

The screenshot shows the MySQL Workbench interface. In the 'Query 1' editor, the command 'use students_db' is entered. The output pane shows the message '0 row(s) affected' and 'Duration / Fetch 0.000 sec'. The status bar at the bottom indicates the session is connected to 'Local instance MySQL80'.

```
use students_db
```

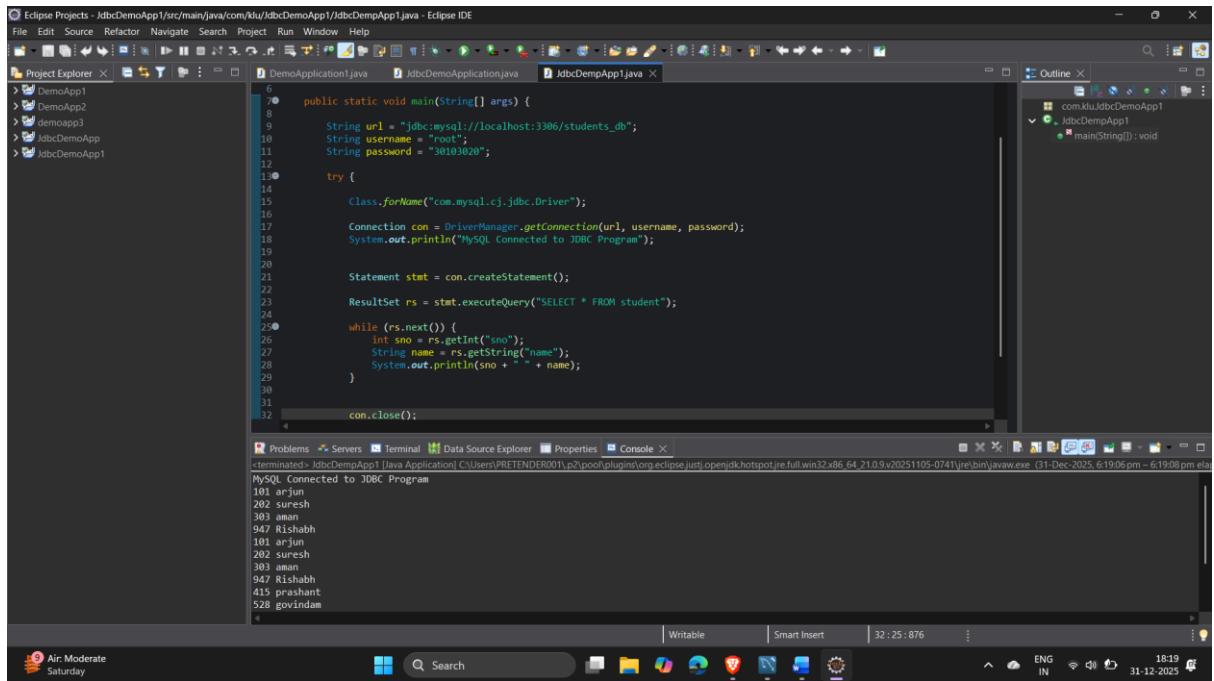
Inserting Values in table student

The screenshot shows the MySQL Workbench interface. In the 'Query 1' editor, multiple rows are inserted into the 'student' table using the following SQL statement:

```
INSERT INTO student (sno, name)
VALUES
(101, 'arjun'),
(202, 'suresh'),
(303, 'aman'),
(947, 'Rishabh'),
(415, 'prashant'),
(528, 'govindam'),
(639, 'neha'),
(742, 'rohit'),
(856, 'kiran'),
(981, 'anita');
```

The output pane shows the results of the insert operation, indicating 10 rows affected and a duration of 0.047 seconds. The status bar at the bottom indicates the session is connected to 'Local instance MySQL80'.

