



**DELHI TECHNICAL CAMPUS**  
**Greater Noida**  
Affiliated to GGSIPU and Approved by AICTE & COA



# PRACTICAL FILE

**SESSION:2024-25**  
**DBMS Lab (CIC-256)**  
**II Year, 4th Sem**

**Submitted to:**

**Name:**Ms. Eirrtly telang

**Designation:**Assistant Professor

**Submitted by:**

**Name :**Kumkum

**Enrollm** 01918012723

Department of Computer Science and Engineering  
Delhi Technical Campus, Greater Noida

# INDEX

S.No	Experiment	Date of Conduction	Date of Submission	Signature
1.	Introduction to DBMS, RDBMS and SQL.			
2.	Experiments based on DDL commands – CREATE, ALTER, DROP and TRUNCATE.			
3.	Apply the integrity constraints like Primary Key, Foreign key, Check, NOT NULL, etc. to the tables.			
4.	Experiments based on basic DML commands – SELECT, INSERT, UPDATE and DELETE.			
5.	Write the queries for implementing Built-in functions, GROUP BY, HAVING and ORDER BY.			
6.	Write the queries to implement the joins and subqueries			
7.	Write the queries to implement the set operations and also create a views based on views.			
8.	Demonstrate the concept of Control Structures also demonstrate the concept of Exception Handling.			
9.	Create employee management system using MongoDB.			
10.	Connect employee management system using JDBC.			
11.	Executing Queries using NoSQL(document-based)			

## **EXPERIMENT-1**

**Title :** Introduction to DBMS, RDBMS and SQL.

**Objective:** To understand the basics of **DBMS, RDBMS, and SQL**, their significance, key differences, and fundamental SQL commands for database operations.

**Theory:**

### **1. Database Management System (DBMS)**

#### **1.1 Definition**

A **Database Management System (DBMS)** is a software system designed to manage databases efficiently. It provides users with tools to store, retrieve, manipulate, and manage data in an organized manner. The DBMS serves as an intermediary between users and the database, ensuring that data is structured, secure, and easily accessible.

A DBMS simplifies data handling by allowing users to create, update, and manage databases without requiring deep knowledge of underlying database operations. It enables multiple users to access and work with data concurrently while maintaining data integrity and security.

#### **Examples of DBMS**

- ☐ **Relational DBMS (RDBMS):** MySQL, PostgreSQL, Oracle, SQL Server
- ☐ **NoSQL DBMS:** MongoDB, Cassandra, Redis
- ☐ **Cloud-Based DBMS:** Google Firebase, Amazon RDS

#### **1.2 Features of DBMS**

A **Database Management System (DBMS)** offers a range of features that make data management efficient, secure, and reliable. These features ensure that data is stored, accessed, and manipulated in a structured manner while maintaining consistency and integrity. Below are the key features of a DBMS explained in detail:

##### **1.2.1 Data Abstraction & Independence**

**Data Abstraction** refers to the process of hiding complex storage details from users while providing a simplified interface for data access. DBMS abstracts the physical data storage details and only presents relevant information to users in a structured manner.

- Users interact with data through queries without knowing how it is physically stored.

- The system handles the complexity of storage mechanisms, indexing, and retrieval processes.

**Data Independence** ensures that changes in the database structure do not affect the applications interacting with the database. It is classified into two types:

- **Logical Data Independence:** Changes in the logical schema (e.g., adding new fields) do not affect the application layer.
- **Physical Data Independence:** Changes in storage mechanisms (e.g., modifying file organization) do not impact the database structure at the logical level.

This feature helps in making the database more flexible and adaptable to evolving business requirements.

### 1.2.2 Data Security & Integrity

Data security and integrity are crucial aspects of a DBMS to ensure that only authorized users can access, modify, or delete data while maintaining data consistency and accuracy.

#### Data Security:

- Implements **user authentication** mechanisms such as login credentials and roles.
- Supports **access control** policies that define who can read, write, or modify data.
- Uses **encryption** techniques to protect sensitive data from unauthorized access.

#### Data Integrity:

Ensures data accuracy and consistency through integrity constraints such as:

- **Primary Key Constraint** (to uniquely identify records)
- **Foreign Key Constraint** (to maintain relationships between tables)
- **Check Constraint** (to enforce valid data values)
- **Not Null Constraint** (to ensure mandatory data fields)

By maintaining strict security and integrity measures, a DBMS prevents unauthorized access, accidental modifications, and data corruption.

### 1.2.3 Concurrent Access & Transaction Management

In multi-user environments, a DBMS allows multiple users to access and manipulate data simultaneously without conflicts or inconsistencies. This feature is essential for business applications that require real-time updates and collaboration.

### Concurrency Control:

- Prevents data inconsistency when multiple users attempt to modify the same data simultaneously.
- Uses techniques like **locking**, **timestamp ordering**, and **multiversion concurrency control (MVCC)** to manage simultaneous transactions

### Transaction Management:

Ensures that database operations are executed reliably using **ACID properties**:

- **Atomicity**: Ensures that a transaction is either fully completed or not executed at all.
- **Consistency**: Ensures that the database remains in a valid state before and after a transaction.
- **Isolation**: Ensures that concurrent transactions do not interfere with each other.
- **Durability**: Ensures that committed transactions are permanently stored even after system failures.

## 1.3 Backup & Recovery

A DBMS provides robust backup and recovery mechanisms to protect data against system failures, hardware crashes, power outages, or accidental data loss.

- **Backup Mechanism:**
  - Supports **automated and manual backups** to store database copies at scheduled intervals.
  - Offers **incremental backups**, where only the modified data since the last backup is stored.
  - Cloud-based DBMS solutions provide **off-site backups** for disaster recovery.
- **Recovery Mechanism:**
  - Uses **log-based recovery** to restore data by replaying transactions from a log file.
  - Implements **shadow paging**, where old copies of pages are preserved before modifications.
  - Supports **rollback operations** to revert unintended changes.

## 1.4 Types of DBMS

A **Database Management System (DBMS)** can be classified into different types based on how data is structured, stored, and accessed. Each type of DBMS is designed to handle specific use cases and offers unique advantages. The four primary types of DBMS are:

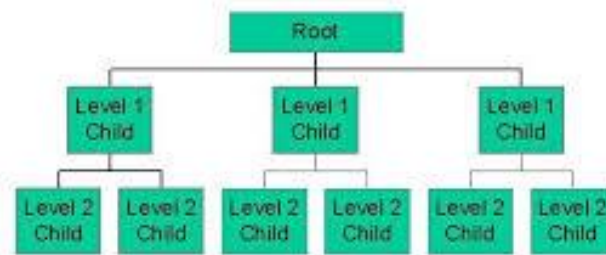
### 1.4.1 Hierarchical DBMS

A **Hierarchical Database Management System** organizes data in a tree-like structure, where records have a **parent-child relationship**. In this model:

- Each parent node can have multiple child nodes, but a child node has only one parent (one-to-many relationship).
- Data is stored in **hierarchical levels**, similar to a file system.

**Example Structure:**

Hierarchical database model

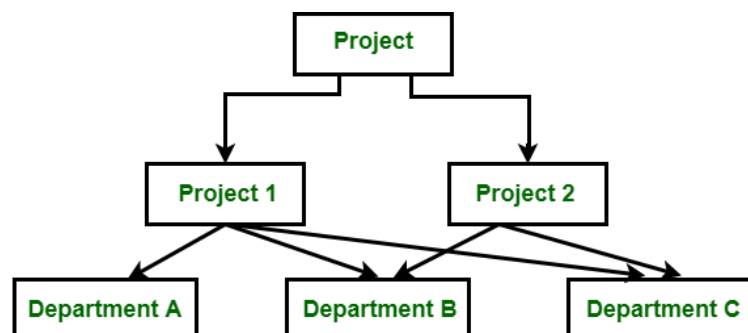


### 1.4.2 Network DBMS

A **Network Database Management System** extends the hierarchical model by allowing **many-to-many relationships** using graph structures. In this model:

- Each record (node) can have multiple parent and child records.
- Relationships are represented using **pointers** (or links), making it more flexible than hierarchical DBMS.

**Example Structure:**





### 1.3.3 Relational DBMS (RDBMS)

A **Relational Database Management System (RDBMS)** is the most widely used DBMS type, where data is stored in **tables (relations)** with rows (records) and columns (fields). Relationships between tables are established using **primary keys** and **foreign keys**.

#### Example Structure: Employee database

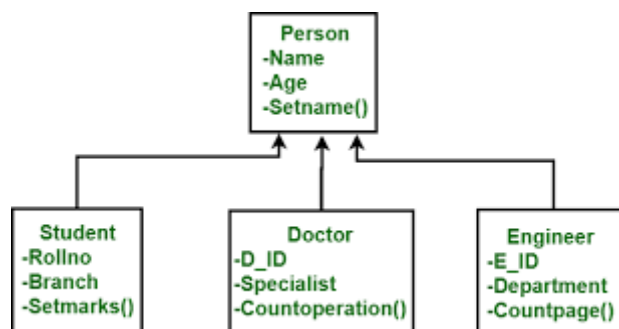
Employee_ID	Name	Department	Manager_ID
101	Alice	IT	201
102	Bob	HR	202
103	Charlie	IT	201

- **Primary Key:** Employee\_ID (uniquely identifies records)
- **Foreign Key:** Manager\_ID (references another table)

### 1.3.4 Object-Oriented DBMS (OODBMS)

An **Object-Oriented Database Management System (OODBMS)** stores data in the form of **objects**, similar to object-oriented programming languages like Java, C++, and Python.

- Objects contain **data (attributes)** and **methods (functions)** that operate on the data.
- Supports **inheritance, encapsulation, and polymorphism**, making it useful for applications that require complex data modeling.



**Example Structure:**

## 1.4 Advantages and Disadvantages of DBMS

Advantages	Description
Reduces Data Redundancy	Eliminates duplicate data storage by using normalization techniques.
Ensures Data Security	Implements authentication, encryption, and access control to prevent unauthorized access.
Supports Multiple User Access	Allows concurrent data access using locking mechanisms and concurrency control.
Facilitates Data Integrity	Maintains data accuracy and consistency using constraints like Primary Keys and Foreign Keys.
Provides Backup and Recovery	Ensures data safety with automated backups, incremental backups, and recovery mechanisms.

Disdvantages	Description
High Setup Cost	Requires expensive software licenses, server infrastructure, and maintenance costs.
Complex System Management	Needs continuous optimization, tuning, and administrative tasks for efficient performance.
Requires Trained Professionals	Database management requires skilled administrators with SQL and security knowledge.
Performance Issues with Large Databases	Large databases slow down query execution due to complex joins and high transaction loads.
Increased Storage Requirements	Needs more storage for metadata, logs, indexes, and backup files compared to file-based systems.

## 2 Relational Database Management System (RDBMS)

### 2.3 Definition

A **Relational Database Management System (RDBMS)** is an advanced form of a **Database Management System (DBMS)** that organizes data into structured tables (relations). Each table consists of **rows (records/tuples)** and **columns (attributes/fields)**, and predefined relationships exist between these tables. RDBMS follows the principles of **relational model theory**, ensuring efficient data storage, retrieval, and management using structured query language (**SQL**).

Unlike traditional **file-based storage systems**, RDBMS eliminates data redundancy, enhances security, and supports multi-user access. It is widely used in applications such as **banking, e-commerce, enterprise systems, and cloud databases** due to its **scalability, consistency, and ease of data management**.



## 2.4 Key Features of RDBMS

### 1. Data is Stored in Tables

Data in an RDBMS is stored in a structured **tabular format**, where:

- **Columns (Attributes)** represent different fields of data (e.g., Name, Age, Email).
- **Rows (Tuples)** contain individual records of information.
- This tabular organization provides **clarity, flexibility, and easy retrieval of data**.
- Example: A **Student table** with columns **ID, Name, Age, Course** and rows representing student records.

