C O N T E N T

Done By:

RA. No.: - RA2211056010132
Name: - Manas Sharma
Class: - AF-2

I N T R O D U C T I O N T O 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 EMPLOYEES and PASSENGERS.

```
CREATE TABLE employees (
   Emp_ID INT PRIMARY KEY,
   E_Name VARCHAR(20),
   Address VARCHAR(50),
   Age INT,
   Email_ID VARCHAR(50),
   Contact VARCHAR(20),
   Air_code VARCHAR(10)
);
```

```
CREATE TABLE passengers (
 Ps_ID INT PRIMARY KEY,
   Ps_Name VARCHAR(20),
   Address VARCHAR(50),
   Age INT,
   Sex VARCHAR(1),
   Contacts VARCHAR(10),
   Flight_ID VARCHAR(15)
);
```

Emp_ID	E_Name	Address	Age	Email_ID	Contact	Air_code
1234	Rekha Tiwary	202-Meeta Apt,Yogi Nagar,Mumbai	30	rekha1234@gmail.com	+918530324018	DEL
2458	Johny Paul	45-Balaji Apt,Ajit Nagar,Jalandhar	32	johnypaul8@gmail.com	+919785425154	AIP
3246	John Dsouza	302-Fountain Apt,ElizaBeth Street,Newham	26	john2346@gmail.com	+447911123456	BOM
4521	Nidhi Maroliya	6-Matruchaya Apt,Park Road,Jammu	31	nidhi785@gmail.com	+918211954901	IXJ
5123	Lara Jen	28-Mark road,Victoria street,New York City	31	jenlara4@gmail.com	+448000751234	CPH
7512	Akshay Sharma	Akshay Villa,Queens Street,Copenhagen	20	akshay27@gmail.com	+45886443210	JFK
8512	Hafsa Iqmar	1023-Prajwal Apt,Newark	41	hafsa964@gmail.com	6465554468	EWR
9321	Sanjay Rathod	62-Patwa Apt,Pradeep Nagar,Delhi	36	sanjay78@gmail.com	+917504681201	LCY
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Q2) Confirm table creation

desc EMPLOYEES***;***

desc PASSENGERS***;***

Field	Type	Null	Key	Default	Extra
Emp_ID	int(11)	NO	PRI	NULL	
E_Name	varchar(20)	YES		NULL	
Address	varchar(50)	YES		NULL	
Age	int(11)	YES		NULL	
Email_ID	varchar(20)	YES		NULL	
Contact	varchar(20)	YES		NULL	
Air_code	varchar(10)	YES	MUL	NULL	

Q3) List name of the tables created by the user

SHOW TABLES;

Tables_in_dbms
airfare
airplane_type
airport
can_land
countries
employees
flight
passengers
ps
route
transactions
travels_on

Q4) Describe tables owned by the user

desc EMPLOYEES*;*

desc PASSENGERS*;*

Field	Type	Null	Key	Default	Extra
Ps_ID	int(11)	NO	PRI	NULL	
Ps_Name	varchar(20)	NO		NULL	
Address	varchar(50)	NO		NULL	
Age	int(11)	NO		NULL	
Sex	varchar(1)	NO		NULL	
Contacts	varchar(10)	NO		NULL	
Flight_ID	varchar(15)	NO	MUL	NULL	

Q5) View distinct object types owned by the user

SELECT DISTINCT permission.

FROM can_land;

permission
True
False

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

```
SHOW TABLES;
WHERE Table_Type = 'VIEW';
```

Tables_in_dbms	Table_type
ps	VIEW

SQL Table Management Questions:

Q1) How to add a new column named "Max_Speed" (int) to the Airplane_type table?

```
ALTER TABLE Airplane_type
ADD Max_Speed int;
```

A_ID	Capacity	A_weight	Company	Max_Speed
738	852	394	Indigo	NULL
745	769	405	GoAir	NULL
750	789	364	AirIndia	NULL
768	866	387	AirAsia	NULL
777	799	380	Vistara	NULL
785	834	410	Alliance Air	NULL
790	849	390	SpiceJet	NULL
821	789	355	TruJet	NULL
NULL	NULL	NULL	NULL	NULL

Q2) How to rename the "City" column in the Airport table to "Location"?

```
ALTER TABLE Airport
CHANGE COLUMN `City` `Location` VARCHAR(100);
```

Air_code	Air_Name	Location	State	Country_code
AIP	Adampur Airport	Jalandhar	Punjab	91
BOM	Chhatrapati Shivaji Maharaj International Airport	Mumbai	Maharashtra	91
CPH	Copenhagen Airport	Copenhagen	Denmark	45
DEL	Indira Gandhi International Airport	Delhi	UP	91
EWR	Newark Liberty International Airport	Newark	New Jersey	1
IXJ	Satwari Airport	Jammu	Jammu & Kashmir	91
JFK	John F.Kennedy International Airport	New York City	New York	1
LCY	London City Airport	Newham	London	44
NULL	NULL	NULL	NULL	NULL

Q3) How to delete the "Max speed" column from the airplane_type table?

```
ALTER TABLE airplane_type
DROP COLUMN Max_Speed;
```

A_ID	Capacity	A_weight	Company
738	852	394	Indigo
745	769	405	GoAir
750	789	364	AirIndia
768	866	387	AirAsia
777	799	380	Vistara
785	834	410	Alliance Air
790	849	390	SpiceJet
821	789	355	TruJet
NULL	NULL	NULL	NULL

Q4) How to modify the data type of the "Contacts" column in the Passengers table to varchar(15)?

```
ALTER TABLE Passengers
MODIFY COLUMN Contacts varchar(15);
```

Q5) How to add a foreign key constraint on the "Country_code" column in the Airport table referencing the "Country_code" column in the Countries table?

```
ALTER TABLE Airport
ADD CONSTRAINT fk_airport_country
FOREIGN KEY (Country_code)
REFERENCES Countries(Country_code);
```

Q6) How to create a new table named "Crew" with columns for "Crew_ID" (int, primary key), "Crew_Name" (varchar), and "Position" (varchar)?

```
CREATE TABLE Crew (
Crew_ID int PRIMARY KEY,
Crew_Name varchar(50),
Position varchar(50)
);
```

Crew_ID	Crew_Name	Position
NULL	NULL	NULL

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

EMP_ID	E_NAME	ADDRRESS	AGE	EMAIL_ID	CONTACT	AIRCODE
1234	Rekha Tiwary	202-Meeta Apt,Yogi Nagar,Mumbai	30	rekha1234@gmail.com	+918530324018	DEL
2458	Johny Paul	45-Balaji Apt,Ajit Nagar,Jalandar	32	johnypaul8@gmail.com	+919785425154	AIP
3246	John Dsouza	302-Fountain Apt,ElizaBeth Street, Newham	26	john2346@gmail.com	+447911123456	BOM
4521	Nidhi Maroliya	6-Matruchaya Apt,Park Road, Jammu	31	nidhi785@gmail.com	+918211954901	IXJ
5123	Lara Jen	28-Mark road,Victoria street,New York City	31	jenlara4@gmail.com	+448000751234	CPH
7512	Akshay Sharma	Akshay Villa,Queens Street,Copenhagen	20	akshay27@gmail.com	+45886443210	JFK
8512	Hafsa Iqmar	1023-Prajwal Apt,Newark	41	hafsa964@gmail.com	6465554468	EWR
9321	Sanjay Rathod	62-Patwa Apt,Pradeep Nagar, Delhi	36	sanjay78@gmail.com	+917504681201	LCY
1234	Rekha Tiwary	202-Meeta Apt,Yogi Nagar,Mumbai	30	rekha1234@gmail.com	+918530324018	DEL
2458	Johny Paul	45-Balaji Apt,Ajit Nagar,Jalandar	32	johnypaul8@gmail.com	+919785425154	AIP
3246	John Dsouza	302-Fountain Apt,ElizaBeth Street, Newham	26	john2346@gmail.com	+447911123456	BOM
4521	Nidhi Maroliya	6-Matruchaya Apt,Park Road, Jammu	31	nidhi785@gmail.com	+918211954901	IXJ
5123	Lara Jen	28-Mark road,Victoria street,New York City	31	jenlara4@gmail.com	+448000751234	CPH
7512	Akshay Sharma	Akshay Villa,Queens Street,Copenhagen	20	akshay27@gmail.com	+45886443210	JFK

Data for AIRPORT table

AIP	Adampur Airport	Jalandhar	Punjab	91
BOM	Chhatrapati Shivaji Maharaj International Airport	Mumbai	Maharashtra	91
CPH	Copenhagen Airport	Copenhagen	Denmark	45
DEL	Indira Gandhi International Airport	Delhi	UP	91
EWR	Newark Liberty International Airport	Newark	New Jersey	1

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.