Now run as - java application



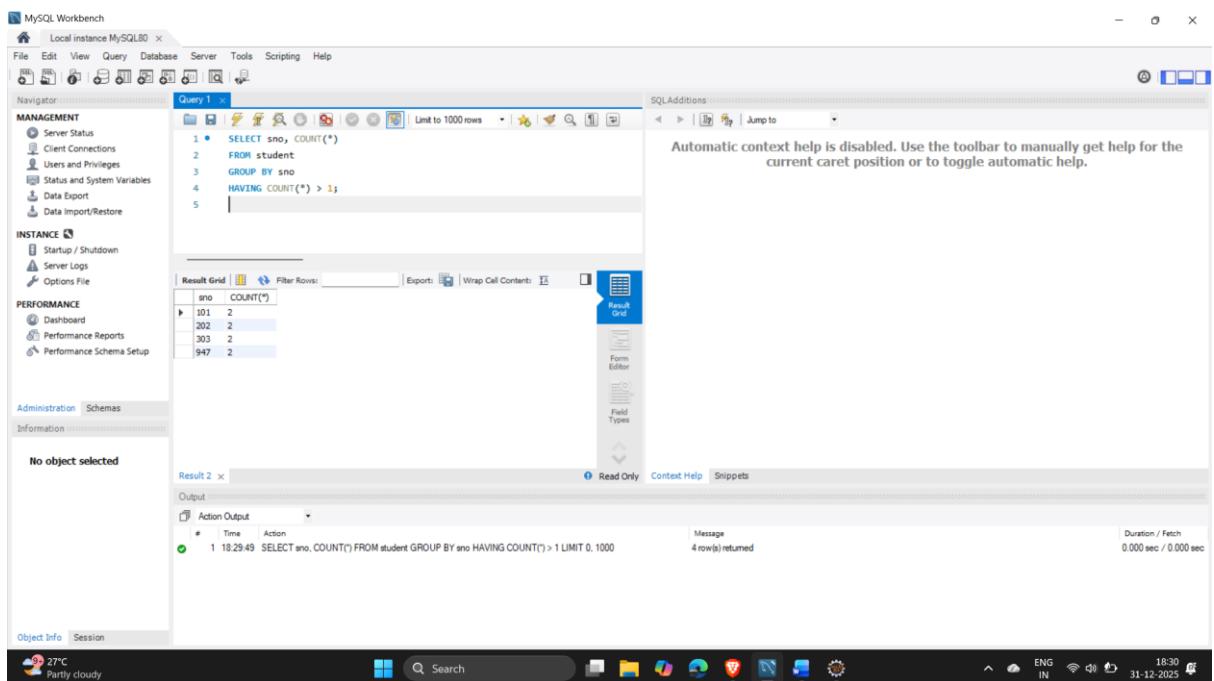
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows several Java projects: DemoApp1, DemoApp2, demoapp3, JdbcDemoApp, and JdbcDemoApp1.
- Code Editor:** Displays the Java code for `JdbcDemoApp1.java`. The code establishes a JDBC connection to a MySQL database named "students_db" and prints student records to the console.
- Console:** Shows the output of the executed code, displaying student records with sno and name.
- OS Taskbar:** Shows system information like battery level (Air: Moderate), date (31-12-2025), and time (18:19).

```
public static void main(String[] args) {
    String url = "jdbc:mysql://localhost:3306/students_db";
    String username = "root";
    String password = "30103020";
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(url, username, password);
        System.out.println("MySQL Connected to JDBC Program");
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM student");
        while (rs.next()) {
            int sno = rs.getInt("sno");
            String name = rs.getString("name");
            System.out.println(sno + " " + name);
        }
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
MySQL Connected to JDBC Program
101 arjun
202 suresh
303 aman
947 Rishabh
101 arjun
202 suresh
303 aman
947 Rishabh
415 prashant
528 govindam
```

Query for finding duplicates



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the MySQL instance "Local instance MySQL80".
- Query Editor:** Displays the SQL query to find student records with more than one occurrence:

```
1 • SELECT sno, COUNT(*)
  2   FROM student
  3   GROUP BY sno
  4   HAVING COUNT(*) > 1;
```

- Result Grid:** Shows the results of the query, which are empty (no duplicates found).
- Output:** Shows the execution details: 1 row(s) returned, duration 0.000 sec / 0.000 sec.
- OS Taskbar:** Shows system information like battery level (27°C), date (31-12-2025), and time (18:30).