### 2. Follows ACID Properties

- RDBMS ensures **data integrity and reliability** by following **ACID** properties:
  - **Atomicity:** A transaction is **all or nothing** (either fully completed or fully rolled back).
  - **Consistency:** Ensures that the database remains in a **valid state** before and after a transaction.
  - **Isolation:** Concurrent transactions are processed **independently** without conflicts.
  - **Durability:** Once a transaction is committed, changes are **permanent** even after system failure.
  - These properties make RDBMS suitable for **critical applications like banking and online transactions**.

### 3. Uses Primary Key and Foreign Key for Maintaining Relationships

- RDBMS establishes **relationships between tables** using:
  - **Primary Key (PK):** A unique identifier for each record in a table.
  - **Foreign Key (FK):** A column in one table that references the **Primary Key** of another table.
- This relationship ensures **data consistency** and eliminates redundancy.
- Example: In a **University Database**, the **Student table** has a **Student\_ID (Primary Key)**, while the **Course table** may have a **Student\_ID as a Foreign Key**, linking both tables.

### 4. Supports Constraints for Data Integrity

- RDBMS enforces rules on data using **constraints**, ensuring accuracy and consistency:
  - **NOT NULL:** Prevents empty values in a column.
  - **UNIQUE:** Ensures all values in a column are distinct.
  - **PRIMARY KEY:** Uniquely identifies each record.
  - **FOREIGN KEY:** Maintains referential integrity between tables.

- **CHECK:** Ensures a column meets a specific condition.
- **DEFAULT:** Assigns a default value if no value is provided.
- Example: In a **Customer table**, the **Email column** can have a **UNIQUE constraint** to prevent duplicate entries.

## 2.5 Important Terms in RDBMS

### 1. Database

- A database is an organized collection of data that is stored and accessed electronically. In RDBMS, this data is stored in the form of tables.

### 2. Table

- A table is a collection of related data in rows and columns. Each table represents a specific entity, and the columns represent attributes of that entity.

### 3. Row (Tuple)

- A row in a table represents a single record or instance of an entity. It contains data that corresponds to each column of the table.

### 4. Column (Attribute)

- A column represents a specific attribute or field of an entity. For example, in a "Students" table, columns might be "StudentID," "Name," "Age," etc.

### 5. Primary Key

- A primary key is a unique identifier for a record in a table. It ensures that each row in the table can be uniquely identified.

### 6. Foreign Key

- A foreign key is a column or set of columns in one table that refers to the primary key in another table. It is used to establish and enforce a link between the data in the two tables.

### 7. Index

- An index is a data structure that improves the speed of data retrieval operations on a table at the cost of additional space and increased maintenance time.

### 10. SQL (Structured Query Language)

- SQL is the language used to interact with relational databases. It includes commands for querying, updating, and managing the database.

### 11. View

- A view is a virtual table created by a query that selects data from one or more tables. It is not a physical table but a result set that behaves like a table.

### 12. Transaction

- A transaction is a sequence of operations that are performed as a single unit. It ensures the database remains in a consistent state.

### 13. Entity-Relationship (ER) Model

- The ER model is a conceptual framework used to describe the data and relationships in a database. It uses entities (tables), attributes (columns), and relationships (foreign keys) to represent the data structure.

**14. Schema** A schema is the structure that defines the organization of data in the database. It includes definitions of tables, views, indexes, and relationships.

### 15. Join

- A join is an operation in SQL that combines rows from two or more tables based on a related column, typically a foreign key. Common types of joins include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN.

### 16. Constraints

- Constraints are rules enforced on the data in a table to ensure integrity. Common types of constraints include NOT NULL, UNIQUE, CHECK, and DEFAULT.

### 17. Trigger

- A trigger is a special type of stored procedure that is automatically executed or fired when certain events (like insert, update, or delete) occur on a table.

### 18. Stored Procedure

- A stored procedure is a set of SQL statements that can be stored and executed on the database server. It allows you to encapsulate logic and reuse it.

### 19. Cascade

- Cascade refers to an operation that automatically propagates changes from one table to related tables. For example, when a record in a parent table is deleted, related records in the child table may also be deleted, depending on the cascade rule.

## 2.6 Differences Between DBMS and RDBMS

Feature	DBMS	RDBMS
<b>Data Storage</b>	Stored in files or simple formats.	Stored in tables (rows and columns).
<b>Data Structure</b>	No fixed structure.	Data stored in a relational structure.
<b>Normalization</b>	Not enforced.	Enforces normalization.
<b>Relationships</b>	No relationships between data.	Supports relationships via keys.
<b>Data Integrity</b>	Not enforced.	Enforced through constraints.
<b>Scalability</b>	Limited.	Highly scalable.
<b>Data Redundancy</b>	Higher redundancy.	Reduces redundancy through normalization.
<b>Examples</b>	File systems, XML databases.	MySQL, Oracle, SQL Server, PostgreSQL.
<b>Security</b>	Basic access control.	Advanced security features.
<b>Query Language</b>	Simple or proprietary.	Uses SQL.
<b>Backup &amp; Recovery</b>	Basic or none.	Robust backup and recovery.

## Structured Query Language (SQL)

### 2.7 Definition

Structured Query Language (SQL) is a standardized programming language used for managing and manipulating relational databases. It allows users to store, retrieve, update, and delete data efficiently. SQL is essential for database management and is widely used in applications ranging from web development to enterprise software.

### 2.8 SQL Process

The SQL process involves several stages that ensure the execution of queries within a database system. The process includes:

1. **Parsing:** The SQL query is checked for syntax errors.
2. **Optimization:** The database engine determines the most efficient way to execute the query.
3. **Execution:** The query is executed, and results are retrieved or modified accordingly.
4. **Result Return:** The processed data is returned to the user or application.

### 2.9 Types of SQL Commands (SQL Languages)

SQL commands are categorized into different types based on their functionality:

1. **Data Definition Language (DDL)** – Defines database structures.
  - CREATE, ALTER, DROP, TRUNCATE
2. **Data Manipulation Language (DML)** – Manipulates data in tables.
  - SELECT, INSERT, UPDATE, DELETE
3. **Data Control Language (DCL)** – Controls access to data.
  - GRANT, REVOKE
4. **Transaction Control Language (TCL)** – Manages transactions.
  - COMMIT, ROLLBACK, SAVEPOINT

### 2.10 SQL Constraints

SQL constraints enforce rules on data to maintain accuracy and integrity. Common constraints include:

- **NOT NULL** – Ensures a column cannot have a NULL value.
- **UNIQUE** – Ensures all values in a column are distinct.
- **PRIMARY KEY** – Uniquely identifies a row in a table.
- **FOREIGN KEY** – Establishes a relationship between tables.
- **CHECK** – Ensures a column meets a specific condition.
- **DEFAULT** – Assigns a default value if no value is provided.

### 2.11 Data Integrity in SQL

Data integrity ensures the accuracy and consistency of data within a database. It is maintained using:

- **Entity Integrity:** Ensures each row has a unique primary key.
- **Referential Integrity:** Maintains valid relationships between tables using foreign keys.
- **Domain Integrity:** Ensures data within a column falls within a valid range or type.

## EXPERIMENT-2

**Title :** Experiments based on DDL commands – CREATE, ALTER, DROP and TRUNCATE.

**Objective:** To practice creating, modifying, removing, and managing database structures while ensuring data integrity and efficient data handling.

### Theory:

#### 1. CREATE Command

The CREATE command is used to create a new table, database, view, or other objects in a database.

Example:

```
mysql> CREATE TABLE Students (  
-> StudentID INT PRIMARY KEY,  
-> Name VARCHAR(100),  
-> Age INT,  
-> Grade CHAR(1) );
```

```
mysql> CREATE TABLE Students (  
-> StudentID INT PRIMARY KEY,  
-> Name VARCHAR(100),  
-> Age INT,  
-> Grade CHAR(1)  
-> );  
Query OK, 0 rows affected (0.04 sec)
```

#### 2. ALTER Command

The ALTER command is used to modify an existing database object, such as a table. You can add, delete, or modify columns in an existing table.

Example:

```
mysql> ALTER TABLE Students  
DROP COLUMN Grade;
```

```
mysql> ALTER TABLE Students  
-> DROP COLUMN Grade;  
Query OK, 0 rows affected (0.08 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

#### 3. DROP Command

The DROP command is used to delete an entire table, database, or other database objects.

Example:

```
mysql> DROP TABLE Students;
```

```
mysql> DROP TABLE Students;  
Query OK, 0 rows affected (0.02 sec)
```

#### 4. TRUNCATE Command

The TRUNCATE command is used to delete all the rows in a table, but it does not remove the table structure. It is faster than DELETE and does not log individual row deletions.

Example:

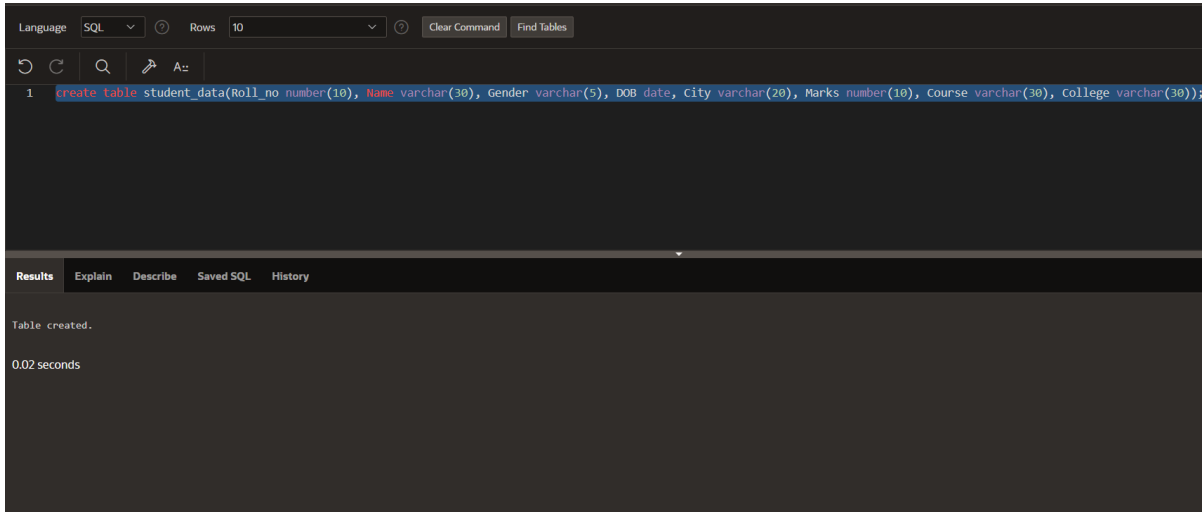
mysql>**TRUNCATE TABLE Products;**

```
mysql> TRUNCATE TABLE products;  
Query OK, 0 rows affected (0.05 sec)
```



## Lab Assignment -1

**Q.1) Create a table student which contain attributes like Roll no., Dob, Gender, Class, Collage, City and Marks.**



