KGiSL Institute of Technology

(Approved by AICTE, New Delhi; Affiliated to Anna University, Chennai)
Recognized by UGC, Accredited by NBA (IT)
365, KGiSL Campus, Thudiyalur Road, Saravanampatti, Coimbatore – 641035.

CS3481 – DATABASE MANAGEMENT SYSTEMS

NAME : ……

REG. NO. : ……

COURSE : ……

SEMESTER : …….

BATCH : …….

KGiSL Institute of Technology

(Approved by AICTE, New Delhi; Affiliated to Anna University, Chennai)
Recognized by UGC, Accredited by NBA (IT)
365, KGiSL Campus, Thudiyalur Road, Saravanampatti, Coimbatore – 641035.

NAME :

CLASS :

UNIVERSITY REG NO :

Certified that, this is a bonafide record of work done by ……...

of …….. branch in **DATABASE MANAGEMENT SYSTEMS LABORATORY**, during fourth semester of academic year 2022-2023.

Faculty In-charge **Head of the Department**

Submitted during Anna University Practical Examination held on …… at KGiSL Institute of Technology, Coimbatore – 641 035.

Internal examiner **External Examiner**

S.NO	DATE	LIST OF THE EXPERIMENTS	PAGE NO	MARKS	SIGNATURE
1		Create database and retrieve information from the database			
2		Database languages			
3		Constraints			
4		Nested queries and aggregate functions			
5		Sub queries and Join operations			
6		Join Functions			
7		User defined functions and stored procedures			
8		Triggers			
9		View and Index			
10		Create an XML database and validate it using XML schema			
11		Document, column and graph based data using NOSQL database tools			
12		Develop a simple GUI based database application			
13		Mini Projects			

EX NO : 01 DATE :	**CREATE DATABASE AND RETRIEVE INFORMATION FROM THE DATABASE**

AIM:

To create a database and writing SQL queries to retrieve information from the database.

LIST OF DDL COMMANDS:

1) CREATE TABLE Command.
2) DESCRIBE Command.
3) ALTER TABLE Command.
4) DROP TABLE Command.
5) TRUNCATE TABLE Command.

LIST OF DML COMMANDS:

1) INSERT Command.
2) SELECT Command.
3) UPDATE Command.
4) DELETE Command.

LIST OF DCL COMMANDS:

1) GRANT Command.
2) REVOKE Command.

LIST OF TCL COMMANDS:

1) COMMIT Command.
2) ROLLBACK Command.

PROCEDURE TO CREATE AN USER:

1. Create your own user first.
2. Create one user name with password.
3. If you are in any other system other than sys manager, do the following steps:
4. SQL> connect system/manager; It'll display as "CONNECTED".
5. SQL> create user com1 identified by student; Where, Com1 - user name. Student - password.
6. CREATE SESSION must be granted by the system manager to the user (com1).
7. SQL>connect com1/student; Connected.

DESCRIPTION OF DDL COMMANDS:

i) CREATE:

This command is used to create a table.

SYNTAX:

create table <table name> (column_name1 datatype size, column_name2 datatype size...);
create table tablename (column_name data_ type constraints, ...)

EXAMPLE:

SQL> create table labour(empno number(5) primary key, ename char(20), dept (10), salary number(7));

ii) DESCRIBE:

This command displays the table structure.

SYNTAX:

SQL>desc tablename;

EXAMPLE:

desc labour;

iii)INSERT:

This command is used to insert the values in to the table.

SYNTAX:

insert into <table name> values (value1,value2,...);

EXAMPLE:

SQL> insert into labour values(101,'Swetha','CSE',12000);
SQL> insert into labour values(102,'Pooja','MECH',10000);

iii) SELECT:

This command is used to display the entire table.

SYNTAX:

Select * from <table_name>;

EXAMPLE:

SQL> select * from labour;

PROCEDURE:

TYPES OF SELECT COMMAND:

1) GROUP BY
2) ORDER BY
3) HAVING and BETWEEN
4) DISTINCT

SET OPERATORS:

1) UNION
2) UNION ALL
3) INTERSECT
4) MINUS

DESCRIPTION OF COMMANDS:

SQL> create table employee(eid number(5) primary key, name char(20), age number(3), salary number(8),dept_id number(5));

SQL> insert into employee values(1,' Swetha',20,5000,2);

SQL> select * from employee;

EID	NAME	AGE	SALARY	DEPT_ID
1	Swetha	20	5000	2
2	Sharmi	25	7000	2
3	Yazhini	20	4000	4

i) GROUP BY:

This command is used to display the group of values.

SYNTAX:

Select column1, column2... from <table_name> group by column1 having condition;

EXAMPLE:

SQL> select dept_id,avg(salary) from employee group by dept_id;

DEPT_ID	AVG(SALARY)
1	5000
2	7000
3	4000

ii) ORDERBY:

This command is used to display the column in an order (ascending order is default in all SQL queries).

SYNTAX:

Select * from <table_name> order by condition;

EXAMPLE:

SQL> select * from employee order by name;

EID	NAME	AGE	SALARY	DEPT_ID
2	Sharmi	25	7000	2
1	Swetha	20	5000	2
3	Yazhini	20	4000	4

iii) HAVING and BETWEEN:

This command is used to display the value having a condition.

SYNTAX:

Select column1,column2.. from <table_name> where column having condition;

EXAMPLE:

SQL> select dept_id,avg(salary) from employee group by dept_id having avg(salary)>=5000;

DEPT_ID	AVG(SALARY)
1	5000
2	7000

BETWEEN:

This command is used to display the values between any conditions.

SYNTAX:

Select column_name from table_name group by column_name between condition;

EXAMPLE:

SQL>select dept_id,avg(salary) from employee group by dept_id having avg(salary) between 5000 and 7000;

DEPT_ID	AVG(SALARY)
1	5000
2	7000

iv) DISTINCT:

This command is used to eliminate the duplicate entries from the table.

SYNTAX:

Select distinct * from table_name;

EXAMPLE :

SQL> select distinct age from employee order by age;

AGE
20
25

SET OPERATORS:

SQL> create table student(rollno number(3) primary key, name char(20), mark1 number(2), mark2 number(2),mark3 number(2));

SQL> insert into student values(1,' Pooja',90,60,90);

Table1: STUDENT

ROLL NO	NAME	MARK1	MARK2	MARK3
1	Pooja	90	60	90
2	Pavithra	80	70	50
3	Shankari	90	80	50

SQL> create table stud(rollno number(3) primary key, name char(20), mark1 number(2), mark2 number(2),mark3 number(2));

SQL> insert into stud values(1,' Pooja',90,60,90);

Table2: STUD

ROLL NO	NAME	MARK1	MARK2	MARK3
1	Pooja	90	60	90
2	Pavithra	80	70	50
4	Maha	80	80	60

i)UNION:

The union operator returns all distinct rows selected by the component queries.

SQL>select * from student union select * from stud;

OUTPUT:

ROLL NO	NAME	MARK1	MARK2	MARK3
1	Pooja	90	60	90
2	Pavithra	80	70	50
3	Shankari	90	80	50
4	Maha	80	80	60

ii)UNION ALL:

The union all operator returns all rows selected by both queries including duplicate.

SQL>select * from student unionall select * from stud;

OUTPUT:

ROLL NO	NAME	MARK1	MARK2	MARK3
1	Pooja	90	60	90
2	Pavithra	80	70	50
3	Shankari	90	80	50
1	Pooja	90	60	90
2	Pavithra	80	70	50
4	Maha	80	80	60

iii)INTERSECT:

This operator returns only rows that are common to both queries.

SQL>select * from student intersect select * from stud;

OUTPUT:

ROLL NO	NAME	MARK1	MARK2	MARK3
1	Pooja	90	60	90
2	Pavithra	80	70	50

v)MINUS:

This operator returns all distinct rows selected only by the first query and not by the second query.

SQL>select * from student minus select * from stud;

OUTPUT:

ROLL NO	NAME	MARK1	MARK2	MARK3
3	Shankari	90	80	50

RESULT:

Thus the above program for creation of a database and writing SQL queries to retrieve information from the database is executed and the output is verified successfully.

EX NO : 02 DATE :	**CONSTRAINTS**

AIM:

To study and execute the constraints using SQL.

LIST OF CONSTRAINTS:

It is a declarative way of defining a business rule for a column of a table. The types are

1) NULL CONSTRAINTS.
2) NOT NULL CONSTRAINTS.
3) PRIMARY KEY CONSTRAINTS.
4) CHECK CONSTRAINTS.
5) FOREIGN KEY CONSTRAINTS.

1) DOMAIN INTEGRITY CONSTRAINTS:

i) NOT NULL Constraint:

When a not null constraint is enforced on a column or a group of columns in a table, oracle will not allow null values in the columns.

Null value is different from zero value. Also a null value in one column is not equivalent to the null value in another column.

SYNTAX:

SQL> create table table_name(column_name data_type not null, ...);

EXAMPLE:

SQL> create table student(sno number(10) NOT NULL, name CHAR(20) NOT NULL, address varchar2(10));

After this, oracle will not allow inserting a row with null values in the columns sno and name.

SQL> insert into master values(1,'Swetha','Chennai');

OUTPUT :

Sno	Name	Address
1	Swetha	Chennai
2	Madhu	Ooty
3	Harshini	Coimbatore

ii)CHECK Constraint:

If there is a necessity to enforce an integrity rule based on a logical or a Boolean expression; we can use the CHECK constraint.

EXAMPLE:

SQL>create table student(roll_no number(3) check (roll_no between 101 and 200), name char(10),address varchar2(20));

Checks the roll number between 101 and 20.

OPERATORS USED IN CHECK CONSTRAINT:

1) IN
2) NOT IN
3) BETWEEN
4) LIKE

LIKE : is used to perform pattern matching. There are two patterns % and _ (underscore).

(a) % represents any number of spaces or characters.

(b) _ represents one space or character.

2) ENTITY INTEGRITY CONSTRAINTS:

i)UNIQUE CONSTRAINTS:

Unique key constraint is used to prevent duplication of values within the row of a specified column or a group of columns in a table.

This does not allow a duplicate value in a column or set of columns.

SYNTAX:

SQL>create table tname (column name data type unique……..));

EXAMPLE:

SQL>create table tname(roll_no number(4) unique, name char(10), address varchar2(10));

At the output oracle will not allow the same entry inside the ROLL_NO column.

ii)PRIMARY KEY CONSTRAINTS:

A primary key is one or more columns in a table which identifies each row in the table uniquely.

This constraint does not allow NULL values and duplicate values across the columns comprising the primary key.

SYNTAX:

SQL>create table tname(col name data type primary key……));

EXAMPLE:

SQL>create table stud(roll_no number(4) primary key, name char(10), address varchar2(20));

OUTPUT :

ROLL NO	NAME	ADDRESS
101	Sharmi	Ooty
102	Pavithra	Coimbatore
103	Shankari	Bangalore

3) REFERENTIAL INTEGRITY CONSTRAINTS:

Referential integrity constraints are used to establish a parent-child relationship between two tables having a common column.

To implement this integrity the column should be defined as primary key in the parent table and the same column as a foreign key in the child table.

SYNTAX:

SQL>create table tname(col name data type references<ref tname>ON <delete/update>CASCADE);

(or)

SQL> create table tname(col1 name data type, constraint constraint_name foreign key(col1 name) references other_table_name(col name), col2 name data type,….);

EXAMPLE:

SQL>create table master(id number(4) foreign key references student(roll_no), mark number(3));

(or)

SQL>create table master(id number(4), constraint fkey foreign key(id) references student(roll_no),mark number(3));

Here the foreign key ID references the primary key ROLL_NO from the table student.

It allows each value in a column or set of columns match a value in related table's unique or primary key.

OUTPUT :

ID	NAME	MARK
1	Sri	90
2	Maha	95
3	Vyshka	80

RESULT:

Thus the above program for studying and executing the integrity constraints has been executed successfully and the output is verified.

EX NO : 03 DATE :	**NESTED QUERIES AND AGGREGATE FUNCTIONS**

AIM:

To study and execute the queries for nested queries &aggregate functions.

PROCEDURE:

Writing and Practice of Simple Queries.

1. Get the description of EMP table.

SQL>desc emp;

2. Get the description DEPT table.

SQL>desc dept;

3.List all employee details.

SQL>select * from emp;

4.List all employee names and their salaries, whose salary lies between 1500/- and 3500/- both inclusive.

