHERE TO_CHAR(STDATE,'MMYY') = TO_CHAR(SYSDATE,'MMYY');
```

Q: DISPLAY EMPNO,FIRST NAME,SALARY AND EXPERIENCE IN YEARS IN THE COMPANY.

```
SELECT EMPNO, SUBSTR(EMPNAME, 1,DECODE(INSTR(EMPNAME,' '),0,LENGTH(EMPNAME),INSTR(EMPNAME,' '))) "FIRSTNAME", SAL, MONTHS_BETWEEN(SYSDATE,DJ) / 12 "EXP.IN YEARS"

FROM EMPLOYEE;
```

Q: DISPLAY THE DETAILS OF EMPLOYEE WHO JOINED IN THE MONTHS OF JULY IRRESPECTIVE OF THE YEAR.

```
SELECT * FROM EMPLOYEE

WHERE TO_CHAR(DJ,'MM') = 7;
```

Q: DISPLAY DETAILS OF EMPLOYEE WHO HAVE MORE THAN 10 CHARS IN THE NAME OR HAVING LETTER 'G' AND 'C' IN THE NAME.

```
SELECT * FROM EMPLOYEE

WHERE LENGTH(EMPNAME) > 10 AND EMPNAME LIKE '%G%C';
```

Q: CHANGE THE NAME OF EMPLOYEE 105 TO UPPERCASE AND REMOVE ALL LEADING AND TRAILING SPACES.

```
UPDATE  EMPLOYEE SET EMPNAME = UPPER(TRIM(EMPNAME)) WHERE EMPNO = 105;
```

Q: DISPLAY EMPNO,LEAVETYPE,THE MONTH IN WHICH LEAVE STARTED AND THE MONTH IN WHICH LEAVE ENDED FOR LEAVES WHERE THESE TWO MONTHS ARE NOT SAME.

```
SELECT EMPNO, LEAVETYPE, TO_CHAR(STDATE,'MONTH'), TO_CHAR(ENDDATE,'MONTH')

FROM EMP_LEAVES

WHERE TO_CHAR(STDATE,'MM') != TO_CHAR(ENDDATE,'MM');
```

Q: DISPLAY LEAVES THAT ENDED IN PREVIOUS MONTH.

```
SELECT * FROM EMP_LEAVES

WHERE  TO_CHAR(ENDDATE,'MMYY') =  TO_CHAR( ADD_MONTHS(SYSDATE,-1), 'MMYY');
```

Q: DELETE DETAILS OF LEAVES WHERE THE LEAVE STARTED IN THE FIRST WEEK OF THE PREVIOUS MONTH.

```
SELECT * FROM EMP_LEAVES

WHERE STDATE BETWEEN LAST_DAY(ADD_MONTHS(SYSDATE,-2))+1 AND

                                LAST_DAY(ADD_MONTHS(SYSDATE,-2)) + 7;
```

Q: DISPLAY EMPNAME IN UPPERCASE,DAY OF JOINING AND DATE OF FIRST SALARY AND WEEK DAY OF FIRST

```
SALARY. SELECT UPPER(EMPNAME), DJ, LAST_DAY(DJ) + 1, TO_CHAR( LAST_DAY(DJ) + 1, 'DAY')

FROM EMPLOYEE;
```

Q: DISPLAY MONTHS IN WHICH EMPLOYEES JOINED IN THE CURRENT

```
YEAR. SELECT DISTINCT TO_CHAR(DJ,'MONTH') "MONTH"

FROM  EMPLOYEE

WHERE TO_CHAR(DJ,'YYYY') = TO_CHAR(SYSDATE,'YYYY');
```

Q: DISPLAY AVG.SALARY OF ALL THE EMPLOYEES.

```
SELECT AVG(SAL)

FROM EMPLOYEE;
```

Q: DISPLAY LEAVETYPE,NO.OF TIMES EMPLOYEES HAVE TAKEN THAT LEAVE.

```
SELECT LEAVETYPE, SUM(ENDDATE-STDATE)
FROM  EMP_LEAVES
GROUP BY LEAVETYPE;
```

Q: DISPLAY DEPTNO,AND NO.OF EMPLOYEES JOINED IN THE CURRENT YEAR. SELECT DEPTNO, COUNT(*)

```
FROM EMPLOYEE
WHERE TO_CHAR(DJ,'YYYY') = TO_CHAR(SYSDATE,'YYYY')
GROUP BY DEPTNO;
```

Q: DISPLAY MONTH NAME AND HOW MANY LEAVES STARTED IN THAT MONTH. SELECT TO_CHAR(STDATE,'MONTH'), COUNT(*)

```
FROM EMP_LEAVES
GROUP BY TO_CHAR(STDATE,'MONTH');
```

Q: DISPLAY THE EMPLOYEE WHO HAVE TAKEN MORE THAN 10 LEAVES SO FAR.

```
SELECT EMPNO
FROM EMP_LEAVES
GROUP BY EMPNO
HAVING SUM(ENDDATE-STDATE) > 10;
```

Q: DISPLAY THE EMPLOYEE WHO HAS TAKEN MORE THAN 5 SICK LEAVES IN THE CURRENT YEAR.

```
SELECT EMPNO
FROM EMP_LEAVES
WHERE TO_CHAR(STDATE,'YYYY') = TO_CHAR(SYSDATE,'YYYY')
GROUP BY EMPNO
HAVING SUM(ENDDATE-STDATE) > 5;
```

Q: DISPLAY DEPTNO,DESG AND AVG.SALARY.