```
Language: SQL Rows: 10 Clear Command Find Tables
```

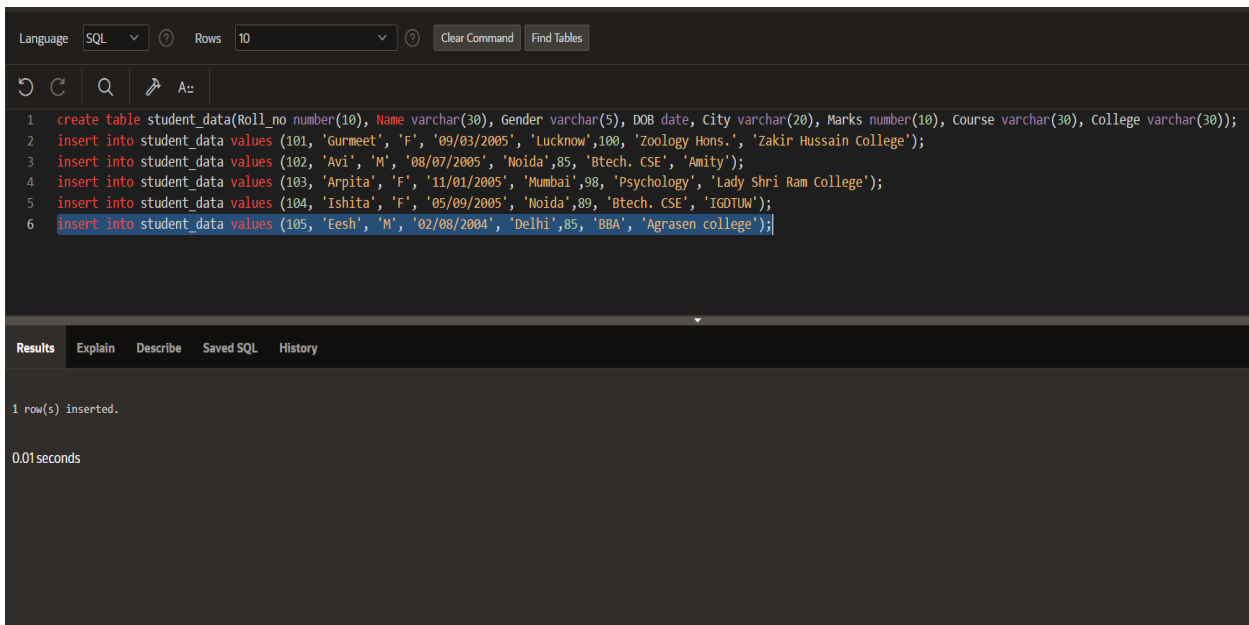
```
1 create table student_data(Roll_no number(10), Name varchar(30), Gender varchar(5), DOB date, City varchar(20), Marks number(10), Course varchar(30), College varchar(30));
```

Results Explain Describe Saved SQL History

Table created.

0.02 seconds

**Q.2) Insert any 5 records.**



```
Language: SQL Rows: 10 Clear Command Find Tables
```

```
1 create table student_data(Roll_no number(10), Name varchar(30), Gender varchar(5), DOB date, City varchar(20), Marks number(10), Course varchar(30), College varchar(30));
2 insert into student_data values (101, 'Gurmeet', 'F', '09/03/2005', 'Lucknow', 100, 'Zoology Hons.', 'Zakir Hussain College');
3 insert into student_data values (102, 'Avi', 'M', '08/07/2005', 'Noida', 85, 'Btech. CSE', 'Amity');
4 insert into student_data values (103, 'Arpita', 'F', '11/01/2005', 'Mumbai', 98, 'Psychology', 'Lady Shri Ram College');
5 insert into student_data values (104, 'Ishita', 'F', '05/09/2005', 'Noida', 89, 'Btech. CSE', 'IGDTUW');
6 insert into student_data values (105, 'Eesh', 'M', '02/08/2004', 'Delhi', 85, 'BBA', 'Agrasen college');
```

Results Explain Describe Saved SQL History

1 row(s) inserted.

0.01 seconds

### Q.3) Display the information of all the students.

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```
7 desc student_data;
8 select * from student_data;
```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	GENDER	DOB	CITY	MARKS	COURSE	COLLEGE
101	Gurmeet	F	9/3/2005	Lucknow	100	Zoology Hons.	Zakir Hussain College
102	Avi	M	8/7/2005	Noida	85	Btech. CSE	Amity
104	Ishita	F	5/9/2005	Noida	89	Btech. CSE	IGDTUW
105	Eesh	M	2/8/2004	Delhi	85	BBA	Agrasen college
103	Arpita	F	11/1/2005	Mumbai	98	Psychology	Lady Shri Ram College

5 rows returned in 0.02 seconds Download

### Q.4) Display detailed structure of student table.

Language: SQL Rows: 10 Clear Command Find Tables

```
7 desc student_data;
```

Results Explain Describe Saved SQL History

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
STUDENT_DATA	ROLL_NO	NUMBER	-	10	0	-	✓	-	-
	NAME	VARCHAR2	30	-	-	-	✓	-	-
	GENDER	VARCHAR2	5	-	-	-	✓	-	-
	DOB	DATE	7	-	-	-	✓	-	-
	CITY	VARCHAR2	20	-	-	-	✓	-	-
	MARKS	NUMBER	-	10	0	-	✓	-	-
	COURSE	VARCHAR2	30	-	-	-	✓	-	-
	COLLEGE	VARCHAR2	30	-	-	-	✓	-	-

### Q.5) Display rollno,name and class of Noida students.

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

7 select * from student_data;
8 select * from student_data;
9 select Roll_no, Name, course from student_data where City= 'Noida';

```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	COURSE
102	Avi	Btech. CSE
104	Ishita	Btech. CSE

2 rows returned in 0.01 seconds Download

**Q.6) Display all the information in ascending of marks.**

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

8 select * from student_data;
9 select Roll_no, Name, Course from student_data where City= 'Noida';
10 select * from student_data order by Marks asc;

```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	GENDER	DOB	CITY	MARKS	COURSE	COLLEGE
105	Eesh	M	2/8/2004	Delhi	85	BBA	Agrasen college
102	Avi	M	8/7/2005	Noida	85	Btech. CSE	Amity
104	Ishita	F	5/9/2005	Noida	89	Btech. CSE	IGDTUW
103	Arpita	F	11/1/2005	Mumbai	98	Psychology	Lady Shri Ram College
101	Gurmeet	F	9/3/2005	Lucknow	100	Zoology Hons.	Zakir Hussain College

5 rows returned in 0.01 seconds Download

**Q.7) Change the marks of roll no 103 to 90.**

SQL Commands

Language: SQL Rows: 10 Clear Command Find Tables

```

9 select Roll_no, Name, course from student_data where City= 'Noida';
10 select * from student_data order by Marks asc;
11 update student_data set Marks=90 where Roll_no = 103;

```

Results Explain Describe Saved SQL History

1 row(s) updated.

0.01 seconds

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

10 select * from student_data order by marks asc;
11 update student_data set Marks=90 where Roll_no = 103;
12 select * from student_data;

```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	GENDER	DOB	CITY	MARKS	COURSE	COLLEGE
101	Gurmeet	F	9/3/2005	Lucknow	100	Zoology Hons.	Zakir Hussain College
102	Avi	M	8/7/2005	Noida	85	Btech. CSE	Amity
104	Ishita	F	5/9/2005	Noida	89	Btech. CSE	IGDTUW
105	Eesh	M	2/8/2004	Delhi	85	BBA	Agrasen college
103	Arpita	F	11/1/2005	Mumbai	90	Psychology	Lady Shri Ram College

5 rows returned in 0.00 seconds Download

**Q.8) Change name and city of roll no 102.**

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

13 update student_data set name='Aavya', City='Gr. Noida' where Roll_no =102;
14 select * from student_data;

```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	GENDER	DOB	CITY	MARKS	COURSE	COLLEGE
101	Gurmeet	F	9/3/2005	Lucknow	100	Zoology Hons.	Zakir Hussain College
102	Aavya	M	8/7/2005	Gr. Noida	85	Btech. CSE	Amity
104	Ishita	F	5/9/2005	Noida	89	Btech. CSE	IGDTUW
105	Eesh	M	2/8/2004	Delhi	85	BBA	Agrasen college
103	Arpita	F	11/1/2005	Mumbai	90	Psychology	Lady Shri Ram College

5 rows returned in 0.00 seconds Download

**Q.9) Delete information of Delhi students.**

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

13 update student_data set name='Aavya', City='Gr. Noida' where Roll_no =102;
14 select * from student_data;
15 delete from student_data where City= 'delhi';

```

Results Explain Describe Saved SQL History

1 row(s) deleted.  
0.01 seconds

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

14 select * from student_data;
15 delete from student_data where City= 'delhi';
16 select * from student_data;

```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	GENDER	DOB	CITY	MARKS	COURSE	COLLEGE
101	Gurmeet	F	9/3/2005	Lucknow	100	Zoology Hons.	Zakir Hussain College
102	Aavya	M	8/7/2005	Gr. Noida	85	Btech. CSE	Amity
104	Ishita	F	5/9/2005	Noida	89	Btech. CSE	IGDTUW
103	Arpita	F	11/1/2005	Mumbai	90	Psychology	Lady Shri Ram College

4 rows returned in 0.00 seconds Download

**Q.10) Delete all records of student where marks>91.**

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

16 select * from student_data;
17 delete from student_data where Marks>91;
18

```

Results Explain Describe Saved SQL History

1 row(s) deleted.

0.01 seconds

Language: SQL Rows: 10 Clear Command Find Tables Save Run

```

16 select * from student_data;
17 delete from student_data where Marks>91;
18 select * from student_data;
19

```

Results Explain Describe Saved SQL History

ROLL_NO	NAME	GENDER	DOB	CITY	MARKS	COURSE	COLLEGE
102	Aavya	M	8/7/2005	Gr. Noida	85	Btech. CSE	Amity
104	Ishita	F	5/9/2005	Noida	89	Btech. CSE	IGDTUW
103	Arpita	F	11/1/2005	Mumbai	90	Psychology	Lady Shri Ram College

3 rows returned in 0.00 seconds Download

## **EXPERIMENT-3**

**Title :** Apply the integrity constraints like Primary Key, Foreign key, Check, NOT NULL, etc. to the tables.

**Objective:** To ensure data accuracy and consistency in the database, apply integrity constraints such as Primary Key for unique identification, Foreign Key for referential integrity, NOT NULL to enforce required fields, and Check constraints for valid data entry.

### **Theory:**

#### **Introduction to Integrity Constraints**

Integrity constraints are rules enforced on database tables to maintain accuracy, consistency, and reliability of data. They ensure that only valid and meaningful data is stored in the database.

#### **Types of Integrity Constraints:**

1. Primary Key Constraint
2. Foreign Key Constraint
3. NOT NULL Constraint
4. Unique Constraint
5. Check Constraint
6. Default Constraint

#### **1.Primary Key Constraint**

A **Primary Key (PK)** is a column or a combination of columns that uniquely identifies each row in a table. It ensures that:

- The values in the primary key column(s) are **unique**.
- The column cannot contain **NULL** values.

**Example:** In a table called Students, the student\_id column can be the primary key, ensuring each student has a unique identifier.

```
CREATE TABLE Students (  
student_id INT PRIMARY KEY,  
name VARCHAR(100),  
age INT  
);
```

#### **2.Foreign Key Constraint**

A **Foreign Key (FK)** is a column in one table that refers to the **Primary Key** of another table. It establishes a relationship between two tables and enforces **referential integrity**.



**Example:** - Dept\_ID in the Employee table is a Foreign Key referencing Dept\_ID in the Department table.

- This ensures that an employee cannot be assigned to a non-existent department.

```
CREATE TABLE Department (  
Dept_ID INT PRIMARY KEY,  
Dept_Name VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE Employee (  
Emp_ID INT PRIMARY KEY,  
Name VARCHAR(50) NOT NULL,  
Dept_ID INT,  
FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)  
);
```

### 3. NOT NULL Constraint

The **NOT NULL** constraint ensures that a column cannot store NULL values. It is used when a field must have a value.

**Example:-** Name and Age must have values; inserting a NULL will result in an error.

```
CREATE TABLE Student (  
Student_ID INT PRIMARY KEY,  
Name VARCHAR(50) NOT NULL,  
Age INT NOT NULL  
);
```

### 4. Unique Constraint

The **Unique Constraint** ensures that all values in a column are distinct. Unlike a Primary Key, it allows one NULL value in the column.

**Example: -** No two users can have the same Email or Username, ensuring uniqueness.

```
CREATE TABLE Users (  
User_ID INT PRIMARY KEY,  
Email VARCHAR(100) UNIQUE,  
Username VARCHAR(50) UNIQUE  
);
```

## 5. Check Constraint

The **Check Constraint** enforces a condition on a column's values. If the condition is not met, the insertion or update operation fails.

**Example:-** Price must be greater than 0.

- Quantity must be at least 1.