(ii) Insert a new row containing values for each column**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 Airport table using syntax.

INSERT INTO Airport (Air_code, Air_Name, City, State, Country_code) VALUES ('JFK', 'John F. Kennedy International Airport', 'New York', 'NY', 1);

INSERT INTO Airport (Air_code, Air_Name, City, State, Country_code) VALUES ('LAX', 'Los Angeles International Airport', 'Los Angeles', 'CA', 1);

INSERT INTO Airport (Air_code, Air_Name, City, State, Country_code) VALUES ('LHR', 'London Heathrow Airport', 'London', NULL, 2);

INSERT INTO Airport (Air_code, Air_Name, City, State, Country_code) VALUES ('CDG', 'Charles de Gaulle Airport', 'Paris', NULL, 3);

Air_code	Air_Name	City	State	Country_code
AIP	Adampur Airport	Jalandhar	Punjab	91
BOM	Chhatrapati Shivaji Maharaj International Airport	Mumbai	Maharashtra	91
CDG	Charles de Gaulle Airport	Paris	NULL	3
CPH	Copenhagen Airport	Copenhagen	Denmark	45
DEL	Indira Gandhi International Airport	Delhi	UP	91
EWR	Newark Liberty International Airport	Newark	New Jersey	1
IXJ	Satwari Airport	Jammu	Jammu & Kashmir	91
JFK	John F.Kennnedy International Airport	New York City	New York	1
LAX	Los Angeles International Airport	Los Angeles	CA	1
LCY	London City Airport	Newham	London	44
LHR	London Heathrow Airport	London	NULL	2
NULL	NULL	NULL	NULL	NULL

Q2) Insert first & second rows of Employees table using syntax (ii)

INSERT INTO Employees (Emp_ID, E_Name, Address, Age, Email_ID, Contact, Air_code)
VALUES (1001, 'John Doe', '123 Main St, New York', 30, 'john.doe@example.com', '1234567890', 'JFK');

INSERT INTO Employees (Emp_ID, E_Name, Address, Age, Email_ID, Contact, Air_code)
VALUES (1002, 'Jane Smith', '456 Elm St, Los Angeles', 28, 'jane.smith@example.com', '9876543210', 'LAX');

Emp_ID	E_Name	Address	Age	Email_ID	Contact	Air_code
1001	John Doe	123 Main St, New York	30	john.doe@example.com	1234567890	JFK
1002	Jane Smith	456 Elm St, Los Angeles	28	jane.smith@example.com	9876543210	LAX

Q3) Insert the remaining rows of Employees table using syntax (iii).

```
INSERT INTO Employees (Emp_ID, E_Name, Address, Age, Email_ID, Contact, Air_code)
VALUES (1003, 'Michael Johnson', '789 Oak St, London', 35, 'michael.johnson@example.com',
'1112223333', 'LHR');
INSERT INTO Employees (Emp_ID, E_Name, Address, Age, Email_ID, Contact, Air_code)
VALUES (1004, 'Emily Williams', '321 Pine St, Paris', 32, 'emily.williams@example.com',
'4445556666', 'CDG');
```

1003	Michael Johnson	789 Oak St, London	35	michael.johnson@example.com	1112223333	LHR
1004	Emily Williams	321 Pine St, Paris	32	emily.williams@example.com	4445556666	CDG

Q4) Create a table Manager with the columns mgr_id, name, salary, and hire_date

```
CREATE TABLE Manager (
   Mgr_ID INT PRIMARY KEY,
   Name VARCHAR(20),
   Salary DECIMAL(10, 2),
   Hire_Date DATE
);
```

	Mgr_ID	Name	Salary	Hire_Date
*	NULL	NULL	NULL	NULL

Table: manager

Columns:

Mgr_ID	int PK
Name	varchar(20)
Salary	decimal(10,2)
Hire_Date	date

Q5) Change State of all rows of Airport table to 'NY'

```
UPDATE Airport
SET State = 'NY';
```

	Air_code	Air_Name	City	State	Country_code
▶	AIP	Adampur Airport	Jalandhar	NY	91
	BOM	Chhatrapati Shivaji Maharaj International Airport	Mumbai	NY	91
	CDG	Charles de Gaulle Airport	Paris	NY	3
	CPH	Copenhagen Airport	Copenhagen	NY	45
	DEL	Indira Gandhi International Airport	Delhi	NY	91
	EWR	Newark Liberty International Airport	Newark	NY	1
	IXJ	Satwari Airport	Jammu	NY	91
	JFK	John F.Kennnedy International Airport	New York City	NY	1
	LAX	Los Angeles International Airport	Los Angeles	NY	1
	LCY	London City Airport	Newham	NY	44
	LHR	London Heathrow Airport	London	NY	2
*	NULL	NULL	NULL	NULL	NULL

Q7) Change State='CA' for Air_code='LAX' in Airport table

```
UPDATE Airport
SET State = 'CA'
WHERE Air_code = 'LCY';
```

LCY	London City Airport	Newham	CA	44

Q8) Delete the rows from Employees table whose employee name = ‘Lara Jen’

```
DELETE FROM Employees
WHERE E_Name = ‘Lara Jen’;
```

Q9) List all the columns and rows of the Airport table

```
SELECT * FROM Airport;
```

Air_code	Air_Name	City	State	Country_code
AIP	Adampur Airport	Jalandhar	Punjab	91
BOM	Chhatrapati Shivaji Maharaj International Airport	Mumbai	Maharashtra	91
CPH	Copenhagen Airport	Copenhagen	Denmark	45
DEL	Indira Gandhi International Airport	Delhi	UP	91
EWR	Newark Liberty International Airport	Newark	New Jersey	1
IXJ	Satwari Airport	Jammu	Jammu & Kashmir	91
JFK	John F.Kennedy International Airport	New York City	New York	1
LCY	London City Airport	Newham	CA	44
NULL	NULL	NULL	NULL	NULL

Q10) List the name of the employee and ag of Employees table

```
SELECT E_Name AS Employee_Name, Age
FROM Employees;
```

Employee_Name	Age
Rekha Tiwary	30
Johny Paul	32
John Dsouza	26
Nidhi Maroliya	31
Lara Jen	31
Akshay Sharma	20
Hafsa Iqmar	41
Sanjay Rathod	36

Q11) Without duplication, list all names of the City from Airport table.

SELECT DISTINCT City FROM Airport;

City
Jalandhar
Mumbai
Copenhagen
Delhi
Newark
Jammu
New York City
Newham

Q12) Find out the name of an employee whose Emp_ID is 1003.

SELECT E_Name
FROM Employees
WHERE Emp_ID = 1234;

E_Name
Rekha Tiwary

Q13) As a copy of Airport table, create a new table named Airport_Copy using select command.

CREATE TABLE Airport_Copy AS SELECT * FROM Airport;

379 | 09:34:21 | CREATE TABLE Airport_Copy AS SELECT * FROM Airport | 8 row(s) affected Records: 8 Duplicates: 0 Warnings: 0

Q14) List Ps_Name and Age of Passengers table with the column headings NAME and AGE

```
SELECT Ps_Name AS NAME, Age AS AGE
FROM Passengers;
```

NAME	AGE
Alfred Schmidt	30
Ankita Ahir	26
Khyati Mishra	30
Akhilesh Joshi	29
Rom Solanki	60
Lakshmi Sharma	30
Ria Gupta	34
Manan Lakhani	45

Q15) Change State='IL' for Air_code='ORD' in Airport table and COMMIT the transaction.

```
UPDATE Airport
SET State = 'IL'
WHERE Air_code = 'ORD';
COMMIT;
```

385	09:35:58	UPDATE Airport SET State = 'UP' WHERE Air_code = 'ORD'	0 row(s) affected Rows matched: 0 Changed: 0 Warnings: 0
386	09:35:58	COMMIT	0 row(s) affected

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

```
DELETE FROM Employees;
ROLLBACK;
```