```
SELECT DEPTNO, DESG, AVG(SAL)
FROM  EMPLOYEE
GROUP BY DEPTNO, DESG;
```

Q: DISPLAY YEAR,NO.OF EMPLOYEES JOINED WITH DESG

PROGRAMMER. SELECT TO_CHAR(DJ,'YYYY'), COUNT(*)

FROM EMPLOYEE

WHERE DESG = 'PRO'

GROUP BY TO_CHAR(DJ,'YYYY');

Q: DISPLAY EMPNO,EMPNAME,DEPTNAME FOR EMPLOYEES WHO HAVE JOINED IN THE CURRENT MONTH.

SELECT EMPNO, EMPNAME, DEPTNAME

FROM EMPLOYEE E, DEPT D

WHERE TO_CHAR(DJ,'YYYY') = TO_CHAR(SYSDATE,'YYYY') AND E.DEPTNO = D.DEPTNO;

Q: DISPLAY EMPNO,EMPNAME,DEPTNO,DEPTNAME,LEAVENAME,STDATE FOR ALL LEAVES THAT ARE NOT COMPLETED.

SELECT EL.EMPNO,EMPNAME, E.DEPTNO,DEPTNAME, LEAVENAME, STDATE

FROM EMPLOYEE E, DEPT D, LEAVES L, EMP_LEAVES EL

WHERE E.DEPTNO = D.DEPTNO AND E.EMPNO = EL.EMPNO AND L.LEAVETYPE= EL.LEAVETYPE;

Q: DISPLAY EMPNAME AND TOTAL NO.OF LEAVES TAKEN(EMPNAME HAS TO BE UNIQUE)

SELECT EMPNAME, SUM(ENDDATE-STDATE)

FROM EMPLOYEE E, EMP_LEAVES EL

WHERE E.EMPNO = EL.EMPNO

GROUP BY EMPNAME;

Q: DISPLAY EMPNO,EMPNAME,LEAVETYPE,STDATE.INCLUDE EMPLOYEES WHO HAVE NOT TAKEN ANY LEAVE AND DISPLAY THE DATE IN THE ASCENDING ORDER OF EMPNO.

SELECT E.EMPNO, EMPNAME, LEAVETYPE, STDATE

FROM EMPLOYEE E, EMP_LEAVES EL

WHERE E.EMPNO = EL.EMPNO (+);

Q: DISPLAY THE LEAVES THAT WERE TAKEN AFTER EMPNO 106 TOOK SICK LEAVE(ASSUMMING 106 HAS TAKEN

ONLY ONE SICK

LEAVE) SELECT EL1.*

FROM EMP_LEAVES EL1, EMP_LEAVES EL2 WHERE

EL2.EMPNO = 106 AND EL2.LEAVETYPE = 'S'

AND EL1.STDATE > EL2.STDATE;

Q: DISPLAY DETAILS OF DEPT. IN WHICH WE HAVE AN EMPLOYEE WITH THE NAME CONTAINING 'KEVIN'.

SELECT * FROM DEPT

WHERE DEPTNO IN (SELECT DEPTNO

```
FROM EMPLOYEE WHERE EMPNAME LIKE '%KEVIN%');
```

Q: DISPLAY HIGHEST TOTAL NO.OF LEAVES TAKEN BY A SINGLE EMPLOYEE.

```
SELECT MAX(SUM(ENDDATE-STDATE))
FROM EMP_LEAVES
GROUP BY EMPNO;
```

Q: DISPLAY HIGHEST NO.OF DAYS IN SINGLE LEAVE.

```
SELECT MAX(ENDDATE-STDATE)
FROM   EMP_LEAVES;
```

Q: DISPLAY DETAILS OF DEPT WHERE DEPTNO HAS MORE THAN 5 EMPLOYEES.

```
SELECT * FROM DEPT
WHERE DEPTNO IN ( SELECT DEPTNO
                  FROM EMPLOYEE
                  GROUP BY DEPTNO HAVING COUNT(*) > 5);
```

Q: DISPLAY DETAILS OF EMPLOYEE WHO IS DRAWING THE MAX.SALARY.

```
SELECT * FROM EMPLOYEE
WHERE SAL = ( SELECT MAX(SAL) FROM EMPLOYEE);
```

Q: DISPLAY DETAILS OF LEAVES THAT WERE TAKEN BY EMPLOYEES OF DEPT 4.

```
SELECT * FROM EMP_LEAVES
WHERE EMPNO IN ( SELECT EMPNO FROM EMPLOYEE WHERE DEPTNO = 4);
```

Q: DISPLAY DEPTNO AND NO.OF EMPLOYEE WHO HAVE TAKEN LEAVE IN THE CURRENT MONTH.

```
SELECT DEPTNO, COUNT(*)
FROM EMPLOYEE
WHERE EMPNO IN ( SELECT EMPNO FROM EMP_LEAVES
                 WHERE TO_CHAR(STDATE,'MMYY') = TO_CHAR(SYSDATE,'MMYY')
               )
GROUP BY DEPTNO;
```

Q: DISPALY DETAILS OF EMPLOYEES WHO HAVE NOT TAKEN ANY LEAVE SO FAR.

```
SELECT * FROM EMPLOYEE
WHERE EMPNO NOT IN ( SELECT EMPNO FROM EMP_LEAVES);
```

Q: DISPLAY DETAILS OF EMPLOYEES WHO HAVE TAKEN A LEAVE IN THE PREVIOUS MONTH AND HAS NOT TAKEN ANY LEAVE IN ONE CURRENT MONTH.

```
SELECT * FROM EMPLOYEE
WHERE EMPNO IN ( SELECT EMPNO
                 FROM EMP_LEAVES
                 WHERE TO_CHAR(STDATE,'MMYY') = TO_CHAR( ADD_MONTHS(SYSDATE,-1),'MMYY'))
 AND  EMPNO NOT IN
              ( SELECT EMPNO
                FROM EMP_LEAVES
                WHERE TO_CHAR(STDATE,'MMYY') =
              TO_CHAR(SYSDATE,'MMYY') );
```

Q: DISPLAY DETAILS OF DEPT. IN WHICH ATLEAST 2 EMPLOYEE HAVE TAKEN MORE THAN 5 SICK LEAVES.

```
SELECT * FROM DEPT
  WHERE  DEPTNO IN ( SELECT DEPTNO
                     FROM EMPLOYEE
                     WHERE  EMPNO IN ( SELECT EMPNO
                                       FROM EMP_LEAVES WHERE LEAVETYPE='S'
                                       GROUP BY EMPNO HAVING SUM(ENDDATE-STDATE) > 5)
                   );
```

Q: DISPLAY DETAILS OF DEPTS. IN WHICH ATLEAST ONE EMPLOYEE IS CURRENTLY ON LEAVE.

```
SELECT * FROM DEPT
WHERE DEPTNO IN
        (SELECT DEPTNO FROM EMPLOYEE
         WHERE EMPNO IN
           ( SELECT EMPNO
             FROM   EMP_LEAVES
             WHERE  ENDDATE IS NULL));
```

Q: DISPLAY DETAILS OF EMPLOYEES WHO ARE HEADED BY BILL OR WHO HAVE TAKEN A LEAVE ON PREVIOUS 'THURSDAY'

```
SELECT * FROM EMPLOYEE

WHERE DEPTNO IN (SELECT DEPTNO FROM DEPT WHERE HOD = 'BILL')

 OR EMPNO IN( SELECT EMPNO

        FROM EMP_LEAVES

        WHERE  TRUNC(STDATE) = NEXT_DAY(SYSDATE-15,'Thursday'));
```

Q: DISPLAY THE DETAILS OF EMPLOYEES WITH FIRST 2 HIGHEST SALARY.