SQL>select ename from emp where sal between 1500 and 3500;

5. List all employee names and their and their manager whose manager is 7902 or 7566 0r 7789.

SQL>select ename from emp where mgr in(7602,7566,7789);

6. List all employees which starts with either J or T.

SQL>select ename from emp where ename like 'J%' or ename like 'T%';

7. List all employee names and jobs, whose job title includes M or P.

SQL>select ename,job from emp where job like 'M%' or job like 'P%';

8. List all jobs available in employee table.

SQL>select distinct job from emp;

9. List all employees who belongs to the department 10 or 20.

SQL>select ename from emp where deptno in (10,20);

10. List all employee names , salary and 15% rise in salary.

SQL>select ename , sal , sal+0.15* sal from emp;

11. List minimum , maximum , average salaries of employee.

SQL>select min(sal),max(sal),avg(sal) from emp;

12. Find how many job titles are available in employee table.

SQL>select count (distinct job) from emp;

13. What is the difference between maximum and minimum salaries of employees in the organization?

SQL>select max(sal)-min(sal) from emp;

14. Display all employee names and salary whose salary is greater than minimum salary of the company and job title starts with 'M'.

SQL>select ename,sal from emp where job like 'M%' and sal >(select min (sal) from emp);

15. Find how much amount the company is spending towards salaries.

SQL>select sum (sal) from emp;

16. Display name of the dept. With deptno 20.

SQL>select ename from emp where deptno = 20;

17. List ename whose commission is NULL.

SQL>select ename from emp where comm is null;

18. Find no.of dept in employee table.

SQL>select count (distinct ename) from emp;

19. List ename whose manager is not NULL.

SQL>select ename from emp where mgr is not null;

OUTPUT :

ENAME	DEPT	JOB	SALARY	MANAGER

Swetha	10	Admin	5000	7902
Madhu	20	Sales	3000	7566
Sri	30	Sales	5000	7789
Harshini	40	R&D	4500	7606

ENAME	DEPT	JOB	SALARY	MANAGER

ENAME	DEPT	JOB	SALARY	MANAGER
Madhu	20	Sales	3000	7566

ENAME	DEPT	JOB	SALARY	MANAGER
Swetha	10	Admin	5000	7902
Madhu	20	Sales	3000	7566
Sri	30	Sales	5000	7789

ENAME	DEPT	JOB	SALARY	MANAGER
Swetha	10	Admin	5000	7902
Madhu	20	Sales	3000	7566

JOB	SUM(SALARY)
Admin	5000
Sales	7500
R&D	5000

ENAME	SALARY	MANAGER
Harshini	4500	7606

DEPT	COUNT(ENAME)
10	1
20	1
30	1

SOFTWARE:

ENAME	BILL	PLACE	PROF 1	STACK
C	NS	Coimbatore	10	100
JAVA	PS	Ooty	20	200
C++	RS	Bangalore	30	300

ENAME	COUNT
C	1
JAVA	1
C++	1

TITLE	COUNT
NS	1
PS	2
RS	3

TITLE	ENAME	PLACE

Writing Queries using GROUP BY and other clauses.

To write queries using clauses such as GROUP BY, ORDER BY, etc. and retrieving information by joining tables.

Source tables: emp, dept, programmer, software, study.

Order by: The order by clause is used to display the results in sorted order.

Group by: The attribute or attributes given in the clauses are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.

Having: SQL applies predicates (conditions) in the having clause after groups have been formed, so aggregate function be used.

1. Display total salary spent for each job category.

SQL>select job,sum (sal) from emp group by job;

2. Display lowest paid employee details under each manager.

SQL>select ename, sal from emp where sal in (select min(sal) from emp group by mgr);

3. Display number of employees working in each department and their department name.

SQL>select dname, count (ename) from emp, dept where emp.deptno=dept.deptno group by dname;

4. Display the sales cost of package developed by each programmer.

SQL>select pname, sum(scost) from software group by pname;

5. Display the number of packages sold by each programmer.

SQL>select pname, count(title) from software group by pname;

6. Display the number of packages in each language for which the development cost is less than thousand.

SQL>select devin, count(title) from software where dcost<1000 group by devin;

7. Display each institute name with number of students.

SQL>select splace, count(pname) from study group by splace;

8. How many copies of package have the least difference between development and selling cost, were sold?

SQL>select sold from software where scost – dcost=(select min(scost – dcost) from software);

9. Which is the costliest package developed in Pascal.

SQL>select title from software where devin = 'PASCAL' and dcost = (select max(dcost)from software where devin = 'PASCAL');

10. Which language was used to develop most no .of packages.

SQL>select devin, count (*) from software group by devin having count(*) = (select max(count(*)) from software group by devin);

11.Who are the male programmers earning below the average salary of female programmers?

SQL>select pname from programmer where sal<(select avg(sal) from programmer where sex ='F') and sex = 'M';

12. Display the details of software developed by the male programmers earning more than 3000/- .

SQL>select programmer.pname, title, devin from programmer, software where sal >3000 and sex ='M' and programmer.pname = software.pname;

13. Display the details of software developed in c language by female programmers of pragathi.

SQL>select software.pname, title, devin, scost, dcost, sold from programmer, software, study where devin = 'c' and sex ='F' and splace = 'pragathi' and programmer.pname = software.pname and software.pname = study.pname;

14. Which language has been stated by the most of the programmers as proficiency one?

SQL>select prof1, count(*) from programmer group by prof1 having count (*) = (select max (count(*)) from programmer group by prof1);

OUTPUT :

UPPER(ENAME)
C
JAVA

C++

SYSDATE
17.12.2021
18.01.2022
14.02.2022
16.03.2022

SYSDATE
17.12.2021
18.01.2022
14.02.2022

LOWER
Price
Price 1
Price 2

DAY	SYSDATE
Wednesday	18.02.2022
Wednesday	16.08.2022

PROF%	ROUND(PROF 1)
19.5	20
19.1	19
18.2	18
17.7	18

ENAME	SALARY	PLACE
C	5000	Coimbatore
JAVA	3500	Ooty
C++	5000	Bangalore

ENAME	COUNT
C	2
JAVA	1
C++	1

TITLE	ENAME
NS	100
PS	200
RS	300

Writing Nested Queries.

To write queries using Set operations and to write nested queries.

Set Operations:

UNION - OR
INTERSECT - AND
EXCEPT - - NOT

NESTED QUERY:- A nested query makes use of another sub-query to compute or retrieve the information.

1. Find the name of the institute in which the person studied and developed the costliest package.

SQL>select splace, pname from study where pname = (select pname from software where scost =(select max (scost) from software);

2. Find the salary and institute of a person who developed the highest selling package.

SQL>select study.pname, sal, splace from study, programmer where study.pname =programmer.pname and study.pname = (select pname from software where scost = (select max(scost) from software));

3. How many packages were developed by the person who developed the cheapest package.

SQL>select pname, count (title) from software where dcost = (select min(dcost) from software)group by pname;

4. Calculate the amount to be recovered for those packages whose development cost has not yet recovered.

SQL>select title , (dcost-scost) from software where dcost> scost;

5. Display the title, scost, dcost, difference of scost and dcost in the descending order of difference.

SQL>select title, scost, dcost, (scost - dcost) from software descending order by (scost-dcost);

6. Display the details of those who draw the same salary.

SQL>select p.pname, p.sal from programmer p, programmer t where p.pname<>t.pname and p.sal= t.sal;

(or)

SQL>select pname,sal from programmer t where pname<>t.pname and sal= t.sal;

OUTPUT:

Power Function : 4
TAN Function : 1.557
COS Function : 0.5403
ASCII Function(a) : 97
ASCII Function(A) : 65
Power Function(2,3) : 8
SION Function :
1 (45)
-1 (-45)
Exp Function :
Exp(O) - 1.0
Exp(1) – 2.71828
Floor Function :
45(45.43)
25(25.75)
UNICODE Function : Atlander – 65
REVERSE Function :
SQL Introduction
NoitcudortnI LQS

WORKING WITH SQL COMMANDS:

SINGLE ROW FUNCTIONS:

SQL>select add_months(sysdate,40)from dual;
SQL>select sysdate from dual;
SQL>select last_day(sysdate)from dual;
SQL>select months_between(sysdate,'4-july-1989')from dual;
SQL>select next_day(sysdate,'tuesday')from dual;
SQL>select next_day(sysdate,'friday')from dual;
SQL>select next_day('17-feb-2010','tuesday')from dual;
SQL>select round('sysdate,'year')from dual;
SQL>select round('sysdate,'month')from dual;
SQL>select greatest('10-jan-2008','tuesday')from dual;
SQL>select greatest('10-jan-2008','10-jan-2009')from dual;
SQL>select greatest('sysdate','10-jan-2009')from dual;
SQL>select greatest('10-jan-2008','sysdate','10-jan-2009')from dual;

SQL>select chr(65)from dual;
SQL>select concat('bhava','tharani')from dual;
SQL>select lower('PRAKASH')from dual;
SQL>select lpad(type,4,'*')from turbo;
SQL>select rpad(type,3,'*')from turbo;
SQL>select ltrim('xabcde1jo,'x')from dual;
SQL>select ltrim('xxxxr','x')from duaL;
SQL>select rtrim('xxxrxxx','x')from dual;
SQL>select replace('tamil','t',T')from dual;
SQL>select replace('main','streat')from dual;
SQL>select substr('abcd',2)from dual;
SQL>select upper('type')from dual;
SQL>select ascii('a')from dual;
SQL>select ascii('A')from dual;
SQL>select instr('haifriends','en',3,1) from dual;
SQL>select instr('this is test','is'3,1)from dual;
SQL>select instr('this is test','is')from dual;
SQL>select instr('this is test','is',1,1)from dual;
SQL>select instr('haifriends','en',0) from dual;
SQL>select instr('coso','o',0)from dual;
SQL>select instr('coso co','o',4)from dual;
SQL>select instr('cos0 c0',0) from dual;
SQL>select length('rtq gfdg')from dual;
SQL>select abs(-65.8)from dual;
SQL>select abs(65.8)from dual;
SQL>select ceil(45.04)from dual;
SQL>select ceil(45.90)from dual;
SQL>select ceil(45.00)from dual;
SQL>select cos(45)from dual;
SQL>select cos(45*(2217/180))from dual;
SQL>select cos(0)from dual;
SQL>select cosh(0)from dual;
SQL>select exp(0)from dual;
SQL>select exp(5)from dual;
SQL>select floor(43.46)from dual;
SQL>select exp(1)from dual;
SQL>select ln(1)from dual;
SQL>select mod(12,5)from dual;
SQL>select power(2,2)from dual;
SQL>select power(2,3)from dual;
SQL>select round(45.01)from dual;
SQL>select sign(-45)from dual;
SQL>select sign(45)from dual;
SQL>select tan(0)from dual;
SQL>select tanh(0)from dual;

Writing Queries using functions.

AIM:

To write queries using single row functions and group functions.

1. Display the names and dob of all programmers who were born in january.

SQL>select pname , dob from programmer where to_char (dob,'MON')='JAN';

2. Calculate the experience in years of each programmer and display along with programmer name in descending order.

SQL>select pname, round (months_between(sysdate, doj)/12, 2) "EXPERIENCE"from programmer order by months_between (sysdate, doj) desc;

3. List out the programmer names who will celebrate their birthdays during current month.

SQL>select pname from programmer where to_char(dob,'MON') like to_char (sysdate, 'MON');

4. Display the least experienced programmer's details.

SQL>select * from programmer where doj = (select max (doj) from programmer);

5. Who is the most experienced programmer knowing pascal.

SQL>select pname from programmer where doj = (select min (doj) from programmer);

6. Who is the youngest programmer born in 1965.

SQL> select pname , dob from programmer where dob = (select max (dob) from programmer where to_char (dob,'yy') = 65);

7. In which year, most of the programmers are born.

SQL>select to_char (dob , 'YY') from programmer group by to_char (dob, 'YY') having count(*) =(select max (count(*)) from programmer group by to_char(dob,'YY');

8. In which month most number of programmers are joined.

SQL>select to_char (doj,'YY') from programmer group by to_char (doj,'YY') having count (*) =(select max (count(*)) from programmer group by to_char (doj,'YY');

9. What is the length of the shortest name in programmer table ?

SQL>select length (pname) from programmer where length (pname) = select min (length (pname) from programmer);

10. Display the names of the programmers whose name contains up to 5 characters.

SQL>select pname from programmer where length (pname) <=5;

11. Display all packages names in small letters and corresponding programmer names in uppercase letters.

SQL>select lower (title), upper (pname) from software;

RESULT:

Thus the above queries for nested queries & aggregate functions has been executed successfully and the output is verified.