Emp_ID	E_Name	Address	Age	Email_ID	Contact	Air_code
1234	Rekha Tiwary	202-Meeta Apt,Yogi Nagar,Mumbai	30	rekha1234@gmail.com	+918530324018	DEL
2458	Johny Paul	45-Balaji Apt,Ajit Nagar,Jalandar	32	johnypaul8@gmail.com	+919785425154	AIP
3246	John Dsouza	302-Fountain Apt,ElizaBeth Street, Newham	26	john2346@gmail.com	+447911123456	BOM
4521	Nidhi Maroliya	6-Matruchaya Apt,Park Road, Jammu	31	nidhi785@gmail.com	+918211954901	IXJ
5123	Lara Jen	28-Mark road,Victoria street,New York City	31	jenlara4@gmail.com	+448000751234	CPH
7512	Akshay Sharma	Akshay Villa,Queens Street,Copenhagen	20	akshay27@gmail.com	+45886443210	JFK
8512	Hafsa Iqmar	1023-Prajwal Apt,Newark	41	hafsa964@gmail.com	6465554468	EWR
9321	Sanjay Rathod	62-Patwa Apt,Pradeep Nagar, Delhi	36	sanjay78@gmail.com	+917504681201	LCY
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Q17) Do the following operations one after another

a) Change City='Boston' for Air_code='BOS' in Airport table

```
UPDATE Airport
SET City = 'Boston'
WHERE Air_code = 'BOS';
```

	BOS	Boston International Airport	Boston	NY	1

b) Create SAVEPOINT in the name 'update_over'

```
SAVEPOINT update_over;
```

c) Insert another row in Airport table with your own values

```
INSERT INTO Airport (Air_code, Air_Name, City, State, Country_code)
VALUES ('YYZ', 'Toronto Pearson International Airport', 'Toronto', 'ON', 4);
```

	YYZ	Toronto Pearson International Airport	Toronto	ON	4

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.

Table: customer1

Columns:

Customer_ID	int PK
Name	varchar(50)
Address	varchar(100)
Age	int
Email	varchar(50)
Phone	varchar(20)
Country_code	int

```
CREATE TABLE Customer1 (
   Customer_ID int,
   Name varchar(50),
   Address varchar(100),
   Age int,
   Email varchar(50),
   Phone varchar(20),
   Country_code int NOT NULL,
   CONSTRAINT customer1_pk PRIMARY KEY (Customer_ID));
```

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

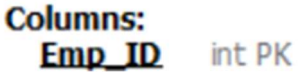
ALTER TABLE *table* ADD CONSTRAINT *const-name* cons-*type* (*column*);

Q2) Add NOT NULL constraint to the columns E_Name and Position of Employees table.

```
SQL> ALTER TABLE Employees
   MODIFY(E_Name varchar(20) NOT
         NULL);
```

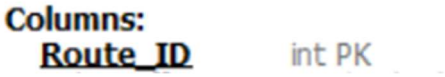
Q3) Add Primary key constraint to the column Emp_ID of Employees table

```
SQL> ALTER TABLE Employees
   ADD CONSTRAINT emp_pk PRIMARY KEY(Emp_ID);
```



Q4) Add Primary key constraint to the column Route_ID of Route table

```
SQL> ALTER TABLE Route
   ADD CONSTRAINT route_pk PRIMARY KEY(Route_ID);
```



Q5) Add Unique key constraint to the column Air_Name of Airport table

```
SQL> ALTER TABLE Airport
   ADD CONSTRAINT airport_uk UNIQUE(Air_Name);
```

Q6) Add Check constraint to the table Passengers to restrict the values of Age to be greater than or equal to 18.

```
SQL> ALTER TABLE Passengers
   ADD CONSTRAINT passengers_ck CHECK(Age >= 18);
```

Q7) Add Foreign key constraint to the column Air_code of Flight table referencing Air_code of Airport table.

SQL> ALTER TABLE Flight
ADD CONSTRAINT flight_fk_airport FOREIGN KEY(Air_code) REFERENCES Airport(Air_code);

Foreign Key Name	Referenced Table
flight_ibfk_1	`airline`.`airplane_type`

Column	Referenced Column
Flight_ID	
Departure	
Arrival	
Flight_date	
A_ID	A_ID

Q8) Add a Foreign key constraint to the Can_Land table indicating that a flight must already exist as a valid flight in the Flight table.

SQL> ALTER TABLE Can_Land
ADD CONSTRAINT can_land_fk_flight FOREIGN KEY(Flight_ID)
REFERENCES Flight(Flight_ID);

Foreign Key Name	Referenced Table
can_land_ibfk_1	`airline`.`airport`
can_land_ibfk_2	`airline`.`flight`

Q9) Remove the Can_Land constraint (added in Q8) from Can_Land table

SQL> ALTER TABLE Can_Land
DROP CONSTRAINT can_land_fk_flight;

Foreign Key Name	Referenced Table
can_land_ibfk_1	`airline`.`airport`

Q10) Remove the primary key constraint on the Transactions table and drop the associated foreign key constraint on the Transactions.Flight_ID column.

SQL> ALTER TABLE Transactions
DROP CONSTRAINT ts_pk CASCADE;

Table: transactions

Columns:
TS_ID int

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

1. Retrieve all airplane types with their capacities

SELECT A_ID, Capacity FROM Airplane_type;

A_ID	Capacity
738	853
745	770
750	790
768	867
777	800
785	835
790	850
821	790
NULL	NULL

2. Get all routes along with their take-off points and destinations

SELECT Route_ID, Take_Off_point, Destination FROM Route;

Route_ID	Take_Off_point	Destination
157306	NewJersey	Mumbai
168806	London	Delhi
178916	Washington	Jodhpur
324567	Chennai	Denmark
421523	Hyderabad	Jammu & Kashmir
452368	Chandigard	NewYork
578425	Beijing	Punjab
894521	Daman	Delhi
NULL	NULL	NULL

3. List all flights with their departure and arrival information

SELECT Flight_ID, Departure, Arrival FROM Flight;

Flight_ID	Departure	Arrival
AA4367	2021-03-25 12:05am	2021-03-25 02:15am
AI2014	2021-01-12 08:45am	2021-01-12 10:25pm
BA1689	2021-03-02 2:15am	2021-03-02 10:00pm
CT7812	2021-04-04 2:15pm	2021-04-04 8:00pm
EY1234	2021-02-10 05:00am	2021-02-10 10:30pm
LH9876	2021-02-25 10:15am	2021-02-25 11:00pm
PF4521	2020-12-25 5:00pm	2020-12-25 10:30pm
QR2305	2020-12-26 12:05pm	2020-12-27 12:25pm
NULL	NULL	NULL

4. Retrieve the fare details for all flights

SELECT * FROM AirFare;

Fare_ID	Charge_Amount	Description	Flight_ID
1	27341	Standard Single	AI2014
2	42176	Key Fare Single	EY1234
3	27373	Business Return	LH9876
4	34837	Standard Return	QR2305
5	8777	Superpex Return	AA4367
6	44592	Advanced Purchase	BA1689
7	9578	Standard Return	CT7812
8	4459	Superpex Return	PF4521
NULL	NULL	NULL	NULL

5. Fetch all passenger details

SELECT * FROM Passengers;

Ps_ID	Ps_Name	Address	Age	Sex	Contacts	Flight_ID
1	Alfred Schmidt	2230 Northside,Apt 11,London	30	M	8080367290	AI2014
2	Ankita Ahir	3456 Vikas Apts,Apt 102,New Jersey	26	F	8080367280	QR2305
3	Khyati Mishra	7820 Mccallum courts,Apt 234,Washington	30	F	8082267280	LH9876
4	Akhilesh Joshi	345 Chatam courts,Apt 678,Chennai	29	M	9080369290	EY1234
5	Rom Solanki	1234 Baker Apts,Apt 208,Chandigard	60	M	9004568903	EY1234
6	Lakshmi Sharma	1110 Fir hills,Apt 90,Daman	30	F	7666190505	AA4367
7	Ria Gupta	B-402,Aditya Apt,Hyderabad	34	F	9819414036	EY1234
8	Manan Lakhani	7720 Mccallum Blvd,Apt 77,Beijing	45	M	8124579635	CT7812
NULL	NULL	NULL	NULL	NULL	NULL	NULL

6. Get the names and ages of passengers younger than 30

SELECT Ps_Name, Age FROM Passengers WHERE Age < 30;

Ps_Name	Age
Ankita Ahir	26
Akhilesh Joshi	29

7. List all flights departing from 'New York City'

SELECT * FROM Flight WHERE Departure LIKE '%New York City%';

