I N T R O D U C T I O N TO S Q L

SQL

- ❖ Structured Query Language (SQL) is a language that all commercial RDBMS implementations understand.
- ❖ SQL is a non-procedural language.
- ❖ SQL is a Unified Language. It provides statements for a variety of tasks.
- ❖ Developed in a prototype relational database management **System R** by IBM in mid 1970s and Oracle Corporation introduced the first commercial SQL in 1979.

SQL Vs SQL *PLUS

SQL	***SQL *PLUS***
▪ A language ▪ ANSI standard ▪ Keyword cannot be abbreviated ▪ Statements manipulate data and table definitions in the database ▪ SQL statements are stored in buffer	▪ An environment ▪ Oracle proprietary ▪ Keywords can be abbreviated ▪ Commands do not allow manipulation of values in the database ▪ SQL*PLUS statements are not stored in Buffer

Oracle RDBMS Limits

Item	***Limit***
Tables in a database	No limit
Rows in a table	No limit
Columns in a table	254
Characters in a row	1,26,495
Characters in character field	240
Digits in a number field	105
Significant digits in a number filed	40
Ranges of values in a date field	1-JAN-4712 BC *to* 31-DEC-4712 AD
Indexes on a table	No limit
Tables or views joined in a query	No limit
Levels of nested sub queries	255
Characters in name	30

Naming Rules

- ❖ Begin with a letter and followed by zero or more of characters A-Z, 0-9, _, $, #
- ❖ Not case sensitive

Data types

- ❖ Integers Decimal numbers--- NUMBER, INTEGER .

  Number is an oracle data type. Integer is an ANSI data type. The syntax for NUMBER is NUMBER(P,S) p is the precision and s is the scale. P can range from 1 to 38 and s from -84 to +127
- ❖ Floating point numbers---- FLOAT
- ❖ Fixed length character strings---- CHAR (len)

  Fixed length character data of length len bytes.
- ❖ Variable length character strings --- Varchar2(len)

  Variable length character string having maximum length *len* bytes.
- ❖ Dates-----DATE
- ❖ Character data of variable size uo to 32760 characters --- LONG
- ❖ Raw binary data, size bytes long. Maximum size is 32767 bytes --- RAW(size)

Constants/Literals

- ❖ ANSI standard defines format for literals
- ❖ Numeric: 21, -32, $0.75,1.2E4
- ❖ String: enclosed within single quote
- ❖ Date Format : 12-mar-03

Operators

- ❖ Arithmetic operators like +,-,*,/
- ❖ Logical operators: AND, OR
- ❖ Relational operators: =,<=,>=, < >

- The Arithmetic operators are used to calculate something like given in the example below:

  ```
  Select * from employee where sal * 1.1 > 1000 ;
  ```
- The logical operators are used to combine conditions like:

  ```
  Select * from employee where (sal > 1000 AND age > 25);
  ```

- The above two examples also illustrate use of relational operators

NULL

- ❖ Missing/unknown/inapplicable data represented as a **null** value
- ❖ NULL is not a data value. It is just an indicator that the value is unknown

Statements

- SQL has three flavours of statements.

DDL is Data Definition Language statements. Some examples:

- 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
- COMMENT - add comments to the data dictionary

DML is Data Manipulation Language statements. Some examples:

- SELECT - retrieve data from the a database
- 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
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

TCL(Transaction Control Language) is a DML

- 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 what rollback segment to use

DCL is Data Control Language statements. Some examples:

- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

Database Objects

❖ Table	-- Basic unit of storage; composed of rows and columns
❖ View	-- Logically represents subsets of data from one or more tables
❖ Sequence	-- Generates primary key values
❖ Index	-- Improves the performance of some queries
❖ Synonym	-- Gives alternative names to objects

SQL *PLUS EDITING COMMANDS

EDIT filename	→ Invokes text editor
L	→ List lines of the current buffer from first to last
C/old/new	→ Change *old* to *new* in the current line of the current buffer. (If *old* is prefixed with '…' then it matches everything up to including the first

occurrence of *old*. If *old* is suffixed with '…' then it matches everything including and after the first occurrence of *old*.)

A *text* → Add *text* to the end of current line in current buffer.

DEL → Delete the current line in current buffer.

SAVE *name* CREATE/ REPLACE/ APPEND → Save the content of current buffer in to the file *name*. If filetype is missing, then CREATE is the default.

@*filename* → Run the file *filename* which contains SQL statements. The file may contain any commands that may be interactively.

RUN → Displays and run the command in the buffer

/ → Run the command in the buffer without displaying.

START *file* → Run the *file*. The file may contain any commands that may not be interactively.

* * *

Ex.No. 1	CREATING DATABASE TABLE	Date :

CREATE TABLE

CREATE TABLE *tablename* (column_name data_ type constraints, …)

- Used to create a table by defining its structure, the data type and name of the various columns, the relationships with columns of other tables etc.

Q1) Create the tables DEPT and EMP as described below

DEPT

Column name	*Data type*	*Description*
DEPTNO	Number	Department number
DNAME	Varchar	Department name
LOC	Varchar	Department Location

EMP

Column name	*Data type*	*Description*
EMPNO	Number	Employee number
ENAME	Varchar	Employee name
JOB	Char	Designation
MGR	Number	Manager's EMP.No.
HIREDATE	Date	Date of joining
SAL	Number	Basic Salary
COMM	Number	Commission
DEPTNO	Number	Department Number

SQL>

SQL>

Q2) Confirm table creation

desc *table-name;*

SQL>
SQL>

Q3) List name of the tables created by the user

SQL>select * from tab;

Q4) Describe tables owned by the user

SQL> SELECT * FROM user_tables;

Q5) View distinct object types owned by the user

SQL> SELECT DISTINCT object_type FROM user_objects;

Q6) View tables, views, synonyms, and sequences owned by the user

SQL> SELECT * FROM user_catalog;

ALTER TABLE

- *Add Column*

  ALTER TABLE *table* ADD(*column data type* [DEFAULT *expr*]
  [, *column data type*]...);
- *Drop Column*

  ALTER TABLE *table* DROP column column_name;

- *Modify Column*

  ALTER TABLE *table* MODIFY(*column data type* [DEFAULT *expr*]
  [, *column data type*]...);

Q7) Add new columns COMNT and MISCEL in DEPT table of character type.

SQL >

Q8) Modify the size of column LOC by 15 in the DEPT table

SQL >

Q9) Set MISCEL column in the DEPT table as unused

SQL >

Q10) Drop the column COMNT from the table DEPT

SQL >

Q11) Drop unused columns in DEPT table

SQL >

Q12) Rename the table DEPT to DEPT12

SQL >

Q13) Remove all the rows in the table DEPT12 (Presently no records in DEPT12)

SQL >

ADDING COMMENTS TO THE TABLES

- You can add comments to a table or column by using the COMMENT statement.
- Comments can be viewed through the data dictionary views.
  - ALL_COL_COMMENTS
  - USER_COL_COMMENTS
  - ALL_TAB_COMMENTS
  - USER_TAB_COMMENTS

Syntax :

COMMENT ON TABLE table **IS** 'xxxxxxxxxx';

Q14) Add some comment to the table DEPT12 and also confirm the inclusion of comment

SQL >

SQL >

Q15) Delete the table DEPT12 from the database.

SQL >

Q16) Confirm the removal of table DEPT12 from the database.

SQL >

Verified by

Staff In-charge Sign :	**Date :**

Ex.No. 2	**Working with Data Manipulation commands**	**Date :**

DML

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

Data for EMP table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Data for DEPT table

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

INSERT STATEMENT

- Add new rows to a table by using the INSERT statement.

  (i) INSERT INTO *table* VALUES*(value1, value2,..);*
  - Only one row is inserted at a time with this syntax.
  - List values in the default order of the columns in the table
  - Enclose character and date values within single quotation marks.
  - Insert a new row containing values for each column

(ii) **INSERT INTO** ***table(column1, column2,..).***
VALUES*(value1, value2,..);*

- Rows can be inserted with NULL values either
  - by omitting column from the column list or
  - by specifying NULL in the value field.

(iii) **INSERT INTO** ***table(column1, column2,...)***
VALUES*(&value1,& value2,..);*

- Substitution variable(&) helps us to write an interactive script for inserting rows

Q1) Insert the rows of DEPT table using syntax (i)
SQL>

Q2) Insert first & second rows of EMP table using syntax (ii)
SQL>

Q3) Insert the remaining rows of EMP table using syntax (iii).
SQL>

Q4) Create a table MANAGER with the columns *mgr-id, name, salary* and *hiredate*
SQL> CREATE TABLE ***manager(mgr-id number(5) primary key,***
name varchar(20), sal number(5), hiredate date);

Q5) Insert values into the table MANAGER by copying the values from EMP table where the designation of the employee is 'MANAGER'
SQL> INSERT INTO ***manager*** **SELECT** ***empno,ename,sal,hiredate***
FROM emp WHERE job='MANAGER';

UPDATE STATEMENT

- Modify existing rows with the UPDATE statement.
- Update more than one row at a time, if required.
- All rows in the table are modified if you omit the WHERE clause

UPDATE ***table*** **SET** ***column* = *value*, *column* = *value,* …. WHERE *condition*;**

Q6) Change the LOC of all rows of DEPT table by 'NEW YORK'
SQL>

Q7) Change the LOC='DALLAS' for deptno=20 in DEPT table.
SQL>

DELETE STATEMENT

- You can remove existing rows from a table by using the DELETE statement.
- All rows in the table are deleted if you omit the WHERE clause.

DELETE FROM *table* WHERE *condition*;

Q8) Delete the rows from EMP table whose employee name = 'PAUL'
SQL>

SELECT STATEMENT

- To perform a query we use select statement

SELECT [DISTINCT] {*, *column* [*alias*],...}
FROM *table;*

- o *Select* Clause determines what columns
- o *From* Clause determines which table.
- o *Where* Clause specifies the conditions

Q9) List all the columns and rows of the table DEPT
SQL>

Q10) List the name of the employee and salary of EMP table
SQL>

Q11) Without duplication, list all names of the department of DEPT table.
SQL>

Q12) Find out the name of an employee whose EMPNO is 7788.
SQL>

Q13) As a copy of DEPT table, create DEPT1 table using select command.

```
SQL> CREATE TABLE dept1 AS SELECT * FROM dept;
```

Q14) List ename and sal of EMP table with the column headings NAME and SALARY

```
SQL> SELECT ename as name, sal salary
     FROM emp;
```

DATABASE TRANSACTIONS

- Begin when the first executable SQL statement is executed
- End with one of the following events:
  - COMMIT or ROLLBACK
  - DDL or DCL statement executes (automatic commit)
  - User exits
  - System crashes

TCL (Transaction Control Language)

- TCL Statements are COMMIT, ROLLBACK & SAVE POINT.
- State of Data Before COMMIT or ROLLBACK
  - The previous state of the data can be recovered.
  - The current user can review the results of the DML operations by using the SELECT statement.
  - Other users *cannot* view the results of the DML statements by the current user.
  - The affected rows are *locked*; other users cannot change the data within the affected rows.
- State of Data After COMMIT
  - Data changes are made permanent in the database.
  - The previous state of the data is permanently lost.
  - All users can view the results.
  - Locks on the affected rows are released; those rows are available for other users to manipulate.
  - All savepoints are erased.

