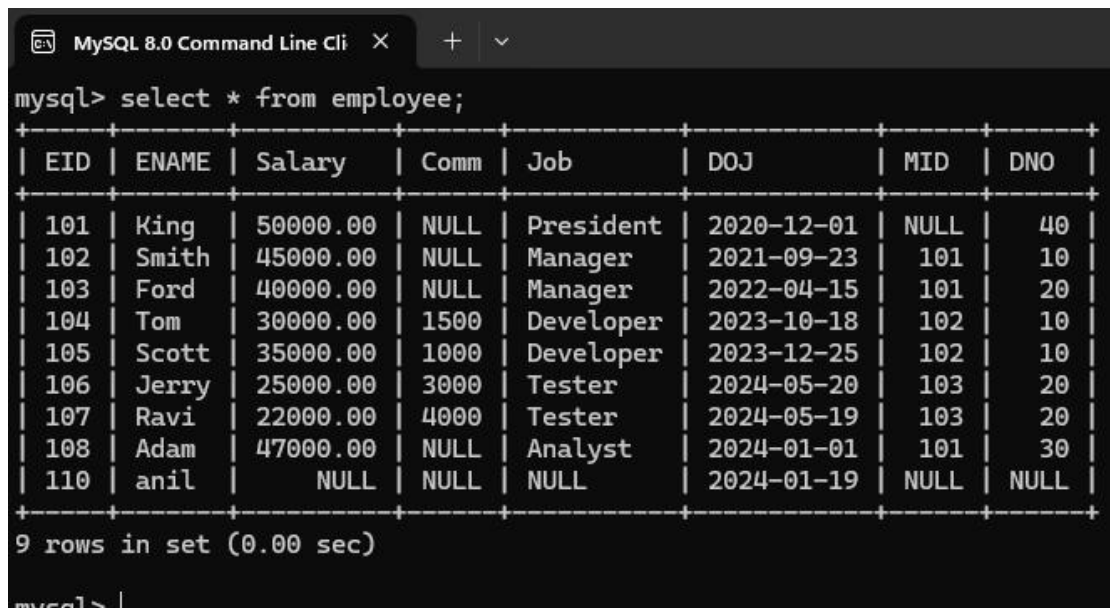


RDBMS and SQL Assignment (part 2)

Assignment 1: Write a SELECT query to retrieve all columns from a 'Employee' table, and modify it to return only the employee name and job for employee in a specific salary range.

To retrieve all columns from a table we use select command.



```
mysql> select * from employee;
```

EID	ENAME	Salary	Comm	Job	DOJ	MID	DNO
101	King	50000.00	NULL	President	2020-12-01	NULL	40
102	Smith	45000.00	NULL	Manager	2021-09-23	101	10
103	Ford	40000.00	NULL	Manager	2022-04-15	101	20
104	Tom	30000.00	1500	Developer	2023-10-18	102	10
105	Scott	35000.00	1000	Developer	2023-12-25	102	10
106	Jerry	25000.00	3000	Tester	2024-05-20	103	20
107	Ravi	22000.00	4000	Tester	2024-05-19	103	20
108	Adam	47000.00	NULL	Analyst	2024-01-01	101	30
110	anil	NULL	NULL	NULL	2024-01-19	NULL	NULL

```
9 rows in set (0.00 sec)
```

```
mysql> |
```

To select or modify to return only specific columns we use the following query.

Syntax: SELECT column1,column2 FROM table_name;

The above syntax returns only specified columns even the table consists of n number of columns.

```
MySQL 8.0 Command Line Cli X + v
mysql> select EID, ENAME, Job from employee;
+-----+-----+-----+
| EID | ENAME | Job |
+-----+-----+-----+
| 108 | Adam | Analyst |
| 110 | anil | NULL |
| 103 | Ford | Manager |
| 106 | Jerry | Tester |
| 101 | King | President |
| 107 | Ravi | Tester |
| 105 | Scott | Developer |
| 102 | Smith | Manager |
| 104 | Tom | Developer |
+-----+-----+-----+
9 rows in set (0.00 sec)

mysql>
```

To return only some rows we use where clause with some condition, if the condition satisfies then the row will be displayed.