Flight_ID	Departure	Arrival	Flight_date	A_ID
AA4367	New York City	2021-03-25 02:15am	2021-03-25	768
AI2014	New York City	2021-01-12 10:25pm	2021-01-12	738
BA1689	New York City	2021-03-02 10:00pm	2021-03-02	745
CT7812	New York City	2021-04-04 8:00pm	2021-04-04	821
NULL	NULL	NULL	NULL	NULL

8. Get the total charge amount for each flight

SELECT Flight_ID, SUM(Charge_Amount) AS Total_Charge FROM AirFare GROUP BY Flight_ID;

Flight_ID	Total_Charge
AA4367	8777
AI2014	27341
BA1689	44592
CT7812	9578
EY1234	42176
LH9876	27373
PF4521	4459
QR2305	34837

9. Retrieve the names and contact details of employees working at 'JFK' airport

```
SELECT E_Name, Contact FROM Employees WHERE Air_code = 'JFK';
```

E_Name	Contact
Akshay Sharma	+45886443210

10. List all flights with their corresponding airplane types

```
SELECT Flight.Flight_ID, Airplane_type.A_ID, Airplane_type.Capacity
FROM Flight
INNER JOIN Airplane_type ON Flight.A_ID = Airplane_type.A_ID;
```

Flight_ID	A_ID	Capacity
AI2014	738	853
BA1689	745	770
EY1234	750	790
AA4367	768	867
QR2305	777	800
PF4521	785	835
LH9876	790	850
CT7812	821	790

11. Get the flight ID and departure date of transactions where the charge amount is greater than 500

```
SELECT Flight_ID, Departure_Date
FROM Transactions
WHERE Charge_Amount > 500;
```

Flight_ID	Departure_Date
AI2014	2021-02-22
AA4367	2021-02-08
CT7812	2021-03-08
LH9876	2021-03-16
QR2305	2021-01-14
EY1234	2020-12-02
BA1689	2021-04-25

12. Fetch the flight IDs and routes where the destination is 'LCY'

```
SELECT Travels_on.Flight_ID, Route.Route_ID
FROM Travels_on
INNER JOIN Route ON Travels_on.Route_ID = Route.Route_ID
WHERE Route.Destination = ' Jammu & Kashmir ';
```

Flight_ID	Route_ID
PF4521	421523

13. Retrieve the unique country codes present in the Airport table

```
SELECT DISTINCT Country_code FROM Airport;
```

Country_code
1
2
3
4
44
45
91

14. List all flights in ascending order of their flight dates

```
SELECT * FROM Flight ORDER BY Flight_date ASC;
```

Flight_ID	Departure	Arrival	Flight_date	A_ID
PF4521	2020-12-25 5:00pm	2020-12-25 10:30pm	2020-12-25	785
QR2305	2020-12-26 12:05pm	2020-12-27 12:25pm	2020-12-26	777
AI2014	New York City	2021-01-12 10:25pm	2021-01-12	738
EY1234	2021-02-10 05:00am	2021-02-10 10:30pm	2021-02-10	750
LH9876	2021-02-25 10:15am	2021-02-25 11:00pm	2021-02-25	790
BA1689	New York City	2021-03-02 10:00pm	2021-03-02	745
AA4367	New York City	2021-03-25 02:15am	2021-03-25	768
CT7812	New York City	2021-04-04 8:00pm	2021-04-04	821
NULL	NULL	NULL	NULL	NULL

15. Get the flight IDs and departure dates of transactions where the transaction type is 'Booking'

```
SELECT Flight_ID, Departure_Date
FROM Transactions
WHERE TS_Type = 'Booking';
```

Flight_ID	Departure_Date
AA4367	2021-02-08
EY1234	2020-12-02
BA1689	2021-04-25

16. Fetch the passenger names and flight IDs of passengers traveling on flights departing from 'DEL' airport

```
SELECT Passengers.Ps_Name, Passengers.Flight_ID
FROM Passengers
INNER JOIN Flight ON Passengers.Flight_ID = Flight.Flight_ID
WHERE Flight.Departure = 'DEL';
```

	Ps_Name	Flight_ID
▶	Lakshmi Sharma	AA4367
	Alfred Schmidt	AI2014
	Manan Lakhani	CT7812

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

SCALAR FUNCTIONS

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

```
SQL> SELECT (Hire_Date) AS "Hire Date"
    FROM Employees
    WHERE Dept_No = 20;
```

	Hire Date
▶	1983-01-12
	1984-02-13
	1985-03-14
	1986-04-15
	1987-05-16

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

```
SQL> SELECT TO_CHAR(Hire_Date, 'YYYY-MM-DD HH24:MI:SS') AS "Hire Date
        with Time"
    FROM Employees
    WHERE Dept_No = 20;
```

	Hire Date with Time
▶	1983-01-12 00:00:00
	1984-02-13 00:00:00
	1985-03-14 00:00:00
	1986-04-15 00:00:00
	1987-05-16 00:00:00

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 works in dept no 20.

```
SQL> SELECT E.E_Name, E.Hire_Date, (E.Hire_Date + 150) AS "First Salary Review
     Date"
    FROM Employees E
    WHERE E.Dept_No = 20;
```

	E_Name	Hire_Date	First Salary Review Date
▶	Employee1	1983-01-12	19830262
	Employee2	1984-02-13	19840363
	Employee3	1985-03-14	19850464
	Employee4	1986-04-15	19860565
	Employee5	1987-05-16	19870666

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('database management') from dual;	
lower(char)	Lower case	Select lower('WELCOME') from dual;	
upper(char)	Upper case	Select upper('srmist') 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;	

GROUP FUNCTIONS

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 :
  - WHERE clause
  - GROUP BY clause
  - HAVING clause

Q9) Find number of rows in the table EMP

SQL > SELECT COUNT(*) AS row_count FROM EMP;

	row_count
▶	5

Q10) Find number of talent according to hire date the table available in EMP table.

SQL> SELECT COUNT(DISTINCT Hire_date) AS "Number of talent"
FROM Employees;

	Number of talent
▶	5

Q11) Find number of employees who earn dept in EMP table.

SQL> SELECT COUNT(*) AS "Number of Employees with dept"
FROM Employees
WHERE Dept_No IS NOT NULL;

	Number of Employees with dept
▶	5

Q12) What is the difference between the following queries

SQL > select count(Dept_No) from emp;

	count(Dept_No)
▶	5

SQL > select count(nvl(Dept_No,0)) from emp;

Error Code: 1305. FUNCTION copydatabase.nvl does not exist

Q13) Find the total salary paid to the employees.

SQL> SELECT SUM(Salary) AS "Total Salary Paid"
FROM Employees;

	Total Salary Paid
▶	NULL

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

SQL> SELECT MAX(Salary) AS "Maximum Salary",
MIN(Salary) AS "Minimum Salary",
AVG(Salary) AS "Average Salary"
FROM Employees;

	Maximum Salary	Minimum Salary	Average Salary
▶	NULL	NULL	NULL

Q15) Find number of employees who work in department number 30

```
SQL> SELECT COUNT(*) AS "Number of Employees in Department 30"
    FROM Employees
    WHERE Dept_No = 30;
```

Q16) Find the maximum salary paid to a ‘CLERK’

```
SQL> SELECT MAX(Salary) AS "Maximum Salary for CLERK"
    FROM Employees
    WHERE Job = 'CLERK';
```

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

```
SQL> SELECT Dept_No, COUNT(*) AS "Number of Employees"
    FROM Employees
    GROUP BY Dept_No;
```

	Dept_No	Number of Employees
▶	20	5

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 Dept_No, COUNT(*) AS "Number of Employees"
    FROM Employees
    GROUP BY Dept_No
    ORDER BY COUNT(*) DESC;
```

	Dept_No	Number of Employees
▶	20	5

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

```
SQL> SELECT Dept_No, SUM(Salary) AS "Total Salary",
          MAX(Salary) AS "Maximum Salary",
          MIN(Salary) AS "Minimum Salary",
          AVG(Salary) AS "Average Salary"
    FROM Employees
    GROUP BY Dept_No;
```

	Dept_No	Total Salary	Maximum Salary	Minimum Salary	Average Salary
▶	20	NULL	NULL	NULL	NULL

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 Dept_No, SUM(Salary) AS "Total Salary",
       MAX(Salary) AS "Maximum Salary",
       MIN(Salary) AS "Minimum Salary",
       AVG(Salary) AS "Average Salary"
    FROM Employees
    WHERE Dept_No = 20
    GROUP BY Dept_No
    HAVING AVG(Salary) > 1000;
```

	Dept_No	Total Salary	Maximum Salary	Minimum Salary	Average Salary

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.