Q15) Change LOC='CHICAGO' for deptno=30 in DEPT table and COMMIT the transaction.

```
SQL>

SQL>
```

- State of Data After ROLLBACK
  - Discard all pending changes by using the ROLLBACK statement.
  - Data changes are undone.
  - Previous state of the data is restored.
  - Locks on the affected rows are released.

Q16) Delete all the rows from EMP table and ROLLBACK the transaction.

SQL>

SQL>

- Rolling Back to a Marker
  - Create a marker within a current transaction by using <SAVEPOINT savepoint-name>
  - Roll back to that marker by using <ROLLBACK TO savepoint-name>

Q17) Do the following operations one after another

a) Change LOC='BOSTON' for deptno=40 in DEPT table

SQL>

b) Create SAVEPOINT in the name 'update_over'

SQL>

c) Insert another row in DEPT table with your won values

SQL>

d) Rollback the transaction upto the point 'update_over'

SQL>

Verified by

Staff In-charge Sign :	**Date :**

Ex.No. 3	**I N T E G R I T Y C O N S T R A I N T S**	**Date :**

CONSTRAINTS

- Constraints enforce rules at the table level.Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid in Oracle:
  - NOT NULL
  - UNIQUE Key
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK
- Name a constraint or the Oracle Server will generate a name by using the SYS_C*n* format.
- Create a constraint:
  - At the same time as the table is created
  - After the table has been created
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

DEFINING CONSTRAINTS

- Column constraint level

  column* [CONSTRAINT *constraint_name*] *constraint_type
- Table constraint level

  [CONSTRAINT *constraint_name*] *constraint_type*(*column*)

```
CREATE TABLE table (column data type, column_constraint,
                    …. …. ....,
                    … …  …,
                    table_constraint);
```

Q1) Create table EMP1 with columns similar to EMP table and create NOT NULL (column) constraint for DEPTNO column and PRIMARY KEY (table) constraint for EMPNO column.

```
SQL> CREATE TABLE emp1( empno  number(4),
        ename  varchar2(10), job char(20), mgr number(10),
        hiredate date, sal number(5), comm number(5),
        deptno  number(7,2) NOT NULL,
        CONSTRAINT emp1_pk  PRIMARY KEY (empno));
```

NOT NULL Constraint

- Ensures that null values are not permitted for the column

CHECK Constraint

- Defines a condition that each row must satisfy

UNIQUE Constraint

- Prevent the duplication of values within the rows of a specified column

PRIMARY KEY Constraint

- Avoids duplication of rows and does not allow NULL values

FOREIGN KEY Constraint

- To establish a 'parent-child' or a 'master-detail' relationship between two tables having a common column, we make use of Foreign key (referential integrity) constraints.
- To do this we should define the column in the parent table as primary key and the same column in the child table as a foreign key referring to the corresponding parent entry.

  FOREIGN KEY
    - Defines the column in the child table at the table constraint level

  REFERENCES
    - Identifies the table and column in the parent table

  ON DELETE CASCADE
    - Allows deletion in the parent table and deletion of the dependent rows in the child table

ADDING A CONSTRAINT

- Add or drop, but not modify, a constraint
- Add a NOT NULL constraint by using the MODIFY clause

ALTER TABLE *table* ADD CONSTRAINT *const-name* cons-*type* (*column*);

Q2) Add NOT NULL constraint to the columns ENAME and JOB of EMP table.

```
SQL> ALTER TABLE emp MODIFY(ename varchar2(20) NOT NULL,
                            job char(20) NOT NULL);
```

Q3) Add Primary key constraint to the column EMPNO of EMP table

```
SQL> ALTER TABLE emp ADD CONSTRAINT emp_pk
     PRIMARY KEY(empno);
```

Q4) Add Primary key constraint to the column DEPTNO of DEPT table

SQL>

Q5) Add Unique key constraint to the column DNAME of DEPT table

SQL>

Q6) Add Check constraint to the table EMP to restrict the values of EMPNO lies between 7000 and 8000.

```
SQL> ALTER TABLE emp ADD CONSTRAINT emp_ck
     CHECK(empno BETWEEN 7000 AND 8000)
```

Q7) Add Foreign key constraint to the column DEPTNO of EMP table references DEPTNO of DEPT table.

```
SQL> ALTER TABLE emp ADD CONSTRAINT emp_fk
     FOREIGN KEY(deptno) REFERENCES DEPT(deptno);
```

Q8) Add a Foreign key constraint to the EMP1 table indicating that a manager must already exist as a valid employee in the EMP1 table.

SQL>

DROPING CONSTRAINTS

- Removing constraints from the table

ALTER TABLE *table* DROP CONSTRAINT *const-name;*

Q9) Remove the Manager constraint (added in Q8) from EMP table

SQL>

Q10) Remove the primary key constraint on the DEPT table and drop the associated foreign key constraint on the EMP.DEPTNO column.

```
SQL> ALTER TABLE dept DROP PRIMARY KEY CASCADE;
```

DISABLE and ENABLE Constraint

- Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.
- Apply the CASCADE option to disable dependent integrity constraints.
- Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.
  A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE key or PRIMARY KEY constraint.

Q11) Disable the primary key constraint of EMP table.

```
SQL>  ALTER TABLE emp DISABLE CONSTRAINT emp_pk CASCADE;
```

Q12) Enable the primary key constraint of EMP table.

```
SQL>
```

Q13) Query the USER_CONSTRAINTS table to view all constraint definitions and names

```
SQL> SELECT constraint_name,  constraint_type, search_condition
     FROM  user_constraints
     WHERE  table_name = 'EMP';
```

Q14) View the columns associated with the constraint names in the USER_CONS_COLUMNS view

```
SQL> SELECT  constraint_name, column_name
     FROM  user_cons_columns
     WHERE  table_name = 'EMP';
```

Verified by

Staff In-charge Sign :	**Date :**

Ex.No. 4	BASIC SELECT STATEMENTS	Date :

Arithmetic Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division

Comparison Operators

=	Equal to
<>	Not Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
IN (List)	Match any of list of values
LIKE	Match a character pattern (% → any no. of characters, - → One Character)
IS NULL	Is a null value
BETWEEN…AND…	Between two values

Logical Operators

AND	Returns TRUE if *both* component conditions are TRUE
OR	Returns TRUE if *either* component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

Concatenation Operator (||)

- Concatenates the Columns of any data type.
- A Resultant column will be a Single column.

Operator Precedence

Order Evaluated	*Operators*
1	Parenthesis
2	All Arithmetic Operators (Multiplication and Division followed by Addition and subtraction)
3	All Comparison Operators
4	NOT
5	AND
6	OR

Where Clause

- Specify the Selection of rows retrieved by the WHERE Clause

    SELECT ***column1, column2, …***
    FROM ***table***
    WHERE ***condition*;**

- The WHERE clause follows the FROM clause

Order by Clause

- Sort rows specified by the order ASC / DESC

    SELECT ***column1, column2, … …***
    FROM ***table***
    ORDER BY ***sort-column* DESC;**

- Sorts *table* by *sort-column* in descending order
- Omitting the keyword DESC will sort the table in ascending order

Note :

- AS Keyword between the column name and the actual alias name
- Date and character literal values must be enclosed within single quotation marks
- Default date format is 'DD-MON-YY'
- Eliminate duplicate rows by using the DISTINCT keyword

Q1) Update all the records of *manager* table by increasing 10% of their salary as bonus.
SQL>

Q2) Delete the records from *manager* table where the salary less than 2750.
SQL>

Q3) Display each name of the employee as "Name" and annual salary as "Annual Salary" (Note: Salary in *emp* table is the monthly salary)
SQL>

Q4) List concatenated value of name and designation of each employee.
SQL>

Q5) List the names of Clerks from *emp* table.
SQL>

Q6) List the Details of Employees who have joined before 30 Sept 81.
SQL>

Q7) List names of employees who's employee numbers are 7369,7839,7934,7788.
SQL>

Q8) List the names of employee who are not Managers.
SQL>

Q9) List the names of employees not belonging to dept no 30,40 & 10
SQL>

Q10) List names of those employees joined between 30 June 81 and 31 Dec 81.
SQL>

Q11) List different designations in the company.
SQL>

Q12) List the names of employees not eligible for commission.
SQL>

Q13) List names and designations of employee who does not report to anybody
SQL>

Q14) List all employees not assigned to any department.
SQL>

Q15) List names of employee who are eligible for commission.
SQL>

Q16) List employees whose name either start or end with ‘s’.
SQL>

Q17) List names of employees whose names have ‘i’ as the second character.
SQL>

Q18) Sort *emp* table in ascending order by *hire-date* and list *ename, job, deptno* and *hire-date.*
SQL>

Q19) Sort *emp* table in descending order by annual salary and list *empno, ename, job* and *annual-salary.* (Note : Salary in *emp* table is the monthly salary)
SQL>

Q20) List *ename, deptno* and *sal* after sorting *emp* table in ascending order by *deptno* and then descending order by *sal.* (Note : Sorting by multiple coluns)
SQL>

Verified by

Staff In-charge Sign :	**Date :**

Ex. No. 5	S Q L F U N C T I O N S	Date :

SQL functions are of two types

(i) Single row functions or scalar functions

- Returns only one value for every row queried in the table
- Can be used in Select clause and where clause
- It can be broadly classified into 5 categories
    - Date Functions
    - Character Functions
    - Conversion functions
    - Numeric functions
    - Miscellaneous functions

(ii) Group functions or multiple-row functions

Discussed in the next exercise (ie.; Ex. No.6)

Note : The exercises that follow mostly uses system table 'dual'. It is a table which is automatically created by Oracle along with the data dictionary. Dual table has one column defined to be of varchar2 type and contains only one row with value 'x'.

SCALAR FUNCTIONS

Q1) List the hiredate of employees who work in deptno 20 in a format like 'WEDNESDAY JANUARY 12, 1983'

(Hint: DAY : Day of the week, MONTH : Name of the month, DD: Day of the month, and YYYY : Year)

SQL>

Q2) Display the hiredate with time of employess who work in deptno 20.