```
SELECT * FROM EMPLOYEE E

WHERE 2 > ( SELECT COUNT(*)

FROM EMPLOYEE WHERE SAL > E.SAL);
```

Q: DISPLAY THE DETAILS OF LEAVETYPES WHERE THE LEAVE HAS BEEN TAKEN FOR MORE THAN 50 TIMES.

```
SELECT * FROM LEAVES

WHERE LEAVETYPE IN ( SELECT  LEAVETYPE

        FROM    EMP_LEAVES

        GROUP   BY LEAVETYPE

        HAVING  COUNT(*) > 50);
```

Q: CHANGE THE DEPTNO OF EMPLOYEE 104 TO DEPTNO. OF 'INTERNET' DEPT.

```
UPDATE EMPLOYEE SET DEPTNO = ( SELECT DEPTNO FROM DEPT WHERE DEPTNAME ='INTERNET')

WHERE EMPNO = 104;
```

Q: DROP COLUMN DESG. FROM EMPLOYEE TABLE.

STEP1 : CREATE TABLE NEWEMP AS SELECT EMPNO,EMPNAME,SAL,DEPTNO, DJ FROM EMPLOYEE;

STEP2 : DROP TABLE EMPLOYEE;

STEP3 : RENAME NEWEMP TO EMPLOYEE;

OTHER IMPORTANT QUERIES

1) To find the nth row of a table

SQL> Select *from emp where rowid = (select max(rowid) from emp where rownum <= 4);

Or

SQL> Select *from emp where rownum <= 4 minus select *from emp where rownum <= 3;

2) To find duplicate rows

SQL> Select *from emp where rowid in (select max(rowid) from emp group by empno,ename, mgr, job, hiredate, comm, deptno, sal);

Or

SQL> Select empno,ename,sal,job,hiredate,comm , count(*) from emp group by empno,ename,sal,job, hiredate, comm having count(*) >=1;

3) To delete duplicate rows

SQL> Delete emp where rowid in (select max(rowid) from emp group by empno,ename,mgr,job, hiredate, sal, comm, deptno);

4) To find the count of duplicate rows

SQL> Select ename, count(*) from emp group by ename having count(*) >= 1;

5) How to display alternative rows in a table?

SQL> select *from emp where (rowid,0) in (select rowid,mod(rownum,2) from emp);

6) Getting employee details of each department who is drawing maximum sal?

SQL> select *from emp where (deptno,sal) in (select deptno,max(sal) from emp group by deptno);

7) How to get number of employees in each department , in which department is having more than 2500 employees?

SQL> Select deptno,count(*) from emp group by deptno having count(*) >2500;

8) To reset the time to the beginning of the day

SQL> Select to_char(trunc(sysdate),'dd-mon-yyyy hh:mi:ss am') from dual;

9) To find nth maximum sal

SQL> Select *from emp where sal in (select max(sal) from (select *from emp order by sal) where rownum <= 5);

FUNCTIONS

Functions can be categorized as follows.

- Single row functions
- Group functions

SINGLE ROW FUNCTIONS

Single row functions can be categorized into five. These will be applied for each row and produces individual output for each row.

- Numeric functions
- String functions
- Date functions
- Miscellaneous functions
- Conversion functions

NUMERIC FUNCTIONS

- Abs
- Sign
- Sqrt
- Mod
- Nvl
- Power
- Exp
- Ln
- Log
- Ceil
- Floor
- Round
- Trunk
- Bitand
- Greatest
- Least
- Coalesce

a) ABS

Absolute value is the measure of the magnitude of value. Absolute value is always a positive number.

Syntax: abs (*value*)

Ex:

```
SQL> select abs(5), abs(-5), abs(0), abs(null) from dual;
ABS(5)      ABS(-5)      ABS(0)      ABS(NULL)
----------  ----------  ----------  ----------
5             -5          0
```

b) SIGN

Sign gives the sign of a value.

Syntax: sign (*value*)

Ex:

```
SQL> select sign(5), sign(-5), sign(0), sign(null) from dual;
SIGN(5)     SIGN(-5)     SIGN(0)     SIGN(NULL)
----------  ----------  ----------  ----------
1           -1            0
```

c) SQRT

This will give the square root of the given value.

Syntax: sqrt (*value*) -- here value must be positive.

Ex:

```
SQL> select sqrt(4), sqrt(0), sqrt(null), sqrt(1) from dual;
SQRT(4)         SQRT(0)         SQRT(NULL)  SQRT(1)
----------      -------------   -----------------  --------------
2               0                                  1
```

d) MOD

This will give the remainder.

Syntax: mod (*value, divisor*)

Ex:

```
SQL> select mod(7,4), mod(1,5), mod(null,null), mod(0,0), mod(-7,4) from dual;
MOD(7,4)        MOD(1,5)        MOD(NULL,NULL)      MOD(0,0)        MOD(-7,4)
------------    ----------      ---------------------   ---------- --    ----------
3               1                                   0                    -3
```

e) NVL

This will substitutes the specified value in the place of null values.

Syntax: nvl (*null_col, replacement_value*)

Ex:

```
SQL> select * from student; -- here for 3rd row marks value is null
NO      NAME MARKS
---     -------  ---------
1       a        100
2       b        200
3       c
```

```
SQL> select no, name, nvl(marks,300) from student;
NO      NAME            NVL(MARKS,300)
---     -------         ---------------------
1       a               100
2       b               200
3       c               300
```

```
SQL> select nvl(1,2), nvl(2,3), nvl(4,3), nvl(5,4) from dual;
NVL(1,2)        NVL(2,3)        NVL(4,3)        NVL(5,4)
----------      ----------      ----------        ----------
1                 2               4               5
```

```
SQL> select nvl(0,0), nvl(1,1), nvl(null,null), nvl(4,4) from dual;
NVL(0,0)        NVL(1,1)        NVL(null,null)          NVL(4,4)
----------      ----------      -----------------       ----------

