Name: Shubham Gupta
Roll no: 1610991834

PRACTICAL-1

AIM: Implementation of indexes, views.

THEORY:

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. When index is created on a column by default the value will be sorted in ascending order.

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.

SINGLE INDEX:

A single index is created based on only one table column.

Syntax:

CREATE INDEX index_name ON table_name(column_name);

```
create table Shubham(id number, name varchar(20), pay integer)
drop table Shubham
insert into Shubham values(1, 'Shubham Gupta',100000);
insert into Shubham values(2, 'Saurabh Thakur',90000);
insert into Shubham values(1, 'Shruti Sheoran',80000);

create index monthly_pay on Shubham(pay);
drop index monthly_pay
select * from Shubham where pay>10000
```

Results Explain Describe Saved SQL History

ID	NAME	PAY
1	Shruti Sheoran	80000
2	Saurabh Thakur	90000
1	Shubham Gupta	100000

3 rows returned in 0.00 seconds Download

COMPOSITE INDEX:

A composite index is created based on more than one table column.

Syntax:

CREATE INDEX index_name ON table_name(column1_name, column2_name);

UNIQUE INDEX: 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.

Syntax:

CREATE UNIQUE INDEX index_name ON table_name(column_name);

REVERSE INDEX: In reverse indexes data is sorted in descending order.

Syntax:

CREATE INDEX index_name ON table_name(column_name) REVERSE;

```
create index monthly_pay on Shubham(pay);
drop index monthly_pay
select * from Shubham where pay>10000
delete from shubham where name='Shruti Sheoran'
create unique index id_index on Shubham(id) reverse
select id, name from shubham
```

Results Explain Describe Saved SQL History

ID	NAME
3	Shruti Sheoran
1	Shubham Gupta
2	Saurabh Thakur

3 rows returned in 0.00 seconds Download

BIT-MAP INDEX: This index is created on those columns which have only two values Example: Gender, Marital Status etc.

Syntax:

CREATE BITMAP INDEX index_name ON table_name(column_name);

```
create bitmap index gender_index on Shubham1(gender)
```

```
ORA-00439: feature not enabled: Bit-mapped indexes
```

View

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.

SINGLE VIEW:

A single view includes only one table column.

Syntax:

CREATE VIEW view_name AS SELECT column1

FROM table_name WHERE [condition];

COMPOSITE VIEW:

A composite view includes more than one table column.

Syntax:

CREATE VIEW view_name AS SELECT column1, column2 ….

FROM table_name WHERE [condition];

COMPLETE VIEW:

A complete view includes every column of the table.

Syntax:

CREATE VIEW view_name AS SELECT *

FROM table_name WHERE [condition];

```
create view simple_view as select * from shubham where pay>1000
select * from simple_view
```

Results | Explain | Describe | Saved SQL | History

ID	NAME	PAY
3	Shruti Sheoran	80000
2	Saurabh Thakur	90000
1	Shubham Gupta	100000

3 rows returned in 0.01 seconds Download

FORCED VIEW:

A forced view is created before the creation of the table. As soon as table is created the view will perform its normal functionality.

Syntax:

CREATE FORCE VIEW view_name AS SELECT column1, column2, …

FROM table_name WHERE [condition];

```
create force view sg as select name from guptaji;
create table guptaji(name varchar(20), id integer)
insert into guptaji values('Shubham Gupta', 1)
select * from sg
```

Results | Explain | Describe | Saved SQL | History

NAME
Shubham Gupta

1 rows returned in 0.01 seconds Download

MATERIALIZED VIEW

Materialized views are stored on disk and are updated periodically based upon the query definition.

Syntax:

CREATE MATERIALIZED VIEW view_name AS SELECT column1, column2, …

FROM table_name WHERE [condition];

```
create materialized view materialized_view as select name from Shubham1 where gender='M'
select * from materialized_vieew
```

Results Explain Describe Saved SQL History

NAME
Shubham Gupta

1 rows returned in 0.01 seconds Download

Name: Shubham Gupta
Roll no: 1610991834

PRACTICAL-2

AIM: Implementation of sequences, clusters.

THEORY:

SEQUENCE

Sequence is a set of integers 1, 2, 3, … that are generated to produce unique values on demand.

Sequences are frequently used in databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.