MySQL Workbench - Local instance MySQL80

Query 1:

```
1 • SELECT * FROM student WHERE sno = 101;
2 |
```

Result Grid:

sno	name
101	arjun
101	arjun

Output:

Action	Time	Message	Duration / Fetch
SELECT sno, COUNT(*) FROM student GROUP BY sno HAVING COUNT(> 1) LIMIT 0, 1000	18:29:49	4 row(s) returned	0.000 sec / 0.000 sec
SELECT * FROM student WHERE sno = 101 LIMIT 0, 1000	18:31:30	2 row(s) returned	0.016 sec / 0.000 sec

Object Info Session

27°C Partly cloudy

ENG IN 18:31 31-12-2025

Fixing duplicates

MySQL Workbench - Local instance MySQL80

Query 1:

```
1 • DELETE FROM student
2 WHERE sno = 101
3 LIMIT 1;
4 |
```

Output:

Action	Time	Message	Duration / Fetch
SELECT sno, COUNT(*) FROM student GROUP BY sno HAVING COUNT(> 1) LIMIT 0, 1000	18:29:49	4 row(s) returned	0.000 sec / 0.000 sec
SELECT * FROM student WHERE sno = 101 LIMIT 0, 1000	18:31:30	2 row(s) returned	0.016 sec / 0.000 sec
DELETE FROM student WHERE sno = 101 LIMIT 1	18:33:07	1 row(s) affected	0.032 sec

Object Info Session

27°C Partly cloudy

ENG IN 18:33 31-12-2025

Delete all duplicates one by one like above until you get something like this

```

MySQL Workbench
Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help
Navigator
Query 1 ×
SELECT sno, COUNT(*)
FROM student
GROUP BY sno
HAVING COUNT(*) > 1;
Result Grid | Filter Rows: Export: Wrap Cell Contents: Result Grid
sno COUNT(*)
Result 6 ×
Output
Action Output
# Time Action
1 18:35:22 SELECT `ano`, COUNT(*) FROM `student` GROUP BY `ano` HAVING COUNT(*) > 1 LIMIT 0, 1000
2 18:35:45 DELETE FROM `student` WHERE `ano` = 202 LIMIT 1
3 18:35:54 DELETE FROM `student` WHERE `ano` = 303 LIMIT 1
4 18:36:15 SELECT `ano`, COUNT(*) FROM `student` GROUP BY `ano` HAVING COUNT(*) > 1 LIMIT 0, 1000
5 18:36:41 DELETE FROM `student` WHERE `ano` = 947 LIMIT 1
6 18:36:59 SELECT `ano`, COUNT(*) FROM `student` GROUP BY `ano` HAVING COUNT(*) > 1 LIMIT 0, 1000
Object Info Session

```

Now add Primary Key(sno)