SQL>

Q3) Each employee receives a salary review after every 150 days of service. Now list employee name, hiredate and first salary review date of each employee who work in dept no 20.

SQL>

Q4. Date Functions

Functions	Value Returned	Input	Output
add_months(d,n)	‘n’ months added to date ‘d’.	Select add_months(sysdate,2) from dual;	
last_day(d)	Date corresponding to the last day of the month	Select last_day(sysdate) from dual;	
to_date(str,’format’)	Converts the string ina given format into Oracle date.	Select to_date(’10-02-09’,’dd-mm-yy’) from dual;	
to_char(date,’format’)	Reformats date according to format	Select to_char(sysdate,’dy dd mon yyyy’) from dual;	
months_between(d1,d2)	No. of months between two dates	Select months_between(sysdate, to_date(’10-10-07’,’dd-mm-yy’)) from dual;	
next_day(d,day)	Date of the ‘day’ that immediately follows the date ‘d’	Select next_day(sysdate,’wednesday’) from dual;	
round(d,’format’)	Date will be rounded to the nearest day.	Select round(sysdate,’year’) from dual;	
		Select round(sysdate,’month’) from dual;	
		Select round(sysdate,’day’) from dual;	
		Select round(sysdate) from dual;	
trunc(d,’format’);	Date will be truncated to the nearest day.	Select trunc(sysdate,’year’) from dual;	
		Select trunc(sysdate,’month’) from dual;	
		Select trunc(sysdate,’day’) from dual;	
		Select trunc(sysdate) from dual;	
greatest(d1,d2,…)	Picks latest of list of dates	Select greatest(sysdate, to_date(‘02-10-06’,’dd-mm-yy’), to-date(’12-07-12’,’dd-mm-yy’)) from dual;	
Date Arithmetic	Add /Subtract no. of days to a date	Select sysdate+25 from dual;	
		Select sysdate-25 from dual;	
	Subtract one date from another, producing a no. of days	Select sysdate - to_date(‘02-10-06’,’dd-mm-yy’) from dual;	

Q5. Character Functions

Functions	Value Returned	Input	Output
initcap(char)	First letter of each word capitalized	Select initcap(‘jesus christ’) from dual;	
lower(char)	Lower case	Select lower(‘DIED’) from dual;	
upper(char)	Upper case	Select upper(‘for Us’) from dual;	
ltrim(char, set)	Initial characters removed up to the character not in set.	Select ltrim(‘lordourgod’,’lord’) from dual;	
rtrim(char, set)	Final characters removed after the last character not in set.	Select rtrim(‘godlovesyou’,’you’) from dual;	
translate(char, from, to)	Translate ‘from’ by ‘to’ in char.	Select translate(‘jack’,’j’,’b’) from dual;	
replace(char, search, repl)	Replace ‘search’ string by ‘repl’ string in ‘char’.	Select replace(‘jack and jue’,’j’,’bl’) from dual;	
substr(char, m, n)	Substring of ‘char’ at ‘m’ of size ‘n’ char long.	Select substr(‘wages of sin is death’,10,3) from dual;	

Q6. Conversion Functions

Functions	Value Returned	Input	Output
to_date(str,’format’)	Converts the string ina given format into Oracle date.	Select to_date(’10-02-09’,’dd-mm-yy’) from dual;	
to_char(date,’format’)	Reformats date according to format	Select to_char(sysdate,’dy dd mon yyyy) from dual;	
to_char(number,’format’)	Display number value as a char.	Select to_char(12345.5,’L099,999.99’) from dual;	
to_number(char)	Char string to number form	Select to_number(‘123’) from dual;	

Q7. Numeric Functions

Functions	Value Returned	Input	Output
Abs(n)	Absolute value of n	Select abs(-15) from dual;	
Ceil(n)	Smallest int >= n	Select ceil(33.645) from dual;	
Cos(n)	Cosine of n	Select cos(180) from dual;	
Cosh(n)	Hyperbolic cosine of n	Select cosh(0) from dual;	
Exp(n)	e^n	Select exp(2) from dual;	
Floor(n)	Largest int <= n	Select floor(100.2) from dual;	
Ln(n)	Natural log of n (base e)	Select ln(5) from dual;	
Log(b,n)	Log n base b	Select log(2,64) from dual;	
Mod(m,n)	Remainder of m divided by n	Select mod(17,3) from dual;	
Power(m,n)	m power n	Select power(5,3) from dual;	
Round(m,n)	m rounded to n decimal places	Select round(125.67854,2) from dual;	
Sign(n)	If n<0, -1 if n=0, 0 otherwise 1.	Select sin(-19) from dual;	
Sin(n)	Sin of n	Select sin(90) from dual;	
Sinh(n)	Hyperbolic sin of n	Select sinh(45) from dual;	
Sqrt(n)	Square root of n	Select sqrt(7) from dual;	
Tan(n)	Tangent of n	Select tan(45) from dual;	
Tanh(n)	Hyperbolic tangent of n	Select tanh(60) from dual;	
Trunc(m,n)	m truncated to n decimal places	Select trunc(125.5764,2) from dual;	

Q8. Miscellaneous Functions

Functions	Value Returned	Input	Output
Uid	User id	Select uid from dual;	
User	User name	Select user from dual;	
Vsize(n)	Storage size of v	Select vsize(‘hello’) from dual;	
NVL(exp1,exp2)	Returns exp1 if not null, otherwise returns exp2.	Select nvl(comm,50) from emp where empno=7369;	

<u>GROUP FUNCTIONS</u>

Common Group Functions

• AVG	:	Average value of a set
• COUNT	:	Numbers of non null values
• MAX	:	Maximum of a set
• MIN	:	Minimum of a set
• STDDEV	:	Standard Deviation of a set
• SUM	:	Sum of a set
• VARIANCE	:	Variance of a set

Syntax :

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_column_or_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

- ➢ Group functions ignore null values
- ➢ *Group by* Clause is used to modularize rows in a table into smaller groups
- ➢ Columns that are not a part of the Group Functions should be included in the Group by clause
- ➢ Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause
- ➢ Group Functions cannot be placed in the where clause
- ➢ HAVING clause is to restrict groups Groups satisfying the HAVING condition are displayed

- ➢ Order of evaluation of the clauses :
    - o WHERE clause
    - o GROUP BY clause
    - o HAVING clause

Q9) Find number of rows in the table EMP

SQL >

Q10) Find number of designations available in EMP table.
SQL>

Q11) Find number of employees who earn commission in EMP table.
SQL>

Q12) What is the difference between the following queries
SQL > select *count(comm)* from *emp*;
SQL > select *count(nvl(comm,0))* from *emp*;

Q13) Find the total salary paid to the employees.
SQL>

Q14) Find maximum, minimum and average salary in EMP table.
SQL>

Q15) Find number of employees who work in department number 30
```
SQL> select count(*) from emp
       where deptno=30;
```

Q16) Find the maximum salary paid to a 'CLERK'
SQL>

Q17) List the department numbers and number of employees in each department
SQL>

Q18) List the jobs and number of employees in each job. The result should be in the descending order of the number of employees.
```
SQL> select job, count(*) from emp
       group by job
       order by count(*) desc;
```

Q19) List the total salary, maximum and minimum salary and average salary of the employees jobwise.
SQL>

Q20) List the total salary, maximum and minimum salary and average salary of the employees jobwise, for department 20 and display only those rows having an average salary > 1000.
SQL> **select** ***job,sum(sal), max(sal), min(sal), avg(sal)***
from ***emp***
group by ***job, deptno***
having ***deptno=20 and avg(sal) > 1000;***

Q21) List the job and total salary of employees jobwise, for jobs other than 'PRESIDENT' and display only those rows having total salary > 5000.
SQL>

Q22) List the job**,** number of employees and average salary of employees jobwise. Display only the rows where the number of employees in each job is more than two.
SQL>

Verified by

Staff In-charge Sign :	**Date :**

Ex. No. 6	**JOINING TABLES**	**Date :**

Set operators

- Set operators combine the results of two queries into a single.

Operator	**Function**
Union	Returns all distinct rows selected by either query
Union all	Returns all rows selected by either query including duplicates
Intersect	Returns only rows that are common to both the queries
Minus	Returns all distinct rows selected only by the first query and not by the second.

Q1) Create the following tables :

depositor(cus_name,acno) & borrower(cus_name,loanno)

SQL>

SQL>

Q2) List the names of distinct customers who have either loan or account

SQL>

Q3) List the names of customers (with duplicates) who have either loan or account

SQL> (select *cus_name* from *borrower*)

union all (select *cus_name* from *depositor*)

Q4) List the names of customers who have both loan and account

SQL>

Q5) List the names of customers who have loan but not account

SQL>

Joins

- Used to combine the data spread across tables

Syntax

SELECT	*table1.column, table2.column*
FROM	*table1, table2*
WHERE	*table1.column1 = table2.column2*;

- A JOIN Basically involves more than one Table to interact with.
- Where clause specifies the JOIN Condition.
- Ambiguous Column names are identified by the Table name.
- If join condition is omitted, then a **Cartesian product** is formed. That is all rows in the first table are joined to all rows in the second table

Types of Joins

▪ Inner Join (Simple Join)	: It retrieves rows from 2 tables having a common column.
o Equi Join	: A join condition with relationship = .
o Non Equi Join	: A join condition with relationship other than = .
▪ Self Join	: Joining of a table to itself
▪ Outer Join	: Returns all the rows returned by simple join as well as those rows from one table that do not match any row from the other table. The symbol (+) represents outer joins.

Q6) List *empno, ename, deptno* from *emp* and *dept* tables.

SQL>

Q7) Create a table *Salgrade* with the following data .

Grade	**Losal**	**Hisal**
1	700	1400
2	1401	2000
3	2001	5000
4	5001	9999

Now, list *ename, sal* and *salgrade* of all employees.

SQL>

Q8) List *ename, deptno* and *deptname* from *emp* and *dept* tables, including the rows of *emp* table that does not match with any of the rows in *dept* table.

SQL>

Q9) List *ename, deptno* and *deptname* from *emp* and *dept* tables, including the rows of *dept* table that does not match with any of the rows in *emp* table.

SQL>

Q10) List the names of the employee with name of his/her manager from *emp* table.

SQL>

Verified by

Staff In-charge Sign :	**Date :**

Ex. No. 7	S U B Q U E R I E S	Date :

- Nesting of queries, one within the other is termed as sub query.

Syntax

```
SELECT      select_list
FROM        table
WHERE       expr operator  ( SELECT      select_list
                             FROM        table);
```

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).

Guidelines for Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison operator.
- Do not add an ORDER BY clause to a subquery.
- Use single-row operators with single-row subqueries.
- Use multiple-row operators with multiple-row subqueries.