Syntax: SELECT column1,coumn2 FROM table_name where condition;

```
MySQL 8.0 Command Line Cli X + v
mysql> select ename, job from employee where salary > 40000;
+-----+-----+
| ename | job |
+-----+-----+
| Adam | Analyst |
| King | President |
| Smith | Manager |
+-----+-----+
3 rows in set (0.00 sec)
```

For Salary less than 40000 we will get other details.

```
MySQL 8.0 Command Line Cli X + v
mysql> select ename, job from employee where salary < 40000;
+-----+-----+
| ename | job   |
+-----+-----+
| Jerry | Tester|
| Ravi  | Tester|
| Scott | Developer|
| Tom   | Developer|
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

For salary equals to 40000 we will get rest details.

```
MySQL 8.0 Command Line Cli X + v
mysql> select ename, job from employee where salary = 40000;
+-----+-----+
| ename | job   |
+-----+-----+
| Ford  | Manager|
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Assignment 2: Craft a query using an INNER JOIN to combine 'library' and 'billing' tables for books in a specified region, and a LEFT JOIN to display all data including those without orders.

INNER JOIN:

INNER JOIN Keyword is used to display or return all values from 2 tables which are matching.

Syntax: `SELECT * FROM table_name1 INNER JOIN table_name2 where condition;`



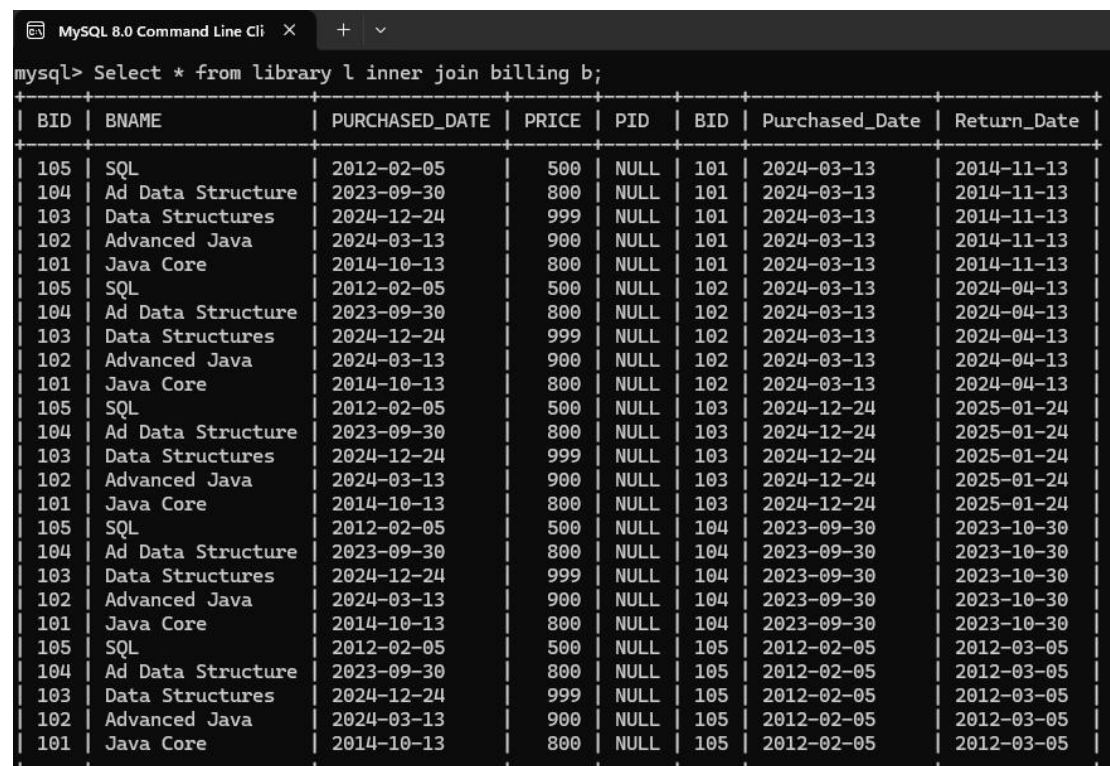
```
mysql> Select * from library l inner join billing b where l.BID = b.BID;
```