```
CREATE TABLE Products (  
Product_ID INT PRIMARY KEY,  
Price DECIMAL(10,2) CHECK (Price > 0),  
Quantity INT CHECK (Quantity >= 1)  
);
```

## 6. Default Constraint

The **Default Constraint** provides a default value for a column when no value is provided.

**Example: -** If no Order\_Date is provided, the system assigns the current date.

- If no Status is provided, it defaults to 'Pending'.

```
CREATE TABLE Orders (  
Order_ID INT PRIMARY KEY,  
Order_Date DATE DEFAULT CURRENT_DATE,  
Status VARCHAR(20) DEFAULT 'Pending'  
);
```

## Importance of Integrity Constraints

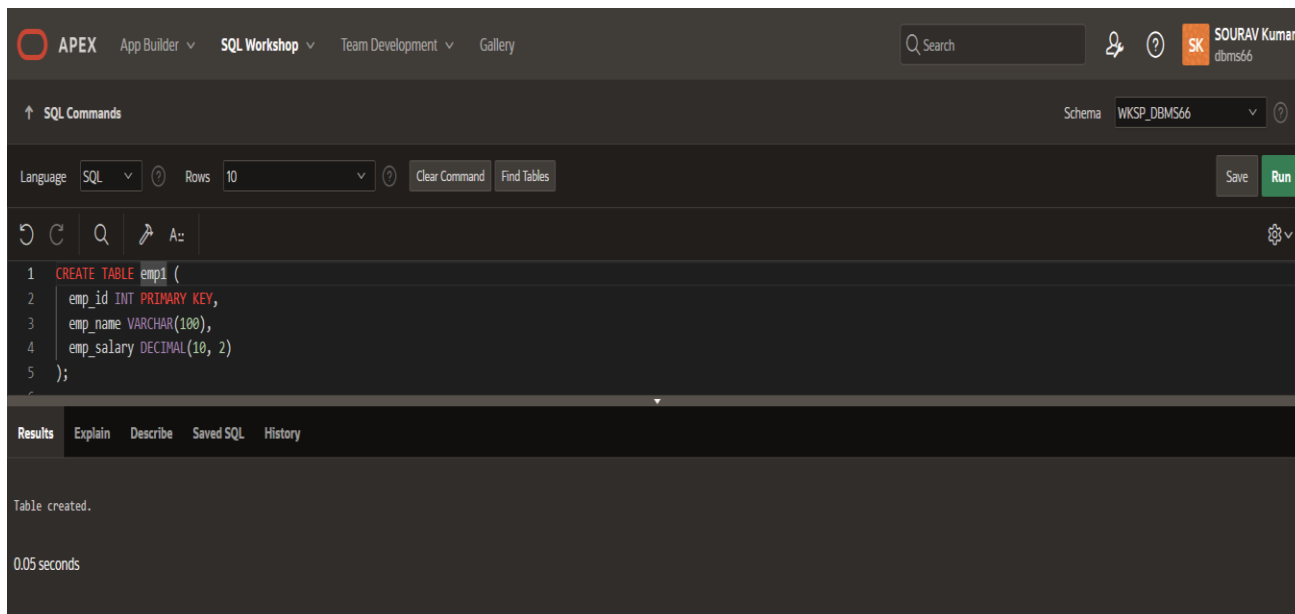
- Prevents inconsistent data entry.
- Ensures data accuracy and validity.
- Maintains relationships between tables.
- Enforces business rules within the database.

## Conclusion

Integrity constraints play a crucial role in database design. They help maintain data consistency and integrity, prevent invalid data entry, and ensure that data follows predefined rules.

## Lab Assignment -2

**Q1: Create table emp having primary key as, emp(emp\_id primary key, emp\_name, emp\_salary)**

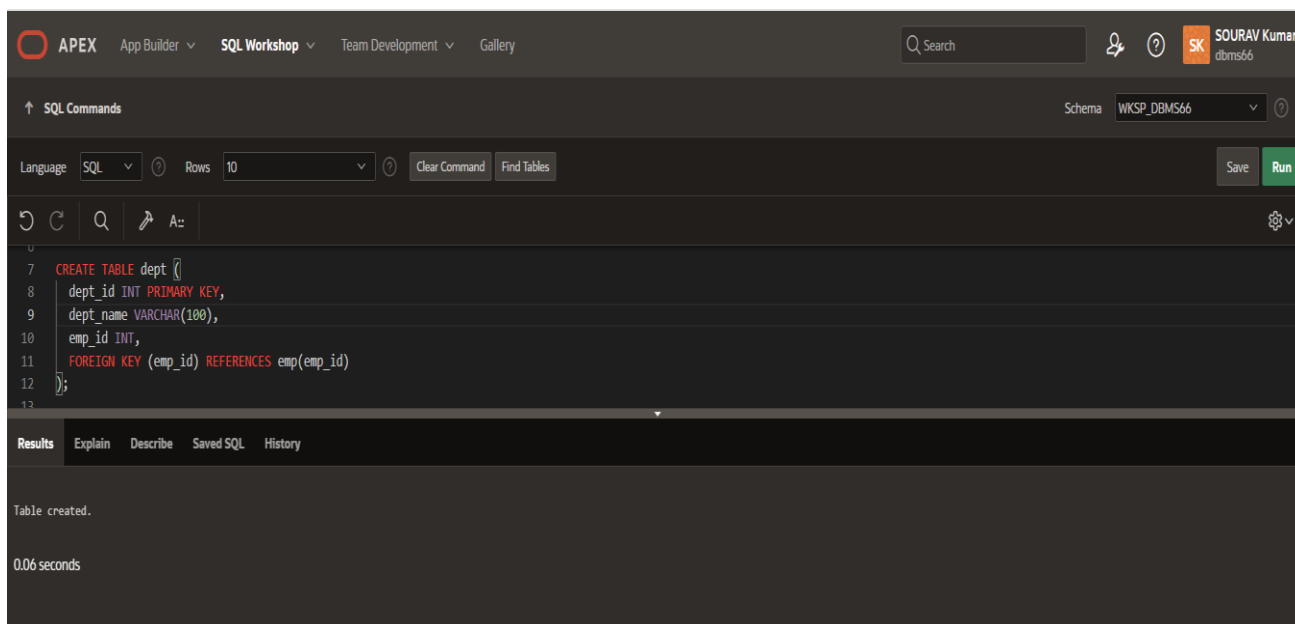


The screenshot shows the APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'SOURAV Kumar dbms66' are on the right. The 'SQL Commands' section is active, showing a schema dropdown set to 'WKSP\_DBMS66'. Below the command area, there are buttons for 'Language' (SQL), 'Rows' (10), 'Clear Command', and 'Find Tables'. The SQL command entered is: 

```
1 CREATE TABLE emp1 (  
2   emp_id INT PRIMARY KEY,  
3   emp_name VARCHAR(100),  
4   emp_salary DECIMAL(10, 2)  
5 );
```

 The 'Results' tab is selected, displaying 'Table created.' and '0.05 seconds'.

**Q2: Create table dept with foreign key (dept\_id primary key, dept\_name, emp\_id foreign key(emp\_id) references emp(emp\_id)).**

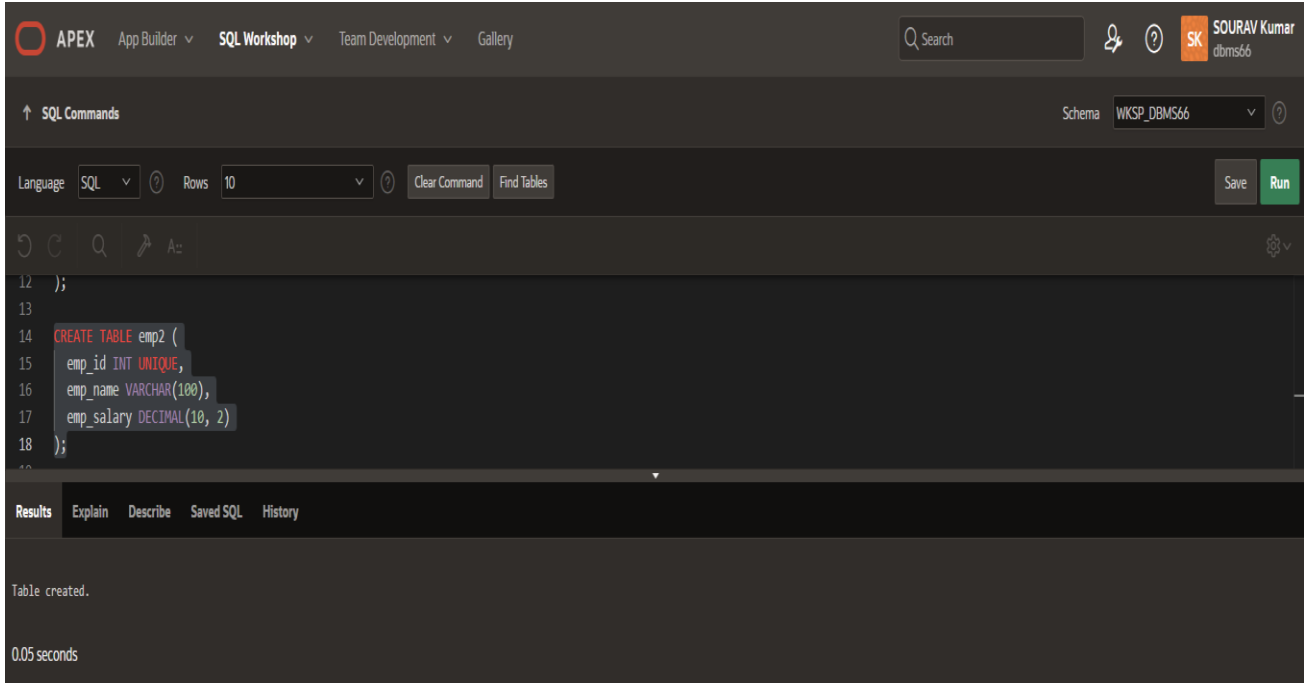


The screenshot shows the APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop', 'Team Development', and 'Gallery'. A search bar and user profile 'SOURAV Kumar dbms66' are on the right. The 'SQL Commands' section is active, showing a schema dropdown set to 'WKSP\_DBMS66'. Below the command area, there are buttons for 'Language' (SQL), 'Rows' (10), 'Clear Command', and 'Find Tables'. The SQL command entered is: 

```
7 CREATE TABLE dept (  
8   dept_id INT PRIMARY KEY,  
9   dept_name VARCHAR(100),  
10  emp_id INT,  
11  FOREIGN KEY (emp_id) REFERENCES emp(emp_id)  
12 );
```

 The 'Results' tab is selected, displaying 'Table created.' and '0.06 seconds'.

### Q.3 Create table emp having one unique column as(emp\_id unique,emp\_name)

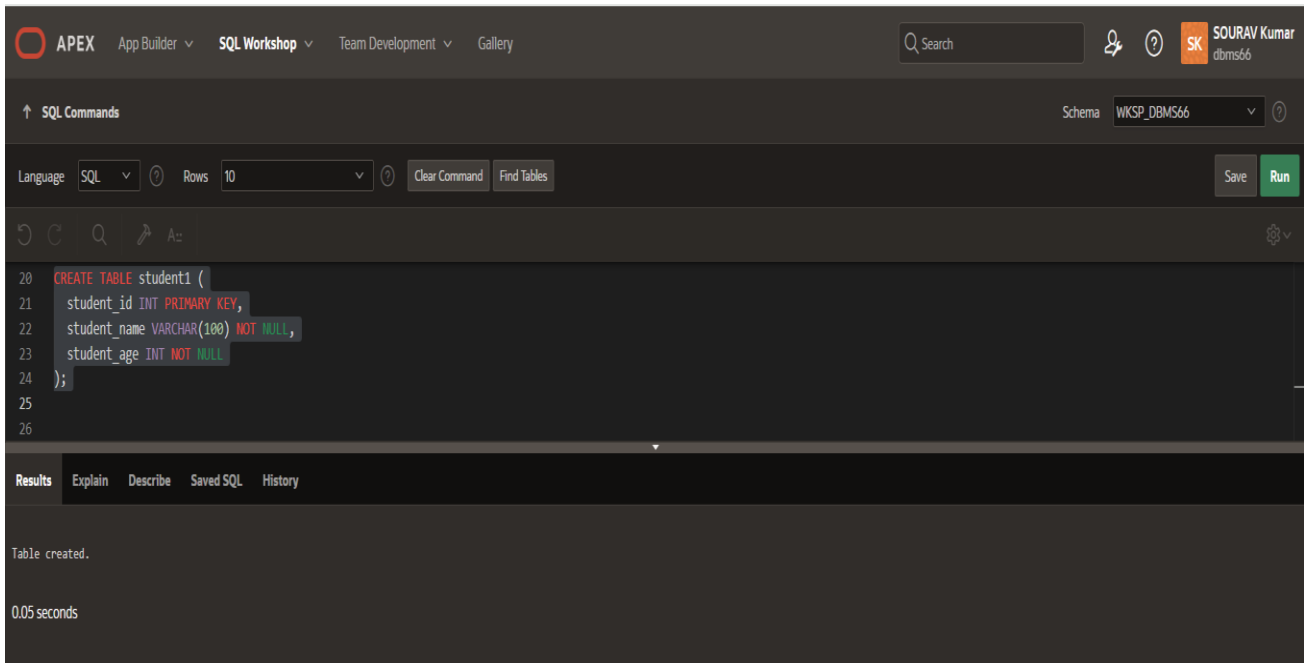


The screenshot shows the APEX SQL Workshop interface. The 'SQL Commands' tab is active, displaying the following SQL code:

```
12 );  
13  
14 CREATE TABLE emp2 (  
15     emp_id INT UNIQUE,  
16     emp_name VARCHAR(100),  
17     emp_salary DECIMAL(10, 2)  
18 );  
19
```

The 'Results' tab is selected, showing the message 'Table created.' and the execution time '0.05 seconds'.

### Q.4 Create table student with not null constraint

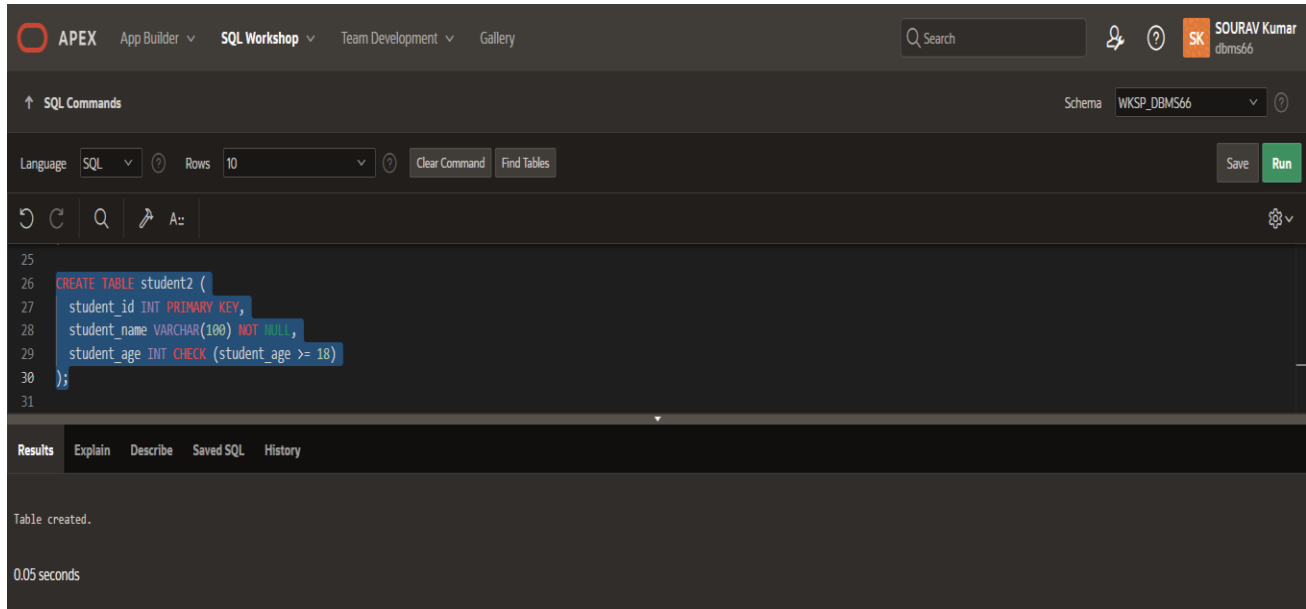


The screenshot shows the APEX SQL Workshop interface. The 'SQL Commands' tab is active, displaying the following SQL code:

```
20 CREATE TABLE student1 (  
21     student_id INT PRIMARY KEY,  
22     student_name VARCHAR(100) NOT NULL,  
23     student_age INT NOT NULL  
24 );  
25  
26
```

The 'Results' tab is selected, showing the message 'Table created.' and the execution time '0.05 seconds'.

### Q5: Apply check constraint in student table

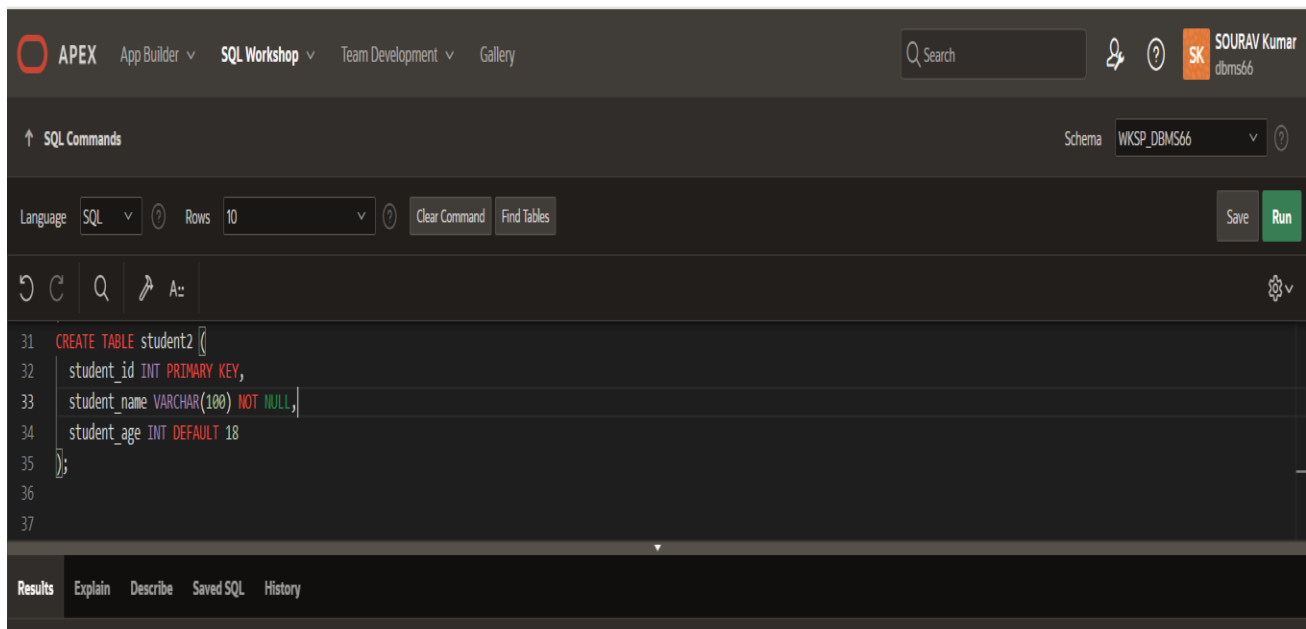


The screenshot shows the APEX SQL Workshop interface. The 'SQL Commands' tab is active, and the schema is set to 'WKSP\_DBMS66'. The SQL command entered is:

```
25  
26 CREATE TABLE student2 (  
27     student_id INT PRIMARY KEY,  
28     student_name VARCHAR(100) NOT NULL,  
29     student_age INT CHECK (student_age >= 18)  
30 );  
31
```

The 'Results' tab shows the message 'Table created.' and the execution time '0.05 seconds'.

### Q6: Apply default constraint in student table



The screenshot shows the APEX SQL Workshop interface. The 'SQL Commands' tab is active, and the schema is set to 'WKSP\_DBMS66'. The SQL command entered is:

```
31 CREATE TABLE student2 (  
32     student_id INT PRIMARY KEY,  
33     student_name VARCHAR(100) NOT NULL,  
34     student_age INT DEFAULT 18  
35 );  
36  
37
```

The 'Results' tab shows the message 'Table created.' and the execution time '0.05 seconds'.

## **Experiment- 4**

**Title:** Experiments based on basic DML commands – SELECT, INSERT, UPDATE and DELETE

**Objective:** To perform and understand basic Data Manipulation Language (DML) operations, including **SELECT**, **INSERT**, **UPDATE**, and **DELETE**, for managing and manipulating data in a relational database.

**Theory:** Data Manipulation Language (DML) is a subset of SQL (Structured Query Language) used to manipulate data stored in a relational database. DML commands enable users to perform various operations like retrieving, adding, updating, and deleting data from tables. The primary DML commands are **SELECT**, **INSERT**, **UPDATE**, and **DELETE**. These commands are essential for efficiently managing and manipulating database records.

### **1. SELECT Command**

The **SELECT** command is used to retrieve data from one or more tables. It allows users to specify columns and apply conditions to filter the results.

#### **SYNTAX:**

SELECT column1, column2, ... FROM table\_name WHERE condition;

- **column1, column2, ...:** The columns to retrieve from the table.
- **table\_name:** The name of the table from which data is fetched.
- **WHERE condition:** An optional clause to filter the results.

**Example 1:** Retrieve specific columns

```
SELECT name, age FROM students;
```

This query retrieves the `name` and `age` columns from the **students** table.

**Example 2:** Retrieve all columns

```
SELECT * FROM students WHERE grade = 'A';
```

This query retrieves all the columns from the `students` table where the grade is 'A'.



## 2. INSERT Command

The **INSERT** command is used to add new records to a table. It can insert data into specific columns or all columns at once.

### Syntax:

```
INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...);
```

- **table\_name:** The table where data is inserted.
- **column1, column2, ...:** The columns to insert data into.
- **value1, value2, ...:** The corresponding values to be inserted.

### Example 1: Inserting data into specific columns

```
INSERT INTO students (name, age, grade) VALUES ('John', 20, 'A');
```

This query adds a new student record with the name "John", age 20, and grade "A".

### Example 2: Inserting data into all columns

```
INSERT INTO students VALUES (102, 'Alice', 19, 'B');
```

This query inserts a new record with all values specified in the order of the table's columns.

## 3. UPDATE Command

The **UPDATE** command is used to modify existing records in a table. It allows users to change the values of one or more columns based on a specified condition.

### Syntax:

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
```

- **table\_name:** The table where the data is to be updated.
- **column1, column2, ...:** The columns whose values are to be updated.
- **value1, value2, ...:** The new values for the specified columns.
- **WHERE condition:** A condition to specify which records to update.

### Example 1: Updating a single field

```
UPDATE students SET grade = 'A+' WHERE name = 'John';
```

This query updates the grade of the student named "John" to "A+".

### Example 2: Updating multiple fields

```
UPDATE students SET age = 21, grade = 'B+' WHERE name = 'Alice';
```

This query updates both the age and grade of the student named "Alice".

## 4. DELETE Command

The **DELETE** command is used to remove records from a table. It is essential to use the **WHERE** clause to prevent accidentally deleting all records.

### Syntax:

```
DELETE FROM table_name WHERE condition;
```

- table\_name: The name of the table from which records are to be deleted.
- WHERE condition: Specifies which records to delete.

### Example 1: Deleting specific records

```
DELETE FROM students WHERE grade = 'D';
```

This query deletes all student records where the grade is "D".