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators (ie; relational operators)

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

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

Note:

‘**=any**’ is equivalent to ‘**in**’

‘**!=all**’ is equivalent to ‘**not in**’

Q1) List the name of the employees whose salary is greater than that of employee with empno 7566.

```
SQL> select ename from employee
         where  sal > (select sal from employee
                            where  empno=7566);
```

Q2) List the name of the employees whose job is equal to the job of employee with empno 7369 and salary is greater than that of employee with empno 7876.

SQL>

Q3) List the *ename,job,sal* of the employee who get minimum salary in the company.

```
SQL> select ename, job, sal from employee
         where  sal = (select    min(sal) from employee);
```

Q4) List *deptno & min(salary)* departmentwise, only if *min(sal)* is greater than the *min(sal)* of *deptno* 20.

```
SQL> select  deptno, min(sal) from employee
         group by deptno
         having min(sal) > (select min(sal) from employee
                                 where deptno = 20);
```

Q5) List *empno, ename, job* of the employees whose *job* is not a 'CLERK' and whose *salary* is less than at least one of the salaries of the employees whose *job* is 'CLERK'.

```
SQL> select  empno, ename, job  from employee
         where sal < any (select sa  from employee
                      where job = 'CLERK')
                  and job <> ' CLERK ';
```

Q6) List *empno, ename, job* of the employees whose salary is greater than the average salary of each department.

SQL>

Q7) Display the *name, dept. no, salary,* and *commission* of any employee whose *salary* and *commission* matches both the *commission* and *salary* of any employee in department 30.

```
SQL> select   ename, deptno, sal, comm
     from     employee
     where    (sal, nvl(comm,-1)) in  ( select sal, nvl(comm,-1)
                                        from   employee
                                        where  deptno = 30);
```

Q8) List *ename sal, deptno, average salary* of the dept where he/she works, if salary of the employee is greater than his/her department average salary.

```
SQL> select  a.ename, a.sal, a.deptno, b.salavg
     from    employee a,   ( select  deptno, avg(sal) salavg
                             from    employee
                             group by deptno) b
     where   a.deptno = b.deptno
     and     a.sal > b.salavg;
```

Q9) Execute and Write the output of the following query in words.

```
SQL> with summary as
     (select  dname,sum(sal) as dept_total   from employee a , department b
      where   a.deptno = b.deptno
      group by dname);
      select dname,dept_total  from summary
      where dept_total > (select sum(dept_total)*1/3  from summary)
      order by dept_total desc;
```

Q10) List *ename, job, sal* of the employees whose salary is equal to any one of the salary of the employee 'SCOTT' and 'WARD'.

SQL>

Q11) List *ename, job, sal* of the employees whose salary and job is equal to the employee 'FORD'.

SQL>

Q12) List *ename, job, deptno, sal* of the employees whose job is same as 'JONES' and *salary* is greater than the employee 'FORD'.

SQL>

Q13) List *ename, job* of the employees who work in *deptno* 10 and his/her *job* is any one of the job in the department 'SALES'.

SQL>

Q14) Execute the following query and write the result in word

```
SQL> select job,ename,empno,deptno from emp s
     where exists (select * from emp
                   where s.empno=mgr)
     order by empno;
```

Verified by

Staff In-charge Sign :	**Date :**

Ex. No. 8	**VIEWS**	**Date :**

VIEWS

- An Imaginary table contains no data and the tables upon which a view is based are called base tables.
- Logically represents subsets of data from one or more tables

Advantages of view

- To restrict database access
- To make complex queries easy
- To allow data independence
- To present different views of the same data

Syntax

CREATE [OR REPLACE] VIEW *view* [*col1 alias*, *col2 alias,...*]
AS *subquery*
[WITH CHECK OPTION [CONSTRAINT *constraint*]]
[WITH READ ONLY]

- You embed a subquery within the CREATE VIEW statement.
- The subquery can contain complex SELECT syntax.
- The subquery cannot contain an ORDER BY clause.

Q1) Create a view *empv10* that contains *empno, ename, job* of the employees who work in *dept* 10. Also describe the structure of the view.

```
SQL> create view empv10 as
         select empno, ename, job  from emp
         where deptno=10;

SQL> desc empv10;
```

Q2) Create a view with column aliases *empv30* that contains *empno, ename, sal* of the employees who work in *dept* 30. Also display the contents of the view.

```
SQL >




SQL> select * from empv30;
```

Rules for Performing DML Operations on a View

- You can perform DML operations on simple views.
- You **cannot remove** a row/ **modify** data/ **add** data if the view contains the following
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
- You cannot modify data in a view if it contains columns defined by expressions or it contains ROWNUM pseudo column
- You cannot add data if any NOT NULL columns in the base tables that are not selected by the view

Q3) Update the view *empv10* by increasing 10% salary of the employees who work as 'CLERK'. Also confirm the modifications in *emp table.*

SQL > update *empv10* set *sal = sal+0.10*sal*
where *job='CLERK'*;

SQL > select *empno,ename,job,sal* from emp;

Q4) Modify the view *empv10* which contains the data *empno, ename, job, sal*. Add an alias for each column name.

SQL > create or replace view *empv10*
***(employee_no, employee_name, job, salary)* as**
select *empno, ename, job,sal* from *emp* where *deptno=10;*

Q5) Using *emp* table, **c**reate a view *pay* which contains *ename, monthly_sal, annual_sal, deptno.*

SQL>

Q6) Create a view *dept_stat* which contains *department no., department name, minimum salary, maximum salary, total salary.*

SQL>

With check option

- You can ensure that DML on the view stays within the domain of the view by using the WITH CHECK OPTION.

Q7) Execute the following query and then try to delete the row with dept no 20. Now write in words that you understand

SQL> create or replace view *empv20*
as select * from *emp* where *deptno = 20*
with check option constraint *empv20_ck;*

Denying DML Operations

- You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition.

Q8) Create a view *empv10* with all the details of employees who work in dept no. 10. Also ensure that no DML operations can be done with the view.

SQL> create or replace view *empv10*
as select * from *emp* where *deptno = 20*
with read only;

Deleting Views

Syntax

DROP VIEW view_*name*;

Q9) Delete the view *empv20*.

SQL>

Ex. No. 9	ADVANCED SELECT STATEMENTS	Date :

Questions Q1 to Q18 pertain to a database with the following tables.

Suppliers	**- S (Supplyno, Name, Status, City)**
Parts	**- P (Partno, Pname, Colour, Weight, City)**
Projects	**- J (Jobno, Jname, City)**
Shipment	**- SPJ (Supplyno, Partno, Jobno, Qty)**

The significance of an SPJ record is that the specified supplier supplies the specified part to the specified project in the specified quantity (and the combination Supplyno-Partno-Jobno uniquely identifies such a record).

Q1) Get full details of all projects in London.
SQL>

Q2) Get Supplyno for suppliers who supply project J1.
SQL>

Q3) Get all part-color/part-city combinations.
SQL>

Q4) Get all Supplyno/Partno/Jobno triples such that all are co-located.
SQL>

Q5) Get al Supplyno, Partno, Jobno triples such that they are not all co-located.
SQL>

Q6) Get Partno for parts supplied by a supplier in London.
SQL>

Q7) Get all pairs of cities such that a supplier in the first city supplies to a Project in the second city.
SQL>

Q8) Get Jobno for projects supplied by at least one supplier not in the same city.
SQL>

Q9) Get all pairs of part numbers such that some supplier supplies both the indicated parts.
SQL>

Q10) Get the total quantity of part P1 supplied by S1.
SQL>

Q11) For each part supplied to a project, get the Partno, Jobno and corresponding total quantity.
SQL>

Q12) Get Partno of parts supplied to some project in an average quantity > 320.
SQL>

Q13) Get project names for projects supplied by supplier S1.
SQL>

Q14) Get colors of parts supplied by S1.
SQL>

Q15) Get Jobno for projects using at least one part available from supplier S1.
SQL>

Q16) Get supplier numbers for suppliers supplying at least one part supplied by at least one supplier who supplies at least one red part.
SQL>

Q17) Get supplier numbers for suppliers with a status lower than that of supplier S1.
SQL>

Q18) Get project numbers for projects not supplied with any red part by any London supplier.
SQL>

Verified by

Staff In-charge Sign :	**Date :**

Ex. No. 10	ADDITIONAL QUERIES	Date :

Solve the queries Q1 to Q32 by considering the following tables

Table 1 : **STUDIES (PNAME, SPLACE, COURSE, CCOST)**
Hint : First 3 columns are of type varchar and the 4th one is number

Table 2 : **SOFTWARE(PNAME, TITLE, DEVIN, SCOST, DCOST, SOLD)**
Hint : First 3 columns are of type varchar and the last 3 are of numbers

Table 3 : **PROGRAMMER(PNAME, DOB, DOJ, SEX, PROF1, PROF2, SAL)**
Hint : Pname : varchar, dob & doj : date, sex : char, prof1 & prof2 : varchar, sal : number.

LEGEND :
PNAME – Programmer Name, SPLACE – Study Place, CCOST – Course Cost, DEVIN – Developed in, SCOST – Software Cost, DCOST – Development Cost, PROF1 – Proficiency 1

Q1) Find out the selling cost average for packages developed in Oracle.
SQL>

Q2) Display the names, ages and experience of all programmers.
SQL>

Q3) Display the names of those who have done the PGDCA course.
SQL>

Q4) What is the highest number of copies sold by a package?
SQL>

Q5) Display the names and date of birth of all programmers born in April.
SQL>

Q6) Display the lowest course fee.
SQL>

Q7) How many programmers have done the DCA course.
SQL>

Q8) How much revenue has been earned through the sale of packages developed in C.
SQL>

Q9) Display the details of software developed by Rakesh.
SQL>

Q10) How many programmers studied at Pentafour.
SQL>

Q11) Display the details of packages whose sales crossed the 5000 mark.
SQL>

Q12) Find out the number of copies which should be sold in order to recover the development cost of each package.
SQL>

Q13) Display the details of packages for which the development cost has been recovered.
SQL>

Q14) What is the price of costliest software developed in VB?
SQL>

Q15) How many packages were developed in Oracle ?
SQL>

Q16) How many programmers studied at PRAGATHI?
SQL>

Q17) How many programmers paid 10000 to 15000 for the course?
SQL>

Q18) What is the average course fee?
SQL>

Q19) Display the details of programmers knowing C.
SQL>

Q20) How many programmers know either C or Pascal?
SQL>

Q21) How many programmers don't know C and C++?
SQL>

Q22) How old is the oldest male programmer?
SQL>

Q23) What is the average age of female programmers?
SQL>

Q24) Calculate the experience in years for each programmer and display along with their names in descending order.
SQL>

Q25) Who are the programmers who celebrate their birthdays during the current month?
SQL>

Q26) How many female programmers are there?
SQL>

Q27) What are the languages known by the male programmers?
SQL>

Q28) What is the average salary?
SQL>

Q29) How many people draw 5000 to 7500?
SQL>

Q30) Display the details of those who don't know C, C++ or Pascal.
SQL>

Q31) Display the costliest package developed by each programmer.
SQL>

Q32) Produce the following output for all the male programmers
Programmer
Mr. Arvind – has 15 years of experience
SQL>

Solve Queries from Q33 to 47 using the table *dept* and *emp*.