BID	BNAME	PURCHASED_DATE	PRICE	PID	BID	Purchased_Date	Return_Date
101	Java Core	2014-10-13	800	NULL	101	2024-03-13	2014-11-13
102	Advanced Java	2024-03-13	900	NULL	102	2024-03-13	2024-04-13
103	Data Structures	2024-12-24	999	NULL	103	2024-12-24	2025-01-24
104	Ad Data Structure	2023-09-30	800	NULL	104	2023-09-30	2023-10-30
105	SQL	2012-02-05	500	NULL	105	2012-02-05	2012-03-05

5 rows in set (0.00 sec)

```
mysql>
```

If we don't mention the condition the table_1 will multiple with table_2 and the result will be



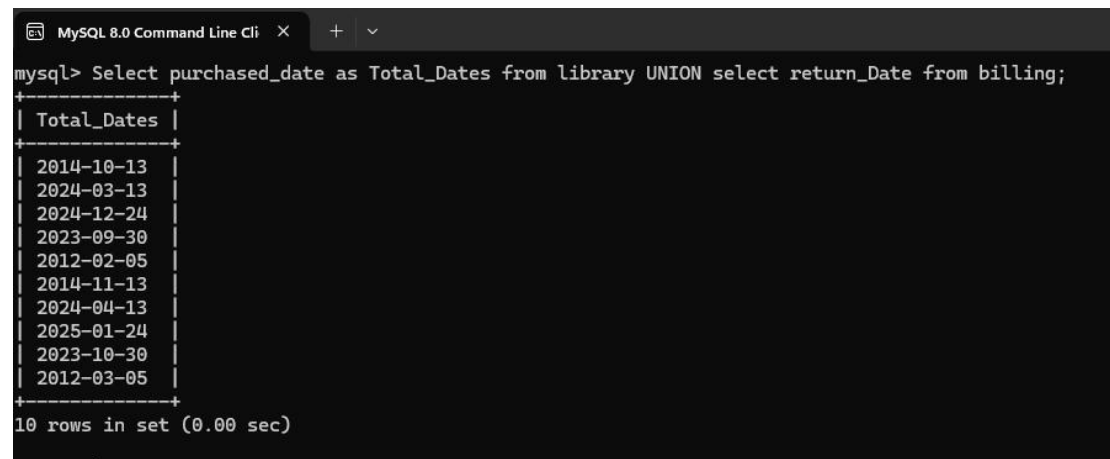
```
mysql> Select * from library l inner join billing b;
```