1. Retrieve all flights along with their departure and arrival locations.

```
SELECT Flight.Flight_ID, Airport.Air_Name AS Departure_Location,
    Airport_1.Air_Name AS Arrival_Location
FROM Flight
JOIN Airport ON Flight.Departure = Airport.Air_code
JOIN Airport AS Airport_1 ON Flight.Arrival = Airport_1.Air_code;
```

Flight_ID	Flight_date
CT7812	2021-04-04

2. List all passengers along with their flight details and respective departure dates.

```
SELECT Passengers.Ps_Name, Flight.Flight_ID, Flight.Departure_Date
FROM Passengers
JOIN Flight ON Passengers.Flight_ID = Flight.Flight_ID;
```

Ps_Name	Flight_ID	Flight_date
Alfred Schmidt	AI2014	2021-01-12
Ankita Ahir	QR2305	2020-12-26
Khyati Mishra	LH9876	2021-02-25
Akhilesh Joshi	EY1234	2021-02-10
Rom Solanki	EY1234	2021-02-10
Lakshmi Sharma	AA4367	2021-03-25
Ria Gupta	EY1234	2021-02-10
Manan Lakhani	CT7812	2021-04-04

3. Find all flights operated by a specific company along with their airplane types and capacities.

```
SELECT Flight.Flight_ID, Airplane_type.Company, Airplane_type.Capacity
FROM Flight
JOIN Airplane_type ON Flight.A_ID = Airplane_type.A_ID
WHERE Airplane_type.Company = 'Company_Name';
```

Flight_ID	Company	Capacity
AI2014	Indigo	853
BA1689	Indigo	770

4. Retrieve all transactions made by passengers, including their names, transaction types, and charge amounts.

```
SELECT Passengers.Ps_Name, Transactions.TS_Type, Transactions.Charge_Amount
FROM Passengers
JOIN Transactions ON Passengers.Ps_ID = Transactions.Ps_ID;
```

Ps_Name	TS_Type	Charge_Amount
Alfred Schmidt	Google Pay	27341
Lakshmi Sharma	Paytm	8777
Manan Lakhani	PhonePe	9578
Khyati Mishra	PhonePe	27373
Ankita Ahir	Credit Card	34837
Akhilesh Joshi	Paytm	42176
Rom Solanki	Paytm	44592

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.
  - Equi Join : A join condition with relationship = .
  - 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.

5. List all flights along with their corresponding route types.

```
SELECT Flight.Flight_ID, Route.R_type
FROM Flight
JOIN Travels_on ON Flight.Flight_ID = Travels_on.Flight_ID
JOIN Route ON Travels_on.Route_ID = Route.Route_ID;
```

Flight_ID	R_type
AI2014	Direct
BA1689	3Hr Break
EY1234	3Hr Break
AA4367	Direct
QR2305	2Hr Break
PF4521	Direct
LH9876	Direct
CT7812	Direct

6. Find all flights departing from a specific city and their respective departure dates.

```
SELECT Flight.Flight_ID, Flight.Departure_Date
FROM Flight
JOIN Airport ON Flight.Departure = Airport.Air_code
WHERE Airport.City = 'City_Name';
```

Flight_ID	Flight_date
CT7812	2021-04-04

7. Retrieve all employees along with the airports they work in and their contact details.

```
SELECT Employees.E_Name, Airport.Air_Name AS Airport_Working,
       Employees.Contact
FROM Employees
JOIN Airport ON Employees.Air_code = Airport.Air_code;
```

E_Name	Airport_Working	Contact
Rekha Tiwary	Indira Gandhi International Airport	+918530324018
Johny Paul	Adampur Airport	+919785425154
John Dsouza	Chhatrapati Shivaji Maharaj International Airport	+447911123456
Nidhi Maroliya	Satwari Airport	+918211954901
Lara Jen	Copenhagen Airport	+448000751234
Akshay Sharma	John F.Kennnedy International Airport	+45886443210
Hafsa Iqmar	Newark Liberty International Airport	6465554468
Sanjay Rathod	London City Airport	+917504681201

8. List all flights along with their corresponding charges and descriptions.

```
SELECT Flight.Flight_ID, AirFare.Charge_Amount, AirFare.Description
FROM Flight
JOIN AirFare ON Flight.Flight_ID = AirFare.Flight_ID;
```

Flight_ID	Charge_Amount	Description
AI2014	27341	Standard Single
BA1689	44592	Advanced Purchase
EY1234	42176	Key Fare Single
AA4367	8777	Superpex Return
QR2305	34837	Standard Return
PF4521	4459	Superpex Return
LH9876	27373	Business Return
CT7812	9578	Standard Return

9. Retrieve all flights along with the names and contact details of the passengers on each flight, sorted by flight ID.

```
SELECT Flight.Flight_ID, Passengers.Ps_Name, Passengers.Contacts
FROM Flight
LEFT JOIN Passengers ON Flight.Flight_ID = Passengers.Flight_ID
ORDER BY Flight.Flight_ID;
```

Flight_ID	Ps_Name	Contacts
AA4367	Lakshmi Sharma	7666190505
AI2014	Alfred Schmidt	8080367290
BA1689	NULL	NULL
CT7812	Manan Lakhani	8124579635
EY1234	Akhilesh Joshi	9080369290
EY1234	Rom Solanki	9004568903
EY1234	Ria Gupta	9819414036
LH9876	Khyati Mishra	8082267280
PF4521	NULL	NULL
QR2305	Ankita Ahir	8080367280

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**'

1. Find the names of passengers who have booked flights departing from a specific airport.

```
SELECT Ps_Name
FROM Passengers
WHERE Flight_ID IN (SELECT Flight_ID FROM Flight WHERE Departure = 'New
        York City');
```

Ps_Name
Lakshmi Sharma
Alfred Schmidt
Manan Lakhani

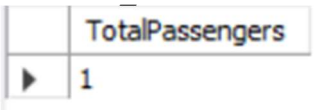
2. Retrieve the flight IDs of all flights departing from a particular city.

```
SELECT Flight_ID
FROM Flight
WHERE Departure IN (SELECT Air_code FROM Airport WHERE City = 'Mumbai');
```

Flight_ID
NULL

3. Find the total number of passengers on flights operated by a specific airline company.

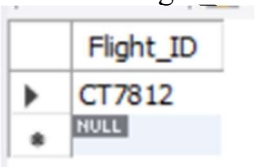
```
SELECT COUNT(*) AS TotalPassengers
FROM Passengers
WHERE Flight_ID IN (SELECT Flight_ID FROM Flight WHERE A_ID IN (SELECT
        A_ID FROM Airplane_type WHERE Company = 'Indigo'));
```



TotalPassengers
1

4. Get the flight IDs where the departure date is after a certain date.

```
SELECT Flight_ID
FROM Flight
WHERE Flight_date > '2021-03-25';
```



Flight_ID
CT7812
NULL

5. Retrieve the names of airports where flights have landed.

```
SELECT Air_Name
FROM Airport
WHERE Air_code IN (SELECT Air_code FROM Can_Land);
```

Air_Name
Adampur Airport
Chhatrapati Shivaji Maharaj International Airport
Copenhagen Airport
Indira Gandhi International Airport
Newark Liberty International Airport
Satwari Airport
John F.Kennnedy International Airport
London City Airport

6. Find the flight IDs where passengers with a specific age have booked tickets.

```
SELECT Flight_ID
FROM Passengers
WHERE Age = (SELECT Age FROM Passengers WHERE Ps_Name = 'Lakshmi
    Sharma');
```

Flight_ID
AI2014
LH9876
AA4367

7. Get the flight IDs where the charge amount exceeds a certain value.

```
SELECT Flight_ID
FROM AirFare
WHERE Charge_Amount > '27341';
```

Flight_ID
EY1234
LH9876
QR2305
BA1689

8. Retrieve the names of passengers who have booked flights to a particular destination.

```
SELECT Ps_Name
FROM Passengers
WHERE Flight_ID IN (SELECT Flight_ID FROM Flight WHERE Arrival = '2021-04-
    04 8:00pm');
```

