PRACTICAL 1

Fundamentals of Database Systems

A **database** is a collection of information that is organized so that it can be easily accessed, managed and updated.
Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.
It is the collection of schemas, tables, queries, reports, views, and other objects.

A **database management system** (DBMS) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, MariaDB, Microsoft SQL Server, Oracle, Sybase, SAP HANA, MemSQL, SQLite and IBM DB2. Database management systems are often classified according to the database model that they support, the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language.

Types of databases :

- Hierarchical databases.
- Network databases.
- Relational databases.
- Entity-Relationship Databases
- Object-oriented databases

- Hierarchical Databases - A hierarchical database model is a data model in which the data is organized into a tree-like structure. The data is stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value. The entity type of a record defines which fields the record contains.
- Networked Databases - A network database is a type of database model wherein multiple member records or files can be linked to multiple owner files and vice versa. The

model can be viewed as an upside-down tree where each member information is the branch linked to the owner, which is the bottom of the tree.

- Relational Databases - A network database is a type of database model wherein multiple member records or files can be linked to multiple owner files and vice versa. The model can be viewed as an upside-down tree where each member information is the branch linked to the owner, which is the bottom of the tree. They use Structured Query Language SQL, which is a standard in the databases.
- Entity-Relationship Databases - An entity–relationship model (ER model) describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types.
- Object-Oriented Databases- An object-oriented database management system (OODBMS) is a database management system that supports the creation and modeling of data as objects. OODBMS also includes support for classes of objects and the inheritance of class properties, and incorporates methods, subclasses and their objects. Most of the object databases also offer some kind of query language, permitting objects to be found through a declarative programming approach.

Standardized language SQL

Structured Query Language is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). In comparison to older read/write APIs like ISAM or VSAM, SQL offers two main advantages: First, it introduced the concept of accessing many records with one single command, and second, it eliminates the need to specify *how* to reach a record, e.g.: with or without an index.

Types of SQL

- SQL is Structured Query Language is a database computer language designed for managing data in relational database management systems (RDBMS).

- PL/SQL (Procedural Language/Structured Query Language) is Oracle Corporation's procedural extension for SQL and the Oracle relational database.

- PostgreSQL is an object-relational database management system (ORDBMS). It is released under a BSD-style license and is thus free software. As with many other open-source programs, PostgreSQL is not controlled by any single company, but has a global community of developers and companies to develop it.

- SQLite is an ACID-compliant embedded relational database management system contained in a relatively small (~225 KB) C programming library. The source code for SQLite is in the public domain.

- MySQL is a relational database management system (RDBMS) which has more than 6 million installations. MySQL stands for "My Structured Query Language". The program runs as a server providing multi-user access to a number of databases.

Types of SQL Statements

The main categories are
- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)
- Transactional Control Commands

Data Definition Language, DDL, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table.
Some of the most fundamental DDL commands :
CREATE TABLE / INDEX / VIEW
ALTER TABLE / INDEX / VIEW
DROP TABLE / INDEX / VIEW

Data Manipulation Language, DML, is the part of SQL used to manipulate data within objects of a relational database.
The basic DML commands:
INSERT
UPDATE
DELETE
SELECT

Data Control Commands in SQL allow you to control access to data within the database. These DCL commands are normally used to create objects related to user access and also control the distribution of privileges among users.
Some data control commands are as follows:
ALTER PASSWORD
GRANT
REVOKE
CREATE SYNONYM

Transactional Control Commands are commands that allow the user to manage database transactions.
Some transaction control statements are as follows:
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

DQL STATEMENTS

DISPLAY THE CONTENTS OF THE TABLE

-SELECT COMMAND SELECTS THE ATTRIBUTES OF THE TABLE TO BE DISPLAYED

SELECT * FROM student95;

Results Explain Describe Saved SQL History

ROLL_NO	NAME	DEPTT	DOB	MARKS
UE163096	SHUBHAM GUPTA	CSE	01/17/1998	97
UE163098	SHUBHKIRTI SHARMA	CSE	11/11/1111	99
UE163102	SOMYADEEP DAS	CSE	02/12/1998	96
UE163064	NAMAN AGGARWAL	CSE	02/12/1998	96
UE163100	SINDHIYA ARYA	CSE	08/19/1998	95
UE163074	PRATIK JOSHI	CSE	08/19/1998	98
UE163095	shivank	cse	08/26/1998	96
UE163092	SHASWAT	CSE	08/10/1997	95

8 rows returned in 0.01 seconds Download

DESCRIBE TABLE COMMAND

-THE DESC COMMAND DESCRIBES THE TABLE IT DISPLAYS THE DETAILS OF COLUMNS ALONG WITH DATA TYPES AND SIZE

DESC student01;

Object Type **TABLE** Object **STUDENT01**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENT01	ROLL_NO	CHAR	8	-	-	1	-	-	-
	NAME	CHAR	20	-	-	-	✓	-	-
	DEPTT	CHAR	3	-	-	-	✓	-	-
	DOB	DATE	7	-	-	-	✓	-	-
	MARKS	NUMBER	22	-	-	-	✓	-	-

1 - 5

DDL STATEMENTS

CREATE TABLE COMMAND

-THS COMMMAND IS USED TO CREATE A TABLE WITH THE DESCRIBED ATTRIBUTES IN THE CURRENT DATABASE

CREATE TABLE student95 (roll_no CHAR(8) PRIMARY KEY , name CHAR(20), deptt CHAR(3), dob DATE , marks NUMBER);

```
Table created.
```

0.08 seconds

CREATE ANOTHER TABLE FROM THE EXISTING TABLE

-A TABLE CAN BE CREATED FROM ANOTHER TABLE USING CREATE TABLE AS STATEMENT

CREATE TABLE student9955 AS SELECT * FROM student95;

```
Table created.
```

0.01 seconds

ALTER TABLE (ADD COLUMN)

-ALTER TABLE STATEMENT IS USED TO ALTER THE SCHEMA OF THE TABLE
-IT CAN BE OF FOUR TYPES
-ADDING A NEW COLUMN IN THE TABLE

ALTER TABLE student9955 ADD (contact_no VARCHAR2(10));

```
Table altered.
```

0.02 seconds

ALTER TABLE (MODIFY COLUMN)

-MODIFYING AN EXISTING ATTRIBUTE

ALTER TABLE student9955 MODIFY (contact_no NUMBER(10));

```
Table altered.
```

0.01 seconds

ALTER TABLE (RENAME COLUMN)

-RENAMING A COLUMN

ALTER TABLE student9955 RENAME COLUMN contact_no TO CONTACT ;

```
Table altered.
```

0.02 seconds

ALTER TABLE (DROP COLUMN)

-DELETING OR DROPPING A TABLE – THIS DELETES ALL THE DATA IN THE COLUMN ALONG WITH THE COLUMN ITSELF

ALTER TABLE student9955 DROP COLUMN contact ;

```
Table altered.
```

0.03 seconds

ALTER TABLE (RENAME TABLE)

-THE RENAME COMMAND ALONG WITH ALTER TABLE CAN BE USED TO RENAME THE TABLE LIKE HERE THE TABLE IS RENAMED FROM student95 TO student01

ALTER TABLE student95 RENAME TO student01;

```
Table altered.
```

0.00 seconds

DROP COLUMN

ALTER TABLE student9955 DROP COLUMN contact ;

```
Table altered.
```

0.03 seconds

RENAME COLUMN

ALTER TABLE student95 RENAME COLUMN contact_no TO CONTACT ;

```
Table altered.
```

0.02 seconds

TRUNCATE THE TABLE

-TRUNCATE STATEMENT TRUNCATES THE TABLE (DELETES ALL THE TUPLES PRESENT IN THE TABLE)

TRUNCATE TABLE student95;

```
Table truncated.
```

0.03 seconds

DROP TABLE COMMAND

-DROP COMMAND DROPS THE TABLE (THIS MEANS THAT THE TALBE'S DATA ALONG WITH SCHEMA(METADATA) IS DELETED)

DROP TABLE student9955;

```
Table dropped.
```

0.02 seconds

PRACTICAL 2

DML STATEMENTS

INSERTING VALUES IN THE TABLE (METHOD 1)

-INSERT INTO COMMAND INSERTS VALUE INTO THE TABLE IN TWO WAYS
-IN FIRST WAY DIRECTLY GINVING VALUES IN THE COMMAND ITSELF

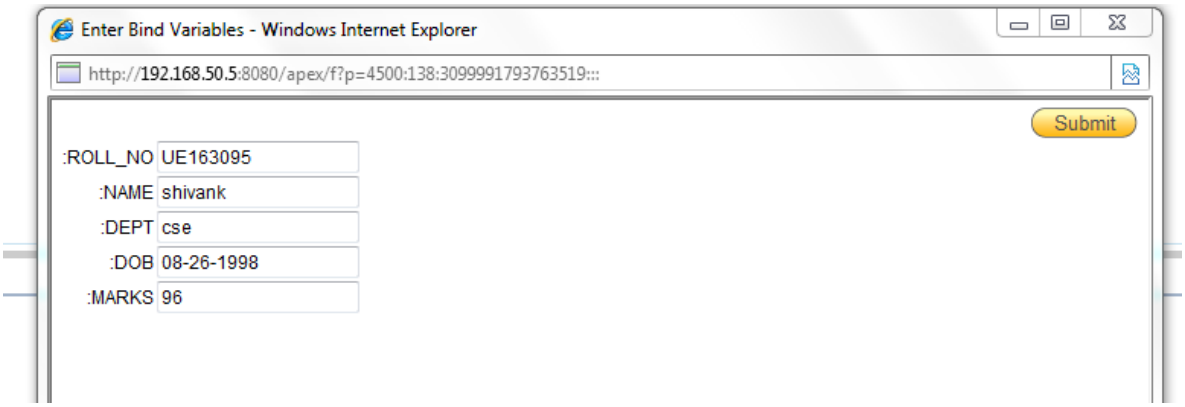
INSERT INTO student95 VALUES ('UE163095', 'SHIVANK', 'CSE', '08-26-1998', 94);

```
1 row(s) inserted.
```

0.01 seconds

INSERTING VALUES IN THE TABLE (METHOD 2)

-IN THIS METHOD THE SAME QUERY CAN BE USED FOR MULTIPLE ENTRIES INTO THE TABLE AS THE VALUES ARE ENTERED IN A DIALOG BOX THAT APPEARS



INSERT INTO student95 VALUES (:ROLLNO, :NAME, :DEPT,:DOB,:MARKS);

```
1 row(s) inserted.
```

0.01 seconds

UPDATING EXISTING VALUES IN THE TABLE

UPDATE student01 SET ROLL_NO='UE163096' WHERE ROLL_NO='UE163095';

```
1 row(s) updated.
```

0.00 seconds

DELETING THE TUPLES FROM THE TABLE

-DELETE TABLE DELETES THE ROWS FROM THE TABLE THE DELETION OF THE ROWS CAN BE SELECTIVE ALSO

DELETE FROM student9955;

```
8 row(s) deleted.
```

0.01 seconds

TCL STATEMENTS

CREATING SAVEPOINT

-THE SAVEPOINT IS THE POINT IN THE DBMS UPTO WHICH WE CAN RECALL ALL THE CHANGES MADE

```
BEGIN
SAVEPOINT A;
END;
```

```
Statement processed.

0.00 seconds
```

ROLLBACK TO A SAVEPOINT

-THE ROLL BACK COMMAND UNDOES ALL THE CHANGES MADE AFTER THE SAVEPOINT AND TAKES THE DATABASE TO THE STAGE WHERE THE SAVEPOINT WAS CREATED

```
ROLLBACK TO A;
```

```
Statement processed.

0.00 seconds
```

COMMIT CHANGES

-THE COMMIT COMMAND COMMITS ALL THE CHANGES TO THE DATABASE AND THOSE CHANGES THAT ARE COMMITTED CANNOT BE REVERTED

```
COMMIT;
```

```
Statement processed.

0.00 seconds
```

DCL STATEMENTS

SQL GRANT REVOKE COMMANDS

-DCL COMMANDS ARE USED TO ENFORCE DATABASE SECURITY IN A MULTIPLE USER DATABASE ENVIRONMENT. TWO TYPES OF DCL COMMANDS ARE GRANT AND REVOKE. ONLY DATABASE ADMINISTRATOR'S OR OWNER'S OF THE DATABASE OBJECT CAN PROVIDE/REMOVE PRIVILEGES ON A DATABASE OBJECT.