EX NO : 04 DATE :	**SUB QUERIES AND JOIN OPERATIONS**

AIM:

To study and execute various select commands and indexes using SQL.

PROCEDURE:

TYPES OF SELECT COMMAND:

1) GROUP BY
2) ORDER BY
3) HAVING and BETWEEN
4) DISTINCT

SET OPERATORS:

1) UNION
2) UNION ALL
3) INTERSECT
4) MINUS

DESCRIPTION OF COMMANDS:

SQL>select * from dept;

D_ID	D_NAME
1	CSE
2	IT
3	MECH
4	ECE
5	Accounts
6	CIVIL
7	AERO
8	EEE

SQL>select * from employee;

EID	NAME	AGE	SALARY	DEPT_ID
1	Swetha	20	5000	2
2	Sharmi	25	7000	2
3	Pavi	21	4000	4
4	Shankari	27	6700	5
5	Madhu	26	5700	3
6	Harshini	24	7400	1

i)GROUP BY:

This command is used to display the group of values.

SYNTAX:

Select column1, column2… from <table_name>group by column1 having condition;

EXAMPLE:

SQL>select dept_id,avg(salary) from employee group by dept_id;

DEPT_ID	AVG(SALARY)
5	6700
2	5000
4	4000
5	6700
1	7400

SQL>select d_name,count(eid) from employee,dept where employee.dept_id=dept.d_id group by d_name;

D_NAME	COUNT(EID)
IT	3
EEE	1
CIVIL	1
CSE	1
MECH	1
ECE	1

ii)ORDERBY:

This command is used to display the column in an order(ascending order is default in all SQL queries).

SYNTAX:

Select * from <table_name> orderby condition;

EXAMPLE:

SQL>select * from employee order by name;

EID	NAME	AGE	SALARY	DEPT_ID
4	Shankari	27	6700	5
5	Madhu	26	5700	3
6	Harshini	24	7400	1
1	Swetha	20	5000	2
2	Sharmi	25	7000	2
3	Pavi	21	4000	4

iii)HAVING and BETWEEN:

This command is used to display the value having a condition.

SYNTAX:

Select column1,column2.. from <table_name>where column having condition;

EXAMPLE:

SQL>select dept_id,avg(salary) from employee group by dept_id having avg(salary)>2000;

DEPT_ID	AVG(SALARY)
5	6700
3	5700
7400	
5000	
2	7000
4	4000

BETWEEN:

This command is used to display the values between any condition.

SYNTAX:

Select column_name from table_name group by column_name between condition;

EXAMPLE:

SQL>select dept_id,avg(salary) from employee group by dept_id having avg(salary) between 2000 and 7000;

DEPT_ID	AVG(SALARY)
5	6700
5700	
2	5000

2 7000
4000

iv)DISTINCT:

This command is used to eliminate the duplicate entries from the table.

SYNTAX:

Select distinct * from table_name;

EXAMPLE:

SQL>select distinct age from employee order by age;

AGE

20
21
24
25
26
27

SET OPERATORS :

Table 1 : STUDENT

ROLL NO	NAME	M1	M2	M3
1	Vyshka	90	60	90
2	Yazhini	80	70	50
3	Maha	90	80	50

Table2: STUD

ROLL NO	NAME	M1	M2	M3
1	Vyshka	90	60	90
2	Yazhini	80	70	50
4	Pooja	80	80	60

i)UNION:

The union operator returns all distinct rows selected by the component queries.

SQL>select * from student union select * from stud;

OUTPUT:

ROLL NO	NAME	M1	M2	M3
1	Vyshka	90	60	90
2	Yazhini	80	70	50
3	Maha	90	80	50
4	Pooja	80	80	60

ii)UNION ALL:

The union all operator returns all rows selected by both queries including duplicate.

SQL>select * from student unionall select * from stud;

OUTPUT:

ROLL NO	NAME	M1	M2	M3
1	Vyshka	90	60	90
2	Yazhini	80	70	50
3	Maha	90	80	50
1	Vyshka	90	60	90
2	Yazhini	80	70	50
4	Pooja	80	80	60

iii)INTERSECT:

This operator returns only rows that are common to both queries.

SQL>select * from student intersect select * from stud;

OUTPUT:

ROLL NO	NAME	M1	M2	M3
1	Vyshka	90	60	90
2	Yazhini	80	70	50

iv)MINUS:

This operator returns all distinct rows selected only by the first query and not by the second query.

SQL>select * from student minus select * from stud;

OUTPUT:

ROLL NO	NAME	M1	M2	M3
3	Maha	90	80	50

INDEXES:

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.

Create an Index:

SYNTAX:

CREATE [UNIQUE] INDEX index_name ON table_name (column1, column2, . column_n);
UNIQUE indicates that the combination of values in the indexed columns must be unique.

EXAMPLE:

CREATE INDEX supplier_idx ON supplier (supplier_name);

In this example, created an index on the supplier table called supplier_idx. It consists of only one field - the supplier_name field.

We could also create an index with more than one field as in the example below:

CREATE INDEX supplier_idx ON supplier (supplier_name, city);

Rename an Index:

SYNTAX:

ALTER INDEX index_name RENAME TO new_index_name;

EXAMPLE:

ALTER INDEX supplier_idx RENAME TO supplier_index_name;

In this example, we're renaming the index called supplier_idx to supplier_index_name.

Drop an Index:

SYNTAX:

DROP INDEX index_name;

EXAMPLE:

DROP INDEX supplier_idx;

In this example, we're dropping an index called supplier_idx.

RESULT:

Thus the above various types of select commands and indexes has been executed successfully and the output is verified.

EX NO : 05 DATE :	**JOIN FUNCTIONS**

AIM:

To execute the join function operation using queries.

PROCEDURE:

A join is a query that combines rows from two or more tables, views, or materialized views. Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

1) EQUI JOINS :

An equijoin is a join with a join condition containing an equality operator (=). An equijoin combines rows that have equivalent values for the specified columns.

EXAMPLE:

select e.empno, e.ename, e.sal, e.deptno, d.dname, d.city from emp e, dept d where emp.deptno=dept.deptno;

(OR)

select * from emp,dept where emp.deptno=dept.deptno;

EMPLOYEE :

ENAME	ENO	SALARY
Sabitha	E1	15000
Swetha	E2	20000
Pooja	E3	25000

2)NON EQUI JOINS :

Non equi joins is used to return result from two or more tables where exact join is not possible.

EXAMPLE:

select e.empno, e.ename, e.sal, s.grade from emp e, salgrade s where e.sal between s.lowsal and s.hisal;

DEPT:

DEPT NO	MANAGER NAME	MANAGERID
E1	Madhu	E5
E2	Harshini	E6
E3	Sri	E7

a) SELF JOINS :

A self join is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle combines and returns rows of the table that satisfy the join condition.

EXAMPLE:

Select e.empno, e.ename, m.ename "Manager" from emp e, emp m where e.mgrid=m.empno;

ENO	ENAME	SALARY
E1	Sabitha	15000
E2	Swetha	20000

b) INNER JOIN :

An inner join (sometimes called a "simple join") is a join of two or more tables that returns only those rows that satisfy the join condition.

SYNTAX:

SELECT <column_name>, <column_name> FROM <table_name alias>INNER JOIN <table_name alias>ON <alias.column_name>= <alias.column_name>

EXAMPLE:

SELECT p.last_name, t.title_name FROM person p INNER JOIN title t ON p.title_1 = t.title_abbrev;

ENO	ENAME	SALARY
E3	Pooja	25000

c) OUTER JOINS :

An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other satisfy the join condition.

i) LEFT OUTER JOIN :

To write a query that performs an outer join of tables A and B and returns all rows from A (a left outer join), use the ANSI LEFT [OUTER] JOIN syntax, or apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle returns null for any select list expressions containing columns of B.

SYNTAX:

SELECT <column_name>,<column_name>FROM <table_name alias>LEFT OUTER JOIN<table_name alias>ON<alias.column_name>= <alias.column_name>

EXAMPLE:

SELECT p.last_name, t.title_name FROM person p LEFT OUTER JOIN title t ON p.title_1 = t.title_abbrev;

DEPT NO	MANAGER NAME	MANAGERID
E2	Harshini	E6

ii) RIGHT OUTER JOIN :

To write a query that performs an outer join of tables A and B and returns all rows from B (a right outer join), use the ANSI RIGHT [OUTER] syntax, or apply the outer join operator (+) to all columns of A in the join condition. For all rows in B that have no matching rows in A, Oracle returns null for any select list expressions containing columns of A.

SYNTAX:

SELECT <column_name>, <column_name> FROM <table_name alias> RIGHT OUTER JOIN<table_name alias>ON<alias.column_name>= <alias.column_name>

EXAMPLE:

SELECT p.last_name, t.title_name FROM person p RIGHT OUTER JOIN title t ON p.title_1 = t.title_abbrev;

BOOK DETAILS :

BID	FIRST_NAME	LAST_NAME	TITLE
001	Swetha	Ramasamy	Frozen
002	Varsha	Parthiban	Great King
003	Sathya	Priya	Myself

BID	ABBREV
001	FN
002	GK
003	MS

iii) FULL OUTER JOIN :

To write a query that performs an outer join and and returns all rows from A and B, extended with nulls if they do not satisfy the join condition (a full outer join), use the ANSI FULL [OUTER] JOIN syntax.

SYNTAX:

SELECT <column_name>, <column_name> FROM <table_name alias>FULL OUTER JOIN<table_name alias> ON <alias.column_name>= <alias.column_name>

EXAMPLE:

SELECT p.last_name, t.title_name FROM person p FULL OUTER JOIN title t ON p.title_1 = t.title_abbrev;

select e.empno,e.ename,e.sal,e.deptno,d.dname,d.city from emp e, dept d where e.deptno(+)=d.deptno;
That is specify the (+) sign to the column which is lacking values.

OUTPUT:

BID	FIRST_NAME	TITLE	ABBREV
001	Swetha	Frozen	FN
002	Varsha	Great king	GK
003	Sathya	Myself	MS

RESULT:

Thus the above program for studying and executing the joins has been executed successfully and the output is verified.

EX NO : 06 DATE :	**USER DEFINED FUNCTIONS AND STORED PROCEDURES**

AIM :

To write user defined functions and stored procedures in SQL.

DEFINITION :

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. They are essentially sub-programs. Procedures and functions are made up of,

- Declarative part .
- Executable part
- Optional exception handling part .

These procedures and functions do not show the errors. .

KEYWORDS AND THEIR PURPOSES :

REPLACE: It recreates the procedure if it already exists.

PROCEDURE: It is the name of the procedure to be created.

ARGUMENT: It is the name of the argument to the procedure. Paranthesis can be omitted if no arguments are present. .

IN: Specifies that a value for the argument must be specified when calling the Procedure ie., it is used to pass values to a sub-program. This is the default parameter.

OUT: Specifies that the procedure passes a value for this argument back to it's calling environment after execution ie., it is used to return values to a caller of the program.

INOUT: Specifies that a value for the argument must be specified when calling the Procedure and that procedure passes a value for this argument back to it's calling environment after execution.

RETURN: It is the data type of the function's return value because every nction must return a value, this clause is required.

ROCEDURES - SYNTAX :

create Or replace procedure <procedure name> (argument {in,out,inout} datatype) {is,as}
variable declaration;
constant declaration;
begin
PL/SQL subprogram body; exception
exception PL/SQL block; end;

CREATING THE TABLE 'ITITEMS' AND DISPLAYING THE CONTENTS:

SQL> create table ititems(itemid number(3), actualprice number(S), ordid number(4), prodid number(4));
Table created.
SQL> insert into ititems values(101, 2000, 500, 201);
1 row created.
SQL> insert into ititems values(102, 3000, 1600, 202);
I row created.
SQL> insert into ititems values(103, 4000, 600, 202); .
I row created.
SQL> select * from ititems;

ITEMID	ACTUALPRICE	ORDID	PRODID
101	2000	500	201
102	3000	1600	202
103	4000	600	202

PROGRAM FOR GENERAL PROCEDURE - SELECTED RECORD'S PRICE IS INCREMENTED BY 500, EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE.

