Name: Jyothirmayi Lakkoju
Email: Jyothi.lakkoju@gmail.com

`

Student Management Database Project

Table of Contents

Objective:

The aim of this project is to design and implement a student management database system that efficiently organizes and maintains student records, grades, and related information. This system aims to ensure data integrity, support data retrieval and analysis through queries and stored procedures, and provide a clear and normalized database structure for optimal performance and scalability.

Creation of Database:

In this database I have created 3 tables namely, branches, grades and student.

The student table consist of 250 records with fields:

- student_id
- name
- DOB
- branch (AI, CSE,ECE,EEE,ME)

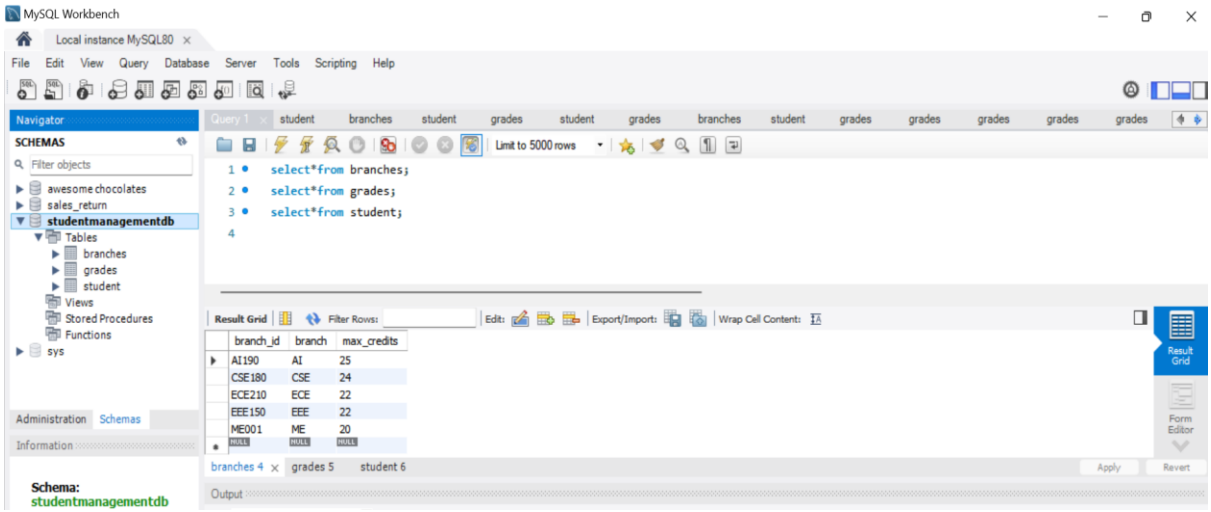
The branch table consist of 5 records with fields:

- branch_id (There is a specific ID for each branch)
- branch
- max_credits (max score a student can get in their respective course)

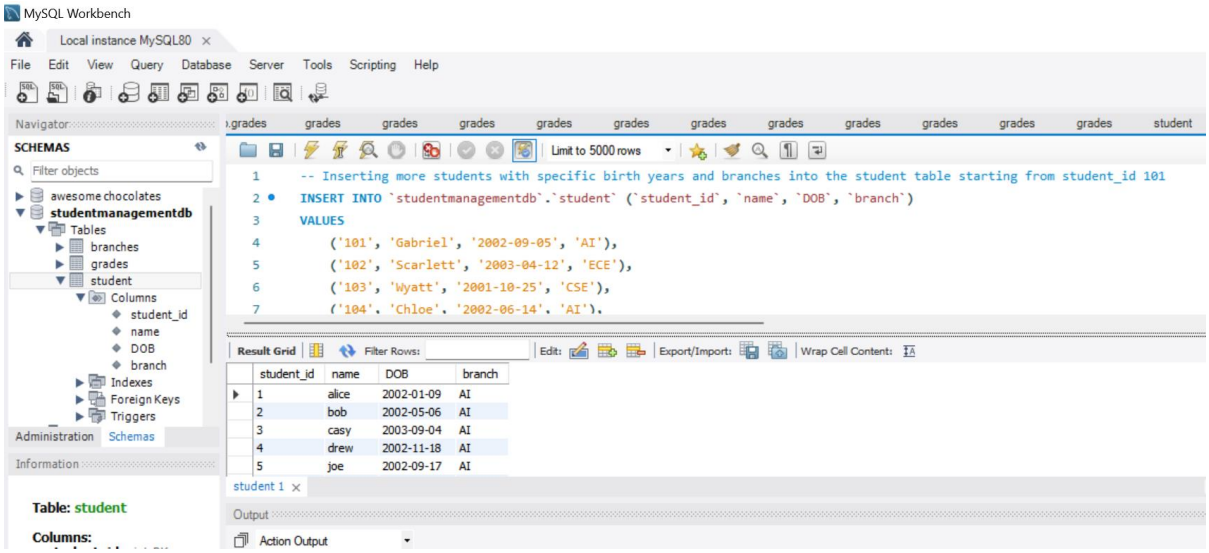
The grades table consist of 250 records with fields:

- student_id
- branch_id
- credit_score (score a student has obtained in their respective course)

Final_grade (based on the credit_score and the respective branch's max score a final grade is given to the student)



Screenshots on creating the database:



```sql
UPDATE grades g
JOIN branches b ON g.branch_id = b.branch_id
SET g.final_grade =
    CASE
        WHEN g.credit_score >= 20 THEN 'A'
        WHEN g.credit_score >= 15 THEN 'B'
        WHEN g.credit_score >= 10 THEN 'C'
        WHEN g.credit_score >= 5 THEN 'D'
        ELSE 'F'
    END
WHERE b.branch = 'AI190';
```

Query Testing:

Select all records:

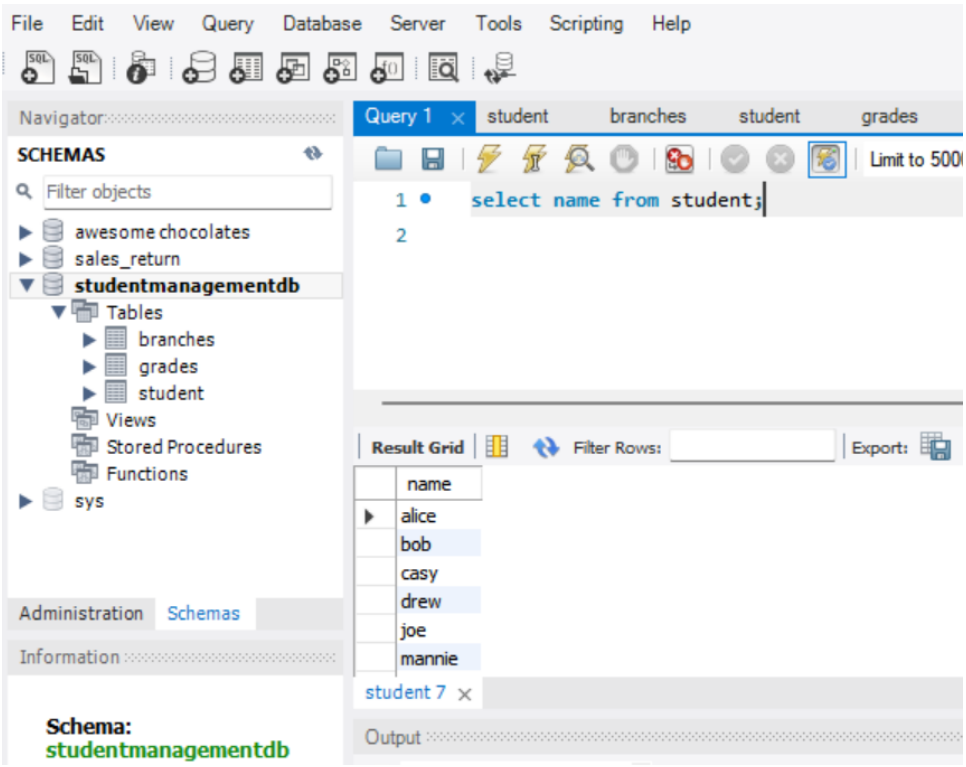
We use the select statement:

```sql
select*from branches;
select*from grades;
select*from student;
```

student_id	branch_id	credit_score	Final_grade
246	AI190	0	F
247	ECE210	18	B
248	CSE180	20	A
249	EEE150	0	F
250	ME001	14	C