Ps_Name
Manan Lakhani

9. Find the flight IDs where passengers of a specific gender have booked tickets.

```
SELECT Flight_ID
FROM Passengers
WHERE Sex = 'F';
```

	Flight_ID
▶	QR2305
	LH9876
	AA4367
	EY1234

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.

use airline;

1: Create a view to display the details of flights along with the names of their corresponding departure and arrival cities

```
CREATE VIEW FlightDetails AS
SELECT Flight.Flight_ID, Flight.Departure, Flight.Arrival, Departure_City.City AS
    Departure_City, Arrival_City.City AS Arrival_City
FROM Flight
JOIN Airport AS Departure_City ON Flight.Departure = Departure_City.Air_code
JOIN Airport AS Arrival_City ON Flight.Arrival = Arrival_City.Air_code;
```

2: Retrieve data from the created view to get flight details

```
SELECT * FROM FlightDetails;
```

3: Create a view to display the total charge amount for each flight

```
CREATE VIEW TotalChargeAmount AS
SELECT Flight_ID, SUM(Charge_Amount) AS Total_Charge_Amount
FROM Transactions
GROUP BY Flight_ID;
```

4: Retrieve data from the created view to get the total charge amount for each flight

```
SELECT * FROM TotalChargeAmount;
```

5: Create a view to display the details of passengers along with the names of their corresponding departure and arrival cities

```
CREATE VIEW PassengerDetails AS
SELECT Passengers.Ps_ID, Passengers.Ps_Name, Passengers.Age, Departure_City.City
    AS Departure_City, Arrival_City.City AS Arrival_City
FROM Passengers
JOIN Flight ON Passengers.Flight_ID = Flight.Flight_ID
JOIN Airport AS Departure_City ON Flight.Departure = Departure_City.Air_code
JOIN Airport AS Arrival_City ON Flight.Arrival = Arrival_City.Air_code;
```

6: Retrieve data from the created view to get passenger details

```
SELECT * FROM PassengerDetails;
```

7: Create a view to display the details of flights along with the corresponding airplane type details

```
CREATE VIEW FlightWithAirplaneDetails AS
SELECT Flight.Flight_ID, Flight.Departure, Flight.Arrival, Flight.Flight_date,
    Airplane_type.Capacity, Airplane_type.A_weight, Airplane_type.Company
FROM Flight
JOIN Airplane_type ON Flight.A_ID = Airplane_type.A_ID;
```

8: Retrieve data from the created view to get flight details along with airplane type details

```
SELECT * FROM FlightWithAirplaneDetails;
```

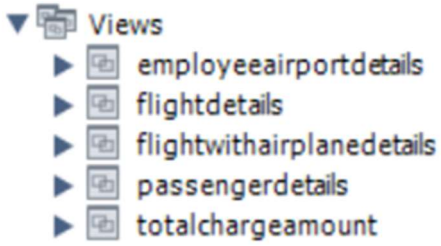
9: Create a view to display the details of employees along with their associated airport details

```
CREATE VIEW EmployeeAirportDetails AS
SELECT Employees.Emp_ID, Employees.E_Name, Employees.Address,
    Employees.Age, Employees.Email_ID, Employees.Contact, Airport.Air_Name,
    Airport.City, Airport.State
FROM Employees
JOIN Airport ON Employees.Air_code = Airport.Air_code;
```

10: Retrieve data from the created view to get employee details along with associated airport details

```
SELECT * FROM EmployeeAirportDetails;
```

Outputs



Flight_ID	Total_Charge_Amount
AA4367	8777
AI2014	27341
BA1689	44592
CT7812	9578
EY1234	42176
LH9876	27373
QR2305	34837

Flight_ID	Departure	Arrival	Flight_date	Capacity	A_weight	Company
AA4367	New York City	2021-03-25 02:15am	2021-03-25	867	387	AirAsia
AI2014	New York City	2021-01-12 10:25pm	2021-01-12	853	394	Indigo
BA1689	New York City	2021-03-02 10:00pm	2021-03-02	770	405	Indigo
CT7812	New York City	2021-04-04 8:00pm	2021-04-04	790	355	TruJet
EY1234	2021-02-10 05:00am	2021-02-10 10:30pm	2021-02-10	790	364	AirIndia
LH9876	2021-02-25 10:15am	2021-02-25 11:00pm	2021-02-25	850	390	SpiceJet
PF4521	2020-12-25 5:00pm	2020-12-25 10:30pm	2020-12-25	835	410	Alliance Air
QR2305	2020-12-26 12:05pm	2020-12-27 12:25pm	2020-12-26	800	380	Vistara

Emp_ID	E_Name	Address	Age	Email_ID	Contact	Air_Name	City	State
1234	Rekha Tiwary	202-Meeta Apt,Yogi Nagar,Mumbai	30	rekha1234@gmail.com	+918530324018	Indira Gandhi International Airport	Delhi	NY
2458	Johny Paul	45-Balaji Apt,Ajit Nagar,Jalandar	32	johnypaul8@gmail.com	+919785425154	Adampur Airport	Jalandhar	NY
3246	John Dsouza	302-Fountain Apt,ElizaBeth Street, Newham	26	john2346@gmail.com	+447911123456	Chhatrapati Shivaji Maharaj International Airport	Mumbai	NY
4521	Nidhi Maroliya	6-Matruchaya Apt,Park Road, Jammu	31	nidhi785@gmail.com	+918211954901	Satwari Airport	Jammu	NY
5123	Lara Jen	28-Mark road,Victoria street,New York City	31	jenlara4@gmail.com	+448000751234	Copenhagen Airport	Copenhagen	NY
7512	Akshay Sharma	Akshay Villa,Queens Street,Copenhagen	20	akshay27@gmail.com	+45886443210	John F.Kennnedy International Airport	New York City	NY
8512	Hafsa Iqmar	1023-Prajwal Apt,Newark	41	hafsa964@gmail.com	6465554468	Newark Liberty International Airport	Newark	NY
9321	Sanjay Rathod	62-Patwa Apt,Pradeep Nagar, Delhi	36	sanjay78@gmail.com	+917504681201	London City Airport	Newham	NY

Verified by

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

Ex. No. 9	ADVANCED SELECT STATEMENTS	Date :

Questions Q1 to Q9 pertain to a database with the following tables.
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).

use airline;

1. Retrieve the names and ages of passengers who booked a flight operated by a specific airline.

```
SELECT P.Ps_Name, P.Age
FROM Passengers P
JOIN Transactions T ON P.Ps_ID = T.Ps_ID
JOIN Flight F ON T.Flight_ID = F.Flight_ID
JOIN Airplane_type AT ON F.A_ID = AT.A_ID
WHERE AT.Company = 'Indigo';
```

Ps_Name	Age
Alfred Schmidt	30
Rom Solanki	60

2. Retrieve the total number of passengers booked on each flight along with the flight details.

```
SELECT F.Flight_ID, COUNT(P.Ps_ID) AS Total_Passengers
FROM Flight F
LEFT JOIN Passengers P ON F.Flight_ID = P.Flight_ID
GROUP BY F.Flight_ID;
```

Flight_ID	Total_Passengers
AA4367	1
AI2014	1
BA1689	0
CT7812	1
EY1234	3
LH9876	1
PF4521	0
QR2305	1

3. List all flights along with their departure and arrival dates.

```
SELECT Flight_ID, Departure, Arrival, Flight_date
FROM Flight;
```

Flight_ID	Departure	Arrival	Flight_date
AA4367	New York City	2021-03-25 02:15am	2021-03-25
AI2014	New York City	2021-01-12 10:25pm	2021-01-12
BA1689	New York City	2021-03-02 10:00pm	2021-03-02
CT7812	New York City	2021-04-04 8:00pm	2021-04-04
EY1234	2021-02-10 05:00am	2021-02-10 10:30pm	2021-02-10
LH9876	2021-02-25 10:15am	2021-02-25 11:00pm	2021-02-25
PF4521	2020-12-25 5:00pm	2020-12-25 10:30pm	2020-12-25
QR2305	2020-12-26 12:05pm	2020-12-27 12:25pm	2020-12-26

4. Find the average age of passengers traveling on each flight.

```
SELECT F.Flight_ID, AVG(P.Age) AS Average_Age
FROM Flight F
JOIN Passengers P ON F.Flight_ID = P.Flight_ID
GROUP BY F.Flight_ID;
```

Flight_ID	Average_Age
AI2014	30.0000
QR2305	26.0000
LH9876	30.0000
EY1234	41.0000
AA4367	30.0000
CT7812	45.0000