Q33) List all the employees who have at least one person reporting to them.
SQL>

Q34) List the employee details if and only if more than 10 employees are present in department no 10.
SQL>

Q35) List the name of the employees with their immediate higher authority.
SQL>

Q36) List all the employees who do not manage any one.
SQL>

Q37) List the employee details whose salary is greater than the lowest salary of an employee belonging to deptno 20.
SQL>

Q38) List the details of the employee earning more than the highest paid manager.
SQL>

Q39) List the highest salary paid for each job.
SQL>

Q40) Find the most recently hired employee in each department.
SQL>

Q41) In which year did most people join the company? Display the year and the number of employees.
SQL>

Q42) Which department has the highest annual remuneration bill?
SQL>

Q43) Write a query to display a '*' against the row of the most recently hired employee.
SQL>

Q44) Write a correlated sub-query to list out the employees who earn more than the average salary of their department.
SQL>

Q45) Find the nth maximum salary.
SQL>

Q46) Select the duplicate records (Records, which are inserted, that already exist) in the EMP table.
SQL>

Q47) Write a query to list the length of service of the employees (of the form n years and m months).
SQL>

Q48) Create the following tables with constraints specified below.

(i) customer (cust_id, cust_name, annual_revenue, cust_type)
Constraints are :
cust_id must be between 100 and 10,000
annual_revenue defaults to Rs. 20,000
cust_type must be manufacturer, wholesaler, or retailer

SQL>

(ii) shipment (shipment_no, cust_id, weight, truck_no, destination, ship_date)
Constraints are :
foreign key: cust_id references customer, on deletion cascade
foreign key: truck_# references truck, on deletion set to null
foreign key: destination references city, on deletion set to null
weight must be under 1000 and defaults to 10

SQL>

(iii) truck (truck_no, driver_name)
SQL>

(iv) city (city_name, population)
SQL>

Answer Questions Q49 to Q67 using the tables created in Q48.

Q49) What are the names of customers who have sent packages (shipments) to Sioux City?
SQL>

Q50) To what destinations have companies with revenue less than $1 million sent packages?
SQL>

Q51) What are the names and populations of cities that have received shipments weighing over 100 pounds?
SQL>

Q52) Who are the customers having over $5 million in annual revenue who have sent shipments weighing less than 1 pound?
SQL>

Q53) Who are the customers having over $5 million in annual revenue who have sent shipments weighing less than 1 pound or have sent a shipment to San Francisco?
SQL>

Q54) Who are the drivers who have delivered shipments for customers with annual revenue over $20 million to cities with populations over 1 million?
SQL>

Q55) List the cities that have received shipments from customers having over $15 million in annual revenue.
SQL>

Q56) List the names of drivers who have delivered shipments weighing over 100 pounds.
SQL>

Q57) List the name and annual revenue of customers who have sent shipments weighing over 100 pounds.
SQL>

Q58) List the name and annual revenue of customers whose shipments have been delivered by truck driver Jensen.
SQL>

Q59) List customers who had shipments delivered by every truck. (use NOT EXISTS)
SQL>

Q60) List cities that have received shipments from every customer. (use NOT EXISTS)
SQL>

Q61) List drivers who have delivered shipments to every city. (use NOT EXISTS)
SQL>

Q62) Customers who are manufacturers or have sent a package to St. Louis.
SQL>

Q63) Cities of population over 1 million which have received a 100-pound package from customer 311.
SQL>

Q64) Trucks driven by Jake Stinson which have never delivered a shipment to Denver.
SQL>

Q65) Customers with annual revenue over $10 million which have sent packages under 1 pound to cities with population less than 10,000.
SQL>

Q66) Create views for each of the following:

a. Customers with annual revenue under $1 million.
b. Customers with annual revenue between $1 million and $5 million.
c. Customers with annual revenue over $5 million.

SQL>

SQL>

SQL>

Q67) Use these views to answer the following queries:

a. Which drivers have taken shipments to Los Angeles for customers with revenue over $5 million?
b. What are the populations of cities which have received shipments from customers with revenue between $1 million and $5 million?
c. Which drivers have taken shipments to cities for customers with revenue under $1 million, and what are the populations of those cities?

SQL>

SQL>

SQL>

Verified by

Staff In-charge Sign :	**Date :**

P L / S Q L   I N T R O D U C T I O N

PL/SQL

- PL/SQL bridges the gap between database technology and procedural programming languages.
- PL/SQL uses the facilities of the sophisticated RDBMS and extends the standard SQL database language
- Not only PL/SQL allow you to insert, delete, update and retrieve data, it lets you use procedural techniques such as looping and branching to process the data.
- Thus PL/SQL provides the data manipulating power of SQL with the data processing power of procedural languages

Advantage of PL/SQL

PL/SQL is a completely portable, high performance transaction processing language. It provides the following advantages :

- Procedural Capabilities
  - It supports many of constructs like constants, variable, selection and iterative statements
- Improved Performance
  - Block of SQL statements can be executed at a time
- Enhanced Productivity
  - PL/SQL brings added functionality to non procedural tools such as SQL Forms.
- Portability
  - PL/SQL can run anywhere the RDBMS can run
- Integration with RDBMS
  - Most PL/SQL variables have data types native to the RDBMS data dictionary. Combined with the direct access to SQL, these native data type declarations allow easy integration of PL/SQL with RDBMS.

Character Set

It is either ASCII or EBCDIC format

Identifiers

It begins with a letter and can be followed by letters, numbers, $ or #. Maximum size is 30 characters in length.

Variable Declaration

The data types (number, varchar2, real, date, …) discussed in SQL are all applicable in PL/SQL.

Ex.	Salary	*Number(7,2);*
	Sex	*Boolean;*
	Count	*smallint :=0;*
	Tax	*number default 750;*
	Name	*varchar2(20) not null;*

Constant declaration

Ex. Phi *Constant Number(7,2)* := 3.1417;

Comment

Line can be commented with double hyphen at the beginning of the line.

Ex. - - This is a comment line

Assignments

Variable assignment sets the current value of a variable. You can assign values to a variable as follows

(i) Assignment operator (:=)

Ex. d := b*b – 4*a*c;

(ii) Select … into statement

Ex. *Select* sal *into* salary *from* emp *where* empno=7655;

Operators

Operators used in SQL are all applicable to PL/SQL also.

Block Structure

PL/SQL code is grouped into structures called blocks. If you create a stored procedure or package, you give the block of PL/SQL code a name. If the block of PL/SQL code is not given a name, then it is called an anonymous block.

The PL/SQL block divided into three section: declaration section, the executable section and the exception section

The structure of a typical PL/SQL block is shown in the listing:

```
declare
        < declaration section >
begin
        < executable  commands>
exception
        <exception handling>
end;
```

Declaration Section :

Defines and initializes the variables and cursor used in the block

Executable commands :

Uses flow-control commands (such as IF command and loops) to execute the commands and assign values to the declared variables

Exception handling :

Provides handling of error conditions

Declaration Using attributes

(i) %type attribute

The %TYPE attribute provides the data type of a variable, constant, or database column. Variables and constants declared using %TYPE are treated like those declared using a data type name.
For example in the declaration below, PL/SQL treats debit like a REAL(7,2) variable.

```
credit  REAL(7,2);
debit   credit%TYPE;
```

The %TYPE attribute is particularly useful when declaring variables that refer to database columns. You can reference a table and column, or you can reference an owner, table, and column.

```
my_dname     dept.dname%TYPE;
```

Using %TYPE to declare my_dname has two advantages.

- First, you need not know the exact datatype of dname.
- Second, if the database definition of dname changes, the datatype of my_dname changes accordingly at run time.

(ii) %rowtype attribute

The %ROWTYPE attribute provides a record type that represents a row in a table (or view). The record can store an entire row of data selected from the table or fetched by a cursor.

```
DECLARE
        emp_rec   emp%ROWTYPE;
        ...
BEGIN
        SELECT  *  INTO  emp_rec  FROM  emp  WHERE ...
        ...
END;
```

Columns in a row and corresponding fields in a record have the same names and data types.

The column values returned by the SELECT statement are stored in fields. To reference a field, you use the dot notation.

```
IF   emp_rec.deptno = 20  THEN ...
```

In addition, you can assign the value of an expression to a specific field.

```
emp_rec.ename := 'JOHNSON';
```

A %ROWTYPE declaration cannot include an initialization clause. However, there are two ways to assign values to all fields in a record at once.

First, PL/SQL allows aggregate assignment between entire records if their declarations refer to the same table or cursor.

```
DECLARE
        dept_rec1    dept%ROWTYPE;
        dept_rec2    dept%ROWTYPE;
        …..
BEGIN
        …..
        dept_rec1  :=  dept_rec2;
        …..
END;
```

..

Second, you can assign a list of column values to a record by using the SELECT and FETCH statement, as the example below shows. The column names must appear in the order in which they were defined by the CREATE TABLE or CREATE VIEW statement.

```
DECLARE
        dept_rec    dept%ROWTYPE;
        ….
BEGIN
        SELECT  deptno, dname, loc INTO  dept_rec  FROM dept
        WHERE deptno = 30;
        ….
END;
```

However, you cannot assign a list of column values to a record by using an assignment statement. Although you can retrieve entire records, you cannot insert them.

For example, the following statement is illegal:

```
INSERT INTO  dept  VALUES (dept_rec); -- illegal
```

Creating and Executing PL/SQL Programs

Edit your PL/SQL program in your favourite editor as text file.

Execute the following command once for a session to get displayed the output.

```
SQL> set serveroutput on;
```

Now execute the program using the following command.

SQL> *start* filename; (or) SQL> @filename;

Note : Give absolute path of the filename if you saved the file in some directory.

Ex. SQL> *start* z:\plsql\ex11; (or) SQL> @ z:\plsql\ex11;

Control Structures

(i) IF Statements

There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF. The third form of IF statement uses the keyword ELSIF (NOT ELSEIF) to introduce additional conditions, as follows:

```
IF condition1 THEN
        sequence_of_statements1;
ELSIF condition2 THEN
        sequence_of_statements2;
ELSE
        sequence_of_statements3;
END IF;
```

(ii) LOOP and EXIT Statements

There are three forms of LOOP statements. They are LOOP, WHILE-LOOP, and FOR-LOOP.