### Example 2: Deleting all records

```
DELETE FROM students;
```

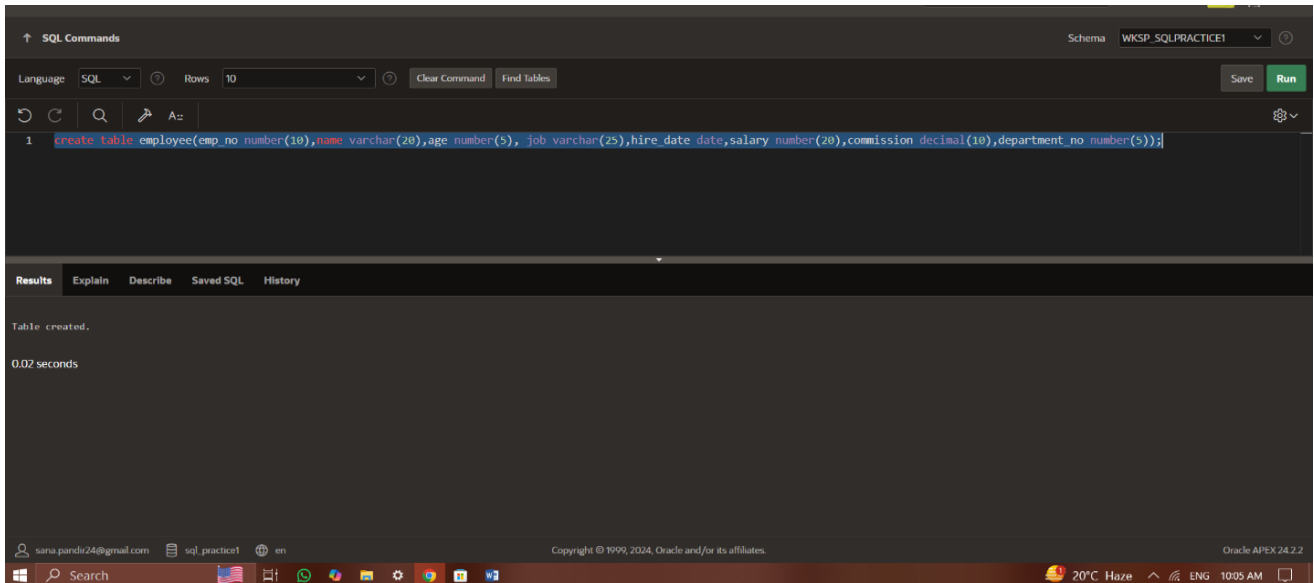
This query removes all records from the **students** table, leaving it empty.

### Conclusion:

In this experiment, we have explored the fundamental DML commands: **SELECT**, **INSERT**, **UPDATE**, and **DELETE**. These commands are essential for performing CRUD (Create, Read, Update, Delete) operations on a database. The **SELECT** command helps retrieve data, while **INSERT** is used for adding new records. **UPDATE** modifies existing records, and **DELETE** removes unwanted data. Mastering these commands ensures effective data management and manipulation within relational databases.

## Lab Assignment -3

Q1. Create a table employee(emp no., emp name, job, hire\_date, salary, commission, department no., age).

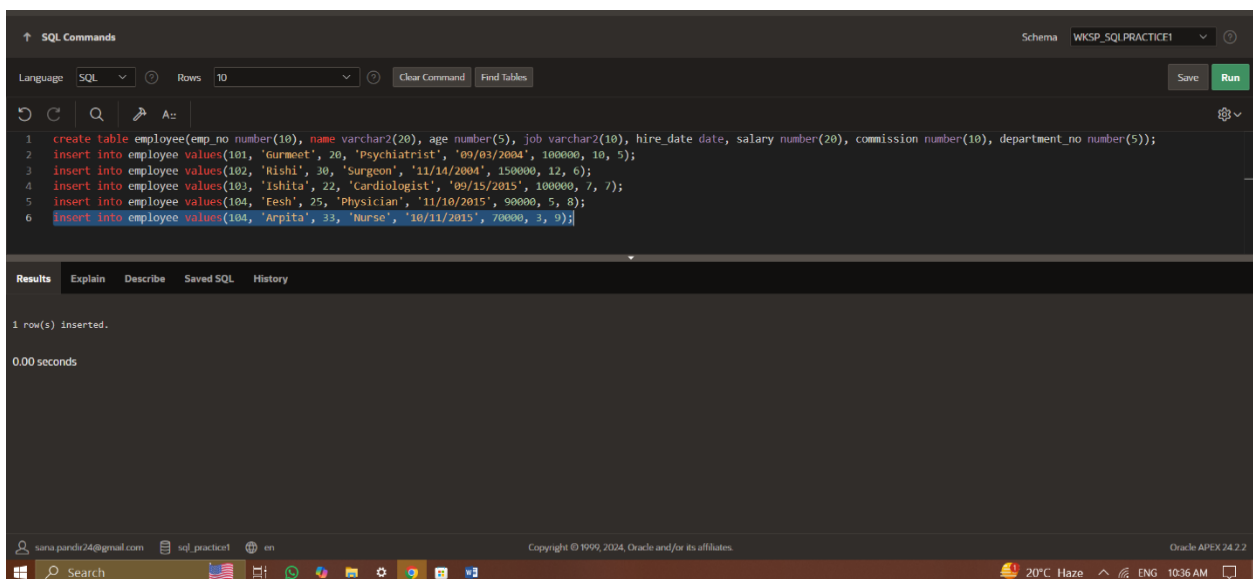


The screenshot shows the SQL Developer interface. The 'SQL Commands' tab is active, displaying the following SQL command:

```
1 create table employee(emp_no number(10), name varchar(20), age number(5), job varchar(25), hire_date date, salary number(20), commission decimal(10), department_no number(5));
```

The 'Results' tab shows the message 'Table created.' and '0.02 seconds'.

Q.2 Insert any 5 records

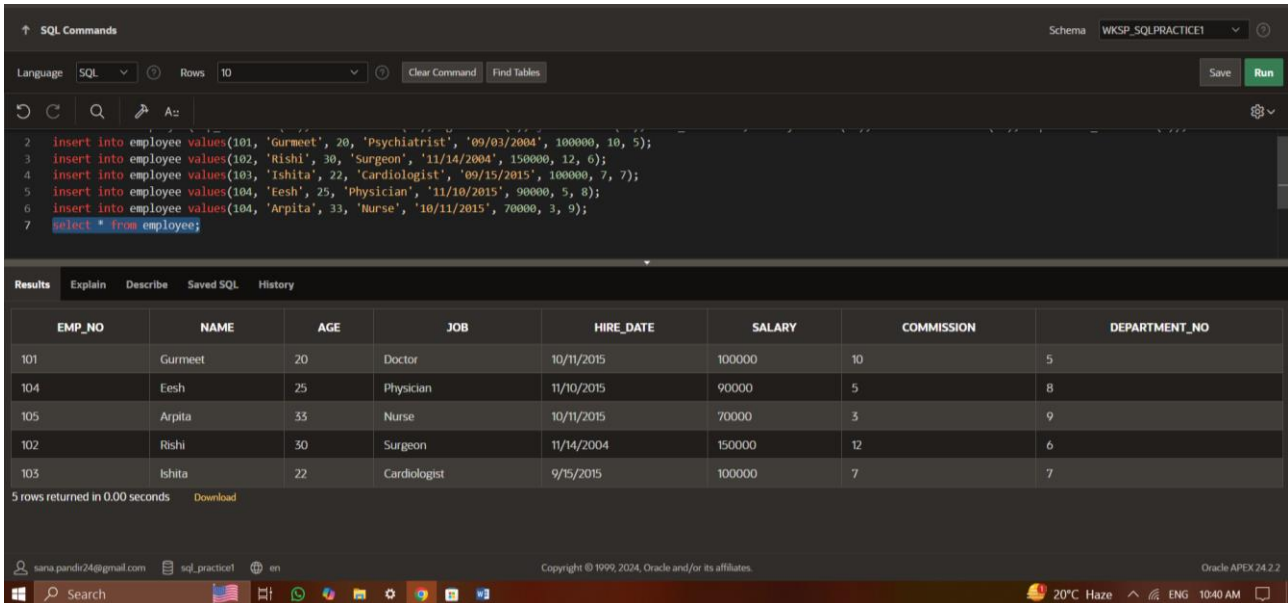


The screenshot shows the SQL Developer interface. The 'SQL Commands' tab is active, displaying the following SQL commands:

```
1 create table employee(emp_no number(10), name varchar2(20), age number(5), job varchar2(10), hire_date date, salary number(20), commission number(10), department_no number(5));
2 insert into employee values(101, 'Gurmeet', 20, 'Psychiatrist', '09/03/2004', 100000, 10, 5);
3 insert into employee values(102, 'Rishi', 30, 'Surgeon', '11/14/2004', 150000, 12, 6);
4 insert into employee values(103, 'Ishita', 22, 'Cardiologist', '09/15/2015', 100000, 7, 7);
5 insert into employee values(104, 'Fash', 25, 'Physician', '11/10/2015', 90000, 5, 8);
6 insert into employee values(104, 'Arpita', 33, 'Nurse', '10/11/2015', 70000, 3, 9);
```

The 'Results' tab shows the message '1 row(s) inserted.' and '0.00 seconds'.

Q.3 Display the info of all employee.



SQL Commands

```

2 insert into employee values(101, 'Gurmeet', 20, 'Psychiatrist', '09/03/2004', 100000, 10, 5);
3 insert into employee values(102, 'Rishi', 30, 'Surgeon', '11/14/2004', 150000, 12, 6);
4 insert into employee values(103, 'Ishita', 22, 'Cardiologist', '09/15/2015', 100000, 7, 7);
5 insert into employee values(104, 'Eesh', 25, 'Physician', '11/10/2015', 90000, 5, 8);
6 insert into employee values(104, 'Arpita', 33, 'Nurse', '10/11/2015', 70000, 3, 9);
7 select * from employee;

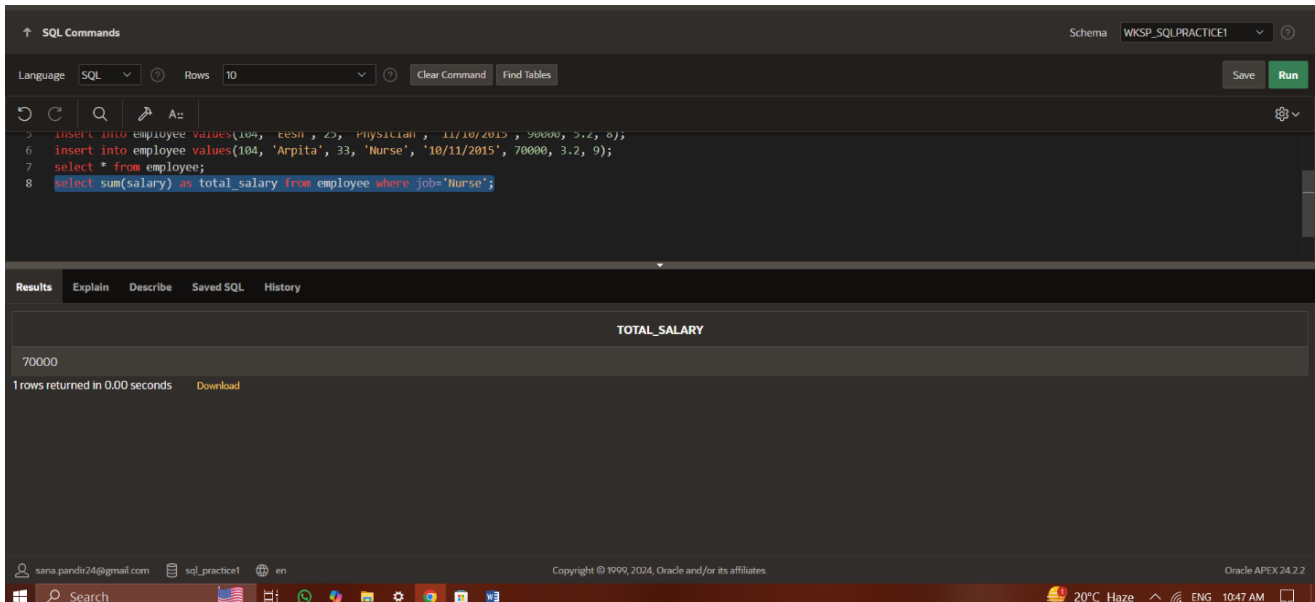
```

Results

EMP_NO	NAME	AGE	JOB	HIRE_DATE	SALARY	COMMISSION	DEPARTMENT_NO
101	Gurmeet	20	Doctor	10/11/2015	100000	10	5
104	Eesh	25	Physician	11/10/2015	90000	5	8
105	Arpita	33	Nurse	10/11/2015	70000	3	9
102	Rishi	30	Surgeon	11/14/2004	150000	12	6
103	Ishita	22	Cardiologist	9/15/2015	100000	7	7

5 rows returned in 0.00 seconds

Q.4 Display total salary paid to nurse.