SQL GRANT Command

-SQL GRANT IS A COMMAND USED TO PROVIDE ACCESS OR PRIVILEGES ON THE DATABASE OBJECTS TO THE USERS.

SYNTAX

GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];

GRANT SELECT, INSERT, UPDATE, DELETE ON dept To students;

```
Statement processed.


0.03 seconds
```

SQL REVOKE Command:

-THE REVOKE COMMAND REMOVES USER ACCESS RIGHTS OR PRIVILEGES TO THE DATABASE OBJECTS.

SYNTAX

REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}

REVOKE ALL ON dept FROM students;

```
Statement processed.


0.04 seconds
```

PRACTICAL 3

CONSTRAINTS

- Constraints are rules and restrictions applied on a column or a table such that unwanted data can't be inserted into tables. This ensures the accuracy and reliability of the data in the database. We can create constraints on single or multiple columns of any table. Constraints maintain the data integrity and accuracy in the table.

CONSTRAINTS CAN BE CLASSIFIED INTO THE FOLLOWING TWO TYPES.

COLUMN TYPES CONSTRAINTS

-DEFINITIONS OF THESE TYPES OF CONSTRAINTS IS GIVEN WHEN THE TABLE IS CREATED.

```
Create Table My_Constraint
(
ID int NOT NULL,
Salary int CHECK(Salary>5000)
)
```

TABLE TYPES CONSTRAINTS

-DEFINITIONS OF THESE TYPES OF CONSTRAINTS IS GIVEN AFTER THE CREATION OF THE TABLE USING THE ALTER COMMAND.

```
Alter Table My_Constraint
Add constraint Check_Constraint Check(Age>65)
```

THE FOLLOWING CONSTRAINTS ARE COMMONLY USED IN SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified

TABLE CREATED FOR TABLE LEVEL CONSTRAINTS CONSTRAINTS

```
CREATE TABLE emp_95(
emp_id VARCHAR2(8),
emp_fname VARCHAR2(15),
emp_lname VARCHAR2(15),
emp_sal NUMBER(10),
contact_no NUMBER(10),
emp_ta NUMBER(10),
emp_da NUMBER(5,2),
emp_td NUMBER(5,2) );
```

```
Table created.
```

0.13 seconds

DESC employee95;

Object Type **TABLE** Object **EMPLOYEE95**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEE95	EMP_ID	VARCHAR2	8	-	-	-	✓	-	-
	EMP_FNAME	VARCHAR2	15	-	-	-	✓	-	-
	EMP_LNAME	VARCHAR2	15	-	-	-	✓	-	-
	EMP_SAL	NUMBER	-	10	0	-	✓	-	-
	CONTACT_NO	NUMBER	-	10	0	-	✓	-	-
	EMP_TA	NUMBER	-	10	0	-	✓	-	-
	EMP_DA	NUMBER	-	5	2	-	✓	-	-
	EMP_TD	NUMBER	-	5	2	-	✓	-	-
									1 - 8

INPUT OUTPUT CONSTRAINTS

ADDING PRIMARY KEY CONSTRAINT USING ALTER TABLE COMMAND

-Primary key is the key that uniquely identifies all of the rows of the table, it cannot be null or same for two rows like two children in same class cannot have same roll number

-ADDING PRIMARY KEY CONSTRAINT TO THE TABLE USING ALTER TABLE STATEMENT

ALTER TABLE emp_95 ADD CONSTRAINT emp_95_emp_id_pk PRIMARY KEY(emp_id);

```
Table altered.
```

0.03 seconds

CREATING ANOTHER TABLE FOR FOREIGN KEY

-A foreign key is a column or a combination of columns whose values match a primary key in a different table. A foreign key is a key used to link two tables together. This sometimes also called as a referencing key.

CREATE TABLE EMP_CONTACT_95 (EMP_ID VARCHAR2(8) PRIMARY KEY, CONTACT_NO NUMBER(10));

```
Table created.
```

0.03 seconds

ADDING FOREIGN KEY CONSTRAINT USING ALTER TABLE COMMAND

-ADDING FOREIGN KEY CONSRAINT CAN BE DONE IN TWO WAYS EITHER AT THE TIME OF CREATION OF THE TABLE OR AFTER THAT USING THE ALTER TABLE STATEMENT.

ALTER TABLE EMP_CONTACT_95 ADD FOREIGN KEY (EMP_ID) REFERENCES EMP_95(EMP_ID);

```
Table altered.
```

0.02 seconds

NOT NULL CONSTRAINT

-NOT NULL CONSTRAINT MAKES SURE THAT A COLUMN DOES NOT HOLD NULL VALUE. WHEN WE DON'T PROVIDE VALUE FOR A PARTICULAR COLUMN WHILE INSERTING A RECORD INTO A TABLE, IT TAKES NULL VALUE BY DEFAULT. BY SPECIFYING NULL CONSTRAINT, WE CAN BE SURE THAT A PARTICULAR COLUMN(S) CANNOT HAVE NULL VALUES.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

```
Table created.

0.02 seconds
```

DEFAULT CONSTRAINT

-THE DEFAULT CONSTRAINT PROVIDES A DEFAULT VALUE TO A COLUMN WHEN THERE IS NO VALUE PROVIDED WHILE INSERTING A RECORD INTO A TABLE.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT  DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

```
Table created.

0.25 seconds
```

UNIQUE CONSTRAINT

-UNIQUE CONSTRAINT ENFORCES A COLUMN OR SET OF COLUMNS TO HAVE UNIQUE VALUES. IF A COLUMN HAS A UNIQUE CONSTRAINT, IT MEANS THAT PARTICULAR COLUMN CANNOT HAVE DUPLICATE VALUES IN A TABLE.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

```
Table created.


0.03 seconds
```

BUISNESS CONSTRAINTS

TABLE LEVEL

CHECKING THAT IF SALARY IS >10000

```
ALTER TABLE EMP_95 ADD CONSTRAINT ck_emp_95_min_salary
 CHECK (emp_sal >= 10000);
```

```
Table altered.

0.01 seconds
```

COLUMN LEVEL

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL CHECK(ROLL_NO >1000) ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

```
Table created.

0.02 seconds
```

PRACTICAL 4

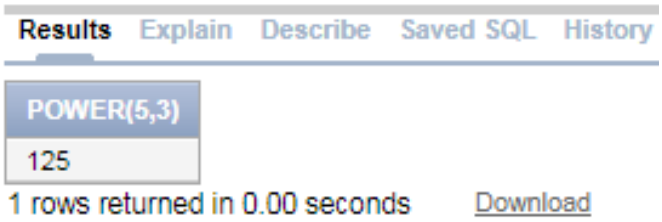
DUAL TABLE

The DUAL is special one row, one column table present by default in all Oracle databases. The owner of DUAL is SYS (SYS owns the data dictionary, therefore DUAL is part of the data dictionary.) but DUAL can be accessed by every user. The table has a single VARCHAR2(1) column called DUMMY that has a value of 'X'.
The DUAL table was created by Charles Weiss of Oracle corporation to provide a table for joining in internal views.

USING THE FUNCTION POWER ON DUAL TABLE

-THE FUNCTION POWER() TAKES TWO INPUTS 1ST THE NUMBER 2ND THE POWER AND THEN IT RETURNS THE VALUE OF IT. IT IS A MATHEMATICAL FUNCTION

SELECT POWER(5,3) FROM DUAL;



ARITHEMETIC OPERATIONS

USING ARITHEMATIC OPERATIONS ON DUAL TABLE

+ : ADDITION

SELECT 5+8 FROM DUAL;

- : SUBTRACTION

SELECT 8-3 FROM DUAL;

Results

8-3
5

* : MULTIPLICATION

SELECT 9*2 FROM DUAL;

Results

9*2
18

/ : DIVISION

SELECT 6/2 FROM DUAL;

Results

6/2
3

CALCULATING TOTAL SALARY BY USING ARITHEMETIC OPERATORS (+, -, *, /)

-ARITHEMETIC OPERATORS SUCH AS (+, -, *, /) ARE USED TO CALCULATE OR TO MANIPULATE THE MATHEMATICAL DATA LIKE CALCULATING THE PERCENTAGE OF MARKS OBTAINED

SELECT EMP_FNAME||' '||EMP_LNAME "NAME", EMP_ID,
EMP_SAL + (EMP_SAL*EMP_DA/100) + EMP_TA – (EMP_SAL*EMP_TD/100) "TOTAL SALARY",
CONTACT_NO FROM EMP_95 ;

Results Explain Describe Saved SQL History

NAME	EMP_ID	TOTAL SALARY	CONTACT_NO
PRATIK JOSHI	EPN12347	775000	987654323
SHIVANK BALI	EPN12345	1055000	9874563210
SHASWAT SINGH	EPN12346	1005000	9874563212
SHUBHKIRTI SHARMA	EPN12348	54000	987654328

4 rows returned in 0.01 seconds Download

INSERTION FROM ANOTHER TABLE

-INSERTING INTO A TABLE FROM ANOTHER TABLE IS POSSIBLE THROUGH INSERT INTO SELECT STATEMENT BUT BOTH THE TABLES MUST BE OF SAME COLUMNS WITH SAME DATATYPES

INSERT INTO student95 SELECT * FROM student01;

```
2 row(s) inserted.
```

0.04 seconds

ALIASING A TABLE COLUMN

SELECT ROLL_NO AS ROLL_NUMBER , NAME AS STUDENT_NAME FROM STUDENT95;