BID	BNAME	PURCHASED_DATE	PRICE	PID	BID	Purchased_Date	Return_Date
105	SQL	2012-02-05	500	NULL	101	2024-03-13	2014-11-13
104	Ad Data Structure	2023-09-30	800	NULL	101	2024-03-13	2014-11-13
103	Data Structures	2024-12-24	999	NULL	101	2024-03-13	2014-11-13
102	Advanced Java	2024-03-13	900	NULL	101	2024-03-13	2014-11-13
101	Java Core	2014-10-13	800	NULL	101	2024-03-13	2014-11-13
105	SQL	2012-02-05	500	NULL	102	2024-03-13	2024-04-13
104	Ad Data Structure	2023-09-30	800	NULL	102	2024-03-13	2024-04-13
103	Data Structures	2024-12-24	999	NULL	102	2024-03-13	2024-04-13
102	Advanced Java	2024-03-13	900	NULL	102	2024-03-13	2024-04-13
101	Java Core	2014-10-13	800	NULL	102	2024-03-13	2024-04-13
105	SQL	2012-02-05	500	NULL	103	2024-12-24	2025-01-24
104	Ad Data Structure	2023-09-30	800	NULL	103	2024-12-24	2025-01-24
103	Data Structures	2024-12-24	999	NULL	103	2024-12-24	2025-01-24
102	Advanced Java	2024-03-13	900	NULL	103	2024-12-24	2025-01-24
101	Java Core	2014-10-13	800	NULL	103	2024-12-24	2025-01-24
105	SQL	2012-02-05	500	NULL	104	2023-09-30	2023-10-30
104	Ad Data Structure	2023-09-30	800	NULL	104	2023-09-30	2023-10-30
103	Data Structures	2024-12-24	999	NULL	104	2023-09-30	2023-10-30
102	Advanced Java	2024-03-13	900	NULL	104	2023-09-30	2023-10-30
101	Java Core	2014-10-13	800	NULL	104	2023-09-30	2023-10-30
105	SQL	2012-02-05	500	NULL	105	2012-02-05	2012-03-05
104	Ad Data Structure	2023-09-30	800	NULL	105	2012-02-05	2012-03-05
103	Data Structures	2024-12-24	999	NULL	105	2012-02-05	2012-03-05
102	Advanced Java	2024-03-13	900	NULL	105	2012-02-05	2012-03-05
101	Java Core	2014-10-13	800	NULL	105	2012-02-05	2012-03-05

UNION:

UNION command is used for combining 2 tables if, but the table must contain common columns.

Syntax: `SELECT column from table_name1 UNION SELECT column from table_name2;`



```
mysql> Select purchased_date as Total_Dates from library UNION select return_Date from billing;
```

Total_Dates
2014-10-13
2024-03-13
2024-12-24
2023-09-30
2012-02-05
2014-11-13
2024-04-13
2025-01-24
2023-10-30
2012-03-05

```
10 rows in set (0.00 sec)
```

Assignment 4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'library' table, COMMIT the transaction, then UPDATE the 'billing' table, and ROLLBACK the transaction.

Transaction:

Transaction in mysql is a group of statements refers to a single unit of work. Which can be revert back if needed, but only before commit.

Syntax:

Start transaction

Statement_1;

Statement_2;

:

:

Statement_n;

Commit;

// if you don't want to save then rollback

```
MySQL 8.0 Command Line Cli X + v
mysql> Start Transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO library VALUES (106,'Data Analysis','2024-05-13',500, 501);
Query OK, 1 row affected (0.05 sec)

mysql> commit;
Query OK, 0 rows affected (0.10 sec)

mysql>
```

Note: In the above statements if commit is not used then the statements are temporarily execute(if we close mysql then data will delete) to save the statements executed we need to use commit statement.

ROLLBACK:

Rollback is used to rollback to the current transaction to the starting o the transaction.

```
MySQL 8.0 Command Line Cli X + v
mysql> Start Transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO billing VALUES (106,'2024-05-13','2024-06-13');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM billing;
+-----+-----+-----+
| BID | Purchased_Date | Return_Date |
+-----+-----+-----+
| 101 | 2024-03-13     | 2014-11-13  |
| 102 | 2024-03-13     | 2024-04-13  |
| 103 | 2024-12-24     | 2025-01-24  |
| 104 | 2023-09-30     | 2023-10-30  |
| 105 | 2012-02-05     | 2012-03-05  |
| 106 | 2024-05-13     | 2024-06-13  |
+-----+-----+-----+
6 rows in set (0.00 sec)

mysql>
```

At this state if we use rollback all the statements in the transaction will get rollback.

```
MySQL 8.0 Command Line Cli X + v
mysql> Rollback;
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM billing;
+-----+-----+-----+
| BID | Purchased_Date | Return_Date |
+-----+-----+-----+
| 101 | 2024-03-13     | 2014-11-13  |
| 102 | 2024-03-13     | 2024-04-13  |
| 103 | 2024-12-24     | 2025-01-24  |
| 104 | 2023-09-30     | 2023-10-30  |
| 105 | 2012-02-05     | 2012-03-05  |
+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

In the above query the inserted statements where got reverted.

Assignment 5: Begin a transaction, perform a series of INSERTs into 'library', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

SAVEPOINT:

Savepoint is a logical rollback point where we can rollback to a certain point in a transaction.

Syntax:

Start Transaction