SQL Commands

```

5 insert into employee values(104, 'Eesh', 25, 'Physician', '11/10/2015', 90000, 5.2, 8);
6 insert into employee values(104, 'Arpita', 33, 'Nurse', '10/11/2015', 70000, 3.2, 9);
7 select * from employee;
8 select sum(salary) as total_salary from employee where job='Nurse';

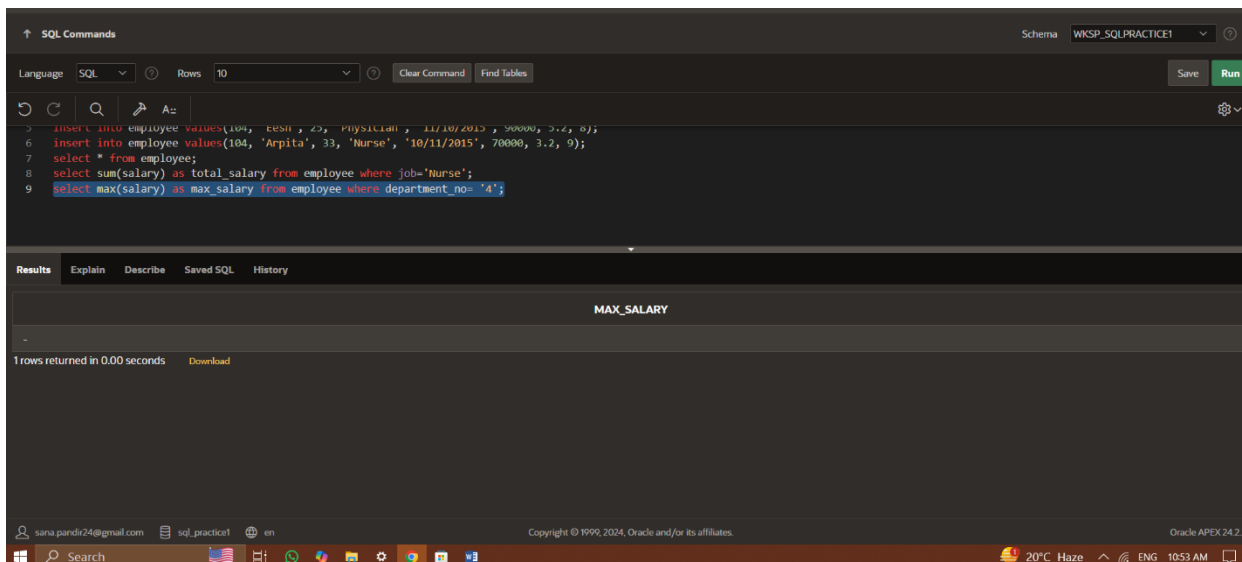
```

Results

TOTAL_SALARY
70000

1 rows returned in 0.00 seconds

Q5. Display max salary of employee who belongs to department no.=4.



SQL Commands

```

1 insert into employee values(100, 'Eesh', '23', 'Physician', '11/10/2015', 90000, 3.2, 0);
2 insert into employee values(104, 'Arpita', '33', 'Nurse', '10/11/2015', 70000, 3.2, 0);
3 select * from employee;
4 select sum(salary) as total_salary from employee where job='Nurse';
5 select max(salary) as max_salary from employee where department_no= '4';

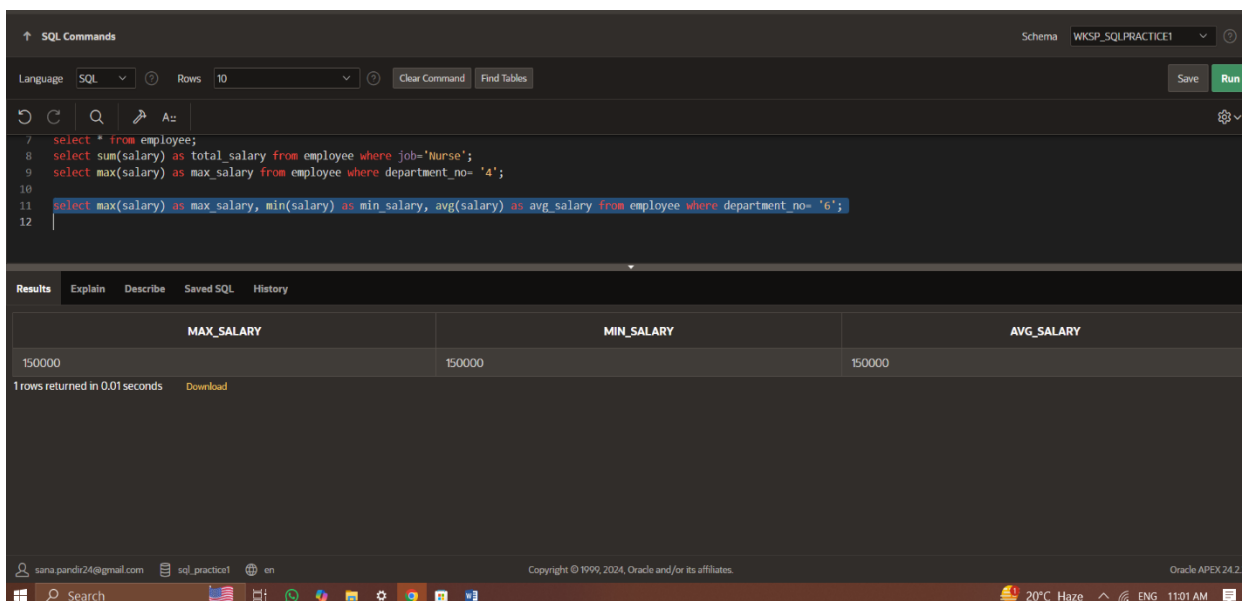
```

Results

MAX_SALARY
70000

1 rows returned in 0.00 seconds

Q6.Display max, min, average salary of department no.=6.



SQL Commands

```

7 select * from employee;
8 select sum(salary) as total_salary from employee where job='Nurse';
9 select max(salary) as max_salary from employee where department_no= '4';
10
11 select max(salary) as max_salary, min(salary) as min_salary, avg(salary) as avg_salary from employee where department_no= '6';
12

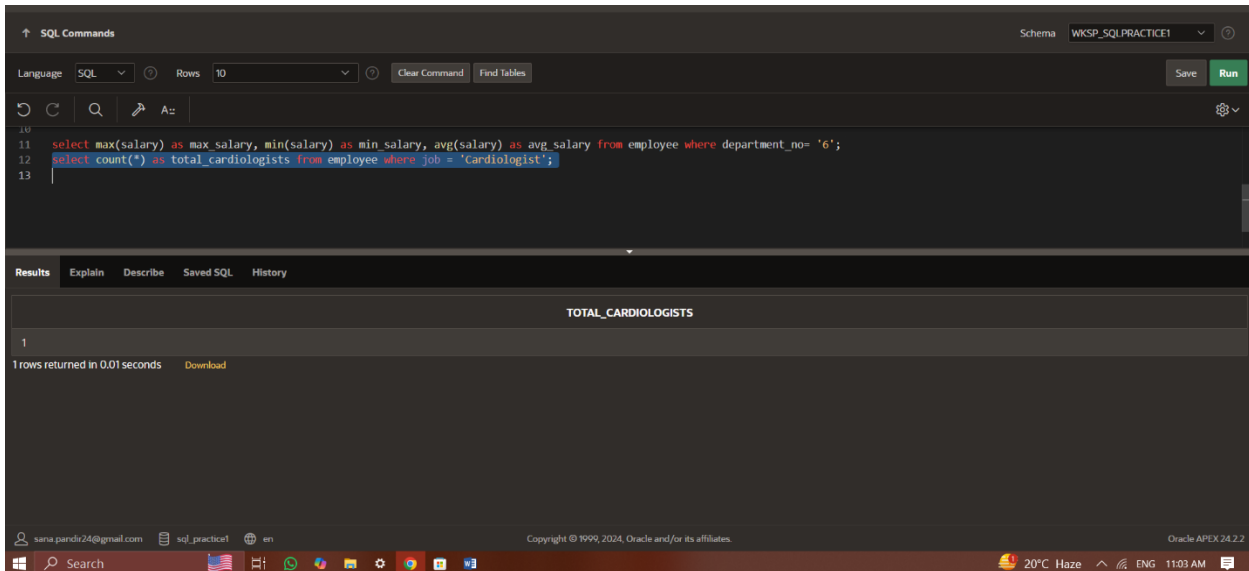
```

Results

MAX_SALARY	MIN_SALARY	AVG_SALARY
150000	150000	150000

1 rows returned in 0.01 seconds

Q7. Count no. of employee where job is cardiologist.



The screenshot shows the SQL Developer interface. The 'SQL Commands' tab is active, displaying the following SQL query:

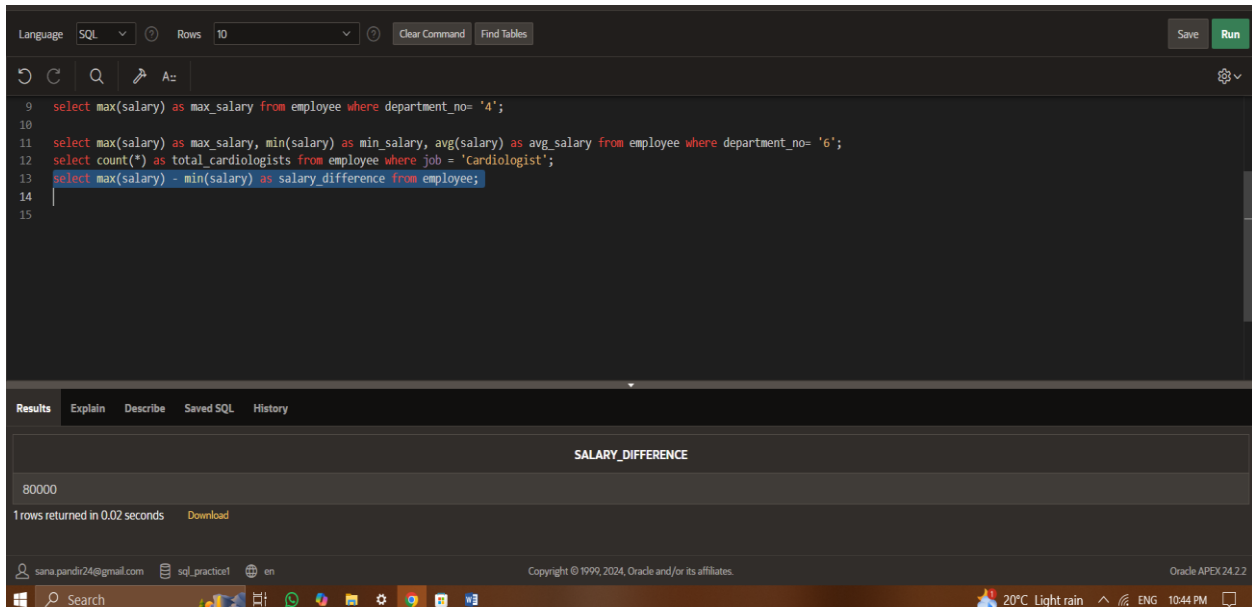
```
10 select max(salary) as max_salary, min(salary) as min_salary, avg(salary) as avg_salary from employee where department_no= '6';
11 select count(*) as total_cardiologists from employee where job = 'Cardiologist';
12
13
```

The 'Results' tab is active, showing a table with one row and one column:

TOTAL_CARDIOLOGISTS
1

1 rows returned in 0.01 seconds

Q.8 What is the difference between maximum and minimum salary of employee in organisation?