SQL> create procedure itsum(identity number, total number) is price number; 2. null_price exception;
3. begin
4. select actualprice into price from itiitems where itemid = identity;

```
5.  if price is null then
6.  raise null_price
7.  else
8.  update ititems set actualprice = actualprice+total where itemid = identity;
9.  end if;
10.  exception
11. when null_price then
12.  dbms_output.put_line('price is null')
13. end;
14. /
Procedure Created
SQL> exec itsum(101, 500);
PL/SQL procedure successfully completed.
SQL> select * from ititems;
```

ITEMID	ACTUALPRICE	ORDID	PRODID
101	2500	500	201
102	3000	1600	202
103	4000	600	202

'PROCEDURE FOR 'IN' PARAMETER - CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure yyy (a IN number) is price number;
2.begin
3.select actualprice into price from ititems where itemid=a;
4.dbms_output.put_line('Actual price is 'II price);
5. if price is null then
6. dbms_output.put_line('Price is null');
7. end if;
8. end;
9. /
Procedure Created
SQL>exec yyy(103);
Actual price is 4000
PL\SQL Procedure successfully completed
```

PROCEDURE FOR 'OUT' PARAMETER – CREATION, EXECUTION

```
SQL> set serveroutput on;
SQL> create procedure zzz(a in number, b out number)is identity
number;
2. begin
3. select ordid into identity from ititems where itemid=a;
4. if identity<1000 then
5. b;=100
6. end if;
7. end;
8. /
Procedure Created
SQL> declare
2. a number;
3. b number;
4. begin
5. zzz(101,b);
6. dbms_output.put_line('The value of b is 'IIb);
7. end;
8. /
The value of b is 100
PL/SQL Procedure successfully completed
```

PROCEDURE FOR 'INOUT' PARAMETER – CREATION, EXECUTION

```
SQL> create procedure itit(a inout number) is
2. begin
3. a;=a+1
4. end;
5. /
Procedure Created
SQL> declare
2. a number=7;
3. begin
4. itit(a);
5. dbms_output.put_line('The updated value is'II a);
6. end;
7. /
8.
```

FUNCTIONS

DATE FUNCTION

1.Add_month

This function returns a date after adding a specified date with specified number of months.

SYNTAX: Add_months(d,n); where d-date n-number of months

EXAMPLE: Select add_months(sysdate,2) from dual;

2.last_day

It displays the last date of that month.

SYNTAX: last_day(d);where d-date

EXAMPLE: Select last_day(,,1-jun-2009'',''1-aug-2009'')from dual;

3.Months_between

It gives the difference in number of months between d1&d2.

SYNTAX: month_between(d1,d2); where d1&d2-dates.

EXAMPLE: Select month_between(,,1-jun-2009'',''1-aug-2009'')from dual;

4.next_day

It returns a day followed the specified date.

SYNTAX: next_day(d,day);

EXAMPLE: Select next_day(sysdate,''wednesday'') from dual.

5.round

This function returns the date, which is rounded to the unit specified by the format model.

SYNTAX: round(d.[fmt]); where d-date,[fmt] – optional.

By default date will be rounded to the nearest day

EXAMPLE: Select round (to_date(,,1-jun-2009'',''dd-mm-yy''),''year'') from dual; Select round(,,1-jun-2009'',''year) from dual;

NUMERICAL FUNCTIONS

COMMAND	QUERY	OUTPUT
Abs(n)	Select abs(-15) from dual;	15
Ceil(n)	Select ceil(55.67) from dual;	56
Exp(n)	Select exp(4) from dual;	54.59
Floor(n)	Select floor(100.2) from dual;	100
Power(m,n)	Select power(4,2) from dual;	16
Mod(m,n)	Select mod(10,3) from dual;	1
Round(m,n)	Select round(100.256,2) from dual;	100.26
Trunc(m,n)	Select trunc(100.25,2) from dual;	100.23
Sqrt(m,n)	Select sqrt(16) from dual;	4

CHARACTER FUNCTIONS

COMMAND	QUERY	OUTPUT
initcap(char)	Select initcap("hello") from dual;	Hello
lower(char);	Select lower(,,HELLO'') from dual;	hello
upper(char);	Select upper(,,hello'') from dual;	HELLO
ltrim(char,[set]);	Select ltrim(,,cseit'', ,,cse'') from dual; select	it
rtrim(char,[set]);	rtrim(,,cseit'', ,,it'') from dual;	cse
replace(char,search string, replace string);	Select replace(,,jack and jue'',,,''bl'') from dual;	black and blue
substr(char,m,n);	Select substr(,,information'',3,4) from dual;	Form

CONVERSION FUNCTION

1.to_char()

SYNTAX: to_char(d,[format]);

This function converts date to a value of varchar type in a form specified by date format. If format is neglected then it converts date to varchar2 in the default date format.

EXAMPLE: select to_char(sysdate,''dd-mm-yy'') from dual;

2. to_date()

SYNTAX: to_date(d,[format]);

This function converts character to date format specified in the form character.

EXAMPLE: select to_date(,,aug 15 2009'',''mm-dd-yy'') from dual;

Miscellaneous Functions

1. **uid-** This function returns the integer value(id) corresponding to the user currently logged in.

   Example: select uid from dual;

2. **user-** This function returns the logins user name.

Example: select user from dual;

3. **nvl-** The null value function is mainly used in the case where we want to consider null values as zero.

   Syntax: nvl(exp1, exp2)

   If exp1 is null, return exp2. If exp1 is not null, return exp1.

   Example: select custid, shipdate, nvl(total,0) from order;

4. **vsize:** It returns the number of bytes in expression.

   Example: select vsize(,,tech'')from dual;

GROUP FUNCTIONS

A group function returns a result based on group of rows.

1. avg - Example: select avg (total) from student;
2. max - Example: select max (percentagel) from student;
3. min - Example: select min (marksl) from student;
4. sum - Example: select sum(price) from product;

COUNT FUNCTION

In order to count the number of rows, count function is used.

1. **Count(*)** – It counts all, inclusive of duplicates and nulls.
   Example: select count(*) from student;
2. **Count(col_name)** – It avoids null value.
   Example: select count(total) from order;
3. **Count(distinct col_name)** – It avoids the repeted and null values.
   Example: select count(distinct ordid) from order;

RESULT:

Thus the above program for user defined functions and stored procedures in SQL has been executed successfully and the output is verified.

EX NO : 07 DATE :	**DCL AND TCL COMMANDS**

AIM :

To write a program to execute complex transactions and realize DCL and TCL commands.

DATA CONTROL LANGUAGE(DCL) , TRANSACTION CONTROL LANGUAGE(TCL) COMMANDS

S.NO	DETAILS OF THE STEP
1	**DCL COMMAND** The DCL language is used for controlling the access to the table and hence securing the database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated.
2	The privilege commands are namely, Grant and Revoke.
3	The various privileges that can be granted or revoked are, Select , Insert, Delete, Update, References, ExecuteAll.
4	**GRANT COMMAND :** It is used to create and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.
5	**REVOKE COMMAND :** Using this command, the DBA can revoke the grant database privileges from the user.
6	**TCL COMMAND** **COMMIT** : Command is used to save the records. **ROLLBACK :** Command is used to undo the records. **SAVE POINT :** Command is used to undo the records in a particular transaction.

SQL Commands
DCL Commands

GRANT Command

Grant <database_priv [database_priv….] > to <user_name> identified by <password> [,<password…]:

Grant <object_priv> |All on <object> to <user | public>[With Grant Option];

REVOKE COMMAND

Revoke <database_priv> from <user[,user]>;

Revoke <object_priv> on <object> from <user| public>;

<database_priv> -- Specifies the system level privileges to be granted to the users or roles.This includes create/alter/delete any object of the system.

<object_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update for tables.

<all> -- Indicates all the privileges.

[With Grant Option] – Allows the recipient user to give further grants on the objects.

The privileges can be granted to different users by specifying their names or to all users by using the “Public” option.

TCL COMMANDS

Syntax :

SAVE POINT : SAVEPOINT<SAVE POINT NAME>;

ROLLBACK : ROLLBACK<SAVE POINT NAME>;

COMMIT : Commit;

Queries

Tables Used

Consider the following tables namely"DEPARTMENTS" and "EMPLOYEES"
Their schemas are as follows,
Departments(dept_no , dept_name , dept_location);
Employees(emp_id , emp_name, emp_salary);

Q1. Develop a query to grant all privileges of employees table into departments table.
Ans.

SQL>Grant all on employees to departments;
Grant succeeded.

Q2. Develop a query to grant some privileges of employees table into departments table.
Ans.

SQL>Grant select, update, insert on departments to departments with grant option;
Grant succeeded.

Q3. Develop a query to revoke all privileges of employees table into departments table.
Ans.

SQL>Revoke all on employees from departments;
Revoke succeeded.

Q4. Develop a query to revoke some privileges of employees table into departments table.
Ans.

SQL>Revoke select, update, insert on departments from departments;
Revoke succeeded.

Q5. Write a query to implement the savepoint.
Ans.

SQL>SAVEPOINT S1;
Savepoint created.
SQL>select*from emp;

	EMPNO	ENAME	JOB	DEPTNO	SAL
1.		Swetha	AP	1	10000
2.		Madhu	ASP	2	15000
3.		Harshini	AP	1	15000
4.		Sri	Prof	2	30000

RESULT :

Thus the above program to execute complex transactions and realize DCL and TCL commands has been executed successfully and the output is verified.

EX NO : 08 DATE :	**TRIGGERS**

AIM :

To write a program to write SQL triggers for insert, delete and update operations in a database table.

DEFINITION

1. A trigger 'is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,
2. Trigger statement: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.
3. Trigger body or trigger action: It is a PL/SQL block that is executed when the triggering statement is used.
4. Trigger restriction: Restrictions on the trigger can be achieved.

The different uses of triggers are as follows,

1. To generate data automatically.
2. To enforce complex integrity constraints.
3. To customize complex securing authorizations.
4. To maintain the replicate table.
5. To audit data modifications.

Types of Triggers

The various types of triggers are as follows,

Before: It fires the trigger before executing the trigger statement.

After: It fires the trigger after executing the trigger statement.

For each row: It specifies that the trigger fires once per row.

For each statement: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

Variables used in Triggers

:new

:old

These two variables retain the new and old value of the column updated in the database. The values in these variables can be used in the database triggers for data manipulation.

TRIGGERS-SYNTAX

Create or replace trigger triggername [before/after] {DML statements} on [tablename] [for each row/statement] begin

---

---

exception

End;

USER DEFINED ERROR MESSAGE

The package "raise_application_error" is used to issue the user defined error messages. Syntax : raise_application_error(error number,'error message');

The error number can lie between -20000 and -20999.

The error message should be a character string.

TO CREATE A SIMPLE TRIGGER THAT DOES NOT ALLOW INSERT UPDATE AND DELETE OPERATIONS ON THE TABLE

SQL>Create trigger ittrigg before insert or update or delete on itempls for each row

2. begin
3. raise_application_error(-20010,'You cannot do manipulation');
4. end;

```
5. /
Trigger created.
SQL>insert into itempls values('aaa',14,34000);
Insert into itempls values('aaa',14,34000)
            *
ERROR at line 1:
ORA-20010 : You cannot do manipulation.
ORA-06512 : at "STUDENT.ITTRIGG",line 2
ORA-04088 : error during execution of trigger 'STUDENT.ITTRIGG'
SQL>delete from itempls where ename='xxx';
Delete from itempls where ename='xxx'
            *
ERROR at line 1:
ORA-20010 : You cannot do manipulation.
ORA-06512 : at "STUDENT.ITTRIGG",line 2
ORA-04088 : error during execution of trigger 'STUDENT.ITTRIGG'
SQL>update itempls set eid=15 where ename='yyy';
Update itempls set eid=15 where ename='yyy'
              *
ERROR at line 1:
ORA-20010 : You cannot do manipulation.
ORA-06512 : at "STUDENT.ITTRIGG",line 2
ORA-04088 : error during execution of trigger 'STUDENT.ITTRIGG'
```

TO DROP THE CREATED TRIGGER

```
SQL>drop trigger ittrigg;
Trigger dropped.
```

TO CREATE A TRIGGER THAT RAISES AN USER DEFINED ERROR MESSAGE AND DOES NOT ALLOW UPDATION AND INSERTION

