K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: A3 Roll No.: 1911034

Experiment / assignment / tutorial No. 8

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementing TCL/DCL

Objective: To be able to Implement TCL and DCL.

Expected Outcome of Experiment:

CO 2: Convert entity-relationship diagrams into relational tables, populate a relational database and formulate SQL queries on the data Use SQL for creation and query the database.

CO 4: Demonstrate the concept of transaction, concurrency control and recovery techniques.

Books/ Journals/ Websites referred:

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Slberchatz, Sudarshan : "Database Systems Concept", 5th Edition , McGraw Hill
4. Elmasri and Navathe,"Fundamentals of database Systems", 4th Edition,PEARSON Education.

Resources used: PostgreSQL

Theory

DCL stands for Data Control Language.

DCL is used to control user access in a database.

This command is related to the security issues.

Using DCL command, it allows or restricts the user from accessing data in database schema.

DCL commands are as follows,

GRANT

REVOKE

It is used to grant or revoke access permissions from any database user.

GRANT command gives user's access privileges to the database.

This command allows specified users to perform specific tasks.

Syntax:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name
[, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]
```

Example

```
GRANT INSERT ON films TO PUBLIC;
GRANT ALL PRIVILEGES ON kinds TO ram;
GRANT admins TO krishna;
```

REVOKE command is used to cancel previously granted or denied permissions.

This command withdraw access privileges given with the GRANT command.

It takes back permissions from user.

Syntax:

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
```

```
          | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [,
...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Example

```
REVOKE INSERT ON films FROM PUBLIC;
REVOKE ALL PRIVILEGES ON kinds FROM Madhav;
REVOKE admins FROM Keshav;
```

TCL stands for **Transaction Control Language.**

This command is used to manage the changes made by DML statements.

TCL allows the statements to be grouped together into logical transactions.

TCL commands are as follows:
1. COMMIT
2. SAVEPOINT
3. ROLLBACK
4. SET TRANSACTION

COMMIT command saves all the work done. It ends the current transaction and makes permanent changes during the transaction

Syntax:
commit;

SAVEPOINT command is used for saving all the current point in the processing of a transaction. It marks and saves the current point in the processing of a transaction. It is used to temporarily save a transaction, so that you can rollback to that point whenever necessary.

Syntax

```
SAVEPOINT savepoint_name
```

ROLLBACK command restores database to original since the last COMMIT. It is used to restores the database to last committed state.

Syntax:

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ]
savepoint_name
```

Example

```
BEGIN;
    INSERT INTO table1 VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (3);
COMMIT;
```

The above transaction will insert the values 1 and 3, but not 2.

SET TRANSACTION is used for placing a name on a transaction. You can specify a transaction to be read only or read write. This command is used to initiate a database transaction.

Syntax:

SET TRANSACTION [Read Write | Read Only];

The SET TRANSACTION command sets the characteristics of the current transaction. It has no effect on any subsequent transactions. SET SESSION CHARACTERISTICS sets the default transaction characteristics for subsequent transactions of a session. These defaults can be overridden by SET TRANSACTION for an individual transaction.

The available transaction characteristics are the transaction isolation level, the transaction access mode (read/write or read-only), and the deferrable mode. In addition,

a snapshot can be selected, though only for the current transaction, not as a session default.
The isolation level of a transaction determines what data the transaction can see when other transactions are running concurrently:

READ COMMITTED
A statement can only see rows committed before it began. This is the default.

REPEATABLE READ
All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

SERIALIZABLE
All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If a pattern of reads and writes among concurrent serializable transactions would create a situation which could not have occurred for any serial (one-at-a-time) execution of those transactions, one of them will be rolled back with a serialization_failure error.