ROLL_NUMBER	STUDENT_NAME
UE163098	Shubhkirti
UE163095	Shivank Bali

2 rows returned in 0.00 seconds Download

ALIASING A TABLE

SELECT * FROM STUDENT95 S WHERE S.ROLL_NO = 'UE163095';

ROLL_NO	NAME	DEPTT	DOB	MARKS
UE163095	Shivank Bali	CSE	08/26/1998	94

1 rows returned in 0.00 seconds Download

PRACTICAL 5

BUILT IN FUNCTOINS

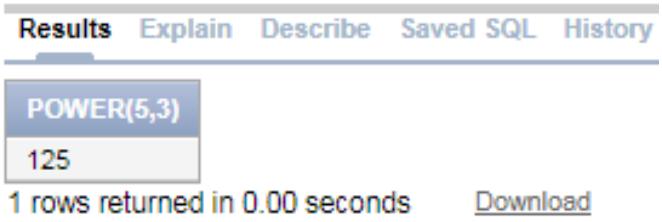
SINGLE ROW FUNCTIONS

-NUMBER FUNCTIONS

POWER FUNCTION

-THE FUNCTION POWER() TAKES TWO INPUTS 1ST THE NUMBER 2ND THE POWER AND THEN IT RETURNS THE VALUE OF IT. IT IS A MATHEMATICAL FUNCTION

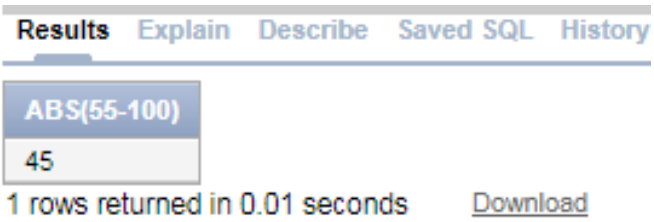
SELECT POWER(5,3) FROM DUAL;



ABSOLUTE (ABS) FUNCTION

-THE FUNCTION ABS() RETURNS THE ABSOLUTE VALUE OF THE VALUE PASSED TO IT

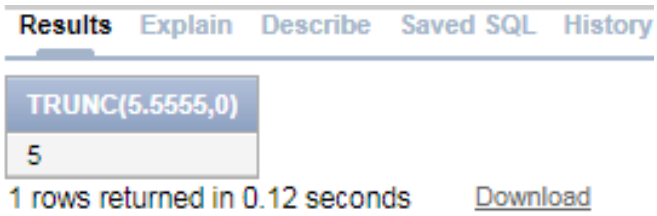
SELECT ABS(55-100) FROM DUAL;



TRUNCATE (TRUNC) FUNCTION

-THE FUNCTION TRUNC() TAKES TWO INPUTS 1ST THE VALUE 2ND TO WHAT DECIMAL PLACE IT IS TO BE TRUNCATED

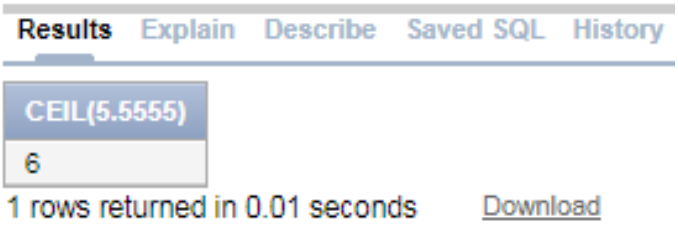
SELECT TRUNC(5.5555,0) FROM DUAL;



CEILING (CEIL) FUNCTION

-THE CEIL() FUNCTION RETURNS THE NEXT INTEGER IF THE NUMBER IS A FRACTION OTHEWISE RETURNS THE NUMBER ITSELF

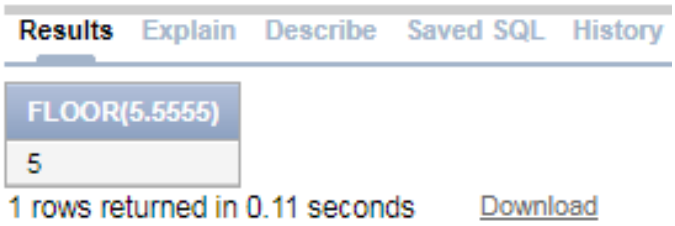
SELECT CEIL(5.5555) FROM DUAL;



FLOOR FUNCTION

-THE FLOOR() FUNCTION RETURNS THE GREATEST INTEGER

SELECT FLOOR(5.5555) FROM DUAL;



EXPONENTIAL (EXP) FUNCTION

-THE FUNCTION EXP() TAKES ONE INPUT THE VALUE OF THE POWER OF THE EXPONENTIAL e AND RETURNS THE VALUE OF IT

SELECT EXP(3) FROM DUAL;

Results Explain Describe Saved SQL History

EXP(3)
20.085536923187667740928529654581717897 1

1 rows returned in 0.01 seconds Download

SQUARE ROOT (SQRT) FUNCTION

-THE FUNCTION SQRT() RETURNS THE SQUARE ROOT OF THE VALUE PASSED TO IT

SELECT SQRT(325) FROM DUAL;

Results Explain Describe Saved SQL History

SQRT(325)
18.027756377319946465596106337352479731 3

1 rows returned in 0.00 seconds Download

GREATEST FUNCTION

-CALCULATES GREATEST FROM THE LIST. VALID FOR BOTH NUMBERS AND CHARACTERS.

select greatest(3,2,1) from dual;

GREATEST(3,2,1)
3

select greatest('a','B','abc') from dual ;

GREATEST('A','B','ABC')
abc

LEAST FUNCTION

-CALCULATES LEAST FROM THE LIST. VALID FOR BOTH NUMBERS AND CHARACTERS

select least(10,13,5) from dual;

LEAST(10,13,5)
5

-CHARACTER FUNCTIONS

USING FUNCTIONS INITCAP, UPPER, LOWER ON STRINGS

-THE FUNCTION INITCAP() IS USED TO CAPITALISE(UPPER CASE) THE FIRST LETTER OF EACH WORD IN THE STRING AND REST IN SMALL (LOWER CASE)

SELECT INITCAP(UPPER(EMP_FNAME||' '||EMP_LNAME)) "NAME", EMP_ID, EMP_SAL, CONTACT_NO FROM EMP_95 ;

Results Explain Describe Saved SQL History

NAME	EMP_ID	EMP_SAL	CONTACT_NO
Pratik Joshi	EPN12347	900000	987654323
Shivank Bali	EPN12345	900000	9874563210
Shaswat Singh	EPN12346	900000	9874563212
Shubhkirti Sharma	EPN12348	80000	987654328

4 rows returned in 0.12 seconds Download

-THE FUNCTION UPPER() IS USED TO CAPITALISE ALL THE LETTERS IN THE STRING
-THE FUNCTION LOWER() IS USED TO CONVERT ALL CAPITAL LETTERS TO LOWER CASE AND DO NOT CHANGE THE LOWER CASE LETTERS

SELECT LOWER(UPPER(EMP_FNAME||' '||EMP_LNAME)) "NAME", EMP_ID, EMP_SAL, CONTACT_NO FROM EMP_95 ;

Results Explain Describe Saved SQL History

NAME	EMP_ID	EMP_SAL	CONTACT_NO
pratik joshi	EPN12347	900000	987654323
shivank bali	EPN12345	900000	9874563210
shaswat singh	EPN12346	900000	9874563212
shubhkirti sharma	EPN12348	80000	987654328

4 rows returned in 0.00 seconds Download

USING CONCATENATION (||) FOR MERGING DETAILS FROM MULTIPLE COLUMNS

-IN THE STATEMENTS IN WHICH THE OUTPUT THAT IS TO BE DISPLAYED THERE SOME DATA IS TO BE MANIPULATED BEFORE DISPLAYING LIKE NAME CONSISTS OF FIRST NAME AND LAST NAME WHICH IS DISPLAYED TOGETHER BY USING OPERATOR ||

SELECT EMP_FNAME||' '||EMP_LNAME "NAME", EMP_ID, EMP_SAL, CONTACT_NO FROM EMP_95 ;

Results Explain Describe Saved SQL History

NAME	EMP_ID	EMP_SAL	CONTACT_NO
PRATIK JOSHI	EPN12347	900000	987654323
SHIVANK BALI	EPN12345	900000	9874563210
SHASWAT SINGH	EPN12346	900000	9874563212
SHUBHKIRTI SHARMA	EPN12348	80000	987654328

4 rows returned in 0.01 seconds Download

USING OPERATIOINS ON STRINGS – SUBSTRING (SUBSTR)

-THE FUNCTION SUBSTR() TAKES 3 ARGUMENTS FIRST THE STRING, SECOND THE BEGINNING OF THE SUBSTRING AND THIRD THE END OF THE SUBSTRING THEN PRINTS THE SUBSTRING ACCORDINGLY

```
SELECT EMP_FNAME||' '||EMP_LNAME "NAME", EMP_ID,
EMP_SAL + (EMP_SAL*EMP_DA/100) + EMP_TA – (EMP_SAL*EMP_TD/100) "TOTAL SALARY",
SUBSTR(CONTACT_NO,1,3)||'-'||SUBSTR(CONTACT_NO,4,3)||'-'||SUBSTR(CONTACT_NO,7,4)
"CONTACT"
FROM EMP_95 ;
```

Results Explain Describe Saved SQL History

NAME	EMP_ID	TOTAL SALARY	CONTACT
PRATIK JOSHI	EPN12347	775000	987-654-323
SHIVANK BALI	EPN12345	1055000	987-456-3210
SHASWAT SINGH	EPN12346	1005000	987-456-3212
SHUBHKIRTI SHARMA	EPN12348	54000	987-654-328

4 rows returned in 0.01 seconds Download

USING FUNCTION INSTRING (INSTR) WITH TWO DIFFERENT VALUES

-THE INSTR() FUNCTION TAKES 3 INPUTS 1ST THE STRING 2ND CHARACTER TO BE SEARCHED FOR 3RD THE NUMBER OF OCCURRENCE AND RETURNS THE POSITION OF THE REQUIRED CHARACTER AND 0 IF THE CHARACTER IS NOT FOUND

```
SELECT EMP_FNAME "NAME",
INSTR(EMP_LNAME,'I',1), EMP_ID,
EMP_SAL + (EMP_SAL*EMP_DA/100) + EMP_TA – (EMP_SAL*EMP_TD/100) "TOTAL SALARY",
SUBSTR(CONTACT_NO,1,3)||'-'||SUBSTR(CONTACT_NO,4,3)||'-'||SUBSTR(CONTACT_NO,7,4)
"CONTACT"
FROM EMP_95 ;
```

NAME	INSTR(EMP_LNAME,'I',1)	EMP_ID	TOTAL SALARY	CONTACT
PRATIK	5	EPN12347	775000	987-654-323
SHIVANK	4	EPN12345	1055000	987-456-3210
SHASWAT	2	EPN12346	1005000	987-456-3212
SHUBHKIRTI	0	EPN12348	54000	987-654-328