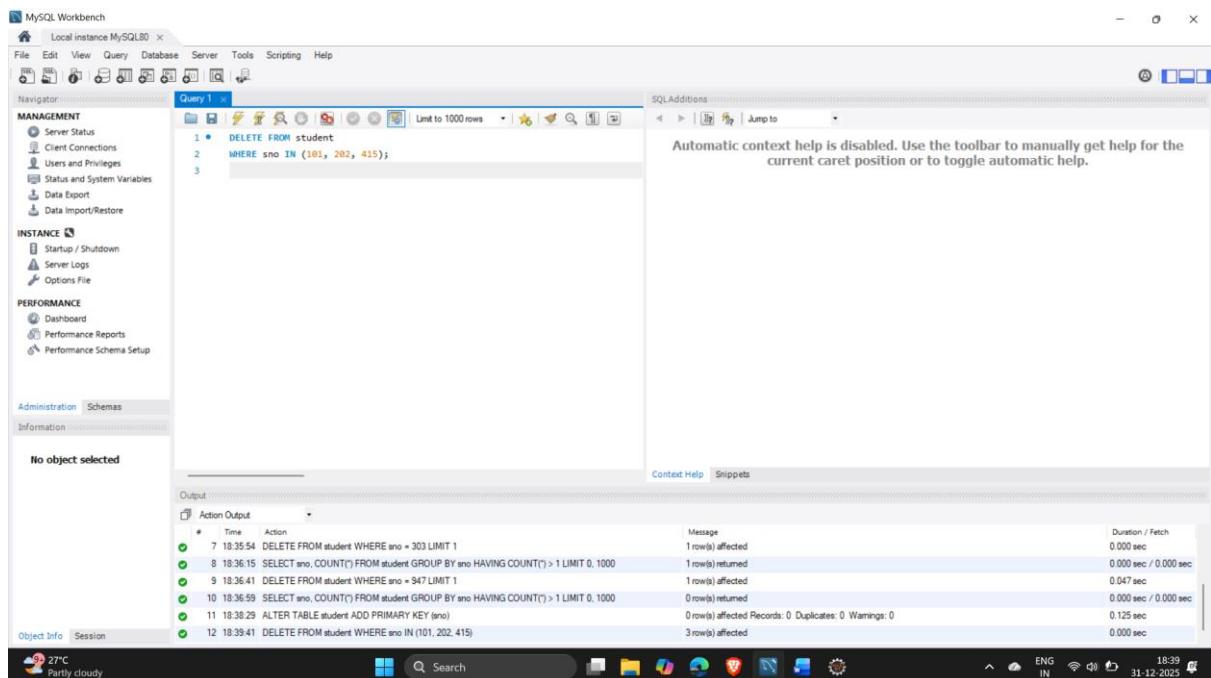
```

MySQL Workbench
Local instance MySQL80

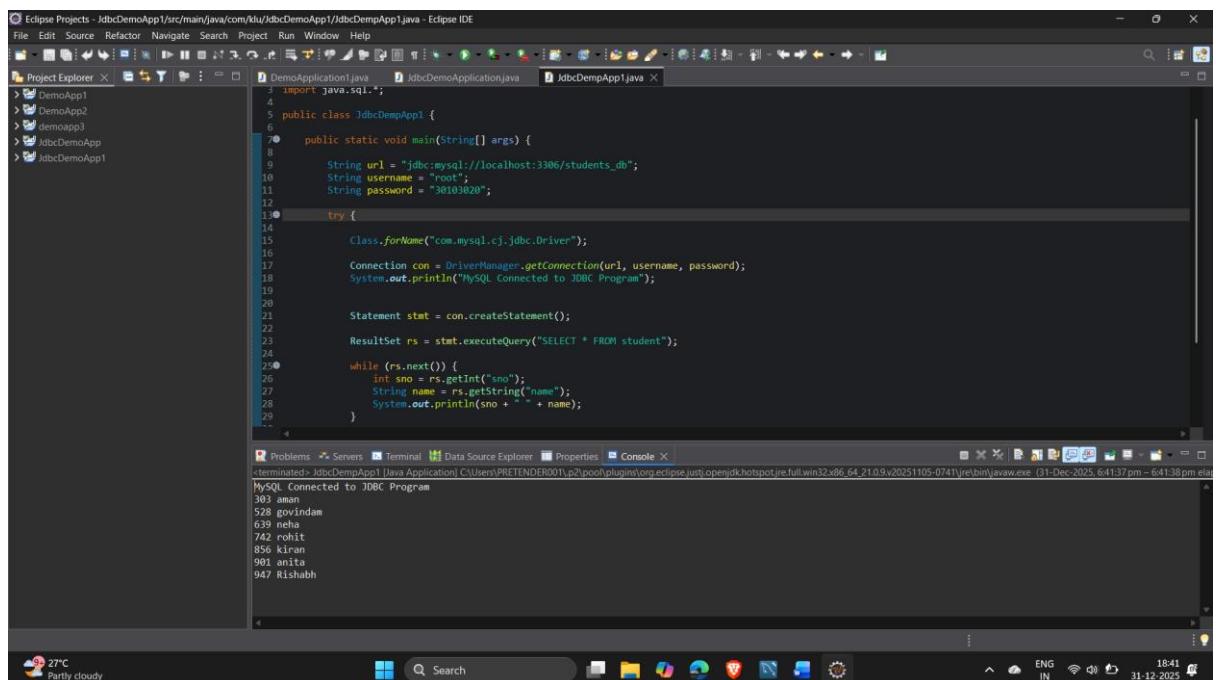
File Edit View Query Database Server Tools Scripting Help
Navigator
Query 1 ×
ALTER TABLE student
ADD PRIMARY KEY (sno);
Result Grid | Filter Rows: Export: Wrap Cell Contents: Result Grid
Result 6 ×
Output
Action Output
# Time Action
1 18:35:45 DELETE FROM `student` WHERE `ano` = 202 LIMIT 1
2 18:35:54 DELETE FROM `student` WHERE `ano` = 303 LIMIT 1
3 18:36:15 SELECT `ano`, COUNT(*) FROM `student` GROUP BY `ano` HAVING COUNT(*) > 1 LIMIT 0, 1000
4 18:36:41 DELETE FROM `student` WHERE `ano` = 947 LIMIT 1
5 18:36:59 SELECT `ano`, COUNT(*) FROM `student` GROUP BY `ano` HAVING COUNT(*) > 1 LIMIT 0, 1000
6 18:38:29 ALTER TABLE student ADD PRIMARY KEY (sno)
Object Info Session