```
SQL>create trigger ittriggs before insert or update of salary on itempls for each row
    2. declare
    3. triggsalitempls.salary%type;
    4. begin
    5. select salary into triggsal from itempls where eid=12;
    6. if(:new.salary>triggsal or :new.salary<triggsal)then
    7. raise_application_error(-20100,'Salary has not been changed');
    8. end if;
    9. end;
    10. /
Trigger created.
SQL>insert into itempls values('bbb',16,45000);
Insert into itempls values('bbb',16,45000)
                  *
ERROR at line 1:
ORA-04098 : trigger ' STUDENT.ITTRIGG' is invalid and failed re-validation
SQL>update itempls set eid=18 where ename='zzz';
Update itempls set eid=18 where ename='zzz'
              *
ERROR at line 1:
ORA-04098 : trigger ' STUDENT.ITTRIGG' is invalid and failed re-validation
```

RESULT :

Thus the above program to write SQL triggers for insert, delete and update operations in a database table has been executed successfully and the output is verified.

EX NO : 09 DATE :	**VIEW AND INDEX FOR DATABSE TABLES**

AIM :

To write a program to create view and index for database tables with a large number of records.

VIEWS

SNO	DETAILS OF THE STEP
1	**VIEWS:** A view is the tailored presentation of data contained in one or more table and can also be said as restricted view to the data's in the tables.A view is a 'virtual table' or a 'stored query' which takes the output of a query and treats it as a table. The table upon which a view is created is called as base table. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables.The view is stored as a SELECT statement in the data dictionary.
2	**Advantages of a view:** Additional level of table security. Hides data complexity. Simplifies the usage by combining multiple tables into a single table. Provides data's in different perspective.
3	**Types of view:** Horizontal-> enforced by where cause Vertical-> enforced by selecting the required columns.

SQL Commands

Creating and dropping view:

Syntax

Create [or replace] view <view name> [column alias names] as <query> [with<options>conditions];

Drop view<view name>;

Example :

Create or replace view empview as select*from emp;

Drop view empview;

Queries

Tables Used

SQL>select*from emp;

	EMPNO	ENAME	JOB	DEPTNO	SAL
1.		Swetha	AP	1	10000
2.		Madhu	ASP	2	15000
3.		Harshini	AP	1	15000
4.		Sri	Prof	2	30000

Q1. The organization wants to display only the details of the employees those who are ASP.(Horizontal portioning)

Solution:

1. Create a view on emp table named managers.
2. Use select from clause to do horizontal portioning.

Ans.

SQL>create view empview as select*from emp where job='ASP';

View created.
SQL>select*from empview;

EMPNO	ENAME	JOB	DEPTNO	SAL
2.	Madhu	ASP	2	15000
3.	Harshini	AP	1	15000

Q2. The organization wants to display only the details like empno, empname, deptno, deptname of the employees.(Vertical portioning)

Solution:

1. Create a view on emp table named general.
2. Use select from clause to do vertical portioning.

Ans.

SQL>create view empview1 as select ename,sal from emp;
View created.

Q3. Display all the views generated.

Ans.

SQL>select*from tab;

TNAME	TABTYPE	CLUSTERED
DEPT	TABLE	
EMP	TABLE	
EMPVIEW	VIEW	
EMPVIEW	VIEW	

Q4. Execute the DML commands on the view created.

Ans.

SQL>select*from empview;

EMPNO	ENAME	JOB	DEPTNO	SAL
2.	Madhu	ASP	2	15000
3.	Harshini	AP	1	15000

Q5. Drop a view.

Ans.

SQL>drop view empview1;
View dropped.

Indexes

1. An index can be created in a table to find data more quickly and efficiently.
2. The users cannot see the indexes; they are just used to speed up searches/queries.
3. Updating a table with indexes takes more time than updating a table without; because the indexes also need an update. So we should only create indexes on columns(and tables) that will be frequently searched against.

Syntax :

Create Index :

CREATE INDEX index_name ON table_name(column_name)

SQL>create table splr(name varchar(10), sid number(10) , scity varchar(10));
Table created.
SQL>insert into splr values('hcl',01,'chennai');
1 row created.
SQL>insert into splr values('dell',04,'madurai');
1 row created.
SQL>insert into splr values('HP',02,'kovai');
1 row created.
SQL>insert into splr values('Lenovo',03,'trichy');

1 row created.
SQL>select*from splr;

SNAME	SID	SCITY
Hcl	1	Chennai
Dell	4	Madurai
HP	2	Kovai
Lenovo	3	Trichy

SQL>create index sp1 on splr(sid);
Index created.
SQL>create index sp2 on splr(sid,scity);
Index created.

Drop Index :

SQL>drop index sp1;
Index created.
SQL>drop index sp2;
Index created.

RESULT:

Thus the above program to create view and index for database tables with a large number of records has been executed successfully and the output is verified.

EX NO : 10 DATE :	**XML DATABASE AND VALIDATE IT USING XML SCHEMA**

AIM :

To write a program to create an xml database and validate it using XML schema.

Step 1 : Create an XML schema

We now need to create an XML schema using the XML Schema Editor in DWB. This section will walk you through creating a new XML Schema from scratch. (Note: The example user has already created the XML Schema.)

1. Expand the Data Development Project in the Data Project Explorer. Locate the folder named XML Schema Document's. Right-click on the XML Schema Documents folder and find New.
2. After you click on New, you see a popup window titled "Create XML Schema." The parent folder is the name of your data development project. In the File Name textbox, create a name for your XML Schema document.You can name the XML Schema document any name you choose; however, it MUST have the .xsd extension.

Before going any further, I want to show you the finished XML Scherna Document graphically so that you have some idea of what you are going to be building. Look at the figure below. On the right side, you should see the Outline pane. It shows_the tree structure of the schema. Remember that this is the finished product that you will build.
Let's start working on it. Right-click in the Elements pane. Click on Add Element.

3. You will now have the "root" element of your XML docwnent, which is named NewGlobalElement.
4. Rename the root element from NewG1oba1Element to customerinfo. Note: Sometimes, you are *able to* immediately change the name; if you can't, *right-click* on NewGlobalElement and find Refactor and then Rename.
5. Once the root element has been renamed, right-click on the customerinfo element, select Set Type, and then select New Complex Type.
6. Double-click on the customerinfo element and you will get an element hierarchy.
The box to the right of the customerinfo box, the one with the 3 dots that also has the cursor pointing to it, is called a sequence box. If you right click on the sequence box, you will see an Add Element choice. This is where you add the elements that belong or go under the root element customerinfo. Remember that customerinfo is a Complex Type.
7. Right-click the sequence box(…box). Click Add Element. Next , you will add the elements.
8. You can immediately change NewElement to name. If you don't do it right away, then right-click on the element and choose Refactor and then Rename. Notice that the data type is String.
9. After you remove the element to address, notice that it is a String type. The address element will further contain other elements, so it must be a Complex Type.
10. Change the "type" from string to anonymous.
11. Add the element balance(use a lowercase B). Set the type to decimal. Use Set Existing Type to change the type.
Now, we will add the elements under the address element.
12. Click on the +sign to the right of the address element. You should get a sequence box.
13. Now add street, city, state, and zip to the address element. Follow the same process you used to create name, address and balance.

Step 2: XML Schema Registration

In this step, there is one more thing to do. We need to register the XML Schema with the XSR- XML Schema Repository. This is done in the DWB. The registration is necessary so that the database can decompose (shred) and validate XML data. Once this is done you will be able to view and manage XML schemas and documents in the XSR.

1. Close the graphical XML schema editor.
2. In the Data Project Explorer Pane, make sure that you have expanded your data development project tree.
3. Locate the XML schema documents folder and click on the +sign next to it. You should see your XML Schema . I am going to use the one that I just created named NanaLiuNewXMLSchema.
4. Right-click on your schema an look for register an XML schema.
5. Click on Register an XML Schema. You should notice that the name of the schema you highlighted (the one you just created) is in the "XMLschema name" text box. If you try to change the name, you may be prompted to reconnect to the database by a User ID and Password dialog box. Don't worry if you don't get the login prompt.
6. Click Next. You should get the "Register an XML Schema" dialog. Notice that it is asking for a "Schema location.c x."

   In the box labeled "XML schema documents and dependencies," you should see a path reference to your U:\ drive and workspace and Schema,.
7. Simply click on the path(U:\) , and that path will then appear in the schema location box.
8. Click finish. It might take 30 seconds or more to register.

Possible reasons for XSR Registration Errors :

1. Rememher when we first started this lab, we talked about a possible XSR Registration error that may occur when the system is IPLed (restarted) after maintenance downtime and the DeVry "Bind" has not.yet run. Well, if this BIND procedure has not run or fails, then you will NOT be able to register your XML Schema.
2. The other possible reason for XSR Registration failure is that you have not completed all of the steps correctly.
3. If you get an error, take a screenshot of your error message, review the lab steps, and consult your instructor for assistance.

   It is important to know that these types of errors are common in the industry, so you are getting direct industry experience.

9. When your XSR Registration is successful you will see a message:
   "Register successful".

RESULT :

Thus the above program to create an XML database and validate it using XML schema has been executed successfully and the output is verified.

EX NO : 11 DATE :	**DOCUMENT, COLUMN, AND GRAPH-BASED DATA NOSQL DATABASE TOOLS**

AIM :

To write a program to create document, column, and graph-based data NOSQL database tools.

DEFINITION

Document based data

A document database is a type of non-relational database that is designed to store and query data as JSON-like documents. Document databases make it easier for developers to store and query data in a database by using the same document-model the format they use in their application code. The flexible, semi-structured, and hierarchical nature of documents and document databases allows them to evolve with applications' needs. The document model works well with use cases such as catalogs, user profiles, and content management systems where

```
[
{
"year": 2013,
 "title": "Turn It Down, Or Else!",
 "info": {
   "directors": [ "Alice Smith", "Bob Jones"],
   "release_date" : "2013-01-18T00:00:00Z",
   "rating": 6.2,
   "genres": [ " Comedy", "Drama"],

"image_url"
 "plot": "A rock band plays their music at high volumes, annoying the
neighbors.",
 "actors" : ["David Matthewman", "Jonathan G. Neff"]
}
},
{
"year": 2015,
 "title": "The Big New Movie",
 \ "info":
   "plot": "Nothing happens at all.",
   "rating": 0
}
}
]
```

The following is an example of a document that might appear in a document database like MongoDB. This sample document represents a company contact card, describing an employee called Sammy:

```
{
"_id": "sammyshark",
 "firstName": "Sammy",
 "lastName": "Shark",
 "email": "sammy.shark@digitalocean.com",
 "department": "Finance"
}
```

Notice that the document is written as a JSON object. JSON is a human-readable the data format has become quite popular in recent years. While many different formats can be used to represent data within a document database, such as XML or YAML, JSON is one of the most common choices. For example, MongoDB adopted JSON is the primary data format to define and manage data.

All data in JSON documents are represented as field-and-value pairs that take the form of field: value. In the previous example, the first line shows an _id field with the value sammyshark. The example also includes fields for the employee's first and last names, their email address, as well as what department they work in.

RESULT :

Thus the above program to create document, column and graph-based data NOSQL database tools has been executed successfully and the output is verified.

EX NO : 12 DATE :	**DEVELOP A SIMPLE GUI BASED DATABSE APPLICATION**

AIM :

To write a program to develop a simple Gui based database application and incorporate all the above-mentioned features.

PROGRAM FOR FORM 1

```
Private Sub emp_click()
Form2.Show
End Sub
Private Sub exit_click()
Unload Me
End Sub
Private Sub salary_click()
Form3.Show
End Sub
```

PROGRAM FOR FORM 2

```
Private Sub add_click()
Adodc1.Recordset.AddNew
MsgBox “Record added”
End Sub
Private Sub clear_click()
Text1.Text = “ “
Text2.Text = “ “
Text3.Text = “ “
Text4.Text = “ “
Text5.Text = “ “
Text6.Text = “ “
End Sub
Private Sub delte_click()
Adodc1.Recordset.Delete
MsgBox”Record Deleted”
If Adodc1.Recordset.EOF = True then
Adodc1.Recordset.moveprevious
End If
End Sub
Private Sub exit_click()
Unload Me
End Sub
Private Sub main_click()
Unload Me
End Sub
Private Sub main_click()
Form1.Show
End Sub
Private Sub modify_click()
Form1.Show
End Sub
Private Sub modify_click()
Adodc1.Recordset.Update
End Sub
```

PROGRAM FOR FORM 3