0               1                                       4
```

f) POWER

Power is the ability to raise a value to a given exponent.

Syntax: power (*value, exponent*)

Ex:

SQL> select power(2,5), power(0,0), power(1,1), power(null,null), power(2,-5) from dual;

POWER(2,5)	POWER(0,0)	POWER(1,1)	POWER(NULL,NULL)	POWER(2,-5)
32	1	1		.03125

g) EXP

This will raise e value to the give power.

Syntax: exp (*value*)

Ex:

SQL> select exp(1), exp(2), exp(0), exp(null), exp(-2) from dual;

EXP(1)	EXP(2)	EXP(0)	EXP(NULL)	EXP(-2)
2.71828183	7.3890561	1		.135335283

h) LN

This is based on natural or base e logarithm.

Syntax: ln (*value*) -- here value must be greater than zero which is positive only.

Ex:

SQL> select ln(1), ln(2), ln(null) from dual;

LN(1)	LN(2)	LN(NULL)
0	.693147181	

Ln and Exp are reciprocal to each other.

EXP (3) = 20.0855369

LN (20.0855369) = 3

i) LOG

This is based on 10 based logarithm.

Syntax: log (10, *value*)-- here value must be greater than zero which is positive only.

Ex:

SQL> select log(10,100), log(10,2), log(10,1), log(10,null) from dual;

LOG(10,100)	LOG(10,2)	LOG(10,1)	LOG(10,NULL)
2	.301029996	0	

LN (value) = LOG (EXP(1), value)

SQL> select ln(3), log(exp(1),3) from dual;

LN(3) LOG(EXP(1),3)

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

1.09861229 1.09861229

j) CEIL

This will produce a whole number that is greater than or equal to the specified value.

Syntax: ceil (*value*)

Ex:

SQL> select ceil(5), ceil(5.1), ceil(-5), ceil(-5.1), ceil(0), ceil(null) from dual;

CEIL(5) CEIL(5.1) CEIL(-5) CEIL(-5.1) CEIL(0) CEIL(NULL)

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

5 6 -5 -5 0

k) FLOOR

This will produce a whole number that is less than or equal to the specified value.

Syntax: floor (*value*)

Ex:

SQL> select floor(5), floor(5.1), floor(-5), floor(-5.1), floor(0), floor(null) from dual;
FLOOR(5) FLOOR(5.1) FLOOR(-5) FLOOR(-5.1) FLOOR(0) FLOOR(NULL)
----------- ------------ ------------ -------------- ----------- ----------------
5 5 -5 -6 0

l) ROUND

This will rounds numbers to a given number of digits of precision.

Syntax: round (*value, precision*)

Ex:

SQL> select round(123.2345), round(123.2345,2), round(123.2354,2) from dual;
ROUND(123.2345) ROUND(123.2345,0) ROUND(123.2345,2) ROUND(123.2354,2)
---------------------- ------------------------ ---------------------- ----------------------
123 123 123.23 123.24

SQL> select round(123.2345,-1), round(123.2345,-2), round(123.2345,-
3), round(123.2345,-4) from dual;
ROUND(123.2345,-1) ROUND(123.2345,-2) ROUND(123.2345,-3) ROUND(123.2345,-4)
------------------------ ------------------------ ------------------------ ------------------------
120 100 0 0

SQL> select round(123,0), round(123,1), round(123,2) from
dual; ROUND(123,0) ROUND(123,1) ROUND(123,2)
----------------- ----------------- ----------------
123 123 123

SQL> select round(-123,0), round(-123,1), round(-123,2) from
dual; ROUND(-123,0) ROUND(-123,1) ROUND(-123,2)
---------- ---------- ----------
-123 -123 -123

SQL> select round(123,-1), round(123,-2), round(123,-3), round(-123,-1), round(-
123,-2), round(-123,-3) from dual;
ROUND(123,-1) ROUND(123,-2) ROUND(123,-3) ROUND(-123,-1) ROUND(-123,-
2) ROUND(-123,-3)
---------- ---------- ---------- ---------- ---------- ----------
120 100 0 -120 -100 0

SQL> select round(null,null), round(0,0), round(1,1), round(-1,-1), round(-2,-2)
from dual;
ROUND(NULL,NULL) ROUND(0,0) ROUND(1,1) ROUND(-1,-1) ROUND(-2,-2)
---------- ---------- ---------- ---------- ----------
0 1 0 0

m) TRUNC

This will truncates or chops off digits of precision from a number.

Syntax: trunc (*value, precision*)

Ex:

SQL> select trunc(123.2345), trunc(123.2345,2), trunc(123.2354,2) from
dual; TRUNC(123.2345) TRUNC(123.2345,2) TRUNC(123.2354,2)

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

123 123.23 123.23

SQL> select trunc(123.2345,-1), trunc(123.2345,-2), trunc(123.2345,-
3), trunc(123.2345,-4) from dual;

TRUNC(123.2345,-1) TRUNC(123.2345,-2) TRUNC(123.2345,-3) TRUNC(123.2345,-4)
---------- ---------- ---------- ----------
120 100 0 0

SQL> select trunc(123,0), trunc(123,1), trunc(123,2) from dual;

TRUNC(123,0) TRUNC(123,1) TRUNC(123,2)
---------- ---------- ----------
123 123 123

SQL> select trunc(-123,0), trunc(-123,1), trunc(-123,2) from
dual; TRUNC(-123,0) TRUNC(-123,1) TRUNC(-123,2)

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

-123 -123 -123

SQL> select trunc(123,-1), trunc(123,-2), trunc(123,-3), trunc(-123,-1), trunc(-123,2), trunc(-123,-3) from dual;

TRUNC(123,-1) TRUNC(123,-2) TRUNC(123,-3) TRUNC(-123,-1) TRUNC(-123,2) TRUNC(-123,-3)

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

120 100 0 -120 -123 0

SQL> select trunc(null,null), trunc(0,0), trunc(1,1), trunc(-1,-1), trunc(-2,-2) from dual;

TRUNC(NULL,NULL) TRUNC(0,0) TRUNC(1,1) TRUNC(-1,-1) TRUNC(-2,-2)

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

0 1 0 0

n) BITAND

This will perform bitwise and operation.

Syntax: bitand (*value1, value2*)

Ex:

SQL> select bitand(2,3), bitand(0,0), bitand(1,1), bitand(null,null), bitand(-2,-3) from dual;

BITAND(2,3) BITAND(0,0) BITAND(1,1) BITAND(NULL,NULL) BITAND(-2,-3)

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

2 0 1 -4

o) GREATEST

This will give the greatest number.

Syntax: greatest (*value1, value2, value3 … valuen*)

Ex:

SQL> select greatest(1, 2, 3), greatest(-1, -2, -3) from dual;

GREATEST(1,2,3) GREATEST(-1,-2,-3)

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

3 -1

If all the values are zeros then it will display zero.

If all the parameters are nulls then it will display nothing.

If any of the parameters is null it will display nothing.

p) LEAST

This will give the least number.

Syntax: least (*value1, value2, value3 … valuen*)

Ex:

SQL> select least(1, 2, 3), least(-1, -2, -3) from dual;

LEAST(1,2,3) LEAST(-1,-2,-3)

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

1 -3

If all the values are zeros then it will display zero.

If all the parameters are nulls then it will display nothing. If any of the parameters is null it will display nothing.

q) COALESCE

This will return first non-null value.

Syntax: coalesce (*value1, value2, value3 … valuen*)

Ex:

SQL> select coalesce(1,2,3), coalesce(null,2,null,5) from dual;

COALESCE(1,2,3) COALESCE(NULL,2,NULL,5)

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

1 2

STRING FUNCTIONS

Initcap	Soundex
Upper	Concat (‘ || ‘ Concatenati on operator)
Lower	Ascii
Length	Chr
Rpad	Substr
Lpad	Instr
Ltrim	Decode
Rtrim	Greatest
Trim	Least
Translate	Coalesce
Replace	

a) INITCAP

This will capitalize the initial letter of the string. **Syntax:** initcap (*string*)

Ex:

SQL> select initcap('computer') from dual;

INITCAP

Computer

b) UPPER

This will convert the string into uppercase.

Syntax: upper (*string*)

Ex:

SQL> select upper('computer') from dual;

UPPER

COMPUTER

c) LOWER

This will convert the string into lowercase.

Syntax: lower (*string*)

Ex:

SQL> select lower('COMPUTER') from dual;

LOWER

---

computer

d) LENGTH

This will give length of the string.

Syntax: length (*string*)

Ex:

SQL> select length('computer') from dual;

LENGTH

8

e) RPAD

This will allows you to pad the right side of a column with any set of characters.

Syntax: rpad (*string, length [, padding_char]*)

Ex:

SQL> select rpad('computer',15,'*'), rpad('computer',15,'*#') from dual;

RPAD('COMPUTER' RPAD('COMPUTER'

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

computer******* computer*#*#*#*

-- Default padding character was blank space.

f) LPAD

This will allows you to pad the left side of a column with any set of characters.

Syntax: lpad (*string, length [, padding_char]*)

Ex:

SQL> select lpad('computer',15,'*'), lpad('computer',15,'*#') from dual;

LPAD('COMPUTER' LPAD('COMPUTER'

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

*******computer *#*#*#*computer

Default padding character was blank space.

g) LTRIM

This will trim off unwanted characters from the left end of string.

Syntax: ltrim (*string [,unwanted_chars]*)

Ex:

SQL> select ltrim('computer','co'), ltrim('computer','com') from dual;

LTRIM(LTRIM

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

mputer puter