```
Statement 1;
SAVEPOINT identifier;
Statement 2;
:
:
```

Rollback to SAVEPOINT identifier;


```
MySQL 8.0 Command Line Cli X + v
mysql> Start Transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO library VALUES (107,'ADS','2024-07-05',600, 507);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT savepoint1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO library VALUES (108,'DSA','2024-08-09',700, 508);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM library;
+-----+-----+-----+-----+-----+
| BID | BNAME          | PURCHASED_DATE | PRICE | PID |
+-----+-----+-----+-----+-----+
| 101 | Java Core      | 2014-10-13     | 800   | NULL |
| 102 | Advanced Java  | 2024-03-13     | 900   | NULL |
| 103 | Data Structures | 2024-12-24     | 999   | NULL |
| 104 | Ad Data Structure | 2023-09-30     | 800   | NULL |
| 105 | SQL            | 2012-02-05     | 500   | NULL |
| 106 | Data Analysis  | 2024-05-13     | 500   | 501 |
| 107 | ADS            | 2024-07-05     | 600   | 507 |
| 108 | DSA            | 2024-08-09     | 700   | 508 |
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql>
```

At this state if we use rollback to specific savepoint then till savepoint1 it will be rollback.

```
MySQL 8.0 Command Line Cli  X  +  v

mysql> Rollback TO SAVEPOINT savepoint1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM library;
+-----+-----+-----+-----+-----+
| BID | BNAME          | PURCHASED_DATE | PRICE | PID |
+-----+-----+-----+-----+-----+
| 101 | Java Core      | 2014-10-13     | 800   | NULL |
| 102 | Advanced Java  | 2024-03-13     | 900   | NULL |
| 103 | Data Structures | 2024-12-24     | 999   | NULL |
| 104 | Ad Data Structure | 2023-09-30     | 800   | NULL |
| 105 | SQL            | 2012-02-05     | 500   | NULL |
| 106 | Data Analysis  | 2024-05-13     | 500   | 501 |
| 107 | ADS            | 2024-07-05     | 600   | 507 |
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.08 sec)

mysql> |
```

In the above query the rollback is happened till the savepoint not to the starting of transaction.

***NOTE:**

Commit must be used after the transaction even if we use savepoints.

Assignment 6: Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Transaction Logs for Data Recovery:

Transaction logs play a crucial role in ensuring data integrity and recoverability in database management systems (DBMS). In this report, we explore the purpose of transaction logs, their characteristics, and scenarios where they are instrumental in data recovery.

Purpose of Transaction Logs

1) **Recording Transactions:** Transaction logs record all modifications made to a database. These include updates, inserts, and deletes performed by transactions.

2) **Point-in-Time Recovery:** Transaction logs allow for point-in-time recovery. In case of system failures or disasters, we can restore the database to a specific point just before the issue occurred, minimizing data loss.

Transaction Log Characteristics

Sequential Records: The transaction log maintains a sequential record of all changes.

Durability: Log entries are durable and survive system crashes.

Circular Buffer: The log operates as a circular buffer, reusing space once it reaches its maximum size.

Hypothetical Scenario:

Unexpected Shutdown Recovery consider the following scenario:

The Incident: A power outage occurs, leading to an unexpected shutdown of the database server.

Database State: At the time of the shutdown, some transactions were incomplete, and not all modifications were written to the data files.

Recovery Process:

Upon server restart, the DBMS runs a recovery process for each database.

The transaction log is used to roll forward all modifications recorded in the log (i.e., apply changes to the data files).

Incomplete transactions are rolled back to maintain database integrity.

Result: The database is brought back to a consistent state, ensuring data consistency and reliability.

Conclusion

Transaction logs are essential for data recovery. They allow to reconstruct databases after system failures, ensuring that no data is lost and maintaining the integrity of the database.

***NOTE:** Never delete or move transaction logs unless fully understanding the ramifications.