```

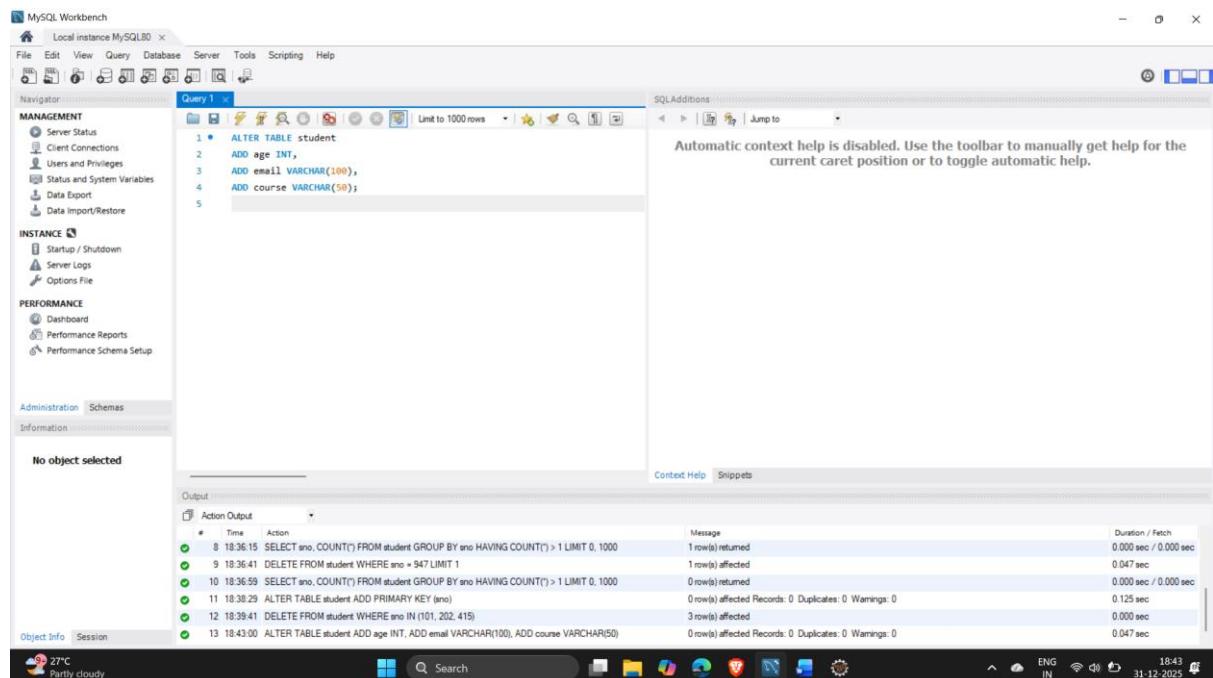
Query for deleting records



Verify in Eclipse Ide



Adding New columns

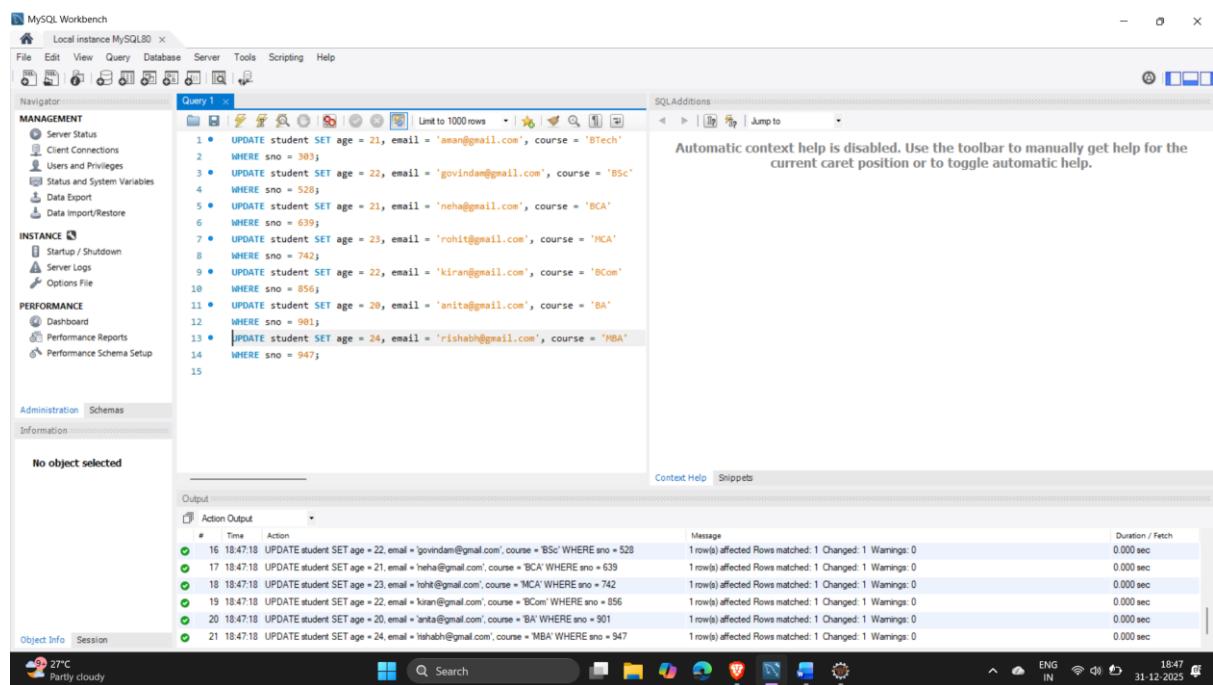


The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with the **MANAGEMENT**, **INSTANCE**, and **PERFORMANCE** sections.
- Query Editor (Query 1):** Contains the following SQL code:

```
1 • ALTER TABLE student
2   ADD age INT,
3   ADD email VARCHAR(100),
4   ADD course VARCHAR(50);
```
- Output:** Displays the execution results of the previous query, showing the creation of the `student` table with three new columns: `age`, `email`, and `course`.
- System Bar:** Shows the system status including temperature (27°C), weather (Partly cloudy), and system time (18:43, 31-12-2025).

Update the records



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with the **MANAGEMENT**, **INSTANCE**, and **PERFORMANCE** sections.
- Query Editor (Query 1):** Contains the following SQL code:

```
1 • UPDATE student SET age = 21, email = 'aman@gmail.com', course = 'BTech'
2 WHERE sno = 303;
3 • UPDATE student SET age = 22, email = 'govindam@gmail.com', course = 'BSc'
4 WHERE sno = 528;
5 • UPDATE student SET age = 21, email = 'neha@gmail.com', course = 'BCA'
6 WHERE sno = 639;
7 • UPDATE student SET age = 23, email = 'rohit@gmail.com', course = 'MCA'
8 WHERE sno = 742;