```
SQL> select ltrim('computer','puter'), ltrim('computer','omputer') from dual;
LTRIM('C LTRIM('C
---------- ----------
computer computer
```

If you haven't specify any unwanted characters it will display entire string.

h) RTRIM

This will trim off unwanted characters from the right end of string.

Syntax: rtrim (*string [, unwanted_chars]*)

Ex:

```
SQL> select rtrim('computer','er'), rtrim('computer','ter') from dual;
RTRIM( RTRIM
-------- ---------
comput compu

SQL> select rtrim('computer','compute') from dual;
RTRIM('C RTRIM('C
---------- ----------
computer computer
```

If you haven't specify any unwanted characters it will display entire string.

i) TRIM

This will trim off unwanted characters from the both sides of string.

Syntax: trim (*unwanted_chars* from *string*)

Ex:

```
SQL> select trim( 'i' from 'indiani') from dual;
TRIM(
-----
ndian
SQL> select trim( leading'i' from 'indiani') from dual; -- this will work as LTRIM
TRIM(L
------
ndiani

SQL> select trim( trailing'i' from 'indiani') from dual; -- this will work as RTRIM
TRIM(T
------
Indian
```

j) TRANSLATE

This will replace the set of characters, character by character.

Syntax: translate (*string*, *old_chars*, *new_chars*)

Ex:

SQL> select translate('india','in','xy') from dual;

TRANS
----
xydxa

k) REPLACE

This will replace the set of characters, string by string.

Syntax: replace (*string*, *old_chars [*, *new_chars]*)

Ex:

SQL> select replace('india','in','xy'), replace('india','in') from dual;

REPLACE REPLACE
----------- -----------
Xydia dia

l) SOUNDEX

This will be used to find words that sound like other words, exclusively used in where clause.

Syntax: soundex (*string*)

Ex:

SQL> select * from emp where soundex(ename) = soundex('SMIT');

EMPNO ENAME JOB MGR HIREDATE SAL DEPTNO
-------- -------- ----- ----- ------------ --------- ----------

7369 SMITH CLERK 7902 17-DEC-80 500
20 **m) CONCAT**

This will be used to combine two strings only.

Syntax: concat (*string1*, *string2*)

Ex:

SQL> select concat('computer',' operator') from dual;

CONCAT('COMPUTER'
----
computer operator

If you want to combine more than two strings you have to use concatenation operator (||).

SQL> select 'how' || ' are' || ' you' from
dual; 'HOW'||'ARE
----
how are you

n) ASCII

This will return the decimal representation in the database character set of the first character of the string.

Syntax: ascii (*string*)

Ex:

SQL> select ascii('a'), ascii('apple') from dual;

ASCII('A') ASCII('APPLE')

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

97 97

o) CHR

This will return the character having the binary equivalent to the string in either the database character set or the national character set.

Syntax: chr (*number*)

Ex:

SQL> select chr(97) from dual;

CHR

---

a

p) SUBSTR

This will be used to extract substrings.

Syntax: substr (*string, start_chr_count [, no_of_chars]*)

Ex:

SQL> select substr('computer',2), substr('computer',2,5), substr('computer',3,7)
from dual;

SUBSTR(SUBST SUBSTR

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

omputer omput mputer

If *no_of_chars* parameter is negative then it will display nothing.

If both parameters except *string* are null or zeros then it will display nothing.

If *no_of_chars* parameter is greater than the length of the string then it ignores and calculates based on the orginal string length.

If *start_chr_count* is negative then it will extract the substring from right end.

1 2 3 4 5 6 7 8

C O M P U T E R -8

-7 -6 -5 -4 -3 -2 -1

q) INSTR

This will allows you for searching through a string for set of characters.

Syntax: instr (*string, search_str [, start_chr_count [, occurrence]]*)

Ex:

```
SQL> select instr('information','o',4,1), instr('information','o',4,2) from dual;
INSTR('INFORMATION','O',4,1) INSTR('INFORMATION','O',4,2)
------------------------------------ ------------------------------------
4 10
```

If you are not specifying *start_chr_count* and *occurrence* then it will start search from the beginning and finds first occurrence only.

If both parameters *start_chr_count* and *occurrence* are null, it will display nothing.

r) DECODE

Decode will act as value by value substitution.

For every value of field, it will checks for a match in a series of if/then tests.

Syntax: decode (*value, if1, then1, if2, then2, ……. else*);

Ex:

```
SQL> select sal, decode(sal,500,'Low',5000,'High','Medium') from emp;
SAL     DECODE
-----   ---------
500     Low
2500    Medium
2000    Medium
3500    Medium
3000    Medium
5000    High
4000    Medium
5000    High
1800    Medium
1200    Medium
2000    Medium
2700    Medium
2200    Medium
3200    Medium
```

```
SQL> select decode(1,1,3), decode(1,2,3,4,4,6) from dual;
DECODE(1,1,3) DECODE(1,2,3,4,4,6)
----------------- ------------------------
3                 6
```

If the number of parameters are odd and different then decode will display nothing.

If the number of parameters are even and different then decode will display last value. If all the parameters are null then decode will display nothing.

If all the parameters are zeros then decode will display zero.

s) GREATEST

This will give the greatest string.

Syntax: greatest (*strng1, string2, string3 … stringn*)

Ex:

SQL> select greatest('a', 'b', 'c'), greatest('satish','srinu','saketh') from dual;

GREAT GREAT

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

c srinu

If all the parameters are nulls then it will display nothing. If any of the parameters is null it will display nothing.

t) LEAST

This will give the least string.

Syntax: greatest (*strng1, string2, string3 … stringn*)

Ex:

SQL> select least('a', 'b', 'c'), least('satish','srinu','saketh') from dual;

LEAST LEAST

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

a saketh

If all the parameters are nulls then it will display nothing. If any of the parameters is null it will display nothing.

u) COALESCE

This will gives the first non-null string.

Syntax: coalesce (*strng1, string2, string3 … stringn*)

Ex:

SQL> select coalesce('a','b','c'), coalesce(null,'a',null,'b') from dual;

COALESCE COALESCE

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

a a

DATE FUNCTIONS

Sysdate	To_char	Greatest
Current_date	To_date	Least
Current_timestamp	Add_months	Round
Systimestamp	Months_between	Trunc
Localtimestamp	Next_day	New_time
Dbtimezone	Last_day	Coalesce
Sessiontimezone	Extract	

Oracle default date format is DD-MON-YY.

We can change the default format to our desired format by using the following command.

SQL> alter session set nls_date_format='DD-MON-YYYY';

But this will expire once the session was closed.

a) SYSDATE

This will give the current date and time.

Ex:

SQL> select sysdate from dual;

SYSDATE

24-DEC-06

b) CURRENT_DATE

This will returns the current date in the session's timezone.

Ex:

SQL> select current_date from dual;

CURRENT_DATE

24-DEC-06

c) CURRENT_TIMESTAMP

This will returns the current timestamp with the active time zone information.

Ex:

SQL> select current_timestamp from dual;

CURRENT_TIMESTAMP

24-DEC-06 03.42.41.383369 AM +05:30

d) SYSTIMESTAMP

This will returns the system date, including fractional seconds and time zone of the database.

Ex:

SQL> select systimestamp from dual;

SYSTIMESTAMP

24-DEC-06 03.49.31.830099 AM +05:30

e) LOCALTIMESTAMP

This will returns local timestamp in the active time zone information, with no time zone information shown.

Ex:

SQL> select localtimestamp from dual;

LOCALTIMESTAMP

24-DEC-06 03.44.18.502874 AM

f) DBTIMEZONE

This will returns the current database time zone in UTC format. (Coordinated Universal Time)

Ex:

SQL> select dbtimezone from dual;

DBTIMEZONE

-07:00

g) SESSIONTIMEZONE

This will returns the value of the current session's time zone.

Ex:

SQL> select sessiontimezone from dual;

SESSIONTIMEZONE

+05:30

h) TO_CHAR

This will be used to extract various date formats.

The available date formats as follows.

Syntax: to_char (*date*, *format*)

DATE FORMATS

D	--	No of days in week
DD	--	No of days in month
DDD	--	No of days in year
MM	--	No of month
MON	--	Three letter abbreviation of month
MONTH	--	Fully spelled out month
RM	--	Roman numeral month
DY	--	Three letter abbreviated day
DAY	--	Fully spelled out day
Y	--	Last one digit of the year
YY	--	Last two digits of the year
YYY	--	Last three digits of the year
YYYY	--	Full four digit year
SYYYY	--	Signed year
I	--	One digit year from ISO standard
IY	--	Two digit year from ISO standard
IYY	--	Three digit year from ISO standard
IYYY	--	Four digit year from ISO standard
Y, YYY	--	Year with comma
YEAR	--	Fully spelled out year
CC	--	Century
Q	--	No of quarters
W	--	No of weeks in month
WW	--	No of weeks in year
IW	--	No of weeks in year from ISO standard
HH	--	Hours
MI	--	Minutes
SS	--	Seconds
FF	--	Fractional seconds
AM or PM	--	Displays AM or PM depending upon time of day
A.M or P.M	--	Displays A.M or P.M depending upon time of day
AD or BC	--	Displays AD or BC depending upon the date
A.D or B.C	--	Displays AD or BC depending upon the date
FM	--	Prefix to month or day, suppresses padding of month or day
TH	--	Suffix to a number
SP	--	suffix to a number to be spelled out
SPTH	--	Suffix combination of TH and SP to be both spelled out
THSP	--	same as SPTH

Ex:

SQL> select to_char(sysdate,'dd month yyyy hh:mi:ss am dy') from dual;

TO_CHAR(SYSDATE,'DD MONTH YYYYHH:MI

---

24 december 2006 02:03:23 pm sun