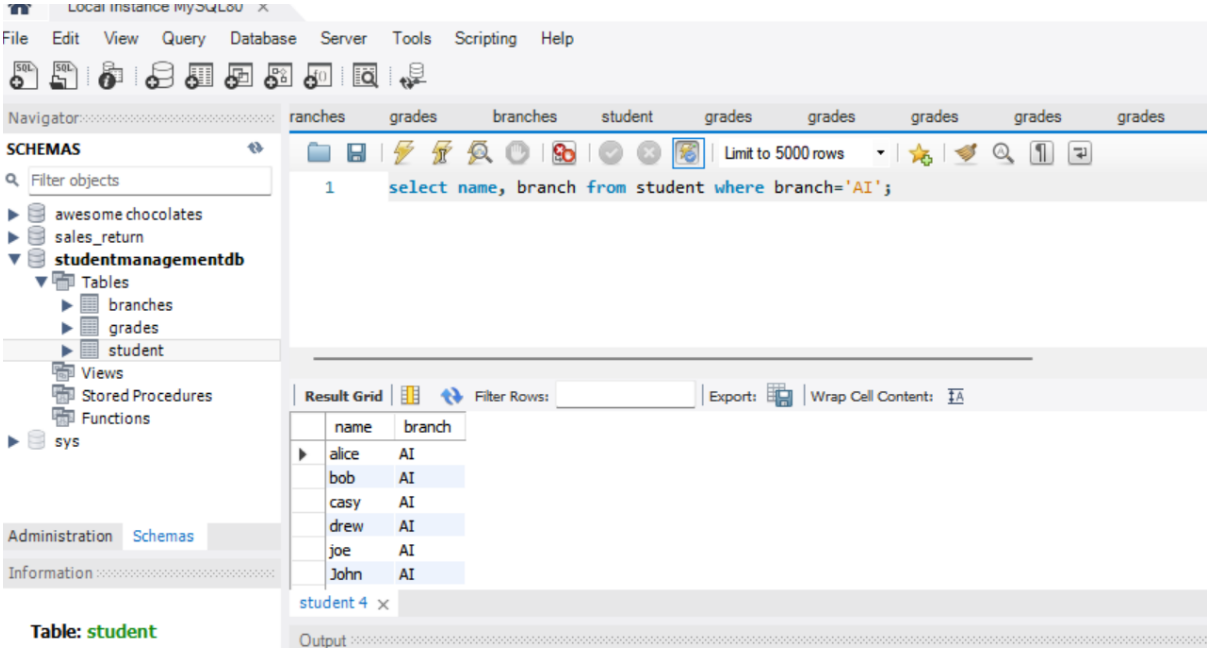
Selecting a specific column from a table:

I have selected all the names from the student table



Filter by condition:

In this query I have selected two columns from student table i.e name , branch with the condition that the selected student name is associated with 'AI' branch.



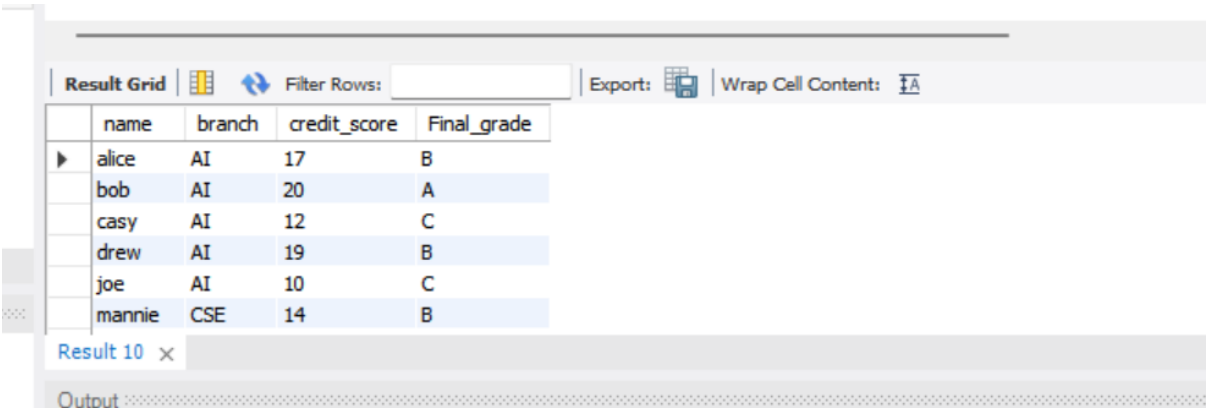
Joins:

To retrieve student names , branch , credit_score and Final_grade:

Query:

```
select s.name, s.branch, g.credit_score,g.Final_grade from student s join grades g on s.student_id = g.student_id;
```

Result:



name	branch	credit_score	Final_grade
alice	AI	17	B
bob	AI	20	A
casy	AI	12	C
drew	AI	19	B
joe	AI	10	C
mannie	CSE	14	B

Aggregate Queries:

1.To retrieve the count of each grade by the students.

Query:

```
select Final_grade,count(*) as grade_count from grades group by Final_grade;
```

Result:

Result Grid | Filter Rows: | Export:

Final_grade	grade_count
B	78
A	58
C	57
D	29
F	28

2. To retrieve count of students per branch.

Query:

```
1   select branch,count(*) as student_count from student group by branch;
```

Result:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

branch	student_count
AI	64
CSE	65
EEE	17
ECE	58
ME	46

Average Credit Score per branch:

Query:

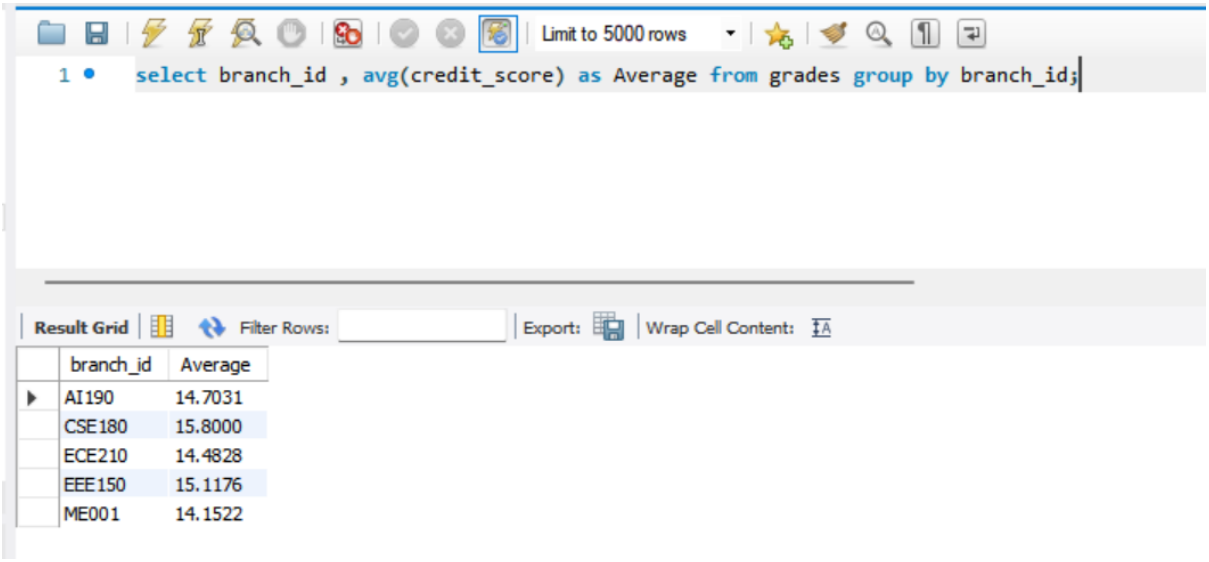
```
1   select b.branch, avg(g.credit_score) as Average from branches b join grades g on b.branch_id = g.branch_id group by b.branch;
```

Result:

Result Grid | Filter Rows: | Export:

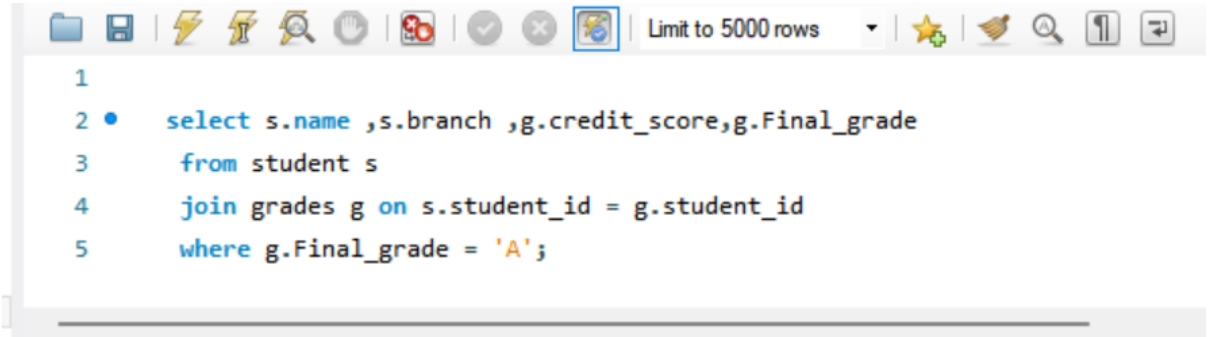
branch	Average
AI	14.7031
CSE	15.8000
ECE	14.4828
EEE	15.1176
ME	14.1522

Incase we want to use only branch_id (without any 'Join' function), we can write:



Finding all students with top grades:

Query:



Result:

Result Grid | Filter Rows: | Exp

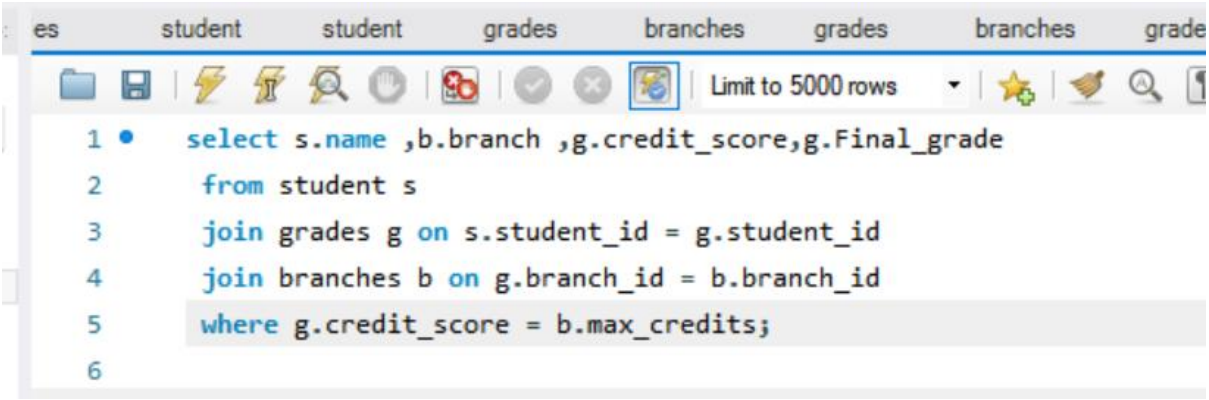
name	branch	credit_score	Final_grade
bob	AI	20	A
jane	CSE	21	A
John	AI	25	A
Benjamin	CSE	20	A
Charlotte	CSE	24	A
Logan	ECE	22	A

Result 30 ×

Output

Finding the student names who reached the max credits for their course:

Query:



Result:

Result Grid | Filter Rows: | Export: | Wrap Cell C

name	branch	credit_score	Final_grade
John	AI	25	A
Leo	AI	25	A
Ravi	AI	25	A
Charlotte	CSE	24	A
Greyson	CSE	24	A
Eli	CSE	24	A