LOOP

The simplest form of LOOP statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

```
LOOP
        sequence_of_statements3;
        ...
END LOOP;
```

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop. If further processing is undesirable or impossible, you can use the EXIT statement to complete the loop. You can place one or more EXIT statements anywhere inside a loop, but nowhere outside a loop. There are two forms of EXIT statements: EXIT and EXIT-WHEN.

The EXIT statement forces a loop to complete unconditionally. When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

```
LOOP
    ...
    IF ... THEN
        ...
        EXIT; -- exit loop immediately
    END IF;
END LOOP;
-- control resumes here
```

The EXIT-WHEN statement allows a loop to complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition evaluates to TRUE, the loop completes and control passes to the next statement after the loop.

```
LOOP
    ….
    EXIT WHEN i>n; -- exit loop if condition is true
    ….
END LOOP;
….
```

Until the condition evaluates to TRUE, the loop cannot complete. So, statements within the loop must change the value of the condition.

Like PL/SQL blocks, loops can be labeled. The label, an undeclared identifier enclosed by double angle brackets, must appear at the beginning of the LOOP statement, as follows:

```
<<label_name>>
LOOP
    sequence_of_statements;
    ...
END LOOP [label_name];
```

Optionally, the label name can also appear at the end of the LOOP statement.

With either form of EXIT statement, you can complete not only the current loop, but any enclosing loop. Simply label the enclosing loop that you want to complete, then use the label in an EXIT statement.

```
<<outer>>
LOOP
    ...
    LOOP
        ...
        EXIT outer WHEN ... -- exit both loops
    END LOOP;
...
END LOOP outer;
```

(iii) WHILE-LOOP

The WHILE-LOOP statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP, as follows:

```
WHILE condition LOOP
    sequence_of_statements;
    ...
END LOOP;
```

Before each iteration of the loop, the condition is evaluated. If the condition evaluates to TRUE, the sequence of statements is executed, then control resumes at the top of the loop. If the condition evaluates to FALSE or NULL, the loop is bypassed and control passes to the next statement. Since the condition is tested at the top of the loop, the sequence might execute zero times.

(iv) FOR-LOOP

Whereas the number of iteration through a WHILE loop is unknown until the loop completes, the number of iterations through a FOR loop is known before the loop is entered. FOR loops iterate over a specified range of integers. The range is part of an iteration scheme, which is enclosed by the keywords FOR and LOOP.

```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
    sequence_of_statements;
    ...
END LOOP;
```

The lower bound need not be 1. However, the loop counter increment (or decrement) must be 1. PL/SQL lets you determine the loop range dynamically at run time, as the following example shows:

```
SELECT COUNT(empno) INTO emp_count FROM emp;
FOR  i  IN  1..emp_count   LOOP
    ...
END LOOP;
```

The loop counter is defined only within the loop. You cannot reference it outside the loop. You need not explicitly declare the loop counter because it is implicitly declared as a local variable of type INTEGER.

The EXIT statement allows a FOR loop to complete prematurely. You can complete not only the current loop, but any enclosing loop.

(v) GOTO and NULL statements

Unlike the IF and LOOP statements, the GOTO and NULL statements are not crucial to PL/SQL programming. The structure of PL/SQL is such that the GOTO statement is seldom needed. Occasionally, it can simplify logic enough to warrant its use. The NULL statement can make the meaning and action of conditional statements clear and so improve readability.

```
BEGIN
        ...
        GOTO insert_row;
        ...
        <<insert_row>>
        INSERT INTO emp VALUES ...
END;
```

A GOTO statement cannot branch into an IF statement, LOOP statement, or sub-block. A GOTO statement cannot branch from one IF statement clause to another. A GOTO statement cannot branch out of a subprogram. Finally, a GOTO statement cannot branch from an exception handler into the current block.

The NULL statement explicitly specifies inaction; it does nothing other than pass control to the next statement. It can, however, improve readability. Also, the NULL statement is a handy way to create stubs when designing applications from the top down.

* * *

Ex. No. 11	B A S I C P L / S Q L	Date :

Q1) Write a PL/SQL Block to find the maximum of 3 Numbers

```
Declare
        a  number;
        b  number;
        c  number;
Begin
        dbms_output.put_line('Enter a:');
        a:=&a;
        dbms_output.put_line('Enter b:');
        b:=&b;
        dbms_output.put_line('Enter c:');
        c:=&c;
        if (a>b) and (a>c) then
                dbms_output.putline('A is Maximum');
        elsif (b>a) and (b>c) then
                dbms_output.putline('B is Maximum');
        else
                dbms_output.putline('C is Maximum');
        end if;
End;
/
```

Q2) Write a PL/SQL Block to find the sum of odd numbers upto 100 using loop statement

```
Declare


Begin










End;
/
```

Q3) Write a PL/SQL block to get the salary of the employee who has empno=7369 and update his salary as specified below

- if his/her salary < 2500, then increase salary by 25%
- otherwise if salary lies between 2500 and 5000, then increase salary by 20%
- otherwise increase salary by adding commission amount to the salary.

```
Declare
        Salary  number(5);
Begin
        Select  sal into salary  from emp where empno=7369;
        -- complete remaining statements









End;
/
```

Q4) Write a PL/SQL Block to modify the department name of the department 71 if it is not 'HRD'.

```
Declare
        deptname   dept.dname%type;
Begin                                   -- complete the block









End;
/
```

C U R S O R

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time.

There are two types of cursors in PL/SQL. They are Implicit cursors and Explicit cursors.

Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed.

Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

For example, When you execute INSERT, UPDATE, or DELETE statements the cursor attributes tell us whether any rows are affected and how many have been affected.

When a SELECT... INTO statement is executed in a PL/SQL Block, implicit cursor attributes can be used to find out whether any row has been returned by the SELECT statement. PL/SQL returns an error when no data is selected.

Implicit Cursor Attributes

%FOUND

The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row or if SELECT ….INTO statement return at least one row. Ex. SQL%FOUND

%NOTFOUND

The return value is FALSE, if DML statements affect at least one row or if SELECT. …INTO statement return at least one row.
Ex. SQL%NOTFOUND

%ROWCOUNT

Return the number of rows affected by the DML operations
Ex. SQL%ROWCOUNT

Q5) Write a PL/SQL Block, to update salaries of all the employees who work in deptno 20 by 15%. If none of the employee's salary are updated display a message *'None of the salaries were updated'*. Otherwise display the total number of employee who got salary updated.

```
Declare
  num number(5);
Begin
  update emp set sal = sal + sal*0.15 where deptno=20;
  if SQL%NOTFOUND then
      dbms_output.put_line('none of the salaries were updated');
  elsif SQL%FOUND then
      num := SQL%ROWCOUNT;
      dbms_output.put_line('salaries for ' || num || 'employees are updated');
  end if;
End;
```

Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

There are four steps in using an Explicit Cursor.

- DECLARE the cursor in the declaration section.
- OPEN the cursor in the Execution Section.
- FETCH the data from cursor into PL/SQL variables or records in the Execution Section.
- CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

Declaring Cursor :

```
CURSOR cursor_name IS select_statement;
```

Opening Cursor :

```
OPEN  cursor_name;
```

Fetching Cursor :

```
FETCH cursor_name INTO variable-list/record-type;
```

Closing Cursor :

```
CLOSE cursor_name;
```

Explicit Cursor Attributes

%FOUND

TRUE, if fetch statement returns at least one row.

Ex. Cursor_name%FOUND

%NOTFOUND

TRUE, , if fetch statement doesn’t return a row.

Ex. Cursor_name%NOTFOUND

%ROWCOUNT

The number of rows fetched by the fetch statement.
Ex. Cursor_name%ROWCOUNT

%ISOPEN
TRUE, if the cursor is already open in the program.
Ex. Cursor_name%ISOPEN

Q6) Create a table *emp_grade* with columns *empno & grade*. Write PL/SQL block to insert values into the table *emp_grade* by processing *emp* table with the following constraints.
If sal <= 1400 then grade is 'C'
Else if sal between 1401 and 2000 then the grade is 'B' Else the grade is 'A'.

SQL> create table emp_grade(empno number, grade char(1));

```
Declare        Emp_rec   emp%rowtype;
               Cursor c is select * into emp_rec from emp;
Begin
          Open c;
          If  c%ISOPEN  then
               Loop
                    Fetch  c  into emp_rec;
                    If c%notfound then  Exit;   Endif;
                    If emp_rec.sal <= 1400 then
                       Insert into emp_grade values(emp_rec.empno,'C');
                    Elsif emp_rec.sal between 1401 and 2000 then
                       Insert into emp_garde values(em_rec.empno,'B');
                    Else
                       Insert into emp_garde values(em_rec.empno,'A');
                    Endif
                End loop;
          Else
              Open c;
          Endif;
End;
```

Q7) Write a PL/SQL block to do the following :
a. Total wages of the company (Sum of the salaries and commission values of all the employees in *emp* table)
b. Total number of highly paid employees. (Employees with salary > 2000)
c. Total number of employees who get commission that is higher than their salary.

Q8) Write a PL/SQL block to find the name and salary of first five highly paid employees.

Cursor for loop
Cursor for loop automatically opens a cursor, fetches each row and closes the cursor when all rows have been processed.
Ex.

```
Declare
        Cursor  s1 is select .. .. ..
Begin
        For  var  in  s1
        Loop
                -- statements ---
        End loop;
        .. .. .. .. ..
```

Q9) Solve the program in question number Q4 using cursor for…loop

Q10) Write a PL/SQL block to find the names of employees & job and total number of employees who have more than 28 years of service in the company.(Use for loop)

II. T R I G G E R

A trigger is a PL/SQL block structure which is fired when DML statements like Insert, Delete and Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Syntax of Trigger

```
CREATE   [OR REPLACE ]  TRIGGER trigger_name
{BEFORE  | AFTER  |  INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF  col_name]
ON  table_name
[REFERENCING OLD AS   o   NEW  AS  n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
        -- SQL Statements
 END;
```

CREATE [OR REPLACE] TRIGGER trigger_name

This clause creates a trigger with the given name or overwrites an existing trigger with the same name.

BEFORE | AFTER | INSTEAD OF

This clause indicates at what time the trigger should get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. Before and after cannot be used to create a trigger on a view.

INSERT [OR] | UPDATE [OR] | DELETE

This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.

OF col_name

This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.

ON table_name

This clause identifies the name of the table/view to which the trigger is associated.

REFERENCING OLD AS o NEW AS n

This clause is used to reference the old and new values of the data being changed. By default, you reference the values as **:old.column_name** or **:new.column_name**. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.