5. Retrieve the details of all routes that a particular airplane type can operate.

```
SELECT R.*
FROM Route R
JOIN Travels_on T ON R.Route_ID = T.Route_ID
JOIN Flight F ON T.Flight_ID = F.Flight_ID
WHERE F.A_ID = 777;
```

Route_ID	Take_Off_point	Destination	R_type
157306	NewJersey	Mumbai	2Hr Break

6. List all flights along with the corresponding fare charges.

```
SELECT F.Flight_ID, AF.Charge_Amount
FROM Flight F
JOIN AirFare AF ON F.Flight_ID = AF.Flight_ID;
```

Flight_ID	Charge_Amount
AI2014	27341
BA1689	44592
EY1234	42176
AA4367	8777
QR2305	34837
PF4521	4459
LH9876	27373
CT7812	9578

7. Find the total revenue generated from each flight.

```
SELECT F.Flight_ID, SUM(AF.Charge_Amount) AS Total_Revenue
FROM Flight F
JOIN AirFare AF ON F.Flight_ID = AF.Flight_ID
GROUP BY F.Flight_ID;
```

Flight_ID	Total_Revenue
AA4367	8777
AI2014	27341
BA1689	44592
CT7812	9578
EY1234	42176
LH9876	27373
PF4521	4459
QR2305	34837

8. Retrieve the names of all passengers who booked a flight departing from a specific state.

```
SELECT DISTINCT P.Ps_Name
FROM Passengers P
JOIN Flight F ON P.Flight_ID = F.Flight_ID
JOIN Airport A ON F.Departure = A.Air_code
WHERE A.State = 'NY';
```

9. List all flights along with the company operating each flight.

```
SELECT F.Flight_ID, AT.Company
FROM Flight F
JOIN Airplane_type AT ON F.A_ID = AT.A_ID;
```

Flight_ID	Company
AI2014	Indigo
BA1689	Indigo
EY1234	AirIndia
AA4367	AirAsia
QR2305	Vistara
PF4521	Alliance Air
LH9876	SpiceJet
CT7812	TruJet

Verified by

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

Ex. No. 10	ADDITIONAL QUERIES	Date :

Solve the queries Q1 to Q10 with Additional queries

1. List all flights departing from a specific airport on a given date.

```
SELECT Flight_ID, Departure, Arrival, Flight_date
FROM Flight
WHERE Departure = 'New York City' AND Flight_date = '2021-03-25';
```

Flight_ID	Departure	Arrival	Flight_date
AA4367	New York City	2021-03-25 02:15am	2021-03-25
NULL	NULL	NULL	NULL

2. Find the total number of passengers on a particular flight.

```
SELECT COUNT(*) AS Total_Passengers
FROM Passengers
WHERE Flight_ID = 'AA4367';
```

Total_Passengers
1

3. Retrieve the details of passengers traveling on a specific route.

```
SELECT Ps_Name, Age, Sex
FROM Passengers
JOIN Flight ON Passengers.Flight_ID = Flight.Flight_ID
JOIN Travels_on ON Flight.Flight_ID = Travels_on.Flight_ID
WHERE Travels_on.Route_ID = '894521';
```

Ps_Name	Age	Sex
Lakshmi Sharma	30	F

4. List all flights operated by a certain company.

```
SELECT Flight_ID, Departure, Arrival, Company
FROM Flight
JOIN Airplane_type ON Flight.A_ID = Airplane_type.A_ID
WHERE Company = 'Indigo';
```

Flight_ID	Departure	Arrival	Company
AI2014	New York City	2021-01-12 10:25pm	Indigo
BA1689	New York City	2021-03-02 10:00pm	Indigo

5. Get the average age of passengers on a particular flight.

```
SELECT AVG(Age) AS Average_Age
FROM Passengers
WHERE Flight_ID = 'EY1234';
```

Average_Age
41.0000

6. Calculate the average charge amount per passenger for a specific flight.

```
SELECT AVG(Charge_Amount) AS Average_Charge_Per_Passenger
FROM AirFare
WHERE Flight_ID = 'EY1234';
```

Average_Charge_Per_Passenger
42176.0000

7. Retrieve the details of employees working at a specific airport.

```
SELECT Emp_ID, E_Name, Address, Age, Email_ID, Contact
FROM Employees
WHERE Air_code = 'DEL';
```

Emp_ID	E_Name	Address	Age	Email_ID	Contact
1234	Rekha Tiwary	202-Meeta Apt,Yogi Nagar,Mumbai	30	rekha1234@gmail.com	+918530324018
NULL	NULL	NULL	NULL	NULL	NULL

8. Calculate the total number of flights departed and arrived at each airport.

```
SELECT Airport.Air_code, Airport.Air_Name,
    COUNT(DISTINCT Departure.Flight_ID) AS Departure_Count,
    COUNT(DISTINCT Arrival.Flight_ID) AS Arrival_Count
FROM Airport
LEFT JOIN Flight AS Departure ON Airport.Air_code = Departure.Departure
LEFT JOIN Flight AS Arrival ON Airport.Air_code = Arrival.Arrival
GROUP BY Airport.Air_code, Airport.Air_Name;
```

Air_code	Air_Name	Departure_Count	Arrival_Count
AIP	Adampur Airport	0	0
BOM	Chhatrapati Shivaji Maharaj International Airport	0	0
CDG	Charles de Gaulle Airport	0	0
CPH	Copenhagen Airport	0	0
DEL	Indira Gandhi International Airport	0	0
EWR	Newark Liberty International Airport	0	0
IXJ	Satwari Airport	0	0
JFK	John F.Kennnedy International Airport	0	0
LAX	Los Angeles International Airport	0	0
LCY	London City Airport	0	0
LHR	London Heathrow Airport	0	0
YYZ	Toronto Pearson International Airport	0	0

9. Get the total revenue generated from a specific flight.

```
SELECT SUM(Charge_Amount) AS Total_Revenue
FROM AirFare
WHERE Flight_ID = 'AI2014';
```

	Total_Revenue
▶	27341

10. List all flights with the corresponding airplane capacity and company name.

```
SELECT Flight_ID, Capacity, Company, Flight_date
FROM Flight
JOIN Airplane_type ON Flight.A_ID = Airplane_type.A_ID;
```

	Flight_ID	Capacity	Company	Flight_date
▶	AA4367	867	AirAsia	2021-03-25
	AI2014	853	Indigo	2021-01-12
	BA1689	770	Indigo	2021-03-02
	CT7812	790	TruJet	2021-04-04
	EY1234	790	AirIndia	2021-02-10
	LH9876	850	SpiceJet	2021-02-25
	PF4521	835	Alliance Air	2020-12-25
	QR2305	800	Vistara	2020-12-26

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

```
DELIMITER //

CREATE PROCEDURE FindMaxOfThreeNumbers()
BEGIN
   DECLARE num1 INT DEFAULT 10;
   DECLARE num2 INT DEFAULT 20;
   DECLARE num3 INT DEFAULT 15;
   DECLARE max_num INT;

   IF num1 >= num2 AND num1 >= num3 THEN
      SET max_num = num1;
   ELSEIF num2 >= num1 AND num2 >= num3 THEN
      SET max_num = num2;
   ELSE
      SET max_num = num3;
   END IF;

   SELECT CONCAT('The maximum number among ', num1, ', ', num2, ', and ', num3, ' is: ', max_num)
AS Result;
END//

DELIMITER ;

CALL FindMaxOfThreeNumbers();
```

t.putlin

	Result
▶	The maximum number among 10, 20, and 15 is: 20

m
s
_
o
u
t
p
u

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

```
DELIMITER //

CREATE PROCEDURE SumOddNumbersUpTo100()
BEGIN
   DECLARE total_sum INT DEFAULT 0;
   DECLARE i INT DEFAULT 1;

   WHILE i <= 100 DO
      IF i % 2 <> 0 THEN
         SET total_sum = total_sum + i;
      END IF;
      SET i = i + 1;
   END WHILE;

   SELECT CONCAT('The sum of odd numbers up to 100 is: ', total_sum) AS Result;
END//

DELIMITER ;

CALL SumOddNumbersUpTo100();
```