4 rows returned in 0.00 seconds Download

```
SELECT EMP_FNAME "NAME",
INSTR(EMP_LNAME,'A',1), EMP_ID,
EMP_SAL + (EMP_SAL*EMP_DA/100) + EMP_TA – (EMP_SAL*EMP_TD/100) "TOTAL SALARY",
SUBSTR(CONTACT_NO,1,3)||'-'||SUBSTR(CONTACT_NO,4,3)||'-'||SUBSTR(CONTACT_NO,7,4)
"CONTACT"
FROM EMP_95 ;
```

NAME	INSTR(EMP_LNAME,'A',1)	EMP_ID	TOTAL SALARY	CONTACT
PRATIK	0	EPN12347	775000	987-654-323
SHIVANK	2	EPN12345	1055000	987-456-3210
SHASWAT	0	EPN12346	1005000	987-456-3212
SHUBHKIRTI	3	EPN12348	54000	987-654-328

4 rows returned in 0.12 seconds Download

USING FUNCTION LEFT - TRIM (LTRIM)

-THE FUNCTION NLTRIM() TAKES TWO INPUTS 1ST THE STRING 2ND THE LETTER TO BE TRIMMED FROM THE LEFT END OF THE STRING

```
SELECT EMP_FNAME "NAME",
LTRIM(EMP_LNAME,'S'), EMP_ID,
EMP_SAL + (EMP_SAL*EMP_DA/100) + EMP_TA – (EMP_SAL*EMP_TD/100) "TOTAL SALARY",
SUBSTR(CONTACT_NO,1,3)||'-'||SUBSTR(CONTACT_NO,4,3)||'-'||SUBSTR(CONTACT_NO,7,4)
"CONTACT"
FROM EMP_95 ;
```

NAME	LTRIM(EMP_LNAME,'S')	EMP_ID	TOTAL SALARY	CONTACT
PRATIK	JOSHI	EPN12347	775000	987-654-323
SHIVANK	BALI	EPN12345	1055000	987-456-3210
SHASWAT	INGH	EPN12346	1005000	987-456-3212
SHUBHKIRTI	HARMA	EPN12348	54000	987-654-328

4 rows returned in 0.01 seconds Download

USING FUNCTION RIGHT - TRIM (RTRIM)

-THE FUNCTIOINN RTRIM() TAKES THE INPUTS 1ST THE STRING 2ND THE CHARACTER TO BE TRIMMED IF THE CHARACTER EXISTS IN THE RAGHT END OF THE STRING THEN IT IS TRIMMED OR WE CAN SAY IT IS REMOVED

```
SELECT EMP_FNAME "NAME",
RTRIM(EMP_LNAME,'I'), EMP_ID,
EMP_SAL + (EMP_SAL*EMP_DA/100) + EMP_TA – (EMP_SAL*EMP_TD/100) "TOTAL SALARY",
SUBSTR(CONTACT_NO,1,3)||'-'||SUBSTR(CONTACT_NO,4,3)||'-'||SUBSTR(CONTACT_NO,7,4)
"CONTACT"
FROM EMP_95 ;
```

Results Explain Describe Saved SQL History

NAME	RTRIM(EMP_LNAME,'I')	EMP_ID	TOTAL SALARY	CONTACT
PRATIK	JOSH	EPN12347	775000	987-654-323
SHIVANK	BAL	EPN12345	1055000	987-456-3210
SHASWAT	SINGH	EPN12346	1005000	987-456-3212
SHUBHKIRTI	SHARMA	EPN12348	54000	987-654-328

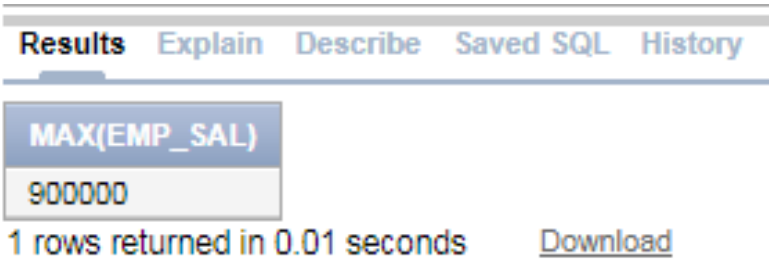
4 rows returned in 0.01 seconds Download

MULTIROW FUNCTIONS

FINDING MAXIMUM VALUE IN A COLUMN USING MAX

-THE MAX() FUNCTION RETURNS ONLY SINGLE TUPLE WITH THE MAXIMUM VALUE IN THAT PARTICULAR ATTRIBUTE

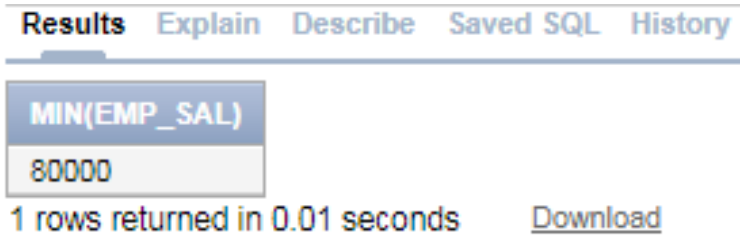
SELECT MAX(EMP_SAL) FROM EMP_95 ;



FINDING MINIMUM VALUE IN A COLUMN USING MIN

-THE MIN() FUNCTION RETURNS THE MINIMUM VALUE OF THAT IS PRESENT IN THAT PARTICULAR ATTRIBUTE IN THE TABLE

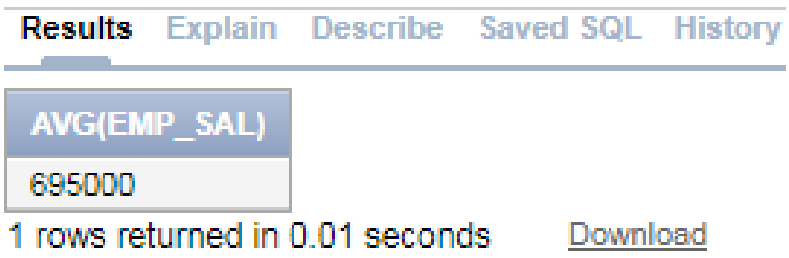
SELECT MIN(EMP_SAL) FROM EMP_95 ;



FINDING AVERAGE OF VALUES IN A COLUMN USING AVG

-THE AVG() FUNCTION IS USED TO RECEIVE THE AVERAGE VALUE OF ALL THE VALUES PRESENT IN THE COLUMN

SELECT AVG(EMP_SAL) FROM EMP_95;



USING THE FUNCTION COUNT

-THE FUNCTION COUNT() RETURNS THE NUMBER OF TUPLES THAT EXIST IN THE COLUMN THAT IS PASSED TO THE FUNCTION

SELECT COUNT(EMP_SAL) FROM EMP_95;

Results Explain Describe Saved SQL History

COUNT(EMP_SAL)
4

1 rows returned in 0.01 seconds Download

COMPARISON DATA USING LOGICAL CONDITION
USING WHERE CONDITION WITH BETWEEN CONSTRAINT

SELECT * FROM EMP_95 WHERE EMP_SAL BETWEEN 80000 AND 900000;

Results Explain Describe Saved SQL History

EMP_ID	EMP_FNAME	EMP_LNAME	EMP_SAL	CONTACT_NO	EMP_TA	EMP_DA	EMP_TD
EPN12347	PRATIK	JOSHI	900000	987654323	100000	15	40
EPN12345	SHIVANK	BALI	900000	9874563210	200000	15	20
EPN12346	SHASWAT	SINGH	900000	9874563212	150000	10	15
EPN12348	SHUBHKIRTI	SHARMA	80000	987654328	10000	5	50

4 rows returned in 0.01 seconds Download

NVL FUNCTION

NVL is a type of general function

General functions are used to handle NULL values in database. The objective of the general NULL handling functions is to replace the NULL values with an alternate value.

```
SELECT empno,ename,job, NVL(TO_CHAR(mgr), 'Not Applicable')
  "MANAGER ID" FROM emp;
```

EMPNO	ENAME	JOB	MANAGER ID
7839	KING	PRESIDENT	Not Applicable
7698	BLAKE	MANAGER	7839
7782	CLARK	MANAGER	7839
7566	JONES	MANAGER	7839
7788	SCOTT	ANALYST	7566
7902	FORD	ANALYST	7566
7369	SMITH	CLERK	7902
7499	ALLEN	SALESMAN	7698
7521	WARD	SALESMAN	7698
7654	MARTIN	SALESMAN	7698
7844	TURNER	SALESMAN	7698
7876	ADAMS	CLERK	7788
7900	JAMES	CLERK	7698
7934	MILLER	CLERK	7782

14 rows returned in 0.00 seconds Download

CREATED TABLE FOR USE IN FURTHER QUERIES

TABLE EMPLOYEE

```
CREATE TABLE employee (
empno VARCHAR2(4) PRIMARY KEY CHECK( empno LIKE 'E%'),
ename VARCHAR2(20),
ejob VARCHAR2(20),
joiningdate DATE,
esal NUMBER(10) CHECK (esal > 0),
manid VARCHAR2(4),
deptno VARCHAR2(4)
);
```

TABLE DEPARTMENT

```
CREATE TABLE department (
deptno VARCHAR2(4) PRIMARY KEY CHECK( deptno LIKE 'D%'),
dname VARCHAR2(20),
dloc VARCHAR2(10)
);
```

TABLES AFTER INSERTING ATTRIBUTES (USING NATURAL JOIN)

SELECT * FROM EMPLOYEE NATURAL JOIN DEPARTMENT ORDER BY DEPTNO;

Results | Explain | Describe | Saved SQL | History

DEPTNO	EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DNAME	DLOC
D001	E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000000	E005	SHUBH	UIET-101
D002	E009	SOMYADEEP	DBA	04/04/2010	4550000	E005	TECH	MOHALI
D002	E008	TANVEER	DBA	04/04/2017	3500000	E005	TECH	MOHALI
D003	E003	SHUBH	CTO	12/12/2005	4000000	E001	MANAGEMENT	PEC
D004	E005	PRATIK	SERVICES MANAGER	04/04/2007	2500000	E003	SERVICES	ELANTE
D005	E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000000	E005	FOOD	DELHI
D006	E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400000	E005	HR	MUMBAI
D006	E002	SHUBHAM GUPTA	CEO	12/12/2000	5000000	E001	HR	MUMBAI
D007	E007	NAMAN	OSD	04/04/2009	3400000	E001	RESEARCH	CHENNAI

9 rows returned in 0.00 seconds Download

PRACTICAL 6

USING GROUP BY AND ORDER BY CLAUSE

THE ORDER BY CLAUSE

The ORDER BY keyword is used to sort the result-set in ascending or descending order.
The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY SYNTAX

SELECT *column_name(s)*
FROM *table_name*
ORDER BY *column_name(s)* ASC|DESC;

THE GROUP BY CLAUSE

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

SQL Nested subqueries

A subquery can be nested inside other subqueries. SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. SQL executes innermost subquery first, then next level.