The screenshot shows the SQL Developer interface. The 'SQL Commands' tab is active, displaying the following SQL query:

```
9 select max(salary) as max_salary from employee where department_no= '4';
10
11 select max(salary) as max_salary, min(salary) as min_salary, avg(salary) as avg_salary from employee where department_no= '6';
12 select count(*) as total_cardiologists from employee where job = 'Cardiologist';
13 select max(salary) - min(salary) as salary_difference from employee;
14
15
```

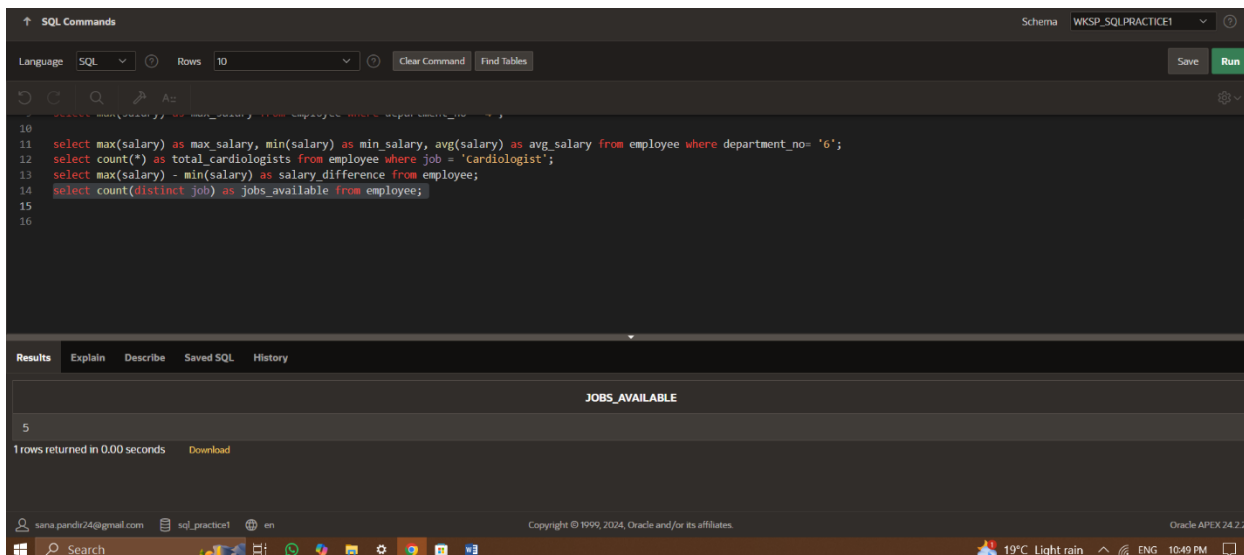
The 'Results' tab is active, showing a table with one row and one column:

SALARY_DIFFERENCE
80000

1 rows returned in 0.02 seconds



Q.9 How many jobs are available in organisation?



SQL Commands

```

10
11 select max(salary) as max_salary, min(salary) as min_salary, avg(salary) as avg_salary from employee where department_no= '6';
12 select count(*) as total_cardiologists from employee where job = 'Cardiologist';
13 select max(salary) - min(salary) as salary_difference from employee;
14 select count(distinct job) as jobs_available from employee;
15
16

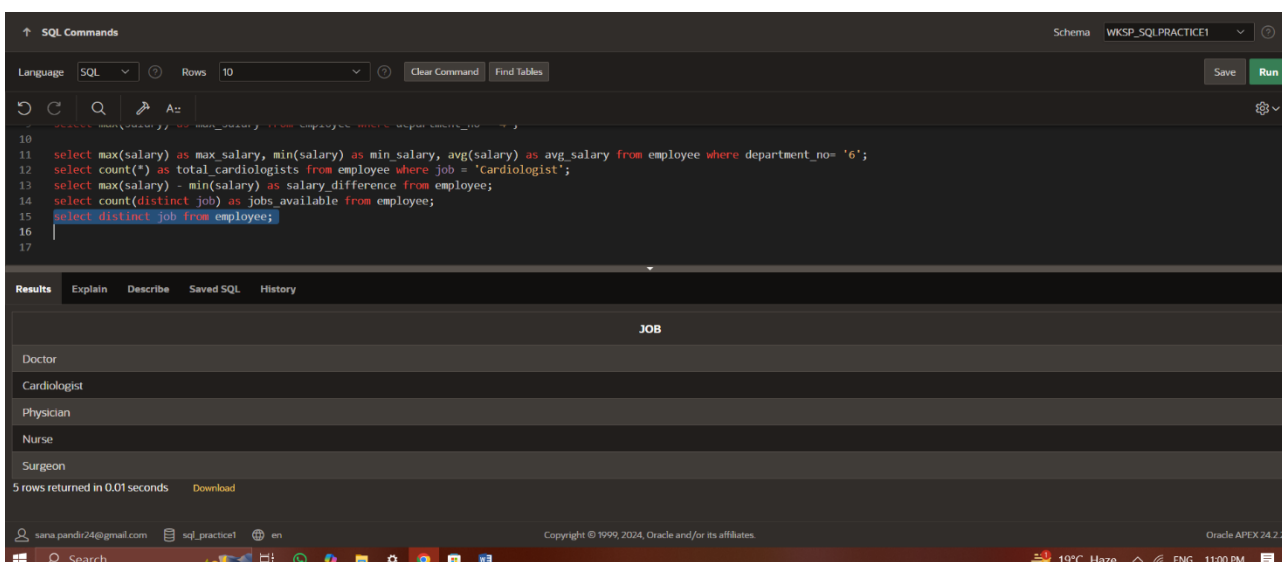
```

Results

JOB
5

1 rows returned in 0.00 seconds

Q.10 How many job titles are available?



SQL Commands

```

10
11 select max(salary) as max_salary, min(salary) as min_salary, avg(salary) as avg_salary from employee where department_no= '6';
12 select count(*) as total_cardiologists from employee where job = 'Cardiologist';
13 select max(salary) - min(salary) as salary_difference from employee;
14 select count(distinct job) as jobs available from employee;
15 select distinct job from employee;
16
17

```

Results

JOB
Doctor
Cardiologist
Physician
Nurse
Surgeon

5 rows returned in 0.01 seconds

## **Experiment – 5**

**Title:** Write the queries for implementing Built-in functions, GROUP BY, HAVING and ORDER BY.

**Objective:** To understand and implement SQL built-in functions for performing calculations and data manipulations.

To learn the use of GROUP BY, HAVING, and ORDER BY clauses to organize and filter query results.

**Theory:** SQL provides a range of built-in functions to perform operations on data, such as mathematical calculations, string manipulations, and data formatting. The most commonly used built-in functions are:

### **1. Aggregate Functions:**

- SUM(), AVG(), MIN(), MAX(), COUNT()
- These functions perform calculations on a set of values and return a single value.

### **2. String Functions:**

- UPPER(), LOWER(), LENGTH(), CONCAT()
- These functions manipulate and return string values.

### **3. Date Functions:**

- NOW(), DATE(), YEAR(), MONTH()
- These functions deal with date and time values.

### **GROUP BY Clause:**

The GROUP BY clause groups rows that have the same values into summary rows. It is often used with aggregate functions to group the result set by one or more columns.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name;
```

### **HAVING Clause:**

The HAVING clause filters the results produced by GROUP BY. It is similar to WHERE, but WHERE cannot be used with aggregate functions.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
HAVING AGGREGATE_FUNCTION(column_name) condition;
```

## ORDER BY Clause:

The ORDER BY clause is used to sort the result set in ascending (ASC) or descending (DESC) order based on one or more columns.

Syntax:

```
SELECT column_name  
FROM table_name  
ORDER BY column_name ASC|DESC;
```

## Examples

### 1. Built-in Functions Examples

Example 1: Finding the Average Salary

```
SELECT AVG(salary) AS AverageSalary  
FROM employees;
```

Example 2: Converting a Name to Uppercase

```
SELECT UPPER(first_name) AS UppercaseName  
FROM employees;
```

### 2. GROUP BY Clause

#### Examples

Example 1: Grouping by Department with Sum of Salaries

```
SELECT department, SUM(salary) AS TotalSalary  
FROM employees  
GROUP BY department;
```

Example 2: Counting Employees by Job Title

```
SELECT job_title, COUNT(*)  
FROM employees  
GROUP BY job_title;
```

### 3. HAVING Clause

#### Examples

Example 1: Filtering Groups with Total Salary Greater than 50,000

```
SELECT department, SUM(salary) AS TotalSalary  
FROM employees  
GROUP BY department  
HAVING SUM(salary) > 50000;
```

Example 2: Displaying Departments with More Than 5 Employees

```
SELECT department, COUNT(*)  
FROM employees  
GROUP BY department  
HAVING COUNT(*) > 5;
```

#### 4. ORDER BY Clause

##### Examples

Example 1: Sorting Employees by Salary in Descending Order

```
SELECT first_name, salary  
FROM employees  
ORDER BY salary DESC;
```

Example 2: Sorting Departments Alphabetically

```
SELECT department  
FROM employees  
ORDER BY department ASC;
```

**Conclusion:** In this experiment, we explored the usage of SQL built-in functions, including aggregate, string, and date functions, to perform various data operations. We also demonstrated the use of GROUP BY, HAVING, and ORDER BY clauses to group, filter, and sort query results efficiently. These concepts are essential for data analysis and manipulation in relational databases.

## Lab Assignment – Group by having

You have a table called with the following structure:

**SaleID**

**Product**

**Category**

**Amount**

**SaleDate**

1

Laptop

Electronics

50000

2024-01-15

2

TV

Electronics

40000

2024-01-18

3

Shirt

Clothing

2000

2024-01-20

4

Laptop

Electronics

55000

2024-02-05

5

Shirt

Clothing

2500

2024-02-10

**Question 2: Find products that have been sold more than once.**

**Question 3: Find the average sale amount per category, but only for categories with more than one sale.**

```

SQL Commands
Schema: WKSP_DBMS66

Language: SQL Rows: 10 Clear Command Find Tables Save Run

1 create table group54(sales_id int primary key ,product varchar(34),category varchar(45),amount decimal(10,2),sales_date date);
2 INSERT INTO group54(sales_id, product, category, amount, sales_date)
3 VALUES(1, 'laptop', 'electronics', 60000, '02-03-2024'),
4 INSERT INTO group54(sales_id, product, category, amount, sales_date)
5 values(2, 'tv', 'electronics', 40000, '03-05-2024');
6 INSERT INTO group54(sales_id, product, category, amount, sales_date)
7 values(3, 'shirt', 'clothing', 2000, '03-08-2024');
8 INSERT INTO group54(sales_id, product, category, amount, sales_date)
9 values(4, 'laptop', 'electronics', 55000, '09-03-2024');
10 INSERT INTO group54(sales_id, product, category, amount, sales_date)
11 values(5, 'jeans', 'clothing', 2500, '07-05-2025');
12 select *from group54;

```

SALES_ID	PRODUCT	CATEGORY	AMOUNT	SALES_DATE
2	tv	electronics	40000	3/5/2024
3	shirt	clothing	2000	3/8/2024
1	laptop	electronics	60000	2/3/2204
4	laptop	electronics	55000	9/3/2024
5	jeans	clothing	2500	7/5/2025

5 rows returned in 0.01 seconds Download

```

13
14 select category ,sum(amount) as totalamount from group54
15 group by category having sum(amount)>50000;

```

CATEGORY	TOTALAMOUNT
electronics	155000

1 rows returned in 0.00 seconds Download



2 rows returned in 0.01 seconds [Download](#)