```
SQL> select to_char(sysdate,'dd month year') from dual;
TO_CHAR(SYSDATE,'DDMONTHYEAR')
------------------------------------------------------
24 december two thousand six

SQL> select to_char(sysdate,'dd fmmonth year') from dual;
TO_CHAR(SYSDATE,'DD FMMONTH YEAR')
------------------------------------------------------
24 december two thousand six

SQL> select to_char(sysdate,'ddth DDTH') from dual;
TO_CHAR(S
------------
24th 24TH

SQL> select to_char(sysdate,'ddspth DDSPTH') from dual;
TO_CHAR(SYSDATE,'DDSPTHDDSPTH
-----------------------------------------
twenty-fourth TWENTY-FOURTH

SQL> select to_char(sysdate,'ddsp Ddsp DDSP ') from dual;
TO_CHAR(SYSDATE,'DDSPDDSPDDSP')
-------------------------------------------------
twenty-four Twenty-Four TWENTY-FOUR
```

i) TO_DATE

This will be used to convert the string into data format.

Syntax: to_date (*date*)

Ex:

```
SQL> select to_char(to_date('24/dec/2006','dd/mon/yyyy'), 'dd * month * day') from
dual; TO_CHAR(TO_DATE('24/DEC/20
--------------------------
24 * december * Sunday
-- If you are not using to_char oracle will display output in default date format.
```

j) ADD_MONTHS

This will add the specified months to the given date.

Syntax: add_months (*date, no_of_months*)

Ex:

SQL> select add_months(to_date('11-jan-1990','dd-mon-yyyy'), 5) from dual;

ADD_MONTHS

11-JUN-90

SQL> select add_months(to_date('11-jan-1990','dd-mon-yyyy'), -5) from dual;

ADD_MONTH

11-AUG-89

If *no_of_months* is zero then it will display the same date.

If *no_of_months* is null then it will display nothing.

k) MONTHS_BETWEEN

This will give difference of months between two dates.

Syntax: months_between (*date1, date2*)

Ex:

SQL> select months_between(to_date('11-aug-1990','dd-mon-yyyy'), to_date('11-jan-1990','dd-mon-yyyy')) from dual; MONTHS_BETWEEN(TO_DATE('11-AUG-1990','DD-MON-YYYY'),TO_DATE('11-JAN-1990','DD-MON-YYYY'))

7

SQL> select months_between(to_date('11-jan-1990','dd-mon-yyyy'), to_date('11-aug-1990','dd-mon-yyyy')) from dual; MONTHS_BETWEEN(TO_DATE('11-JAN-1990','DD-MON-YYYY'),TO_DATE('11-AUG-1990','DD-MON-YYYY'))

-7

l) NEXT_DAY

This will produce next day of the given day from the specified date. **Syntax:** next_day (*date, day*)

Ex:

```
SQL> select next_day(to_date('24-dec-2006','dd-mon-yyyy'),'sun') from dual;

NEXT_DAY(

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

31-DEC-06
```

-- If the day parameter is null then it will display nothing.

m) LAST_DAY

This will produce last day of the given date. **Syntax:** last_day (*date*)

Ex:

```
SQL> select last_day(to_date('24-dec-2006','dd-mon-yyyy'),'sun') from dual;
LAST_DAY(
-------------
31-DEC-06
```

n) EXTRACT

This is used to extract a portion of the date value.

Syntax: extract ((year | month | day | hour | minute | second), *date*)

Ex:

```
SQL> select extract(year from sysdate) from dual;
EXTRACT(YEARFROMSYSDATE)
-----------------------------------
2006
```

-- You can extract only one value at a time. **o) GREATEST**

This will give the greatest date.

Syntax: greatest (*date1, date2, date3 … daten*)

Ex:

```
SQL> select greatest(to_date('11-jan-90','dd-mon-yy'),to_date('11-mar-90','dd-monyy'),to_date('11-
apr-90','dd-mon-yy')) from dual;
GREATEST(
-------------
11-APR-90
```

p) LEAST

This will give the least date.

Syntax: least (*date1, date2, date3 … daten*)

Ex:

SQL> select least(to_date('11-jan-90','dd-mon-yy'),to_date('11-mar-90','dd-monyy'), to_date('11-apr-90','dd-mon-yy')) from dual;

LEAST(

11-JAN-90

q) ROUND

Round will rounds the date to which it was equal to or greater than the given date.

Syntax: round (*date, (*day | month | year*)*)

If the second parameter was *year* then round will checks the month of the given date in the following ranges. JAN -- JUN

JUL -- DEC

If the month falls between JAN and JUN then it returns the first day of the current year. If the month falls between JUL and DEC then it returns the first day of the next year.

If the second parameter was *month* then round will checks the day of the given date in the following ranges. 1 -- 15 16 -- 31

If the day falls between 1 and 15 then it returns the first day of the current month. If the day falls between 16 and 31 then it returns the first day of the next month.

If the second parameter was *day* then round will checks the week day of the given date in the following ranges.

SUN -- WED

THU -- SUN

If the week day falls between SUN and WED then it returns the previous sunday. If the weekday falls between THU and SUN then it returns the next sunday.

If the second parameter was null then it returns nothing.

If the you are not specifying the second parameter then round will resets the time to the begining of the current day in case of user specified date.

If the you are not specifying the second parameter then round will resets the time to the begining of the next day in case of sysdate.

Ex:

```
SQL> select round(to_date('24-dec-04','dd-mon-yy'),'year'), round(to_date('11-mar- 06','dd-mon-yy'),'year') from dual;
ROUND(TO_ ROUND(TO_
---------- ----------
01-JAN-05 01-JAN-06


SQL> select round(to_date('11-jan-04','dd-mon-yy'),'month'), round(to_date('18-jan- 04','dd-mon-yy'),'month') from dual;
ROUND(TO_ ROUND(TO_
---------- ----------
01-JAN-04 01-FEB-04

SQL> select round(to_date('26-dec-06','dd-mon-yy'),'day'), round(to_date('29-dec-06','dd-mon-yy'),'day') from dual;
ROUND(TO_ ROUND(TO_
---------- ----------
24-DEC-06 31-DEC-06