```
Private Sub add_click()
```

```
Adodc1.Recordset.AddNew
MsgBox “Record added”
End Sub
Private Sub clear_click()
Text1.Text = “ “
Text2.Text = “ “
Text3.Text = “ “
Text4.Text = “ “
Text5.Text = “ “
Text6.Text = “ “
End Sub
Private Sub delte_click()
Adodc1.Recordset.Delete
MsgBox”Record Deleted”
If Adodc1.Recordset.EOF = True then
Adodc1.Recordset.moveprevious
 End If
 End Sub
 Private Sub exit_click()
 Unload Me
 End Sub
 Private Sub main_click()
 Unload Me
 End Sub
 Private Sub main_click()
 Form1.Show
 End Sub
 Private Sub modify_click()
 Form1.Show
 End Sub
 Private Sub modify_click()
 Adodc1.Recordset.Update
 End Sub
```

BANKING SYSTEM

Facilities required to do the experiment :

SNO	FACILITIES REQUIRED	QUANTITY
1	System	1
2	Operating System	Windows XP
3	Front end	VB/VC ++/JAVA
4	Back end	Oracle11g,my SQL,DB2

Procedure for doing the experiment

SNO	DETAILS OF THE STEP
1	Create the DB for banking system source request the using SQL
2	Establishing ODBC connection
3	Click add button and select oracle in ORA home 90 click finished
4	A window will appear give the data source name as oracle and give the user id as scott
5	Now click the test connection a window will appear with server and user name give user As scott and password tiger click ok

6	**VISUAL BASIC APPLICATION :** Create standard exe project in to and design ms from in request format To add ADODC project select component and check ms ADO data control click ok Now the control is added in the tool book Create standard exe project in to and design ms from in request format
7	**ADODC CONTEOL FOR ACCOUNT FROM :** Click customs and property window and window will appear and select ODBC data source name as oracle and click apply as the some window

Program

CREATE A TABLE IN ORACLE

SQL>create table account(cname varchar(20),accnonumber(10),balance number); Table Created
SQL> insert into account values('&cname',&accno,&balance); Enter value for
cname: Swetha
Enter value for accno: 1234
Enter value for balance: 10000
oldl: insert into account values('&cname',&accno,&balance)
newl: insert into emp values('Swetha',1234,10000) 1 row created.

SOURCE CODE FOR FORM1

```
Private Sub ACCOUNT_Click()
Form2.show
End Sub
Private Sub EXIT_Click()
Unload Me
End Sub
Private Sub TRANSACTION_Click()
Form3.Show
End Sub
```

SOURCE CODE FOR FORM2

```
Private Sub CLEAR_Click()
Text1.Text = " "
Text2.Text = " "
Text3.Text = " "
End Sub
Private Sub DELETE_Click()
Adodc1.Recordset.DELETE
Msgbox "record deleted"
Adodc1.Recordset.MoveNext
If Adodc1.Recordset.EOF = True Then
Adodc1.Recordset.MovePrevious
End If
End Sub
Private Sub EXIT_Click()
Unload Me
End Sub
Private Sub HOME_Click()
Form1.Show
End Sub
Private Sub INSERT_Click()
Adodc1.Recordset.Addnew
End Sub
Private Sub TRANSACTION_Click()
Form3.Show
End Sub
Private Sub UPDATE_Click()
Adodc1.Recordset.UPDATE
```

```
        MsgBox  “record updated successfully”
        End sub
```

SOURCE CODE FOR FORM3

```
        Private Sub ACCOUNT_Click()
        Form2.show
        End Sub
        Private Sub CLEAR_Click()
        Text1.Text = “ “
        Text2.Text = “ “
        End Sub
        Private Sub
        DEPOSIT_Ci
        Ck() Dim s As
        String
        S = InputBox(“enter the amount to be deposited”) Text2.text =
val(Text2.Text) + val(s)
        A = Text2.text
        MsgBox “ current balance is Rs” + Str(A)
        Adodc1.Recorset.save Adodc1.Recordset.UPDATE
        End Sub
        Private Sub EXIT_Click()
        Unload Me
        End Sub
        Private Sub HOME_Click()
        Form1.Show
        End Sub
        Private Sub WITHDRAW_Click() Dim s As String
        S = InputBox(“enter the amount to be deleted”) Text2.text =
val(Text2.Text) + val(s)
        A = Text2.text
        MsgBox “ current balance is Rs” + Str(A)
        Adodc1.Recorset.save Adodc1.Recordset.UPDATE
        End Sub
```

RESULT:

Thus the above program to develop a simple GUI database application and incorporate all the above-mentioned features has been executed successfully and the output is verified.

EX NO : 13(a) DATE :	**Inventory Management for an E-Mart Grocery Shop**

AIM:

To build Inventory Management for an E-Mart Grocery Shop using Java and SQL.

ALGORITHM:

Step-1: Define the database structure:

- Create tables for entities like customers, products, orders, and order items.
- Identify the necessary attributes for each table, such as customer ID, product ID, order ID, quantity, price, etc.
- Set appropriate primary keys and foreign key relationships between tables.

Step-2: Create the necessary SQL tables:

- Use the CREATE TABLE statement to create tables with the defined structure.
- Specify the data types and constraints for each column.

Step-3: Populate the tables:

- Use the INSERT INTO statement to insert sample data into the tables.
- Add records for customers, products, and orders.

Step-4: Implement basic operations:

- Create SQL queries to perform basic operations like selecting, inserting, updating, and deleting records.
- For example, you can write queries to retrieve customer details, list available products, add new customers, update product quantities, etc.

Step-5: Implement advanced operations:

- Develop queries to handle more complex operations such as processing orders, calculating total prices, generating reports, etc.
- For example, you can create queries to calculate the total price of an order, retrieve order history for a specific customer, generate sales reports for a given period, etc.

Program:

Adminlog.java

```
 import java.io.IOException; import
javax.servlet.ServletException; import
javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import
javax.servlet.http.HttpServletRequest;
import
javax.servlet.http.HttpServletResponse;
@WebServlet("/AdminLog") public class AdminLog
extends HttpServlet {             private static final long
serialVersionUID = 1L;         public AdminLog() {
       super();
       // TODO Auto-generated constructor stub
   }
          protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                 // TODO Auto-generated method stub
               String username = request.getParameter("username");
               String password = request.getParameter("password");
                    //System.out.println("hello!");
if(!username.equals("admin") || !password.equals("kitkat")){
          response.sendRedirect("adminLogin.jsp?status=1");
                    }else{
                        //public void setAttribute(String name, Object value)
                      //This method binds an object to this session, using the name specified.
```

```
                        request.getSession().setAttribute("Admin","1");
//System.out.println("log:"+request.getAttribute("Admin"));
response.sendRedirect("adminPage.jsp");
                    //System.out.println("HI");
                }
        }
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
  // TODO Auto-generated method stub   doGet(request,
response) }
}
```

BACKEND SQL

```
-- Create Customers table
CREATE TABLE Customers (
  CustomerID INT PRIMARY KEY,
  CustomerName VARCHAR(50),
  Email VARCHAR(50),
  Address VARCHAR(100)
);
-- Create Products table
CREATE TABLE Products (
  ProductID INT PRIMARY KEY,
  ProductName VARCHAR(50),
  Price DECIMAL(10, 2),
  Quantity INT
);
-- Create Orders table
CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  CustomerID INT,
  OrderDate DATE,
  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
-- Create OrderItems table
CREATE TABLE OrderItems (
  OrderID INT,
  ProductID INT,
  Quantity INT,
  FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
  FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
-- Insert sample data
INSERT INTO Customers (CustomerID, CustomerName, Email, Address)
VALUES (1, 'John Doe', 'john@example.com', '123 Main St');
INSERT INTO Products (ProductID, ProductName, Price, Quantity)
VALUES (1, 'Apples', 1.99, 50),
      (2, 'Bananas', 0.99, 100);
INSERT INTO Orders (OrderID, CustomerID, OrderDate)
VALUES (1, 1, '2023-05-19');
INSERT INTO OrderItems (OrderID, ProductID, Quantity)
VALUES (1, 1, 5),
      (1, 2, 10);
-- Selecting data
SELECT * FROM Customers;
-- Inserting data
```

```
INSERT INTO Customers (CustomerID, CustomerName, Email, Address)
VALUES (2, 'Jane Smith', 'jane@example.com', '456 Elm St');
- Updating data
UPDATE Products SET Quantity = Quantity - 5 WHERE ProductID = 1;
-- Deleting data
DELETE FROM Customers WHERE CustomerID = 2;
```

OUTPUT:

RESULT:

Thus the above program to build an E-Mart Grocery Shop using Java and SQL has been executed and the output is verified successfully.

EX NO : 13(c) DATE :	**COP – FRIENDLY APPLICATION**

AIM:

To build cop – friendly application by using Database management system.

ALGORITHM:

Step-1: Define the database structure:

- Create tables for entities like home , add staff , view staff and View cases.
- Identify the necessary attributes for each table, such as staff ID, case ID, admin ID, etc.
- Set appropriate primary keys and foreign key relationships between tables.

Step-2: Create the necessary SQL tables:

- Use the CREATE TABLE statement to create tables with the defined structure.
- Specify the data types and constraints for each column.

Step-3: Populate the tables:

- Use the INSERT INTO statement to insert sample data into the tables.
- Add records for customers, products, and orders.

Step-4: Implement basic operations:

- Create SQL queries to perform basic operations like selecting, inserting, updating, and deleting records.
- For example, you can write queries to retrieve customer details, list available products, add new customers, update product quantities, etc.

Step-5: Implement advanced operations:

- Develop queries to handle more complex operations such as processing orders, calculating total prices, generating reports, etc.
- For example, you can create queries to create , read , update and delete an user ; retrieve case studies for a specific user, generate crime reports for a given period, etc.

Program:

Adminlog.java

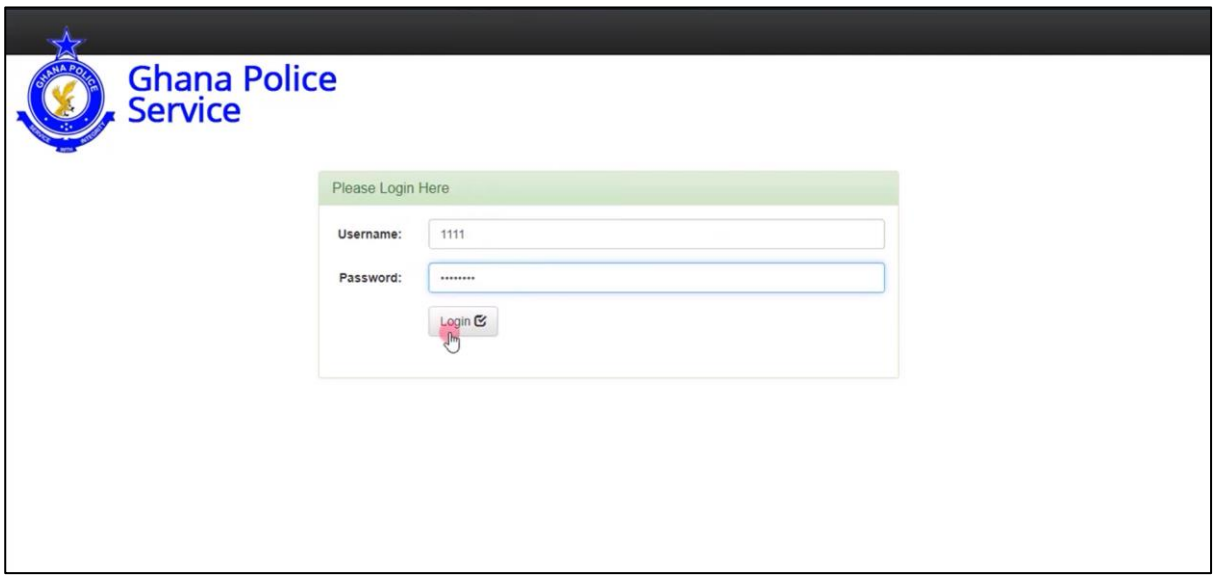
```
 SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";
-- Table structure for table `case_table`