-DISPLAY ALL THE DETAILS IN ASCENDING ORDER OF THEIR SALARY

SELECT * FROM employee ORDER BY esal ASC;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007
E008	TANVEER	DBA	04/04/2017	3500500	E005	D002
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006

9 rows returned in 0.01 seconds Download

-DISPLAY ALL THE DETAILS IN DESCENDING ORDER OF THEIR SALARY

SELECT * FROM employee ORDER BY esal DESC;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003
E008	TANVEER	DBA	04/04/2017	3500500	E005	D002
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001

9 rows returned in 0.00 seconds Download

-GIVE TOTAL SALARY ISSUED BY EACH DEPARTMENT

SELECT deptno, SUM(esal) FROM employee GROUP BY deptno;

Results Explain Describe Saved SQL History

DEPTNO	SUM(ESAL)
D003	4000000
D006	7400000
D004	2500000
D002	8050000
D001	1000000
D005	2000000
D007	3400000

7 rows returned in 0.01 seconds Download

-AVERAGE SALARY OF EACH JOB

SELECT ejob, AVG(esal) FROM employee GROUP BY ejob;

Results Explain Describe Saved SQL History

EJOB	AVG(ESAL)
DBA	4025000
CTO	4000000
SERVICES MANAGER	2500000
TRAINING OFFICER	2400000
INTERVIEWING OFFICER	2000000
DEVELOPER	1000000
CEO	5000000
OSD	3400000

8 rows returned in 0.00 seconds Download

-LIST THE TOTAL SALARY OF EACH JOB IN EACH DEPARTMENT

SELECT deptno, ejob, SUM(esal) FROM employee GROUP BY deptno, ejob;

Results Explain Describe Saved SQL History

DEPTNO	EJOB	SUM(ESAL)
D006	CEO	5000000
D007	OSD	3400000
D002	DBA	8050000
D003	CTO	4000000
D005	INTERVIEWING OFFICER	2000000
D004	SERVICES MANAGER	2500000
D001	DEVELOPER	1000000
D006	TRAINING OFFICER	2400000

8 rows returned in 0.00 seconds Download

-LIST TOTAL, MAX, MIN, AVG, SALARY OF DEPARTMENT NO D006 JOB WISE

SELECT ejob, SUM(esal),MAX(esal),MIN(esal),AVG(esal) FROM employee WHERE (deptno='D006') GROUP BY ejob ;

Results Explain Describe Saved SQL History

EJOB	SUM(ESAL)	MAX(ESAL)	MIN(ESAL)	AVG(ESAL)
TRAINING OFFICER	2400000	2400000	2400000	2400000
CEO	5000000	5000000	5000000	5000000

2 rows returned in 0.00 seconds Download

-LIST EJOB AND AVERAGE SALARY OF JOBS HAVING AVERAGE SALARY GREATER THAN 100000

SELECT ejob, AVG(esal) FROM employee HAVING (AVG(esal) >100000) GROUP BY ejob;

Results Explain Describe Saved SQL History

EJOB	AVG(ESAL)
DBA	4025000
CTO	4000000
SERVICES MANAGER	2500000
TRAINING OFFICER	2400000
INTERVIEWING OFFICER	2000000
DEVELOPER	1000000
CEO	5000000
OSD	3400000

8 rows returned in 0.00 seconds Download

NESTED QUERIES

SINGLE ROW AND MULTI ROW SUBQUERIES

-(SINGLE ROW) LIST NAME AND SALARY OF EMPLOYEE WHOSE SALARY IS GREATER THAN MINIMUM SALARY OF DEPARTMENT RESEARCH

SELECT ename, esal FROM employee WHERE esal > (SELECT MIN(esal) FROM employee NATURAL JOIN department WHERE dname='RESEARCH') ;

Results Explain Describe Saved SQL History

ENAME	ESAL
SOMYADEEP	4550000
SHUBHAM GUPTA	5000000
SHUBH	4000000
TANVEER	3500000

4 rows returned in 0.00 seconds Download

-(SINGLE ROW) LIST NAME AND SALARY OF EMPLOYEE WHOSE SALARY IS GREATER THAN MAXIMUM SALARY OF DEPARTMENT HR

SELECT ename, esal FROM employee WHERE esal > (SELECT AVG(esal) FROM employee NATURAL JOIN department WHERE dname='HR') ;

ENAME	ESAL
SOMYADEEP	4550000
SHUBHAM GUPTA	5000000
SHUBH	4000000

3 rows returned in 0.01 seconds Download

-(MULTI ROW) LIST THE DETAILS OF DEPARTMENT WHERE MANAGER ID IS E005

SELECT * FROM department WHERE deptno IN (SELECT deptno FROM employee WHERE manid='E005');

Results Explain Describe Saved SQL History

DEPTNO	DNAME	DLOC
D001	SHUBH	UIET-101
D002	TECH	MOHALI
D005	FOOD	DELHI
D006	HR	MUMBAI

4 rows returned in 0.00 seconds Download

-(SINGLE ROW) LIST THE EMPLOYEES BELONGING TO THE DEPARTMENT SHUBH

SELECT ename FROM EMPLOYEE WHERE deptno = (SELECT deptno FROM employee WHERE ENAME='SHUBH');

ENAME
SHUBH

1 rows returned in 0.01 seconds Download

-(SINGLE ROW) LIST THE NAME OF EMPLOYEE WHO DO THE JOB SAME AS OF EMPLOYEE WITH EMPNO = E008

SELECT ename FROM employee WHERE ejob=(SELECT ejob FROM employee WHERE empno ='E008');

ENAME
SOMYADEEP
TANVEER

2 rows returned in 0.02 seconds Download

-(SINGLE ROW) LIST THE NAME OF EMPLOYEE WHO DO THE JOB SAME AS OF EMPNO = E008 AND WHOSE SALARY IS GREATER THAN EMPNO = E003

SELECT ename FROM employee WHERE ejob=(SELECT ejob FROM employee WHERE empno ='E008') AND esal > (SELECT esal FROM employee WHERE empno ='E003') ;

Results Explain Describe Saved SQL History

ENAME
SOMYADEEP

1 rows returned in 0.00 seconds Download

-(SINGLE ROW) LIST NAME AND SALARY OF EMPLOYEE WHOSE SALARY IS GREATER THAN SALARY OF DEPARTMENT RESEARCH

SELECT ename,esal FROM employee WHERE esal>(SELECT SUM(esal) FROM employee NATURAL JOIN department WHERE dname='RESEARCH' GROUP BY dname) ;

Results Explain Describe Saved SQL History

ENAME	ESAL
SOMYADEEP	4550000
SHUBHAM GUPTA	5000000
SHUBH	4000000
TANVEER	3500000

4 rows returned in 0.01 seconds Download

-(MULTI ROW) LIST THE DETAILS OF THE DEPARTMENT WHERE MANAGER ID IS E001

SELECT * FROM department WHERE deptno IN (SELECT deptno FROM employee WHERE manid='E001');

Results Explain Describe Saved SQL History

DEPTNO	DNAME	DLOC
D003	MANAGEMENT	PEC
D006	HR	MUMBAI
D007	RESEARCH	CHENNAI

3 rows returned in 0.00 seconds Download

-(MULTI ROW) LIST THE EMPLOYEE WHO DO NOT MANAGE ANY EMPLOYEE

SELECT ename FROM employee WHERE empno NOT IN (SELECT manid FROM employee GROUP BY manid);

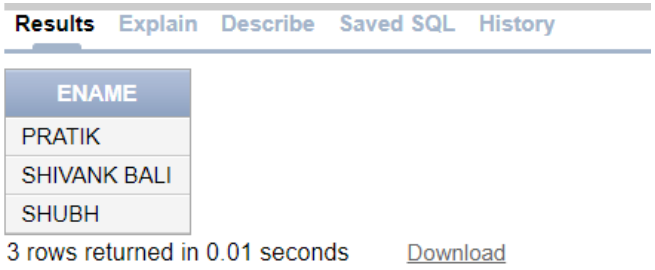
Results Explain Describe Saved SQL History

ENAME
TANVEER
SINDHIYA
SHUBHAM GUPTA
SHASWAT
NAMAN
SOMYADEEP

6 rows returned in 0.00 seconds Download

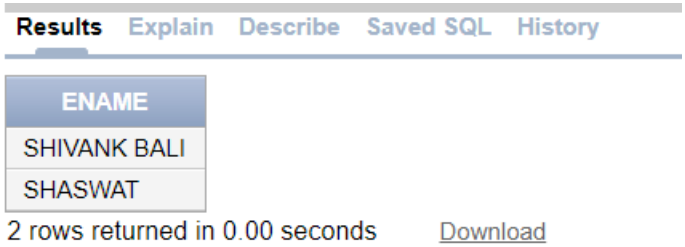
-(MULTI ROW) LIST THE EMPLOYEE WHO MANAGE ATLEAST ONE EMPLOYEE

SELECT ename FROM employee WHERE empno IN (SELECT manid FROM employee GROUP BY manid);



-(SINGLE ROW) LIST ALL EMPLOYEE WHOSE SALARY IS LESS THAN MINIMUM SALARY OF HR DEPARTMENT AND THEY SHOULD BE OF DIFFERENT DEPARTMENT

SELECT ename FROM employee NATURAL JOIN department WHERE esal < (SELECT MIN(esal) FROM employee NATURAL JOIN department WHERE dname='HR') AND NOT dname = 'HR';



PRACTICAL 7

JOINS

An SQL join clause combines columns from one or more tables in a relational database. It creates a set that can be saved as a table or used as it is.
A JOIN is a means for combining columns from one (self-join) or more tables by using values common to each. ANSI-standard SQL specifies five types of JOIN: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS.