The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when exceeds max_value

Syntax:

CREATE SEQUENCE sequence_name

START WITH initial_value

INCREMENT BY increment_value

MINVALE minimum value

MAXVALUE maximum value

CYCLE | NOCYCLE;

INSERT INTO table_name(sequence_name.NEXTVAL);

```
create sequence Shubham_id
start with 1
increment by 1
maxvalue 30;
```

Results Explain Describe Saved SQL History

```
Sequence created.

0.03 seconds
```

Name: Shubham Gupta
Roll no: 1610991834

CLUSTER

A **cluster** is a schema object that contains data from one or more tables, all of which have one or more columns in common. A cluster ties data values to disk location.
A cluster key is used to group data together. All rows of all tables with the same value of the cluster key are stored in the same data block.

Syntax:

CREATE CLUSTER cluster_name(column_name datatype);

CREATE INDEX index_name ON CLUSTER cluster_name;

CREATE TABLE table_name(column_name datatype, …)

CLUSTER cluster_name(column_name);

To see list of cluster:

SELECT * FROM USER_CLUSTERS;

```
create cluster shubham_cluster(deptno varchar2(20));
create index shubham_index on cluster shubham_cluster;
create table shubham_emp(name varchar2(20), deptno varchar2(20)) cluster shubham_cluster(deptno);
create table shubham_dept(deptid  integer, deptno varchar2(20)) cluster shubham_cluster(deptno);
select * from user_clusters;
```

Results | Explain | Describe | Saved SQL | History

CLUSTER_NAME	TABLESPACE_NAME	PCT_FREE	PCT_USED	KEY_SIZE	INI_TRANS	MAX_TRANS	INITIAL_EXTENT	NEXT_EXTENT
SHUBHAM_CLUSTER	USERS	10	-	-	2	255	65536	1048576

1 rows returned in 0.09 seconds Download

Name: Shubham Gupta
Roll no: 1610991834

PRACTICAL-3

AIM: Introduction to PL/SQL Concepts, its features and shows how PL/SQL meets the challenges of database programming, and how you can reuse techniques that you know from other programming languages

THEORY:

PL/SQL

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. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.
PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle.

BLOCK STRUCTURE:
1. DECLARE
2. BEGIN
3. EXCEPTION
4. END

BEGIN and END are mandatory blocks other two are optional.

PL/SQL has the following features −

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

PL/SQL meets the challenges of database programming −

- SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.
- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.

- PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
- PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- PL/SQL provides high security level.
- PL/SQL provides access to predefined SQL packages.

PL/SQL reuses techniques used by other programming languages −

- PL/SQL provides access to predefined SQL packages.
- PL/SQL provides support for Object-Oriented Programming.
- PL/SQL provides support for developing Web Applications and Server Pages.

```
declare
    user varchar(20):=:user;
begin
    dbms_output.put_line('Hello ' || user);
end
```

Results Explain Describe Saved SQL History

```
Hello Shubham Gupta

Statement processed.

0.00 seconds
```

Name: Shubham Gupta
Roll no: 1610991834

PRACTICAL-4

AIM: How to structure the flow of control through a PL/SQL program. Connect the statements by simple but powerful control structures that have a single entry and exit point. Collectively, these structures can handle any situation. Their proper use should leads naturally to a well-structured program.

THEORY:

FLOW CONTROL

IF-THEN-ELSE statement

IF statement adds the keyword **ELSE** followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.

IF-THEN-ELSIF statement

It allows you to choose between several alternatives.

```
Shubham Gupta

DECLARE
    java number :=:java;
    oracle number :=:oracle;
    C number:=:C;
    CPP number :=:CPP;
    avg1 number;
BEGIN
    avg1 :=(java + oracle + C + CPP)/4;
    if(avg1 > 80) then
        dbms_output.put_line('First Division');
        elsif(avg1>60 and avg1 < 80) then
             dbms_output.put_line('Second Division');
        else
           dbms_output.put_line('Fail');
    end if;
    end;
```

Results Explain Describe Saved SQL History

```
hello
First Division

Statement processed.

0.01 seconds
```

```
DECLARE
   balance number:=10000;
   choice number:=:choice;
   cash number:=:cash;
BEGIN
   dbms_output.put_line('Enter 1 for deposit 2 for withdraw');
   if(choice=1) then
      balance:= balance + cash;
      dbms_output.put_line(balance);
   elsif(choice=2) then
       balance:= balance - cash;
       dbms_output.put_line(balance);
   else
       dbms_output.put_line('fail');
   end if;
END;
```

Results Explain Describe Saved SQL History

```
Enter 1 for deposit 2 for withdraw
12000

Statement processed.

0.00 seconds
```

CASE statement

Like the IF statement, the **CASE statement** selects one sequence of statements to execute.