CREATE TABLE `case_table` (
  `case_id` varchar(20) NOT NULL,`statement` varchar(200) NOT NULL,
  `caseid` int(11) NOT NULL,
  `date_added` datetime DEFAULT
NULL,
  `staffid` varchar(30) NOT NULL,
  `case_type` varchar(50) NOT NULL,
  `status` varchar(50) NOT NULL,
  `cid` varchar(20) NOT NULL
DEFAULT
   'Not Yet',
  `complete_date` date NOT NULL,
  `diaryofaction` varchar(200) NOT NULL
)
 ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;
```

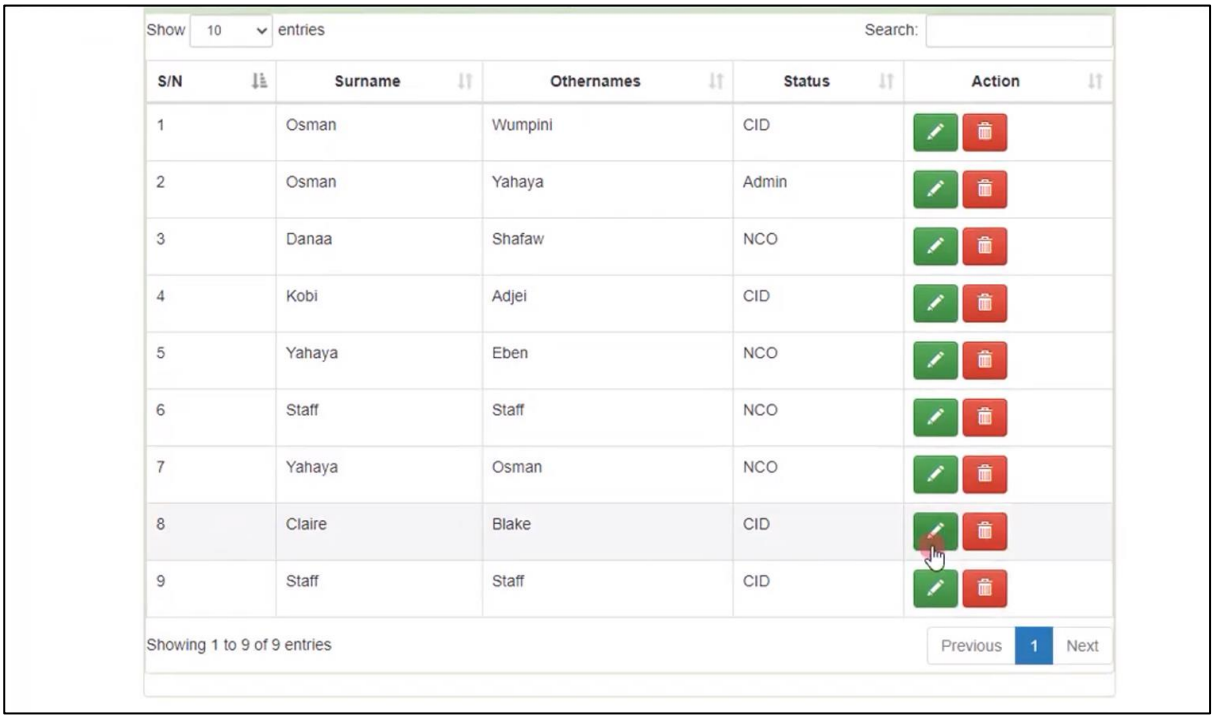
```
-- Dumping data for table `case_table`
INSERT INTO `case_table` (`case_id`, `statement`, `caseid`, `date_added`, `staffid`, `case_type`,
`status`, `cid`, `complete_date`, `diaryofaction`) VALUES
('210728101', '<p>HelLo,</p>\r\n\r\n<p>This CID Officer yahaya and this my findings so for in the
case</p>\r\n', 56, '2021-07-28 13:13:49', '333', 'Robbing', 'Completed', '005', '0000-00-00', ' This is
case  '),
('210728102', '', 57, '2021-07-28 13:14:53', '333', 'Assault', '', 'cid', '0000-00-00', 'Here is anotehr
acse');

-- Table structure for table `complainant`
CREATE TABLE `complainant` (
  `case_id` varchar(20) NOT NULL,
  `comp_name` varchar(100) NOT NULL,
  `tel` varchar(10) NOT NULL,
  `occupation` varchar(20) NOT NULL,
  `region` varchar(50) NOT NULL,
  `district` varchar(100) NOT NULL,
  `loc` varchar(50) NOT NULL,
  `addrs` varchar(100) NOT NULL,
  `age` int(3) NOT NULL,
  `gender` varchar(6) NOT NULL,
  `date_added` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE
current_timestamp()
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Dumping data for table `complainant`
INSERT INTO `complainant` (`case_id`, `comp_name`, `tel`, `occupation`, `region`, `district`,
`loc`, `addrs`, `age`, `gender`, `date_added`) VALUES
('210707101', 'Hanan Gundaadoo', '234567', '34567', 'Bono Region', 'Tain District', 'Sunyani',
'67788', 22, 'Male', '2021-07-07 10:56:23'),
('210713102', 'New Case', '345678', 'fae', 'Bono Region', 'Jaman South Municipal', 'wertyui', 'ddd', 33,
'Male', '2021-07-13 09:20:49'),
('210713103', 'asdfgh', '567890', 'dfghjk', 'Ahafo Region', 'Asunafo South District', 'ertyu', 'erty', 456,
'Female', '2021-07-13 09:27:48'),
('210728101', 'Yahaya Osman', '0509997797', 'Lecturer', 'Bono Region', 'Dormaa Central Municipal',
'Sunyani', 'Hse No. 60/G, Kotokrom', 55, 'Male', '2021-07-28 13:12:57'),
('210728102', 'Yahaya Osman', '0509997797', 'Lecturer', 'Bono East Region', 'Pru West District',
'Sunyani', 'Hse No. 60/G, Kotokrom', 89, 'Male', '2021-07-28 13:14:41');

-- Table structure for table `crime_type`
CREATE TABLE `crime_type` (
  `id` int(11) NOT NULL,
  `des` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;

-- Dumping data for table `crime_type`
INSERT INTO `crime_type` (`id`, `des`)
VALUES
(1, 'Domestic Violence'),
(2, 'Murder Case'),
(3, 'Assault'),
(4, 'Theft Case'),
(5, 'Defilement'),
(6, 'Robbing'),
(7, 'Fraud'),
```

```
(8, 'Others');

-- Table structure for table `investigation`
CREATE TABLE `investigation` (
  `case_id` varchar(20) NOT NULL,
  `investigator` varchar(20) NOT NULL,
  `statement2` text NOT NULL,
  `assigned_date` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE
current_timestamp(),
  `status2` varchar(100) NOT NULL,
  `completed_date` varchar(20) NOT
NULL,
  `id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Dumping data for table `investigation`
INSERT INTO `investigation` (`case_id`, `investigator`, `statement2`, `assigned_date`, `status2`,
`completed_date`, `id`) VALUES
('210707101', '005', '<p>thi is a cse</p>\r\n', '2021-07-07 11:03:58', 'Completed', '', 55);

-- Table structure for table `userlogin`
CREATE TABLE `userlogin` (
  `id` int(11) NOT NULL,
  `staffid` varchar(20) NOT NULL,
  `status` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL,
  `surname` varchar(50) NOT NULL,
  `othernames` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT
CHARSET=utf8mb4;

-- Dumping data for table `userlogin`
INSERT INTO `userlogin` (`id`, `staffid`, `status`, `password`, `surname`, `othernames`) VALUES
(0, '005', 'CID', '8cb2237d0679ca88db6464eac60da96345513964', 'Osman ', 'Wumpini'),
(0, '1111', 'Admin', 'f865b53623b121fd34ee5426c792e5c33af8c227', 'Osman', 'Yahaya'),
(0, '112', 'NCO', '7c4a8d09ca3762af61e59520943dc26494f8941b', 'Danaa', 'Shafaw'),
(0, '113', 'CID', '7110eda4d09e062aa5e4a390b0a572ac0d2c0220', 'Kobi', 'Adjei'),
(0, '222', 'NCO', '7110eda4d09e062aa5e4a390b0a572ac0d2c0220', 'Yahaya', 'Eben'),
(0, '333', 'NCO', '7110eda4d09e062aa5e4a390b0a572ac0d2c0220', 'Staff', 'Staff'),
(0, '4444', 'NCO', '8cb2237d0679ca88db6464eac60da96345513964', 'Yahaya', 'Osman'),
(0, 'cid', 'CID', '7110eda4d09e062aa5e4a390b0a572ac0d2c0220', 'Staff ', 'Staff');

-- Indexes for dumped tables
-- Indexes for table `case_table`
ALTER TABLE `case_table`
  ADD PRIMARY KEY (`caseid`);

-- Indexes for table `complainant`
ALTER TABLE `complainant`
  ADD PRIMARY KEY (`case_id`);

-- Indexes for table `crime_type`
ALTER TABLE `crime_type`
  ADD PRIMARY KEY (`id`);

-- Indexes for table `investigation`
```

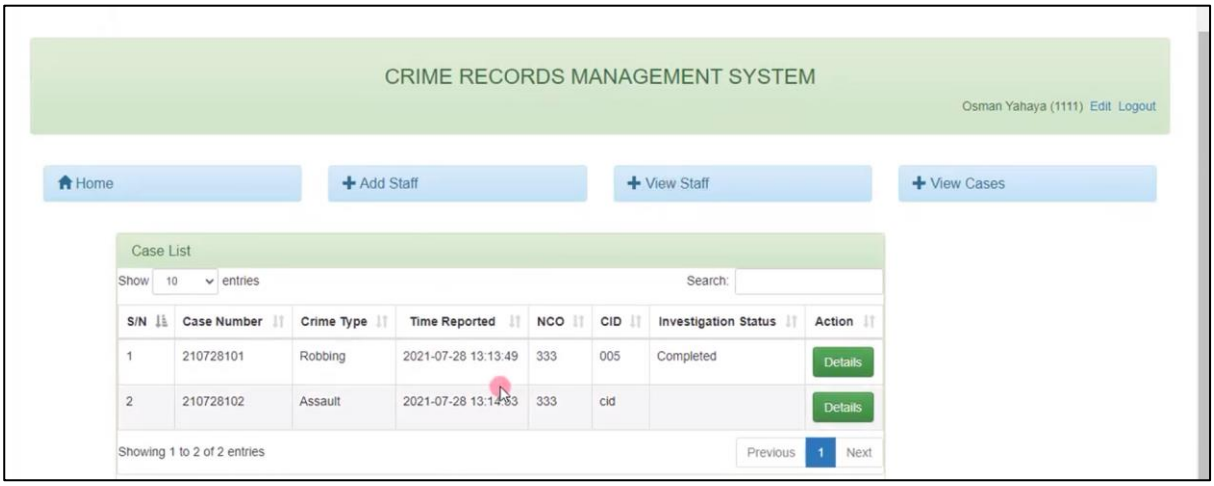
```
ALTER TABLE `investigation`
  ADD PRIMARY KEY (`id`);
-- Indexes for table `userlogin`
ALTER TABLE `userlogin`
  ADD PRIMARY KEY (`staffid`);

-- AUTO_INCREMENT for dumped
tables
-- AUTO_INCREMENT for table `case_table`
ALTER TABLE `case_table`
  MODIFY `caseid` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=59;

-- AUTO_INCREMENT for table `crime_type`
ALTER TABLE `crime_type`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;