Examples

With the default read committed isolation level.

```
process A: BEGIN; -- the default is READ COMMITED

process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 1600

process B: INSERT INTO purchases (value) VALUES (400)
--- process B inserts a new row into the table while
--- process A's transaction is in progress

process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 2000

process A: COMMIT;
```

If we want to avoid the changing sum value in process A during the lifespan of the transaction, we can use the repeatable read transaction mode.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 1600

process B: INSERT INTO purchases (value) VALUES (400)
--- process B inserts a new row into the table while
--- process A's transaction is in progress
```

```
process A: SELECT sum(value) FROM purchases;
--- process A still sees that the sum is 1600

process A: COMMIT;
```

The transaction in process A fill freeze its snapshot of the data and offer consistent values during the life of the transaction.

Repeatable reads are not more expensive than the default read commit transaction. There is no need to worry about performance penalties. However, applications must be prepared to retry transactions due to serialization failures.

Let's observe an issue that can occur while using the repeatable read isolation level — the `could not serialize access due to concurrent update` error.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process B: BEGIN;
process B: UPDATE purchases SET value = 500 WHERE id = 1;
process A: UPDATE purchases SET value = 600 WHERE id = 1;
-- process A wants to update the value while process B is changing it
-- process A is blocked until process B commits

process B: COMMIT;
process A: ERROR:  could not serialize access due to concurrent update
-- process A immidiatly errors out when process B commits
```

If process B would rolls back, then its changes are negated and repeatable read can proceed without issues. However, if process B commits the changes then the repeatable read transaction will be rolled back with the error message because it can not modify or lock the rows changed by other processes after the repeatable read transaction has began.

demonstrate the differences between the two isolation modes.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process A: SELECT sum(value) FROM purchases;
process A: INSERT INTO purchases (value) VALUES (100);
process B: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process B: SELECT sum(value) FROM purchases;
process B: INSERT INTO purchases (id, value);
process B: COMMIT;
process A: COMMIT;
```

With Repeatable Reads everything works, but if we run the same thing with a Serializable isolation mode, process A will error out.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
process A: SELECT sum(value) FROM purchases;
process A: INSERT INTO purchases (value) VALUES (100);
process B: BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
process B: SELECT sum(value) FROM purchases;
process B: INSERT INTO purchases (id, value);
process B: COMMIT;
process A: COMMIT;
ERROR: could not serialize access due to read/write
dependencies among transactions
DETAIL: Reason code: Canceled on identification as
a pivot, during commit attempt.
HINT: The transaction might succeed if retried.
```

Both transactions have modified what the other transaction would have read in the select statements. If both would allow to commit this would violate the Serializable behaviour, because if they were run one at a time, one of the transactions would have seen the new record inserted by the other transaction.

Implementation Screenshots (Problem Statement, Query and Screenshots of Results):

Demonstrate DCL and TCL language commands on your database.

DCL commands :

Creating User and Granting Privileges to user:

Database used : Property Management System.

Tables it has : Builder , Contractor , Customer , Employee, property

Screenshot of Property Table:

```
mysql> select * from property;
+------+-----------+---------+-------------+-----------+------+
| area | location  | cost    | no_of_rooms | type_p    | p_id |
+------+-----------+---------+-------------+-----------+------+
|  200 | mumbai    | 2355.16 |           4 | rental    | 1111 |
|  350 | pune      | 1297.66 |           3 | ownership | 2110 |
|  410 | thane     | 3500.01 |           2 | rental    | 2159 |
|  300 | churchgate| 3500.77 |           3 | rental    | 2211 |
|  250 | andheri   | 2200.22 |           2 | rental    | 3221 |
|  500 | ghatkopar | 1799.66 |           5 | ownsership| 7651 |
+------+-----------+---------+-------------+-----------+------+
6 rows in set (0.00 sec)
```

Screenshot of customer table:

```
mysql> select * from customer;
+----------+-------+-------+--------+-----------+-----------+-----------+
| name_c   | age_c | id_no | budget | type_p    | no_of_emi | asc_bank  |
+----------+-------+-------+--------+-----------+-----------+-----------+
| Ashwini  |    48 |  1122 |   5000 | ownership |        12 | HDFC Bank |
| Aditi    |    19 |  1210 |   9000 | rental    |         7 | ICPC Bank |
| Dhruvi   |    19 |  1998 |  10000 | rental    |         9 | HDFC Bank |
| Samiksha |    19 |  2133 |   4500 | ownership |         8 | Canara    |
| Madhuri  |    25 |  9021 |   3000 | rental    |         9 | HDFC Bank |
| Pinky    |    21 |  9987 |   2300 | rental    |         9 | Baroda    |
| Siddhi   |    22 |  9989 |   3000 | ownership |        10 | Canara    |
+----------+-------+-------+--------+-----------+-----------+-----------+
7 rows in set (0.00 sec)
```