	Result
▶	The sum of odd numbers up to 100 is: 2500

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%

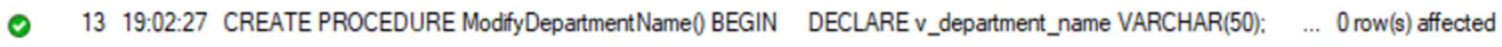
otherwise increase salary by adding commission amount to the salary.

```
DELIMITER //

CREATE PROCEDURE UpdateEmployeeSalary()
BEGIN
   DECLARE v_salary DECIMAL(10, 2);
   DECLARE v_new_salary DECIMAL(10, 2);
   DECLARE v_commission DECIMAL(10, 2) DEFAULT 500; -- Assuming commission amount

   -- Retrieve the current salary of the employee with empno = 7369
   SELECT sal INTO v_salary FROM emp WHERE empno = 7369;
   -- Update the salary based on conditions
   IF v_salary < 2500 THEN
      SET v_new_salary := v_salary * 1.25; -- Increase salary by 25%
   ELSEIF v_salary >= 2500 AND v_salary <= 5000 THEN
      SET v_new_salary := v_salary * 1.20; -- Increase salary by 20%
   ELSE

      SET v_new_salary := v_salary + v_commission; -- Increase salary by adding commission
   END IF;
   -- Update the salary in the database
   UPDATE emp SET sal = v_new_salary WHERE empno = 7369;
   -- Display the updated salary
   SELECT CONCAT('Updated salary for employee 7369: ', v_new_salary) AS Result;
END//
```

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

```
DELIMITER //

CREATE PROCEDURE ModifyDepartmentName()
BEGIN
```

```
    DECLARE v_department_name VARCHAR(50);

    -- Retrieve the current department name for department number 71
    SELECT dname INTO v_department_name FROM dept WHERE deptno = 71;

    -- Check if the department name is not 'HRD'
    IF v_department_name != 'HRD' THEN
        -- Update the department name to 'HRD'
        UPDATE dept SET dname = 'HRD' WHERE deptno = 71;
        SELECT 'Department name for department 71 has been updated to HRD.' AS Result;
    ELSE
        SELECT 'Department name for department 71 is already HRD. No changes made.' AS Result;
    END IF;
END//

DELIMITER ;
```



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.

```
DELIMITER //

CREATE PROCEDURE UpdateSalariesInDept20()
BEGIN
   DECLARE v_total_updated INT DEFAULT 0;

   -- Update salaries of employees in department 20 by 15%
   UPDATE employees
   SET salary = salary * 1.15
   WHERE deptno = 20;

   -- Get the count of updated employees
   SELECT COUNT(*) INTO v_total_updated
   FROM employees
   WHERE deptno = 20;

   -- Display appropriate message
   IF v_total_updated = 0 THEN
      SELECT 'None of the salaries were updated.' AS Result;
   ELSE
      SELECT CONCAT('Total number of employees whose salaries were updated: ', v_total_updated)
AS Result;
   END IF;
END//

DELIMITER ;
```

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

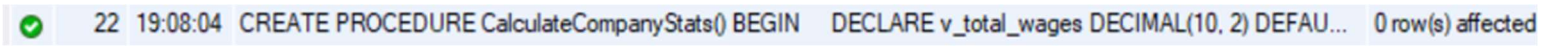
%ISOPEN
TRUE, if the cursor is already open in the program.
Ex. Cursor_name%ISOPEN

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.

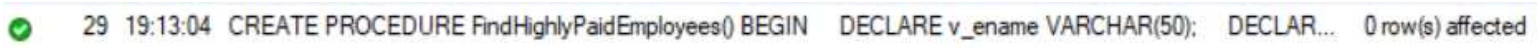
```
DELIMITER //

CREATE PROCEDURE CalculateCompanyStats()
BEGIN
DECLARE v_total_wages DECIMAL(10, 2) DEFAULT 0;
DECLARE v_highly_paid_employees INT DEFAULT 0;
DECLARE v_commission_higher_than_salary INT DEFAULT 0;
```

```
    -- Total wages of the company
    SELECT SUM(sal + IFNULL(comm, 0))
    INTO v_total_wages
    FROM emp;

    -- Total number of highly paid employees
    SELECT COUNT(*)
    INTO v_highly_paid_employees
    FROM emp
    WHERE sal > 2000;

    -- Total number of employees who get commission higher than their salary
    SELECT COUNT(*)
    INTO v_commission_higher_than_salary
    FROM emp
    WHERE comm > sal;

    -- Display results
    SELECT CONCAT('a. Total wages of the company: ', v_total_wages) AS Result;
    SELECT CONCAT('b. Total number of highly paid employees: ',
v_highly_paid_employees) AS Result;
    SELECT CONCAT('c. Total number of employees with commission higher than
salary: ', v_commission_higher_than_salary) AS Result;
    END//
    DELIMITER ;
```



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

```
DELIMITER //

CREATE PROCEDURE FindHighlyPaidEmployees()
BEGIN
   DECLARE v_ename VARCHAR(50);
   DECLARE v_sal DECIMAL(10, 2);
   DECLARE no_more_rows BOOLEAN DEFAULT FALSE;

   -- Declare cursor to fetch data
   DECLARE c_high_paid_employees CURSOR FOR
      SELECT ename, sal
      FROM emp
      WHERE sal > 2000
      ORDER BY sal DESC
      LIMIT 5;

   -- Declare CONTINUE HANDLER for not found condition
   DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET no_more_rows = TRUE;

   -- Open the cursor
   OPEN c_high_paid_employees;

   -- Loop to fetch and display data
   employee_loop: LOOP
      FETCH c_high_paid_employees INTO v_ename, v_sal;
      IF no_more_rows THEN
         LEAVE employee_loop;
      END IF;

      -- Output the name and salary of the employee
      SELECT CONCAT('Name: ', v_ename, ', Salary: ', v_sal) AS Result;
   END LOOP;

   -- Close the cursor
   CLOSE c_high_paid_employees;
END//

DELIMITER ;
```



Cursor for loop

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

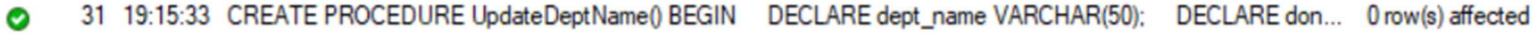
Ex.

```
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

```
DELIMITER //

CREATE PROCEDURE UpdateDeptName()
BEGIN
  DECLARE dept_name VARCHAR(50);
  DECLARE done BOOLEAN DEFAULT FALSE; -- Declare done variable before cursor declaration

  -- Declare cursor to fetch department name
  DECLARE cur_dept CURSOR FOR
    SELECT dname FROM dept WHERE deptno = 71;

  -- Declare CONTINUE HANDLER for not found condition
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET done = TRUE;

  -- Open the cursor
  OPEN cur_dept;

  -- Start the loop
  dept_loop: LOOP
    FETCH cur_dept INTO dept_name;
    IF done THEN
      LEAVE dept_loop;
    END IF;

    -- Check if department name is not 'HRD'
    IF dept_name != 'HRD' THEN
      -- Update department name to 'HRD'
      UPDATE dept SET dname = 'HRD' WHERE deptno = 71;
      COMMIT;
      -- Output the result
      SELECT 'Department name updated to HRD for department 71.' AS Result;
    ELSE
      -- Output message if department name is already 'HRD'
      SELECT 'Department name is already HRD for department 71. No changes made.' AS Result;
    END IF;
  END LOOP;
```

```
    -- Close the cursor
    CLOSE cur_dept;
END//

DELIMITER ;
```



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 job_history table
CREATE TABLE job_history (
   empno INT,
   ename VARCHAR(50),
   job VARCHAR(50),
   pro_date DATE
);

-- Create or replace trigger job_history_trigger
CREATE TRIGGER job_history_trigger
BEFORE UPDATE ON emp
FOR EACH ROW
BEGIN
   IF OLD.job != NEW.job THEN
      INSERT INTO job_history (empno, ename, job, pro_date)
      VALUES (OLD.empno, OLD.ename, OLD.job, SYSDATE());
   END IF;
END;
```

Table 'job_history' already exists

Q12) 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.

```
DELIMITER //

CREATE OR REPLACE TRIGGER dept_adr
AFTER DELETE ON dept
FOR EACH ROW
BEGIN
INSERT INTO deptold VALUES (OLD.deptno, OLD.dname, OLD.loc);
END;
//DELIMITER ;
```

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> ALTER TRIGGER job_history_trigger DISABLE;

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

SQL> ALTER TABLE emp DISABLE ALL TRIGGERS;

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.

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