FOR EACH ROW

This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire SQL statement is executed (i.e.statement level Trigger).

WHEN (condition)

This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Types of Triggers

There are two types of triggers based on which level it is triggered.

- *Row level trigger* : An event is triggered for each row updated, inserted or deleted.
- *Statement level trigger* : An event is triggered for each SQL statement executed.

Before and After Triggers : Since triggers occur because of events, they may be set to occur immediately *before* or *after* those events. Within the trigger, we are able to refer *old* and *new* values involved in transactions. *Old* refers to the data as it existed prior to the transaction. *New* refer to the data that the transaction creates.

Q11) Before Update : Row level Trigger

Employees may get promoted and continue servicing with new designation. To maintain the job history of the employees, create a table *job_history* with columns *empno, ename, job, pro_date*, and create a trigger to update the table *job_history* whenever there is an updation in *job* column of any row in *emp* table.

```
Create or replace trigger job_history_trigger
Before update of job
on emp
For each row
Begin
        Insert into job_history values(:old.empno,:old.ename,:old.job,sysdate);
End;
```

Q12) Before Insert : Row level Trigger

Create a trigger to convert *employee name* into upper case, before we insert any row into the table *emp* with *employee name* in either case.
(Hint. :new.ename refers the value that is to be inserted)

Q13) After delete : Row level Trigger
Consider tables *dept* and *deptold* with same structure. Create a trigger to move the row into second table whenever a row is removed from first table.

Q14) Create a trigger which will not allow you to enter duplicate or null values in column *empno* of *emp* table.

```
create or replace trigger dubb
before insert on emp
for each row
declare
        cursor c1 is select * from emp;
        x   emp%rowtype;
begin
        open c1;
        loop
                fetch c1 into x;
                if :new.empno = x.empno then
                        dbms_output.put_line('you entered duplicated no');
                elseif :new.empno is null then
                        dbms_output.put_line('you empno is null');
                end if;
                exit when c1%notfound;
        end loop;
        close c1;
end;
```

Q15) Before Insert/Update/Delete : Statement level Trigger

Create a database trigger that allows changes to employee table only during the business hours(i.e. from 8 a.m to 5 p.m.) from Monday to Friday. There is no restriction on viewing data from the table

```
Create or replace trigger time_check
Before insert or update or delete  on emp
Begin
        if  to_number(to_char(sysdate,'hh24')) < 8  or
            to_number(to_char(sysdate,'hh24')) >= 17 or
            to_char(sysdate,'DY') = ‘SAT'' or  to_char(sysdate,'day') = 'SUN'   then
                raise_application_error(-20004,'you can access only between 8am
                to 5pm on Monday to Friday');
        end if;
end;
```

Information about Triggers

We can use the data dictionary 'USER_TRIGGERS' to obtain information about any trigger. The below statement shows the structure of 'USER_TRIGGERS'.

SQL> desc user_triggers;

Q16) Find the trigger type, trigger event and table name of the trigger ‘*time_check’*.

SQL> select trigger_type, trigger_event, table_name
from user_triggers where trigger_name = ‘TIME_CHECK’;

Enabling and Disabling Triggers

Syntax : **ALTER TRIGGER *trigger_name* ENABLE | DISABLE** (or)
ALTER TABLE *table_name* ENABLE | DISABLE ALL TRIGGERS;

Q17) Disable the trigger *‘job_history_trigger’*
SQL>

Q18) Disable all the triggers of *emp* table.
SQL>

Q19) Drop the trigger *‘dubb’*
SQL> drop trigger *dub*;

Verified by

Staff In-charge Sign :	**Date :**

A P P E N D I X - A

CODD'S RULES

Codd's rule provides a method for therotical evalution of a product, that claims to be a Relational Data Base Management System.

Rule 1 : The Information Rule
All information should be represented as data values in the rows and columns of a table.

Rule 2 : The Guaranteed Access Rule
Every item of data must be logically addressable by specifying a combination of the table name, the primary key value and the column name.

Rule 3 : The systematic Treatment of Null values
It is fundamental to the DBMS that NULL values are supported in the representation of missing and inapplicable information. This support for null values must be consistent throughout the DBMS and independent of data types.

Rule 4 : The database Description Rule
A description of the database is held and maintained using the same logical structures used to define the data. Thus allowing users with appropriate authority to query such information in the same way and using the same language as they would for other data in the database.

Rule 5 : The Comprehensive Sub-Language Rule
There must be at least one unified language whose statements can be expressed as character strings confirming to some well defined syntax. It should essentially encapsulate the following :-

- Data Definition Language
- Data Manipulation Language
- View Definition Language
- Integrity Constraints
- Authorization or Data Control Language
- Transaction boundaries

Rule 6 : View Update Rule
All views that are theoretically updateable must be updateable by the system.

Rule 7 : The Insert, Update and Delete Rule
The capability of handling a base relation or in fact a derived relation as a single operand must hold good for all retrieve, insert, update and delete activity.

Rule 8 : The Physical Data Independence Rule

User access to the database via terminal monitors or application programs must remain logically consistent whenever changes to the storage representation or access methods to the data are changed.

Rule 9 : The logical Data Independence Rule
Application programs and terminal activities must remain logically unimpaired whenever information preserving changes of any kind that are theoretically permitted are made to the base tables.

Rule 10 : Integrity Independence Rule
All integrity constraints should be stored in the system catalog or in the database as table.

Rule 11 : Distribution Rule
According to this rule a DBMS which claims to be relational must have distribution independence.

Rule 12 : No subversion Rule
Different levels of the language cannot bypass the integrity rules and constraints. That is, if a DBMS supports a lower level language that permits, then it should not bypass any integrity rules any constraints defined in the higher level.

A P P E N D I X - B

A N S W E R F O R A D D I T I O N A L Q U E R I E S

Q1) SELECT AVG(SCOST) FROM SOFTWARE WHERE DEVIN = 'ORACLE';

Q2) SELECT PNAME,TRUNC(MONTHS_BETWEEN(SYSDATE,DOB)/12) "AGE", TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) "EXPERIENCE" FROM PROGRAMMER;

Q3) SELECT PNAME FROM STUDIES WHERE COURSE = 'PGDCA';

Q4) SELECT MAX(SOLD) FROM SOFTWARE;

Q5) SELECT PNAME, DOB FROM PROGRAMMER WHERE DOB LIKE '%APR%';

Q6) SELECT MIN(CCOST) FROM STUDIES;

Q7) SELECT COUNT(*) FROM STUDIES WHERE COURSE = 'DCA';

Q8) SELECT SUM(SCOST*SOLD-DCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = 'C';

Q9) SELECT * FROM SOFTWARE WHERE PNAME = 'RAKESH';

Q10) SELECT * FROM STUDIES WHERE SPLACE = 'PENTAFOUR';

Q11) SELECT * FROM SOFTWARE WHERE SCOST*SOLD-DCOST > 5000;

Q12) SELECT CEIL(DCOST/SCOST) FROM SOFTWARE;

Q13) SELECT * FROM SOFTWARE WHERE SCOST*SOLD >= DCOST;

Q14) SELECT MAX(SCOST) FROM SOFTWARE GROUP BY DEVIN HAVING DEVIN = 'VB';

Q15) SELECT COUNT(*) FROM SOFTWARE WHERE DEVIN = 'ORACLE';

Q16) SELECT COUNT(*) FROM STUDIES WHERE SPLACE = 'PRAGATHI';

Q17) SELECT COUNT(*) FROM STUDIES WHERE CCOST BETWEEN 10000 AND 15000;

Q18) SELECT AVG(CCOST) FROM STUDIES;

Q19) SELECT * FROM PROGRAMMER WHERE PROF1 = 'C' OR PROF2 = 'C';

Q20) SELECT * FROM PROGRAMMER WHERE PROF1 IN ('C','PASCAL') OR PROF2 IN ('C','PASCAL');

Q21) SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C','C++') AND PROF2 NOT IN ('C','C++');

Q22) SELECT TRUNC(MAX(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = 'M';

Q23) SELECT TRUNC(AVG(MONTHS_BETWEEN(SYSDATE,DOB)/12)) FROM PROGRAMMER WHERE SEX = 'F';

Q24) SELECT PNAME, TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) FROM PROGRAMMER ORDER BY PNAME DESC;

Q25) SELECT PNAME FROM PROGRAMMER WHERE TO_CHAR(DOB,'MON') = TO_CHAR(SYSDATE,'MON');

Q26) SELECT COUNT(*) FROM PROGRAMMER WHERE SEX = 'F';

Q27) SELECT DISTINCT(PROF1) FROM PROGRAMMER WHERE SEX = 'M';

Q28) SELECT AVG(SAL) FROM PROGRAMMER;

Q29) SELECT COUNT(*) FROM PROGRAMMER WHERE SAL BETWEEN 5000 AND 7500;

Q30) SELECT * FROM PROGRAMMER WHERE PROF1 NOT IN ('C','C++','PASCAL') AND PROF2 NOT IN ('C','C++','PASCAL');

Q31) SELECT PNAME,TITLE,SCOST FROM SOFTWARE WHERE SCOST IN (SELECT MAX(SCOST) FROM SOFTWARE GROUP BY PNAME);

Q32) SELECT 'Mr.' || PNAME || ' - has ' || TRUNC(MONTHS_BETWEEN(SYSDATE,DOJ)/12) || ' years of experience' “Programmer” FROM PROGRAMMER WHERE SEX = 'M' UNION SELECT 'Ms.' || PNAME || ' - has ' || TRUNC (MONTHS_BETWEEN (SYSDATE,DOJ)/12) || ' years of experience' “Programmer” FROM PROGRAMMER WHERE SEX = 'F';

Q33) SELECT DISTINCT(A.ENAME) FROM EMP A, EMP B WHERE A.EMPNO = B.MGR; or SELECT ENAME FROM EMP WHERE EMPNO IN (SELECT MGR FROM EMP);

Q34) SELECT * FROM EMP WHERE DEPTNO IN (SELECT DEPTNO FROM EMP GROUP BY DEPTNO HAVING COUNT(EMPNO)>10 AND DEPTNO=10);

Q35) SELECT A.ENAME "EMPLOYEE", B.ENAME "REPORTS TO" FROM EMP A, EMP B WHERE A.MGR=B.EMPNO;

Q36) SELECT * FROM EMP WHERE EMPNO IN (SELECT EMPNO FROM EMP MINUS SELECT MGR FROM EMP);

Q37) SELECT * FROM EMP WHERE SAL > (SELECT MIN(SAL) FROM EMP GROUP BY DEPTNO HAVING DEPTNO=20);