However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions. A selector is an expression whose value is used to select one of several alternatives.

```
Shubham Gupta
DECLARE
    choice number:=:choice;
BEGIN
    CASE choice
    when '1' then
        dbms_output.put_line('ONE');
    when '2' then
        dbms_output.put_line('TWO');
    else
        dbms_output.put_line('Greater than two');
    end case;
end
```

Results Explain Describe Saved SQL History

```
TWO

Statement processed.

0.00 seconds
```

PL/SQL WHILE LOOP

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
Shubham Gupta
DECLARE
    num number:=:num;
BEGIN
    while num > 0 loop
       dbms_output.put_line(num);
       num:=num-1;
    end loop;
end
```

Results Explain Describe Saved SQL History

```
5
4
3
2
1

Statement processed.
```

0.00 seconds

```
Shubham Gupta

DECLARE
   num number:=:num;
   sums number:=0;
BEGIN
   while num > 0 loop
      sums:=sums+num;
      num:=num-1;
   end loop;
   dbms_output.put_line(sums);
end
```

Results Explain Describe Saved SQL History

```
15

Statement processed.
```

0.01 seconds

PL/SQL FOR LOOP

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

```
Shubham Gupta
DECLARE
   num number:=:number;
   fact number:=1;
   i number;
BEGIN
   for i in 1..num loop
      fact:= fact*i;
   end loop;
dbms_output.put_line(fact);
   end;
```

Results Explain Describe Saved SQL History

```
24

Statement processed.

0.00 seconds
```

```
Shubham Gupta
declare
    num number:=:num;
    i number;
begin
   for i in 1..num loop
      dbms_output.put_line(i);
   end loop;
end
```

Results Explain Describe Saved SQL History

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Statement processed.

0.00 seconds
```

Name: Shubham Gupta
Roll no: 1610991834

PRACTICAL-5,6

AIM: The Run-time errors arise from design faults, coding mistakes, hardware failures, and many other sources. Although you cannot anticipate all possible errors, you can plan to handle certain kinds of errors meaningful to your PL/SQL program. With PL/SQL mechanism called exception handling design "bulletproof" program so that it can continue operating in the presence of errors.

THEORY:

Exceptions

PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition. There are two types of exceptions −