-- AUTO_INCREMENT for table `investigation`
ALTER TABLE `investigation`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=56;
COMMIT;
```

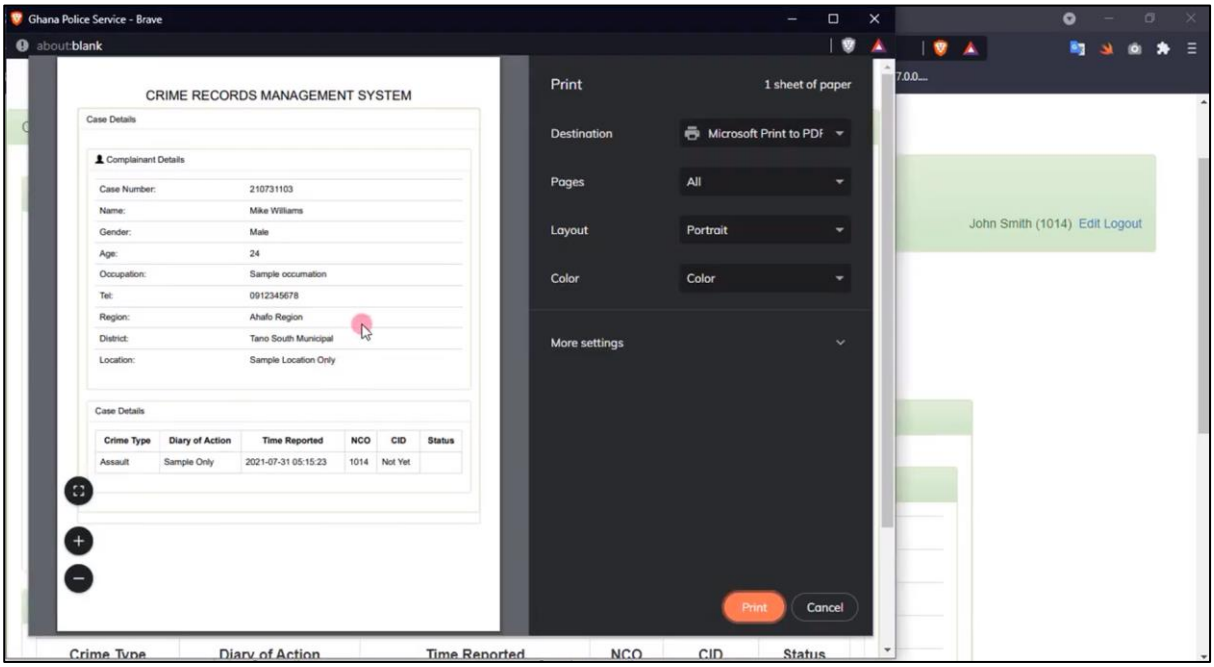
OUTPUT:

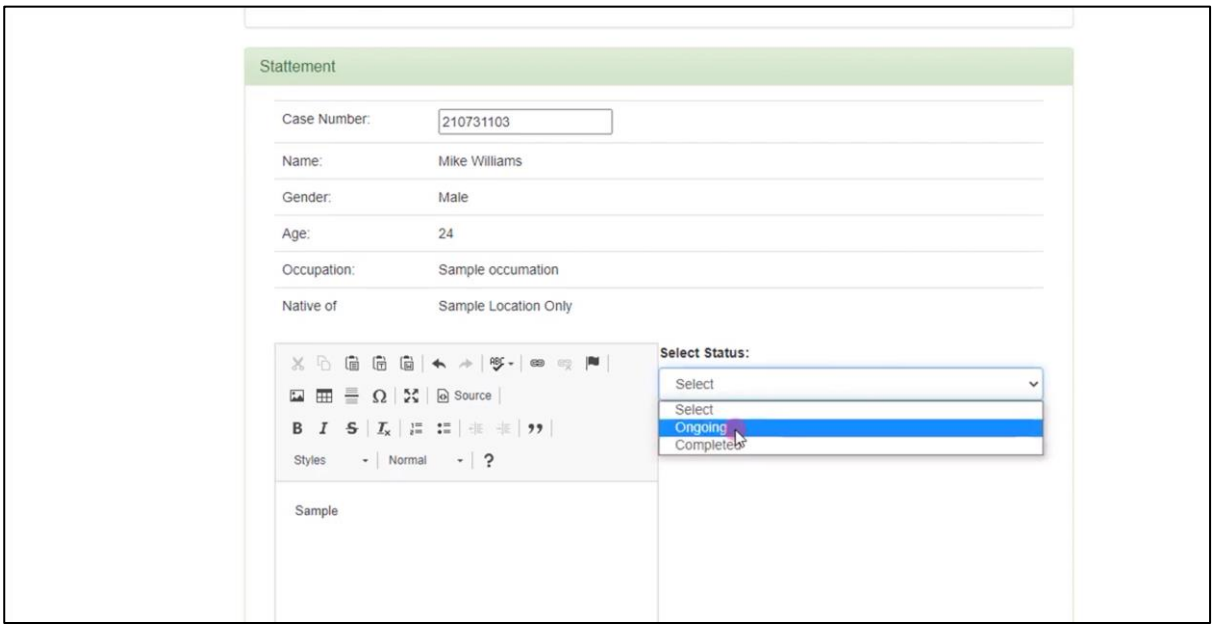



Show 10 entries
Search:
S/N
Surname
Othernames
Status
Action
1 Osman Wumpini CID
2 Osman Yahaya Admin
3 Danaa Shafaw NCO
4 Kobi Adjei CID
5 Yahaya Eben NCO
6 Staff Staff NCO
7 Yahaya Osman NCO
8 Claire Blake CID
9 Staff Staff CID
Showing 1 to 9 of 9 entries
Previous 1 Next


CRIME RECORDS MANAGEMENT SYSTEM
Osman Yahaya (1111) Edit Logout
Home
Add Staff
View Staff
View Cases
Enter Login Details
Firstname:
John
Othernames:
Smith
Staff Number:
1014
Password:
Select Status:
NCO
Save and Continue


CRIME RECORDS MANAGEMENT SYSTEM
Osman Yahaya (1111) Edit Logout
Home
Add Staff
View Staff
View Cases
Case List
Show 10 entries
Search:
S/N
Case Number
Crime Type
Time Reported
NCO
CID
Investigation Status
Action
1 210728101 Robbing 2021-07-28 13:13:49 333 005 Completed Details
2 210728102 Assault 2021-07-28 13:14:33 333 cid Details
Showing 1 to 2 of 2 entries
Previous 1 Next





RESULT:

Thus the above program to build cop – friendly application by using Database management system has been executed and the output is verified successfully.

Exp No: 13(d) Date:	**PROPERTY MANAGEMENT SYSTEM**

Aim:

The aim is to create an application to maintain and manage the properties in the single roof application using the help of SQL.

Algorithm:

Step 1: Define the database schema: Identify tables for property, tenant, rental history, etc. Define columns and data types.

Step 2: Set up the database: Create a new database and establish a connection to it.

Step 3: Create tables: Write SQL queries to create tables based on the schema. Include primary and foreign key constraints for data integrity. Consider adding indices for efficient data retrieval.

Step 4: Implement data entry functionality: Develop forms or user interface components to allow users to enter property and tenant information. Capture relevant data such as property details, tenant details, rental agreements, etc.

Step 5: Implement data retrieval functionality: Build queries to retrieve property and tenant information based on various criteria. Allow users to search for available properties, view tenant details, or generate reports on rental income. Consider using indices to optimize the performance of frequently executed queries.

Step 6: Implement update and delete functionality: Create forms or interfaces to enable users to update or delete property and tenant records. Implement appropriate validation to prevent accidental data loss or corruption. Ensure to update the indices when modifying the data to maintain consistency and efficiency.

Step 7: Implement property allocation functionality: Develop a process to assign tenants to available properties. Implement checks to prevent double booking and update the relevant tables accordingly. Ensure that the indices are updated to reflect the changes in property occupancy.

Step 8: Implement rental payment tracking: Design a mechanism to record and track rental payments made by tenants. Store the payment details in the appropriate tables and update the payment status as. Consider using indices for efficient retrieval of payment records.

Step 9: Implement reporting functionality: Create queries or procedures to generate reports on property occupancy, rental income, tenant information, and other relevant metrics. Customize the reports based on the specific requirements of the property management system. Optimize the queries with appropriate indices to enhance report generation performance.

Step 10: Implement security measures: Add authentication and authorization mechanisms to ensure only authorized users can access and modify the data. Apply encryption and other security measures to protect sensitive information. Consider implementing access control through the use of indices to restrict unauthorized access to specific data.

Step 11: Test the application: Conduct thorough testing to ensure the system functions as expected. Perform both positive and negative tests to verify data entry, retrieval, and other functionalities. Validate the performance of the application with different data loads and analyze the efficiency of index usage.

Step 12: Deploy the application: Once testing is complete, deploy the application to the production environment. Ensure that the necessary hardware and software requirements are met. Monitor the application for any issues or performance bottlenecks.

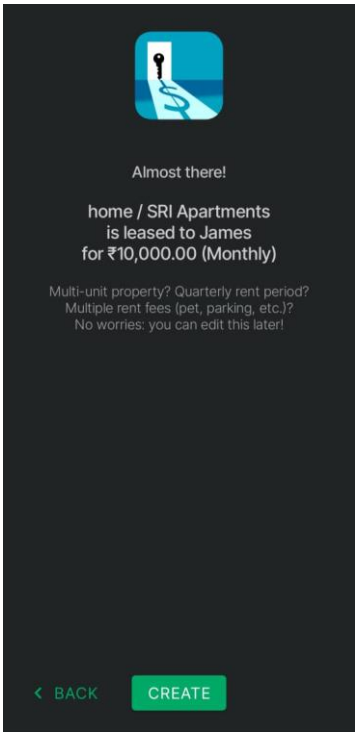
Step 13: Monitor and maintain the system: Regularly monitor the application for any issues or performance degradation. Address reported bugs or user feedback promptly. Perform routine maintenance tasks such as database backups, software updates, and security patches. Optimize index usage periodically to ensure optimal performance.

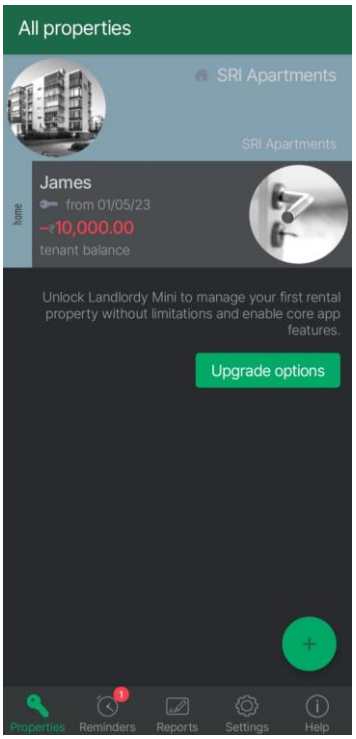
Source Code:

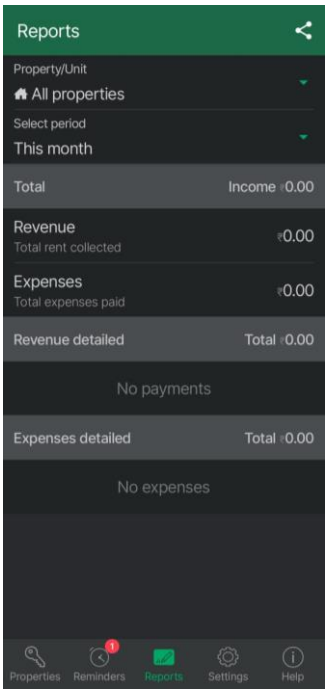
```
import mysql.connector

# Connect to the database
connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password",
    database="property_management"
)

# Create a cursor
cursor = connection.cursor()

# Create a table
sql = """
CREATE TABLE properties (
    property_id INT NOT NULL AUTO_INCREMENT,
    property_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    city VARCHAR(255) NOT NULL,
    state VARCHAR(255) NOT NULL,
    zip_code VARCHAR(10) NOT NULL,
    property_type VARCHAR(255) NOT NULL,
    number_of_bedrooms INT NOT NULL,
    number_of_bathrooms INT NOT NULL,
    rent_amount DECIMAL(10,2) NOT NULL,
    available_date DATE NOT NULL,
    PRIMARY KEY (property_id)
)
"""

cursor.execute(sql)

# Insert data into the table
sql = """
INSERT INTO properties (property_name, address, city, state, zip_code, property_type,
number_of_bedrooms, number_of_bathrooms, rent_amount, available_date)
VALUES
('Apartment 1', '123 Main Street', 'Anytown', 'CA', '91234', 'Apartment', 2, 1, 1200.00, '2023-05-01'),
('Apartment 2', '456 Elm Street', 'Anytown', 'CA', '91234', 'Apartment', 1, 1, 1000.00, '2023-06-01'),
('House 1', '789 Oak Street', 'Anytown', 'CA', '91234', 'House', 3, 2, 2000.00, '2023-07-01'),
('House 2', '1011 Pine Street', 'Anytown', 'CA', '91234', 'House', 4, 3, 3000.00, '2023-08-01'),
('Apartment 3', '1234 Main Street', 'Los Angeles', 'CA', '90001', 'Apartment', 3, 2, 2500.00, '2023-05-
15'),
```

```
('Apartment 4', '5678 Elm Street', 'San Francisco', 'CA', '94105', 'Apartment', 2, 1, 1500.00, '2023-06-15'),
('House 3', '9012 Oak Street', 'San Diego', 'CA', '92101', 'House', 4, 3, 3500.00, '2023-07-15'),
('House 4', '10111 Pine Street', 'Sacramento', 'CA', '95814', 'House', 5, 4, 4500.00, '2023-08-15')
"""

cursor.execute(sql)

# Commit the changes to the database
connection.commit()

# Close the cursor
cursor.close()

# Close the connection
connection.close()
```

Project Screenshots:







Result:

The application has been successfully implemented and executed without any major issues or flaws.