Q38) SELECT * FROM EMP WHERE SAL > (SELECT MAX(SAL) FROM EMP GROUP BY JOB HAVING JOB = 'MANAGER');

Q39) SELECT JOB, MAX(SAL) FROM EMP GROUP BY JOB;

Q40) SELECT * FROM EMP WHERE (DEPTNO, HIREDATE) IN (SELECT DEPTNO, MAX(HIREDATE) FROM EMP GROUP BY DEPTNO);

Q41) SELECT TO_CHAR(HIREDATE,'YYYY') "YEAR", COUNT(EMPNO) "NO. OF EMPLOYEES" FROM EMP GROUP BY TO_CHAR(HIREDATE,'YYYY') HAVING COUNT(EMPNO) = (SELECT MAX(COUNT(EMPNO)) FROM EMP GROUP BY TO_CHAR(HIREDATE,'YYYY'));

Q42) SELECT DEPTNO, LPAD(SUM(12*(SAL+NVL(COMM,0))),15) "COMPENSATION" FROM EMP GROUP BY DEPTNO HAVING SUM(12*(SAL+NVL(COMM,0))) = (SELECT MAX(SUM(12*(SAL+NVL(COMM,0)))) FROM EMP GROUP BY DEPTNO);

Q43) SELECT ENAME, HIREDATE, LPAD('*',8) "RECENTLY HIRED" FROM EMP WHERE HIREDATE = (SELECT MAX(HIREDATE) FROM EMP) UNION SELECT ENAME NAME, HIREDATE, LPAD(' ',15) "RECENTLY HIRED" FROM EMP WHERE HIREDATE != (SELECT MAX(HIREDATE) FROM EMP);

Q44) SELECT ENAME,SAL FROM EMP E WHERE SAL > (SELECT AVG(SAL) FROM EMP F WHERE E.DEPTNO = F.DEPTNO);

Q45) SELECT ENAME, SAL FROM EMP A WHERE &N = (SELECT COUNT (DISTINCT(SAL)) FROM EMP B WHERE A.SAL<=B.SAL);

Q46) SELECT * FROM EMP A WHERE A.EMPNO IN (SELECT EMPNO FROM EMP GROUP BY EMPNO HAVING COUNT(EMPNO)>1) AND A.ROWID!=MIN (ROWID));

Q47) SELECT ENAME "EMPLOYEE",TO_CHAR(TRUNC(MONTHS_BETWEEN(SYSDATE,HIREDATE)/12))||' YEARS '|| TO_CHAR(TRUNC(MOD(MONTHS_BETWEEN (SYSDATE, HIREDATE),12)))||' MONTHS ' "LENGTH OF SERVICE" FROM EMP;

Q48) REFER EXERCISES 1 AND 3 FOR TABLE CREATION.

Q49) SELECT CUST_NAME FROM CUSTOMER WHERE CUST_ID IN
(SELECT CUST_ID FROM SHIPMENT
WHERE DESTINATION='SIOUX CITY');

Q50) SELECT DESTINATION FROM SHIPMENT WHERE 1000000>
(SELECT SUM(ANNUAL_REVENUE) FROM CUSTOMER
WHERE CUSTOMER.CUST_ID = SHIPMENT.CUST_ID
GROUP BY DESTINATION);

Q51) SELECT DESTINATION FROM
(SELECT SUM(WEIGHT)AS SUM_WEIGHT,DESTINATION FROM
SHIPMENT GROUP BY DESTINATION) WHERE SUM_WEIGHT>100;

Q52) SELECT CUST_NAME FROM CUSTOMER,SHIPMENT
WHERE CUSTOMER.CUST_ID=SHIPMENT.CUST_ID AND
ANNUAL_REVENUE>5000000 AND WEIGHT<1;

Q53) SELECT CUST_NAME FROM CUSTOMER,SHIPMENT
WHERE CUSTOMER.CUST_ID=SHIPMENT.CUST_ID AND
ANNUAL_REVENUE>5000000 AND
(DESTINATION='SAN   FRANCISCO'OR WEIGHT<1);

Q54) SELECT DRIVER_NAME FROM TRUCK WHERE TRUCK_# IN
(SELECT TRUCK_# FROM SHIPMENT,CUSTOMER,CITY WHERE
SHIPMENT.CUST_ID=SHIPMENT.CUST_ID AND
ANNUAL_REVENUE>20000000 AND CITY.CITY_NAME =
SHIPMENT.DESTINATION AND POPULATION>1000000);

Q55) SELECT DESTINATION FROM SHIPMENT,CUSTOMER
WHERE CUSTOMER.CUST_ID=SHIPMENT.CUST_ID
AND ANNUAL_REVENUE>15000000;

Q56) SELECT DRIVER_NAME FROM TRUCK WHERE TRUCK_# IN
(SELECT TRUCK_# FROM SHIPMENT WHERE WEIGHT>100);

Q57)
```
SELECT DISTINCT CUST_NAME,ANNUAL_REVENUE
FROM CUSTOMER,SHIPMENT
WHERE CUSTOMER.CUST_ID=SHIPMENT.CUST_ID AND WEIGHT>100;
```

Q58)
```
SELECT DISTINCT CUST_NAME,ANNUAL_REVENUE
FROM CUSTOMER,SHIPMENT,TRUCK
WHERE CUSTOMER.CUST_ID=SHIPMENT.CUST_ID AND
SHIPMENT.TRUCK_#=TRUCK.TRUCK_# AND DRIVER_NAME='JENSEN';
```

Q59)
```
SELECT CUST_NAME FROM CUSTOMER
WHERE NOT EXISTS (SELECT * FROM SHIPMENT,TRUCK
WHERE SHIPMENT.CUST_ID=CUSTOMER.CUST_ID
GROUP BY SHIPMENT.CUST_ID
HAVING COUNT(DISTINCT SHIPMENT.TRUCK_#)<COUNT(DISTINCT
TRUCK.TRUCK_#));
```

Q60)
```
SELECT CITY_NAME FROM CITY
WHERE NOT EXISTS (SELECT * FROM SHIPMENT,CUSTOMER
WHERE SHIPMENT.DESTINATION=CITY.CITY_NAME
GROUP BY SHIPMENT.DESTINATION
HAVING COUNT(DISTINCT SHIPMENT.CUST_ID)<COUNT(DISTINCT
CUSTOMER.CUST_ID));
```

Q61)
```
SELECT DRIVER_NAME FROM TRUCK
WHERE NOT EXISTS (SELECT * FROM SHIPMENT,CITY
WHERE SHIPMENT.TRUCK_#=TRUCK.TRUCK_#
GROUP BY SHIPMENT.TRUCK_#
HAVING COUNT(DISTINCT
SHIPMENT.DESTINATION)<COUNT(DISTINCT CITY.CITY_NAME));
```

Q62)
```
SELECT DISTINCT CUST_NAME FROM CUSTOMER,SHIPMENT
WHERE CUST_TYPE='MANUFACTURER'OR DESTINATION='ST.LOUIS';
          (or)
SELECT DISTINCT CUST_NAME FROM CUSTOMER
WHERE  CUST_TYPE='MANUFACTURER' OR CUST_ID
IN (SELECT CUST_ID FROM SHIPMENT
WHERE DESTINATION='ST.LOUIS');
```

Q63)
```
SELECT CITY_NAME FROM CITY
WHERE POPULATION>1000000 AND CITY_NAME IN
(SELECT DESTINATION FROM SHIPMENT
WHERE WEIGHT=100 AND CUST_ID=311);
```

Q64)
```
SELECT TRUCK_#  FROM TRUCK
WHERE DRIVER_NAME='JAKE STINSON'  AND TRUCK_# NOT IN
(SELECT TRUCK_#   FROM SHIPMENT WHERE
DESTINATION='DENVER');
```

Q65)
```
SELECT CUST_NAME  FROM CUSTOMER
WHERE ANNUAL_REVENUE>10000000 AND CUST_ID IN
(SELECT CUST_ID FROM SHIPMENT,CITY  WHERE WEIGHT<1 AND
POPULATION<10000 AND CITY_NAME=DESTINATION);
```

Q66) a)
```
CREATE VIEW CUST_VIEW1 AS
(SELECT * FROM CUSTOMER
WHERE ANNUAL_REVENUE<1000000);
```

b)
```
CREATE VIEW CUST_VIEW2 AS
(SELECT * FROM CUSTOMER
WHERE ANNUAL_REVENUE BETWEEN 1000000 AND 5000000);
```

b)
```
CREATE VIEW CUST_VIEW3 AS
(SELECT * FROM CUSTOMER
WHERE ANNUAL_REVENUE >5000000);
```

Q67) a)
```
SELECT DRIVER_NAME  FROM TRUCK,CUST_VIEW3,SHIPMENT
WHERE SHIPMENT.CUST_ID=CUST_VIEW3.CUST_ID
AND TRUCK.TRUCK_#=SHIPMENT.TRUCK_#
AND DESTINATION='LOS ANGELES ';
```

c)
```
SELECT CITY_NAME,POPULATION  FROM CITY
WHERE CITY_NAME  IN
(SELECT DESTINATION  FROM SHIPMENT,CUST_VIEW2
WHERE SHIPMENT.CUST_ID=CUST_VIEW2.CUST_ID);
```

d)
```
SELECT  DISTINCT DRIVER_NAME,POPULATION,CITY_NAME
FROM TRUCK,CITY,SHIPMENT
WHERE  SHIPMENT.TRUCK_#=TRUCK.TRUCK_#
AND SHIPMENT.CUST_ID IN  (SELECT CUST_ID FROM
CUST_VIEW1);
```

B I B L I O G R A P H Y

The following books and manuals were referred during the preparation of this work book and suggested for further reading

- SQL * Plus User's Guide and Reference – Oracle Corporation – 2015.
- Introduction to Oracle 9i – Instructors Guide – Oracle corporation – Nancy Greenberg, Priya Nathan – 2015.
- PL/SQL User Guide and Reference – Oracle Corporation – 2007.
- Oracle/SQL Tutorial – Michael Gertz – University of California, Davis – 2010.
- ORACLE SQL*Plus - Prof. Richard Holowczak - City University of New York, USA.
- Database system concepts – Silberschatz, korth and Sundarshan – McGraw Hill Publishers – 6th Edition 2010.
- Peter rob, Carlos Coronel, "Database Systems – Design, Implementation, and Management", 9th Edition, 2009, Thomson Learning, ISBN: 978-0538469685
- Date C.J, "An Introduction to Database", 8th Edition , 2003, Addison-Wesley Pub Co, ISBN: 978-0321197849
- Raghu Ramakrishnan, Johannes Gehrke, "Database Management System", 3rd Edition, 2007, McGraw Hill, ISBN: 978-0072465631