9 • UPDATE student SET age = 22, email = 'kirangmail.com', course = 'BCom'
10 WHERE sno = 856;
11 • UPDATE student SET age = 20, email = 'anita@gmail.com', course = 'BA'
12 WHERE sno = 901;
13 • UPDATE student SET age = 24, email = 'rishabh@gmail.com', course = 'MBA'
14 WHERE sno = 947;
```
- Output:** Displays the execution results of the previous query, showing the update of multiple records in the `student` table.
- System Bar:** Shows the system status including temperature (27°C), weather (Partly cloudy), and system time (18:47, 31-12-2025).

Update the code in Eclipse Ide – Run as – Java Application

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows projects: DemoApp1, DemoApp2, demoapp3, JdbcDemoApp.
- Java Editor:** Displays the `JdbcDemoApp1.java` file content. The code is a Java program that connects to a database, executes a query to select all rows from the student table, prints the column headers, and then prints each row's sno, name, email, age, and course.
- Console:** Shows the output of the executed Java application. The output is a table with columns: SNO, NAME, EMAIL, AGE, COURSE. The data is as follows:

SNO	NAME	EMAIL	AGE	COURSE
303	aman	aman@gmail.com	21	BTech
528	govindam	govindam@gmail.com	22	BSc
639	neha	neha@gmail.com	21	BCA
742	rohit	rohit@gmail.com	23	MCA
856	kirat	kirat@gmail.com	22	BCom
901	anita	anita@gmail.com	20	BA
947	Rishabh	rishabh@gmail.com	24	MBA