First we will create a new user in MySQL:

```
mysql> CREATE USER 'Arvind'@'localhost' IDENTIFIED by '24042001';
Query OK, 0 rows affected (0.02 sec)
```

We have granted the user ‘Arvind’ the rights to select and insert values from and to the property table:

```
mysql> GRANT SELECT, INSERT ON property TO 'Arvind'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

Now we will go to command line client and login to mysql with username and password provided for the user ‘Arvind’

```
C:\Users\arvin>cd C:\Program Files\MySQL\MySQL Server 8.0\bin

C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -uArvind -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

As we have provided the user to perform only 'SELECT' and 'INSERT' on 'property' table to user 'Arvind' , when we try to access the customer table from the user , it will show us access denied:

```
mysql> use propsys;
Database changed
mysql> select * from customer;
ERROR 1142 (42000): SELECT command denied to user 'Arvind'@'localhost' for table 'customer'
```

Let us try to insert a new row into property table and display it's values from user 'Arvind':

```
mysql> INSERT INTO PROPERTY VALUES (350, "kalyan", 4511.17 , 3 , "rental", 3558);
Query OK, 1 row affected (0.01 sec)

mysql> select * from property;
+------+------------+---------+-------------+------------+------+
| area | location   | cost    | no_of_rooms | type_p     | p_id |
+------+------------+---------+-------------+------------+------+
|  200 | mumbai     | 2355.16 |           4 | rental     | 1111 |
|  350 | pune       | 1297.66 |           3 | ownership  | 2110 |
|  410 | thane      | 3500.01 |           2 | rental     | 2159 |
|  300 | churchgate | 3500.77 |           3 | rental     | 2211 |
|  250 | andheri    | 2200.22 |           2 | rental     | 3221 |
|  350 | kalyan     | 4511.17 |           3 | rental     | 3558 |
|  500 | ghatkopar  | 1799.66 |           5 | ownsership | 7651 |
+------+------------+---------+-------------+------------+------+
7 rows in set (0.00 sec)
```

If we try to update 'property' from user 'Arvind' it will throw an error as the user has not been granted the rights to do so:

```
mysql> UPDATE property SET location = "pune" WHERE p_id = 1111;
ERROR 1142 (42000): UPDATE command denied to user 'Arvind'@'localhost' for table 'property'
mysql>
```

Granting EXECUTE privilege for a procedure to user Arvind : for this we log out of the user 'ARVIND' and login to default user where we had created the user 'Arvind'

```
mysql> use propsys;
Database changed
mysql> GRANT EXECUTE ON PROCEDURE loc_cost TO 'Arvind'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Executing loc_cost() from user 'Arvind'

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -uArvind -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 29
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> use propsys;
Database changed
mysql> call loc_cost();
+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| mumbai 2355.16               |
+------------------------------+
1 row in set (0.00 sec)

+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| pune 1297.66                 |
+------------------------------+
1 row in set (0.01 sec)

+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| thane 3500.01                |
+------------------------------+
1 row in set (0.02 sec)

+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| churchgate 3500.77           |
+------------------------------+
1 row in set (0.02 sec)

+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| andheri 2200.22              |
+------------------------------+
1 row in set (0.03 sec)
```

```
+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| kalyan 4511.17               |
+------------------------------+
1 row in set (0.03 sec)

+------------------------------+
| concat(locat , concat(" ",c)) |
+------------------------------+
| ghatkopar 1799.66            |
+------------------------------+
1 row in set (0.04 sec)

Query OK, 0 rows affected (0.04 sec)

mysql>
```

Revoking select privilege from user:

```
mysql> use propsys;
Database changed
mysql> REVOKE SELECT ON property FROM 'Arvind'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql>
```