SQL> select to_char(round(to_date('24-dec-06','dd-mon-yy')), 'dd mon yyyy hh:mi:ss am')from dual;
TO_CHAR(ROUND(TO_DATE('
--------------------------------
24 dec 2006 12:00:00 am
```

r) TRUNC

Trunc will chops off the date to which it was equal to or less than the given date.

Syntax: trunc (*date, (*day | month | year*)*)

If the second parameter was *year* then it always returns the first day of the current year.

If the second parameter was *month* then it always returns the first day of the current month. If the second parameter was *day* then it always returns the previous sunday.

If the second parameter was null then it returns nothing.

If the you are not specifying the second parameter then trunk will resets the time to the begining of the current day.

Ex:

SQL> select trunc(to_date('24-dec-04','dd-mon-yy'),'year'), trunc(to_date('11-mar-06','dd-mon-yy'),'year') from dual;

TRUNC(TO_ TRUNC(TO_

--- ---

01-JAN-04 01-JAN-06

SQL> select trunc(to_date('11-jan-04','dd-mon-yy'),'month'), trunc(to_date('18-jan-04','dd-mon-yy'),'month') from dual;

TRUNC(TO_ TRUNC(TO_

--- ---

01-JAN-04 01-JAN-04

SQL> select trunc(to_date('26-dec-06','dd-mon-yy'),'day'), trunc(to_date('29-dec-06','ddmon- yy'),'day') from dual;

TRUNC(TO_ TRUNC(TO_

--- ---

24-DEC-06 24-DEC-06

SQL> select to_char(trunc(to_date('24-dec-06','dd-mon-yy')), 'dd mon yyyy hh:mi:ss am') from dual; TO_CHAR(TRUNC(TO_DATE('

---

24 dec 2006 12:00:00 am

s) NEW_TIME

This will give the desired timezone's date and time.

Syntax: new_time (*date, current_timezone, desired_timezone*)

Available timezones are as follows.

TIMEZONES

AST/ADT	--	Atlantic standard/day light time
BST/BDT	--	Bering standard/day light time
CST/CDT	--	Central standard/day light time
EST/EDT	--	Eastern standard/day light time
GMT	--	Greenwich mean time
HST/HDT	--	Alaska-Hawaii standard/day light time
MST/MDT	--	Mountain standard/day light time
NST	--	Newfoundland standard time
PST/PDT	--	Pacific standard/day light time
YST/YDT	--	Yukon standard/day light time

Ex:

```
SQL> select to_char(new_time(sysdate,'gmt','yst'),'dd mon yyyy hh:mi:ss am') from dual;

TO_CHAR(NEW_TIME(SYSDAT
-----------------------------------
24 dec 2006 02:51:20 pm

SQL> select to_char(new_time(sysdate,'gmt','est'),'dd mon yyyy hh:mi:ss am') from dual;

TO_CHAR(NEW_TIME(SYSDAT
----------------------
24 dec 2006 06:51:26 pm
```

t) COALESCE

This will give the first non-null date.

Syntax: coalesce (*date1, date2, date3 … daten*)

Ex:

```
SQL> select coalesce('12-jan-90','13-jan-99'), coalesce(null,'12-jan-90','23-mar-98',null)
from dual;

COALESCE( COALESCE(
------------ ------------
12-jan-90 12-jan-90
```

Uid

User

Vsize

Rank

Dense_rank

a) UID

This will returns the integer value corresponding to the user currently logged in.

Ex:

SQL> select uid from dual;

UID

319

b) USER

This will returns the login's user name.

Ex:

SQL> select user from dual;

USER

SAKETH

c) VSIZE

This will returns the number of bytes in the expression.

Ex:

SQL> select vsize(123), vsize('computer'), vsize('12-jan-90') from dual;

VSIZE(123)	VSIZE('COMPUTER')	VSIZE('12-JAN-90')
3	8	9

d) RANK

This will give the non-sequential ranking.

Ex:

SQL> select rownum,sal from (select sal from emp order by sal desc);

ROWNUM	SAL
1	5000
2	3000
3	3000
4	2975
5	2850
6	2450
7	1600
8	1500
9	1300
10	1250
11	1250
12	1100
13	1000
14	950
15	800

SQL> select rank(2975) within group(order by sal desc) from emp;

RANK(2975)WITHINGROUP(ORDERBYSALDESC)

4

d) DENSE_RANK

This will give the sequential ranking.

Ex:

SQL> select dense_rank(2975) within group(order by sal desc) from emp;

DENSE_RANK(2975)WITHINGROUP(ORDERBYSALDESC)

3

CONVERSION FUNCTIONS

Bin_to_num

Chartorowid

Rowidtochar

To_number

To_char

To_date

a) BIN_TO_NUM

This will convert the binary value to its numerical equivalent.

Syntax: bin_to_num(*binary_bits*)

Ex:

SQL> select bin_to_num(1,1,0) from dual;

BIN_TO_NUM(1,1,0)

6

If all the bits are zero then it produces zero.

If all the bits are null then it produces an error.

b) CHARTOROWID

This will convert a character string to act like an internal oracle row identifier or rowid.

c) ROWIDTOCHAR

This will convert an internal oracle row identifier or rowid to character string.

d) TO_NUMBER

This will convert a char or varchar to number.

e) TO_CHAR

This will convert a number or date to character string.

f) TO_DATE

This will convert a number, char or varchar to a date.

GROUP FUNCTIONS

Sum

Avg

Max

Min

Count

Group functions will be applied on all the rows but produces single output.

a) SUM

This will give the sum of the values of the specified column.

Syntax: sum (*column*)

Ex:

SQL> select sum(sal) from emp;

SUM(SAL)

38600

b) AVG

This will give the average of the values of the specified column.

Syntax: avg (*column*)

Ex:

SQL> select avg(sal) from emp;

AVG(SAL)

2757.14286

c) MAX

This will give the maximum of the values of the specified column.

Syntax: max (*column*)

Ex:

SQL> select max(sal) from emp;

MAX(SAL)

5000

d) MIN

This will give the minimum of the values of the specified column.

Syntax: min (*column*)

Ex:

SQL> select min(sal) from emp;

MIN(SAL)

500

e) COUNT

This will give the count of the values of the specified column.

Syntax: count (*column*)

Ex:

SQL> select count(sal),count(*) from emp;

COUNT(SAL)	COUNT(*)
14	14