- System-defined exceptions
- User-defined exceptions

The general syntax for exception handling is as follows:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
    exception1-handling-statements
    ........
  WHEN others THEN
    exception3-handling-statements
END;
```

System-defined Exceptions

The system exception is raised by PL/SQL run-time when it detects an error. For example, **NO_DATA_FOUND** exception is raised if you select a non-existing record from the database.

Some pre-defined exceptions:

- **CASE_NOT_FOUND**

  SQL CODE: -6592

  It is raised when none of the choices in the WHEN clause of a CASE statement is selected, and there is no ELSE clause.

- **DO_NOT_FOUND**

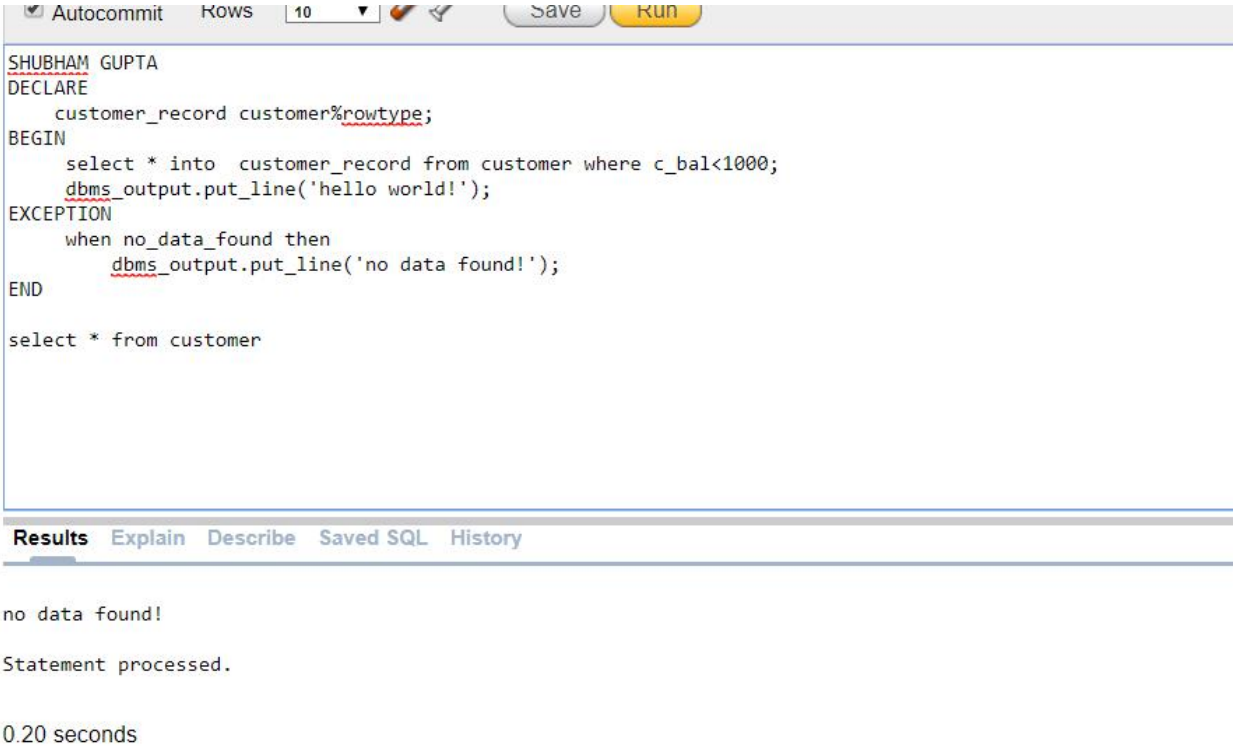
  SQLCODE: -100

  It is raised when a SELECT INTO statement returns no rows.

- **INVALID_CURSOR**

  SQLCODE: -1001

  It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.



Name: Shubham Gupta
Roll no: 1610991834

PRACTICAL-7

AIM: PL/SQL give you control to make your own exception base on oracle rules. User define exception must be declare yourself and RAISE statement to raise explicitly. Use PL/SQL user defined exception to make your own exception.

THEORY:

User-defined Exceptions

The user-defined exception is defined by user in a specific application. User can map exception names with specific Oracle errors using the **EXCEPTION_INIT pragma**.

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command **RAISE**.

SYNTAX:

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
  statement;
END;
```

EXCEPTION_INIT pragma.

Pragma is a keyword directive to execute proceed at compile time. pragma EXCEPTION_INIT function take this two argument,

exception_name

error_number

You can define pragrma EXCEPTION_INIT in DECLARE BLOCK on your program.

PRAGMA EXCEPTION_INIT(exception_name, -error_number);

exception_name is character string up to 2048 bytes support and error_number is a negative integer range from -20000 to -20999.

SYNTAX:

DECLARE

user_define_exception_name EXCEPTION;

PRAGMA EXCEPTION_INIT(user_define_exception_name, -error_number);

BEGIN

statement(s);

IF condition THEN

RAISE user_define_exception_name;

END IF;

EXCEPTION

WHEN user_define_exception_name THEN

END;

```
DECLARE
  amount decimal:=:amount;
  not_enough EXCEPTION;
  balance decimal:=50000.0;

BEGIN
  if(amount>balance) then
    RAISE not_enough;
  else
    balance:=balance-amount;
    Dbms_Output.put_line('Available Balance is: ' || balance);
  end if;

EXCEPTION
  WHEN not_enough THEN
    Dbms_Output.put_line('Sorry Can not process your transaction, not enough balance');

END;
```

Results Explain Describe Saved SQL History

```
Sorry Can not process your transaction, not enough balance

Statement processed.
```

```
SHUBHAM GUPTA
DECLARE
    V_dept dept%rowtype;
    V_deptno dept.deptno%type;
    Child_found exception;
    Pragma Exception_init(Child_found, -2292);
BEGIN
    V_deptno:=:deptno;
    delete from dept where V_deptno=deptno;
EXCEPTION
    when child_found then
        dbms_output.put_line('Child found in other table, cant delete');
END;

select *  from dept;
```

Results Explain Describe Saved SQL History

```
Child found in other table, cant delete
```