After revoking :

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -uArvind -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 8.0.23 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use propsys;
Database changed
mysql> select * from property;
ERROR 1142 (42000): SELECT command denied to user 'Arvind'@'localhost' for table 'property'
mysql>
```

TCL Commands:

TCL Commands :

1. Setting AUTOCOMMIT = “false” and using ROLLBACK to revert the changes done:

```
mysql> SET autocommit =0;
Query OK, 0 rows affected (0.00 sec)
```

First we have inserted a new customer in the customer table with name =”Nina” and id = 2345

```
mysql> INSERT INTO Customer VALUES ("Nina", 25, 2345, 3000, "ownership", 5, "HDFC Bank");
Query OK, 1 row affected (0.01 sec)

mysql> select * from customer;
+----------+-------+-------+--------+-----------+-----------+-----------+
| name_c   | age_c | id_no | budget | type_p    | no_of_emi | asc_bank  |
+----------+-------+-------+--------+-----------+-----------+-----------+
| Ashwini  |    48 |  1122 |   5000 | ownership |        12 | HDFC Bank |
| Aditi    |    19 |  1210 |   9000 | rental    |         7 | ICPC Bank |
| Dhruvi   |    19 |  1998 |  10000 | rental    |         9 | HDFC Bank |
| Samiksha |    19 |  2133 |   4500 | ownership |         8 | Canara    |
| Nina     |    25 |  2345 |   3000 | ownership |         5 | HDFC Bank |
| Madhuri  |    25 |  9021 |   3000 | rental    |         9 | HDFC Bank |
| Pinky    |    21 |  9987 |   2300 | rental    |         9 | Baroda    |
| Siddhi   |    22 |  9989 |   3000 | ownership |        10 | Canara    |
+----------+-------+-------+--------+-----------+-----------+-----------+
8 rows in set (0.00 sec)
```

Let us use ROLLBACK to revert the changes done:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from customer;
+----------+-------+-------+--------+-----------+-----------+-----------+
| name_c   | age_c | id_no | budget | type_p    | no_of_emi | asc_bank  |
+----------+-------+-------+--------+-----------+-----------+-----------+
| Ashwini  |    48 |  1122 |   5000 | ownership |        12 | HDFC Bank |
| Aditi    |    19 |  1210 |   9000 | rental    |         7 | ICPC Bank |
| Dhruvi   |    19 |  1998 |  10000 | rental    |         9 | HDFC Bank |
| Samiksha |    19 |  2133 |   4500 | ownership |         8 | Canara    |
| Madhuri  |    25 |  9021 |   3000 | rental    |         9 | HDFC Bank |
| Pinky    |    21 |  9987 |   2300 | rental    |         9 | Baroda    |
| Siddhi   |    22 |  9989 |   3000 | ownership |        10 | Canara    |
+----------+-------+-------+--------+-----------+-----------+-----------+
7 rows in set (0.00 sec)
```

New customer table after inserting few more values :

```
mysql> SELECT * FROM CUSTOMER;
+-----------+-------+-------+--------+-----------+-----------+-----------+
| name_c    | age_c | id_no | budget | type_p    | no_of_emi | asc_bank  |
+-----------+-------+-------+--------+-----------+-----------+-----------+
| Ashwini   |    48 |  1122 |   5000 | ownership |        12 | HDFC Bank |
| Aditi     |    19 |  1210 |   9000 | rental    |         7 | ICPC Bank |
| Dhruvi    |    19 |  1998 |  10000 | rental    |         9 | HDFC Bank |
| Samiksha  |    19 |  2133 |   4500 | ownership |         8 | Canara    |
| Aakanksha |    25 |  2365 |   3000 | ownership |         5 | HDFC Bank |
| Prerna    |    25 |  2377 |   5000 | rental    |         5 | ICPC Bank |
| Kiara     |    35 |  2775 |   3000 | rental    |         8 | HDFC Bank |
| Khushbu   |    35 |  2945 |   9000 | rental    |         8 | HDFC Bank |
| Madhuri   |    25 |  9021 |   3000 | rental    |         9 | HDFC Bank |
| Pinky     |    21 |  9987 |   2300 | rental    |         9 | Baroda    |
| Siddhi    |    22 |  9989 |   3000 | ownership |        10 | Canara    |
+-----------+-------+-------+--------+-----------+-----------+-----------+
11 rows in set (0.00 sec)
```