-CROSS JOIN

SELECT e.deptno,e.ename,d.deptno AS DDEPTNO FROM EMPLOYEE e CROSS JOIN DEPARTMENT d ORDER BY DEPTNO;

DEPTNO	ENAME	DDEPTNO
D001	SHIVANK BALI	D002
D001	SHIVANK BALI	D003
D001	SHIVANK BALI	D004
D001	SHIVANK BALI	D005
D001	SHIVANK BALI	D006
D001	SHIVANK BALI	D007
D001	SHIVANK BALI	D001
D002	SOMYADEEP	D001
D002	SOMYADEEP	D002
D002	SOMYADEEP	D003
D002	SOMYADEEP	D004
D002	SOMYADEEP	D005
D002	SOMYADEEP	D006
D002	TANVEER	D007
D002	TANVEER	D006
D002	TANVEER	D005
D002	TANVEER	D004
D002	TANVEER	D003
D002	SOMYADEEP	D007
D002	TANVEER	D001
D002	TANVEER	D002
D003	SHUBH	D006
D003	SHUBH	D007
D003	SHUBH	D005
D003	SHUBH	D004
D003	SHUBH	D003
D003	SHUBH	D002
D003	SHUBH	D001
D004	PRATIK	D006
D004	PRATIK	D002
D004	PRATIK	D001
D005	SHASWAT	D001
D005	SHASWAT	D002
D005	SHASWAT	D003
D005	SHASWAT	D004
D005	SHASWAT	D005
D005	SHASWAT	D006
D005	SHASWAT	D007
D006	SHUBHAM GUPTA	D007
D006	SHUBHAM GUPTA	D006
D006	SHUBHAM GUPTA	D005
D006	SHUBHAM GUPTA	D004
D006	SHUBHAM GUPTA	D003
D006	SHUBHAM GUPTA	D002
D006	SHUBHAM GUPTA	D001
D006	SINDHIYA	D007
D006	SINDHIYA	D006
D006	SINDHIYA	D005
D006	SINDHIYA	D004
D006	SINDHIYA	D003
D006	SINDHIYA	D001
D006	SINDHIYA	D002
D007	NAMAN	D006
D007	NAMAN	D005
D007	NAMAN	D004
D007	NAMAN	D003
D007	NAMAN	D002
D007	NAMAN	D001
D007	NAMAN	D007

-NATURAL JOIN

THE SQL NATURAL JOIN IS A TYPE OF EQUI JOIN AND IS STRUCTURED IN SUCH A WAY THAT, COLUMNS WITH THE SAME NAME OF ASSOCIATED TABLES WILL APPEAR ONCE ONLY

SELECT * FROM EMPLOYEE NATURAL JOIN DEPARTMENT ORDER BY DEPTNO;

Results Explain Describe Saved SQL History

DEPTNO	EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DNAME	DLOC
D001	E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000000	E005	SHUBH	UIET-101
D002	E009	SOMYADEEP	DBA	04/04/2010	4550000	E005	TECH	MOHALI
D002	E008	TANVEER	DBA	04/04/2017	3500000	E005	TECH	MOHALI
D003	E003	SHUBH	CTO	12/12/2005	4000000	E001	MANAGEMENT	PEC
D004	E005	PRATIK	SERVICES MANAGER	04/04/2007	2500000	E003	SERVICES	ELANTE
D005	E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000000	E005	FOOD	DELHI
D006	E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400000	E005	HR	MUMBAI
D006	E002	SHUBHAM GUPTA	CEO	12/12/2000	5000000	E001	HR	MUMBAI
D007	E007	NAMAN	OSD	04/04/2009	3400000	E001	RESEARCH	CHENNAI

9 rows returned in 0.00 seconds Download

-OUTER JOIN

--FULL OUTER JOIN

SELECT * FROM employee FULL OUTER JOIN department ON employee.deptno = department.deptno;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO	DEPTNO	DNAME	DLOC
E010	ANUSHUMAN	CTO	05/05/2002	4050505	E002	D010	-	-	-
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006	D006	HR	MUMBAI
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007	D007	RESEARCH	CHENNAI
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002	D002	TECH	MOHALI
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001	D001	SHUBH	UIET-101
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006	D006	HR	MUMBAI
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003	D003	MANAGEMENT	PEC
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005	D005	FOOD	DELHI
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004	D004	SERVICES	ELANTE
-	-	-	-	-	-	-	D011	DEVELOPMENT	PUNJAB

10 rows returned in 0.00 seconds Download

--LEFT OUTER JOIN

SELECT * FROM employee LEFT OUTER JOIN department ON employee.deptno = department.deptno;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO	DEPTNO	DNAME	DLOC
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001	D001	SHUBH	UIET-101
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002	D002	TECH	MOHALI
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003	D003	MANAGEMENT	PEC
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004	D004	SERVICES	ELANTE
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005	D005	FOOD	DELHI
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006	D006	HR	MUMBAI
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006	D006	HR	MUMBAI
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007	D007	RESEARCH	CHENNAI
E010	ANUSHUMAN	CTO	05/05/2002	4050505	E002	D010	-	-	-

9 rows returned in 0.00 seconds Download

--RIGHT OUTER JOIN

SELECT * FROM employee RIGHT OUTER JOIN department ON employee.deptno = department.deptno;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO	DEPTNO	DNAME	DLOC
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006	D006	HR	MUMBAI
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007	D007	RESEARCH	CHENNAI
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002	D002	TECH	MOHALI
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001	D001	SHUBH	UIET-101
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006	D006	HR	MUMBAI
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003	D003	MANAGEMENT	PEC
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005	D005	FOOD	DELHI
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004	D004	SERVICES	ELANTE
-	-	-	-	-	-	-	D011	DEVELOPMENT	PUNJAB

9 rows returned in 0.00 seconds Download

PRACTICAL 8

VIEWS

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

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

Views, which are a type of virtual tables allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

CREATING VIEWS

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables or another view.

SINGLE TABLE

CREATE VIEW emp95 AS SELECT * FROM employee;

```
View created.

0.44 seconds
```

SELECT * FROM emp95;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004
E008	TANVEER	DBA	04/04/2017	3500500	E005	D002

UPDATING A VIEW

A view can be updated under certain conditions which are given below –

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view.

INSERTING ROWS INTO A VIEW

ROWS OF DATA CAN BE INSERTED INTO A VIEW. THE SAME RULES THAT APPLY TO THE UPDATE COMMAND ALSO APPLY TO THE INSERT COMMAND.

INSERT INTO emp95 VALUES ('E010' , 'ANUSHUMAN' , 'CTO' , '05-05-2002' , 4050505 , 'E002' , 'D002');

```
1 row(s) inserted.
```

0.15 seconds

SELECT * FROM EMPLOYEE;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO
E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	D006
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002
E010	ANUSHUMAN	CTO	05/05/2002	4050505	E002	D002
E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	D001
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006
E003	SHUBH	CTO	12/12/2005	4000500	E001	D003
E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	D005
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004
E008	TANVEER	DBA	04/04/2017	3500500	E005	D002

DELETING FROM VIEWS

-THE ROWS CAN BE DELETED FROM VIEWS IN THE SAME WAY INSERTION AND UPDATION TAKES PLACE

DELETE FROM EMP95;

```
9 row(s) deleted.
```

0.01 seconds

DROPPING VIEWS

OBVIOUSLY, WHERE YOU HAVE A VIEW, YOU NEED A WAY TO DROP THE VIEW IF IT IS NO LONGER NEEDED. THE SYNTAX IS VERY SIMPLE AND IS GIVEN BELOW –

DROP VIEW view_name;

DROP VIEW EMP95;

```
View dropped.
```

0.49 seconds

CREATING VIEW OF COMPOSITION OF TWO TABLES USING JOIN STATEMENT

CREATE VIEW empj AS SELECT * FROM employee NATURAL JOIN department;

```
View created.
```

0.00 seconds

SELECT * FROM empj;

DEPTNO	EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DNAME	DLOC
D006	E006	SINDHIYA	TRAINING OFFICER	04/04/2006	2400500	E005	HR	MUMBAI
D007	E007	NAMAN	OSD	04/04/2009	3400500	E001	RESEARCH	CHENNAI
D002	E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	TECH	MOHALI
D001	E001	SHIVANK BALI	DEVELOPER	10/10/2010	1000500	E005	SHUBH	UIET-101
D006	E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	HR	MUMBAI
D003	E003	SHUBH	CTO	12/12/2005	4000500	E001	MANAGEMENT	PEC
D005	E004	SHASWAT	INTERVIEWING OFFICER	12/04/2007	2000500	E005	FOOD	DELHI
D004	E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	SERVICES	ELANTE
D002	E008	TANVEER	DBA	04/04/2017	3500500	E005	TECH	MOHALI

9 rows returned in 0.01 seconds Download

PRACTICAL 9

PL/SQL

INTRODUCTION TO PL/SQL

PL/SQL is an Oracle procedural extension for SQL. They have designed this language for easy use of complex SQL statements.

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

Building Blocks of PL/SQL Programs

PL/SQL is a block-structured language. A PL/SQL block is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END, which break up the block into three sections:

- Declarative: statements that declare variables, constants, and other code elements, which can then be used within that block
- Executable: statements that are run when the block is executed
- Exception handling: a specially structured section you can use to “catch,” or trap, any exceptions that are raised when the executable section runs

BASIC SYNTAX

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

CONTROL STRUCTURES IN PL/SQL

IF THEN Statement

The IF THEN statement has this structure:

```
IF condition THEN
  statements
END IF;
```

IF THEN ELSE Statement

The IF THEN ELSE statement has this structure:

IF *condition* THEN
  statements
ELSE
  else_statements
END IF;

IF THEN ELSIF Statement

The IF THEN ELSIF statement has this structure:

IF *condition_1* THEN
  statements_1
ELSIF *condition_2* THEN
  statements_2
...
[ELSE
   else_statements
]
END IF;

Simple CASE Statement

The simple CASE statement has this structure:

CASE *selector*
WHEN *selector_value_1* THEN *statements_1*
WHEN *selector_value_2* THEN *statements_2*
...
WHEN *selector_value*_n THEN *statements*_n
[ELSE
  else_statements]
END CASE;]

Basic LOOP Statement

The basic LOOP statement has this structure:

[*label*] LOOP
  statements
END LOOP [*label*];

EXIT Statement

The EXIT statement exits the current iteration of a loop unconditionally and transfers control to the end of either the current loop or an enclosing labeled loop.

EXIT WHEN Statement

The EXIT WHEN statement exits the current iteration of a loop when the condition in its WHEN clause is true, and transfers control to the end of either the current loop or an enclosing labeled loop.

WHILE LOOP Statement

The WHILE LOOP statement runs one or more statements while a condition is true. It has this structure:

[*label*] WHILE *condition* LOOP
  statements
END LOOP [*label*];

FOR LOOP Statement

The FOR LOOP statement runs one or more statements while the loop index is in a specified range. The statement has this structure:

[*label*] FOR *index* IN [REVERSE] *lower_bound..upper_bound* LOOP
  statements
END LOOP [*label*];

PRACTICAL 10

PL - SQL PROGRAMS :-

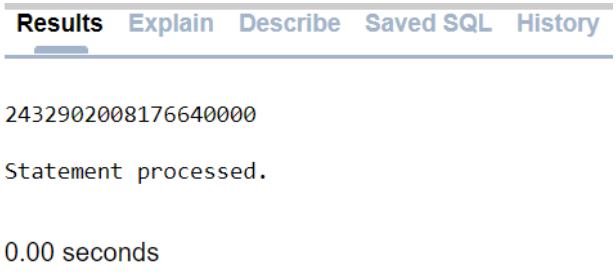
FACTORIAL OF A NUMBER

CODE->

```
DECLARE
   x number;
   val number :=1;
BEGIN
 x:=:ENTER_THE_NUMBER;
 WHILE x>0
 LOOP
    val:=val*x;
    x:=x-1;
 END LOOP;
 dbms_output.put_line(val);
END;
```

-INPUT

-OUTPUT



-FIBONACCI SERIES

CODE->

```
DECLARE
   x number;
   fibp number :=0;
   fibn number :=1;
   t number;
BEGIN
 x:=:ENTER_THE_NUMBER;
 IF x=1
 THEN dbms_output.put_line(fibp);
 ELSIF x=2
 THEN dbms_output.put_line(fibp);
     dbms_output.put_line(fibn);
 ELSE
     dbms_output.put_line(fibp);
     dbms_output.put_line(fibn);
 WHILE x>2
 LOOP
   t:=fibn;
   fibn:=fibn+fibp;
   fibp:=t;
   x:=x-1;
   dbms_output.put_line(fibn);
 END LOOP;
 END IF;
END;
```

-INPUT

5

-OUTPUT

```
0
1
1
2
3

Statement processed.


0.01 seconds
```

-PROGRAM TO CHECK LEAP YEAR OR NOT

CODE->

```
DECLARE
   x number;
BEGIN
 x:=:ENTER_THE_YEAR;
 IF MOD(x,4)=0 AND MOD(x,400)!=0
 THEN dbms_output.put_line('LEAP YEAR');
 ELSE
     dbms_output.put_line('NOT A LEAP YEAR');
 END IF;
END;
```

-INPUT

2016

-OUTPUT

```
LEAP YEAR

Statement processed.


0.01 seconds
```

-CIRCLE CIRCUMFERENCE AND AREA

CODE->

```
DECLARE
   R number;
   PI number:=3.14;
   VAL number;
BEGIN
 R:=:ENTER_THE_RADIUS;
 VAL:=2*PI*R;
    dbms_output.put_line('CIRCUMFERENCE :'||VAL);
 VAL:=PI*R*R;
    dbms_output.put_line('AREA :'||VAL);
END;
```

-INPUT

5

-OUTPUT

```
CIRCUMFERENCE :43.96
AREA :153.86

Statement processed.


0.00 seconds
```

-CHECK IF PRIME NUMBER OR NOT

CODE->

```
DECLARE
 X NUMBER;
 I NUMBER:=2;
BEGIN
 X:=:ENTER_THE_NUMBER;
 WHILE I<X/2
 LOOP
  IF MOD(X,I)=0 THEN
      I:=-1;
      EXIT;
  END IF;
 I:=I+1;
 END LOOP;
 IF I=-1 THEN
 DBMS_OUTPUT.PUT_LINE(X||' IS NOT A PRIME NUMBER');
 ELSE DBMS_OUTPUT.PUT_LINE(X||' IS A PRIME NUMBER');
 END IF;
END;
```

-INPUT

433

-OUTPUT

```
433 IS A PRIME NUMBER

Statement processed.


0.00 seconds
```

-PRODGRAM TO CALCULATE SIMPLE INTEREST

CODE->

```
DECLARE
p number(9,2);
r number(9,2);
t number(9,2);
si number(9,2);
BEGIN
p:=&p;
r:=&r;
t:=&t;
si:=(p*r*t)/100;
dbms_output.put_line('Simple Interest = '||si);
END;
```

-INPUT

P=100
R=15
T=5

-OUTPUT

```
Simple Interest = 75

Statement processed.


0.00 seconds
```

-PROGRAM TO DISPLAY GRADE

CODE->

```
DECLARE
  grade char(1) ;
BEGIN
  grade:=:GRADE;
  CASE grade
    when 'A' then dbms_output.put_line('Excellent');
    when 'B' then dbms_output.put_line('Very good');
    when 'C' then dbms_output.put_line('Well done');
    when 'D' then dbms_output.put_line('You passed');
    when 'F' then dbms_output.put_line('Better try again');
    else dbms_output.put_line('No such grade');
  END CASE;
END;
```

-INPUT

A

-OUTPUT

```
Excellent

Statement processed.


0.01 seconds
```

PRACTICAL 11

CURSORS

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. The set of rows the cursor holds is called the *active* set.

There are mainly two types of cursors

1. Implicit cursors
2. Explicit cursors

Implicit Cursors

Implicit cursors are automatically created and destroyed by the Oracle server whenever we execute an SQL statement inside a PL/SQL block. The Oracle server by default opens, fetches, processes, and closes the implicit cursor automatically without the need of a programmer intervention and that is why the implicit cursors are much faster compared to the explicit cursors, thus resulting in a simple and elegant code.

%FOUND
Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%ISOPEN
Always returns FALSE, because the database closes the SQL cursor automatically after executing its associated SQL statement.
%NOTFOUND
The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ROWCOUNT
Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Explicit Cursors

An explicit cursor is a named pointer to a private SQL area that stores information for processing a specific query or DML statement—typically, one that returns or affects multiple rows. You can use an explicit cursor to retrieve the rows of a result set one at a time.

DECLARATION

- CURSOR cursor_name [parameter] RETURN return_type;
- CURSOR cursor_name [parameter] [RETURN return_type]
  IS SELECT STATEMENT;

OPENING CURSOR

OPEN cursor_name [(cursor_parameter)];

FETCHING FROM CURSORS

FETCH cursor_name INTO variable;

CLOSING CURSORS

CLOSE cursor_name [(cursor_parameter)];

-PROGRAM USING IMPLICIT CURSOR

CODE->

```
DECLARE
  emp employee%rowtype;
BEGIN
  SELECT *
  INTO emp
  FROM employee
  WHERE empno = 'E001';
  dbms_output.put_line('Number of employee with employee number E001
:'||sql%rowcount);
END;
```

-OUTPUT

```
Number of employee with employee number E001 :1

Statement processed.

0.00 seconds
```

-PROGRAM USING EXPLICIT CURSOR

CODE->

```
DECLARE
CURSOR C1 IS SELECT * FROM EMPLOYEE;
REC C1%ROWTYPE;
BEGIN
OPEN C1;
LOOP
FETCH C1 INTO REC;
EXIT WHEN C1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('NAME : '||REC.ENAME);
END LOOP;
CLOSE C1;
END;
```

-OUTPUT

```
NAME : SINDHIYA
NAME : NAMAN
NAME : SOMYADEEP
NAME : SHIVANK BALI
NAME : SHUBHAM GUPTA
NAME : SHUBH
NAME : SHASWAT
NAME : PRATIK
NAME : TANVEER

Statement processed.
```

PRACTICAL 12

PROCEDURES

A procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.
1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

-PROCEDURE SYNTAX

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
IS
  Declaration section
BEGIN
  Execution section
EXCEPTION
  Exception section
END;
```

-CODE TO IMPLEMENT PROCEDURES

```
CREATE OR REPLACE PROCEDURE fibonacci (x IN OUT NUMBER) IS
   fibp number :=0;
   fibn number :=1;
   t number;
BEGIN
  IF x=1
 THEN dbms_output.put_line(fibp);
 ELSIF x=2
 THEN dbms_output.put_line(fibp);
     dbms_output.put_line(fibn);
 ELSE
     dbms_output.put_line(fibp);
     dbms_output.put_line(fibn);
 WHILE x>2
 LOOP
   t:=fibn;
   fibn:=fibn+fibp;
   fibp:=t;
   x:=x-1;
   dbms_output.put_line(fibn);
 END LOOP;
 END IF;
END;
```

```
Procedure created.

0.24 seconds
```

```
declare
  x number:=10;
begin
  fibonacci(x);
end;
```

```
0
1
1
2
3
5
8
13
21
34

Statement processed.

0.00 seconds
```

FUNCTIONS

A stored function (also called a user function or user-defined function) is a set of PL/SQL statements you can call by name. Stored functions are very similar to procedures, except that a function returns a value to the environment in which it is called. User functions can be used as part of a SQL expression.
A function is same as a procedure except that it returns a value.

Creating a Function

A function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows –

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
   Declaration section
BEGIN
   Execution section
EXCEPTION
  Exception section
END;
```

-CODE TO IMPLEMENT FUNCTION

```
CREATE OR REPLACE
FUNCTION FACTORIAL(x IN OUT NUMBER)
RETURN NUMBER
IS
   val number :=1;
BEGIN
 WHILE x>0
 LOOP
    val:=val*x;
    x:=x-1;
 END LOOP;
 dbms_output.put_line(val);
RETURN val;
END FACTORIAL;
```

```
Statement processed.

1.18 seconds
```

```
declare
  x number:=10;
  t number;
begin
  t:= factorial(x);
end;
```

```
3628800

Statement processed.

0.00 seconds
```

PRACTICAL 13

TRIGGERS

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

Syntax of Triggers

```
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
  Execution section
END;
```

-CODE FOR AFTER TRIGGER

```
CREATE TRIGGER LOGU
        AFTER UPDATE on employee
        FOR EACH ROW
        WHEN (NEW.ESAL > 0)
        BEGIN
        DBMS_OUTPUT.PUT_LINE('Before Updation  : ' || :OLD.EMPNO ||' '||
:OLD.ENAME||' '||:OLD.ESAL||' '||:OLD.EJOB);
        DBMS_OUTPUT.PUT_LINE('After Updation  :'  || :NEW.EMPNO ||' '||
:NEW.ENAME||' '||:NEW.ESAL||' '||:NEW.EJOB);
END;
```

```
Trigger created.
```

0.01 seconds

UPDATE employee SET ENAME='SHUBHKIRTI' WHERE ENAME='SHUBH';

```
Before Updation  : E003 SHUBH 4000500 CTO
After Updation  : E003 SHUBHKIRTI 4000500 CTO

1 row(s) updated.
```

0.00 seconds

-CODE FOR BEFORE TRIGGER

```
CREATE TRIGGER LOGD
        BEFORE DELETE on employee
        FOR EACH ROW
        WHEN (OLD.ESAL > 0)
BEGIN
        DBMS_OUTPUT.PUT_LINE('Deleting : ' || :OLD.EMPNO ||' '|| :OLD.ENAME||'
'||:OLD.ESAL||' '||:OLD.EJOB);
END;
```

```
Trigger created.
```

0.01 seconds

DELETE FROM EMPLOYEE WHERE EMPNO='E010';

```
Deleting  : E010 ANUSHUMAN 4050505 CTO

1 row(s) deleted.
```

0.10 seconds

PRACTICAL 14

PACKAGE

A package is a schema object that groups logically related PL/SQL types, variables, constants, subprograms, cursors, and exceptions. A package is compiled and stored in the database, where many applications can share its contents.

A package always has a specification, which declares the public items that can be referenced from outside the package.
Package consists of two parts
1. Package specification
2. Package body

PACKAGE SPECIFICATION

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

PACKAGE BODY

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package.