Creating SAVEPOINT and ROLLBACK to SAVPOINT;

```
mysql> START TRANSACTION ;
Query OK, 0 rows affected (0.00 sec)

mysql> SAVEPOINT initial_save;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE customer set budget = 2000 WHERE id_no = 1122;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from customer;
+-----------+-------+-------+--------+-----------+-----------+-----------+
| name_c    | age_c | id_no | budget | type_p    | no_of_emi | asc_bank  |
+-----------+-------+-------+--------+-----------+-----------+-----------+
| Ashwini   |    48 |  1122 |   2000 | ownership |        12 | HDFC Bank |
| Aditi     |    19 |  1210 |   9000 | rental    |         7 | ICPC Bank |
| Dhruvi    |    19 |  1998 |  10000 | rental    |         9 | HDFC Bank |
| Samiksha  |    19 |  2133 |   4500 | ownership |         8 | Canara    |
| Aakanksha |    25 |  2365 |   3000 | ownership |         5 | HDFC Bank |
| Prerna    |    25 |  2377 |   5000 | rental    |         5 | ICPC Bank |
| Kiara     |    35 |  2775 |   3000 | rental    |         8 | HDFC Bank |
| Khushbu   |    35 |  2945 |   9000 | rental    |         8 | HDFC Bank |
| Madhuri   |    25 |  9021 |   3000 | rental    |         9 | HDFC Bank |
| Pinky     |    21 |  9987 |   2300 | rental    |         9 | Baroda    |
| Siddhi    |    22 |  9989 |   3000 | ownership |        10 | Canara    |
+-----------+-------+-------+--------+-----------+-----------+-----------+
11 rows in set (0.00 sec)
```

IN this , we have updated budget of customer with id number 1122 after creating a savepoint ‘Initial_save’ when we rollback to savepoint , the original budget of that customer is restored again.

```
mysql> rollback to savepoint initial_save;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customer;
+-----------+-------+-------+--------+-----------+-----------+-----------+
| name_c    | age_c | id_no | budget | type_p    | no_of_emi | asc_bank  |
+-----------+-------+-------+--------+-----------+-----------+-----------+
| Ashwini   |    48 |  1122 |   5000 | ownership |        12 | HDFC Bank |
| Aditi     |    19 |  1210 |   9000 | rental    |         7 | ICPC Bank |
| Dhruvi    |    19 |  1998 |  10000 | rental    |         9 | HDFC Bank |
| Samiksha  |    19 |  2133 |   4500 | ownership |         8 | Canara    |
| Aakanksha |    25 |  2365 |   3000 | ownership |         5 | HDFC Bank |
| Prerna    |    25 |  2377 |   5000 | rental    |         5 | ICPC Bank |
| Kiara     |    35 |  2775 |   3000 | rental    |         8 | HDFC Bank |
| Khushbu   |    35 |  2945 |   9000 | rental    |         8 | HDFC Bank |
| Madhuri   |    25 |  9021 |   3000 | rental    |         9 | HDFC Bank |
| Pinky     |    21 |  9987 |   2300 | rental    |         9 | Baroda    |
| Siddhi    |    22 |  9989 |   3000 | ownership |        10 | Canara    |
+-----------+-------+-------+--------+-----------+-----------+-----------+
11 rows in set (0.01 sec)

mysql>
```

Conclusion: In this experiment we have learnt about the DCL and TCL commands and implemented them on our database propsys.

Postlab question:

1. **Discuss ACID properties of transaction with suitable example**

Ans: A transaction is a very small unit of a program and it may contain several low level tasks. A transaction in a database system must
maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- Atomicity − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
  For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

- Consistency − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
  For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

- Isolation − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.
  For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

- Durability − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.