SYNTAX OF PACKAGE

PACKAGE SPECIFICATION

```
CREATE [OR REPLACE] PACKAGE package_name
  IS | AS
  [variable_declaration ...]
  [constant_declaration ...]
  [exception_declaration ...]
  [cursor_specification ...]
      [PROCEDURE [Schema..] procedure_name
            [ (parameter {IN,OUT,IN OUT} datatype [,parameter]) ]
      ]
```

```
    [FUNCTION [Schema..] function_name
        [ (parameter {IN,OUT,IN OUT} datatype [,parameter]) ]
        RETURN return_datatype
    ]
END [package_name];
```

PACKAGE BODY

```
CREATE [OR REPLACE] PACKAGE BODY package_name
  IS | AS
  [private_variable_declaration ...]
  [private_constant_declaration ...]
    BEGIN
        [initialization_statement]
        [PROCEDURE [Schema..] procedure_name
            [ (parameter [,parameter]) ]
            IS | AS
                variable declarations;
                constant declarations;
            BEGIN
                statement(s);
            EXCEPTION
                WHEN ...
            END
        ]
        [FUNCTION  [Schema..] function_name
                [ (parameter [,parameter]) ]
                RETURN return_datatype
            IS | AS
                variable declarations;
                constant declarations;
            BEGIN
                statement(s);
            EXCEPTION
                WHEN ...
            END
        ]
    [EXCEPTION
        WHEN built-in_exception_name_1 THEN
            User defined statement (action) will be taken;
    ]
```

END;

-CODE FOR IMPLEMENTING PACKAGE

```
CREATE OR REPLACE PACKAGE stud_pac AS
FUNCTION insert_stu (name VARCHAR2, marks NUMBER)
   RETURN NUMBER;
PROCEDURE stu_details (name varchar2);
PROCEDURE stu_details (rno number);
END stud_pac;
```

```
Package created.

0.42 seconds
```

```
CREATE OR REPLACE PACKAGE BODY stud_pac AS
FUNCTION insert_stu(name VARCHAR2, marks NUMBER)
   RETURN NUMBER IS
   new_rno NUMBER;
BEGIN
    SELECT studentrno.NEXTVAL INTO new_rno FROM dual;
    INSERT INTO student VALUES (new_rno, name, marks);
    stu_details(name);
    RETURN new_rno;
END insert_stu;

PROCEDURE stu_details(sname varchar2) IS
    smarks number;
    r_no number;
BEGIN
    select rno into r_no from student where sname=name;
    select marks into smarks from student where sname=name;
    DBMS_OUTPUT.PUT_LINE('The details of the new student  ');
    DBMS_OUTPUT.PUT_LINE('ROLL NO :'|| r_no );
    DBMS_OUTPUT.PUT_LINE('NAME :'|| sname );
    DBMS_OUTPUT.PUT_LINE('MARKS :'|| smarks );
END stu_details;

PROCEDURE stu_details (r_no number) IS
    smarks number;
```

```
    sname varchar2(30);
BEGIN
    select name into sname from student where rno=r_no;
    select marks into smarks from student where rno=r_no;
    DBMS_OUTPUT.PUT_LINE('The details of the new student  ');
    DBMS_OUTPUT.PUT_LINE('ROLL NO :'|| r_no );
    DBMS_OUTPUT.PUT_LINE('NAME :'|| sname );
    DBMS_OUTPUT.PUT_LINE('MARKS :'|| smarks );
END stu_details;
END stud_pac;
```

```
Package Body created.

0.07 seconds
```

Using the package created

```
DECLARE
   name varchar2(30);
   marks number;
   rno number;
BEGIN
   name:=:NAME;
   marks:=:MARKS;
   rno:=stud_pac.insert_stu(name,marks);
   stud_pac.stu_details(rno);
end;
```

```
The details of the new student
ROLL NO :3
NAME :shubh
MARKS :50
The details of the new student
ROLL NO :3
NAME :shubh
MARKS :50

Statement processed.
```

0.16 seconds

PRACTICAL 15

CASE STATEMENT

-The CASE function lets you evaluate conditions and return a value when the first condition is met (like an IF-THEN-ELSE statement).

CODE FOR CASE STATEMENT IN SQL

```
SELECT roll_no,name,
CASE deptt
  WHEN 'CSE' THEN 'COMPUTER SCIENCE'
  WHEN 'IT' THEN 'INFORMATION TECHNOLOGY'
  ELSE 'UNKNOWN'
END DEPARTMENT
FROM STUDENT95;
```

ROLL_NO	NAME	DEPARTMENT
UE163098	Shubhkirti	COMPUTER SCIENCE
UE163095	Shivank Bali	COMPUTER SCIENCE

2 rows returned in 0.00 seconds Download

CODE FOR CASE STATEMENT IN PL/SQL

```
DECLARE
  grade char(1) ;
BEGIN
  grade:=:GRADE;
  CASE grade
    when 'A' then dbms_output.put_line('Excellent');
    when 'B' then dbms_output.put_line('Very good');
    when 'C' then dbms_output.put_line('Well done');
    when 'D' then dbms_output.put_line('You passed');
    when 'F' then dbms_output.put_line('Better try again');
    else dbms_output.put_line('No such grade');
  END CASE;
END;
```

-INPUT

A

-OUTPUT

```
Excellent

Statement processed.

0.01 seconds
```

CORELATED QUERIES

SQL Correlated Subqueries are used to select data from a table referenced in the outer query. The subquery is known as a correlated because the subquery is related to the outer query. In this type of queries, a table alias (also called a correlation name) must be used to specify which table reference is to be used.

-CODE FOR IMPLEMENTING CORELATED QUERIES

```
SELECT ename,deptno,esal
  FROM employee emp
  WHERE esal >
    (SELECT AVG(esal)
     FROM employee
     WHERE deptno = emp.deptno);
```

ENAME	DEPTNO	ESAL
SOMYADEEP	D002	4550500
SHUBHAM GUPTA	D006	5000500

2 rows returned in 0.00 seconds Download

SEQUENCE

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.

TABLE CREATED FOR SEQUENCE

create table student(rno number(5),name varchar2(30),marks number(5));

```
Table created.
```

0.67 seconds

CREATING SEQUENCE

```
CREATE SEQUENCE studentrno
 MINVALUE 1
 MAXVALUE 99999
 START WITH 1
 INCREMENT BY 1
 CACHE 20;
```

```
Sequence created.
```

0.04 seconds

USING SEQUENCE

insert into student values(studentrno.NEXTVAL,'Shivank',96);

```
1 row(s) inserted.
```

0.04 seconds

insert into student values(studentrno.CURRVAL,'Shivank',96);

```
1 row(s) inserted.
```

0.00 seconds

SELECT * FROM student;

RNO	NAME	MARKS
1	Shivank	96
1	Shivank	96

2 rows returned in 0.00 seconds

ROWID AND ROWNUM

ROWID

For each row in the database, the ROWID pseudocolumn returns the address of the row. Oracle Database rowid values contain information necessary to locate a row:

- The data object number of the object
- The data block in the datafile in which the row resides
- The position of the row in the data block (first row is 0)
- The datafile in which the row resides (first file is 1). The file number is relative to the tablespace.

Usually, a rowid value uniquely identifies a row in the database. However, rows in different tables that are stored together in the same cluster can have the same rowid. Rowid values have several important uses:

- They are the fastest way to access a single row.
- They can show you how the rows in a table are stored.
- They are unique identifiers for rows in a table.

You should not use ROWID as the primary key of a table. If you delete and reinsert a row with the Import and Export utilities, for example, then its rowid may change. If you delete a row, then Oracle may reassign its rowid to a new row inserted later. Although you can use the ROWID pseudocolumn in the SELECT and WHERE clause of a query, these pseudocolumn values are not actually stored in the database. You cannot insert, update, or delete a value of the ROWID pseudocolumn.

IMPLEMENTATION OF ROWID

SELECT ROWID, ename FROM employee;

ROWID	ENAME
AAAE7gAAEAAAAO+AAA	SINDHIYA
AAAE7gAAEAAAAO+AAB	NAMAN
AAAE7gAAEAAAAO+AAC	SOMYADEEP
AAAE7gAAEAAAAO/AAA	SHIVANK BALI
AAAE7gAAEAAAAO/AAB	SHUBHAM GUPTA
AAAE7gAAEAAAAO/AAC	SHUBHKIRTI
AAAE7gAAEAAAAO/AAD	SHASWAT
AAAE7gAAEAAAAO/AAE	PRATIK

8 rows returned in 0.00 seconds Download

ROWNUM

For each row returned by a query, the ROWNUM pseudo column returns a number indicating the order in which Oracle selects the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on.
ROWNUM can be used to limit the number of rows returned by a query, as in this example:
SELECT * FROM employees WHERE ROWNUM < 10;

If an ORDERBY clause follows ROWNUM in the same query, then the rows will be reordered by the ORDERBY clause. The results can vary depending on the way the rows are accessed. For example, if the ORDERBY clause causes Oracle to use an index to access the data, then Oracle may retrieve the rows in a different order than without the index. Therefore, the following statement will not have the same effect as the preceding example:
SELECT * FROM employees WHERE ROWNUM < 11 ORDER BY ename;

If you embed the ORDERBY clause in a subquery and place the ROWNUM condition in the top-level query, then you can force the ROWNUM condition to be applied after the ordering of the rows.
For example, the following query returns the employees with the 3 smallest employee numbers. This is sometimes referred to as top-N reporting:

IMPLEMENTATION OF ROWNUM

SELECT * FROM (SELECT * FROM employee ORDER BY esal DESC)
WHERE ROWNUM <= 5;

EMPNO	ENAME	EJOB	JOININGDATE	ESAL	MANID	DEPTNO
E002	SHUBHAM GUPTA	CEO	12/12/2000	5000500	E001	D006
E009	SOMYADEEP	DBA	04/04/2010	4550500	E005	D002
E003	SHUBHKIRTI	CTO	12/12/2005	4000500	E001	D003
E007	NAMAN	OSD	04/04/2009	3400500	E001	D007
E005	PRATIK	SERVICES MANAGER	04/04/2007	2500500	E003	D004

5 rows returned in 0.00 seconds Download

INDEX

Indexes are **special lookup tables** that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

An index is a performance-tuning method of allowing faster retrieval of records. An index creates an entry for each value that appears in the indexed columns. By default, Oracle creates B-tree indexes.

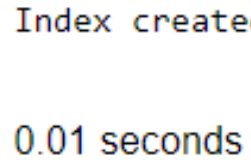
Syntax
The syntax for creating an index in Oracle/PLSQL is:

```
CREATE [UNIQUE] INDEX index_name
 ON table_name (column1, column2, ... column_n)
 [ COMPUTE STATISTICS ];
```

SINGLE-COLUMN INDEXES

A single-column index is created based on only one table column. The basic syntax is as follows.

```
CREATE INDEX index_emp ON EMP(ENAME);
```



UNIQUE INDEXES

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

CREATE UNIQUE INDEX index_emp ON EMP (ENAME);

```
Index created.
```

0.01 seconds

-Unique indexes cannot be created on columns with duplicate values
CREATE UNIQUE INDEX index_emp ON EMP (manid);

```
ORA-01452: cannot CREATE UNIQUE INDEX; duplicate keys found
```

COMPOSITE INDEXES

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

CREATE INDEX index_emp_name_job ON EMP(ENAME,JOB);

```
Index created.
```

0.00 seconds

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.
Should there be only one column used, a single-column index should be the choice.
Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

IMPLICIT INDEXES

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

THE DROP INDEX COMMAND

An index can be dropped using SQL **DROP** command. Care should be taken when dropping an index because the performance may either slow down or improve.

DROP INDEX index_emp;

```
Index dropped.
```

0.01 seconds