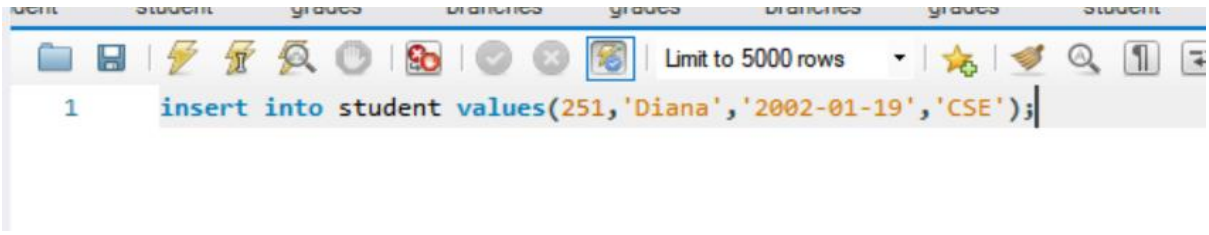
Result 5 ×

Output

Insert, Update, Delete Queries:

Insert:

Query:



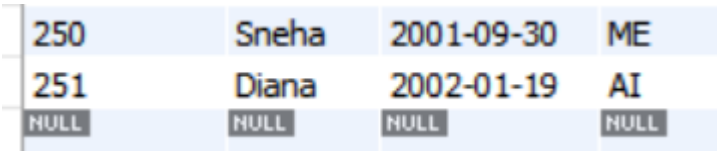
Result:

250	Sneha	2001-09-30	ME
251	Diana	2002-01-19	CSE
NULL	NULL	NULL	NULL

Update:

Query:

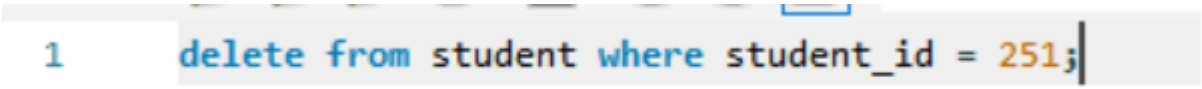
Result:



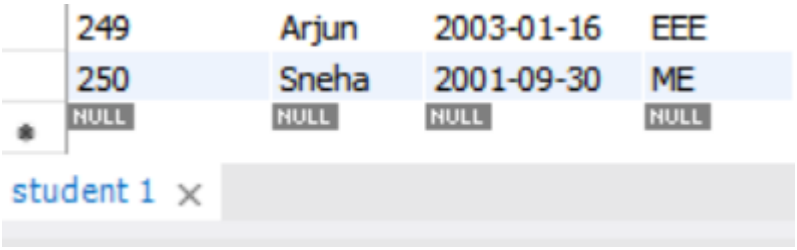
250	Sneha	2001-09-30	ME
251	Diana	2002-01-19	AI
NULL	NULL	NULL	NULL

Delete:

Query:



Result:



249	Arjun	2003-01-16	EEE
250	Sneha	2001-09-30	ME
NULL	NULL	NULL	NULL

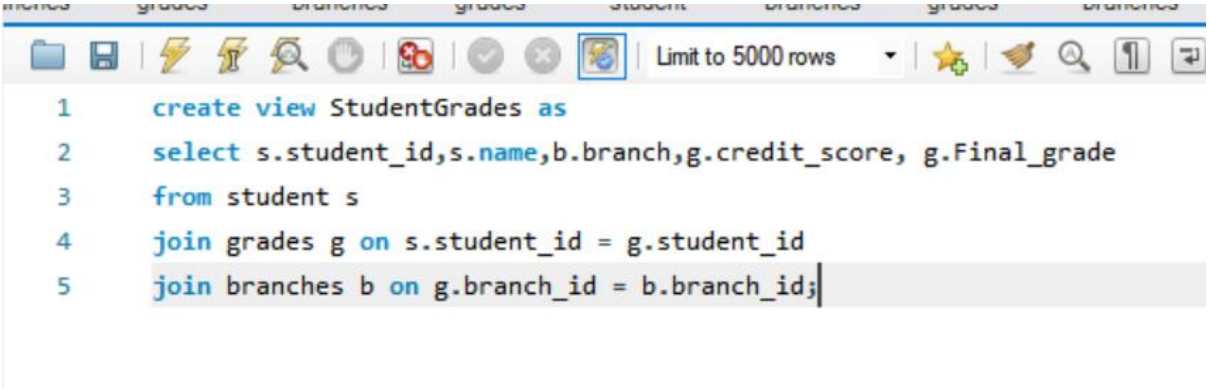
student 1

There is no record after student_id = 250

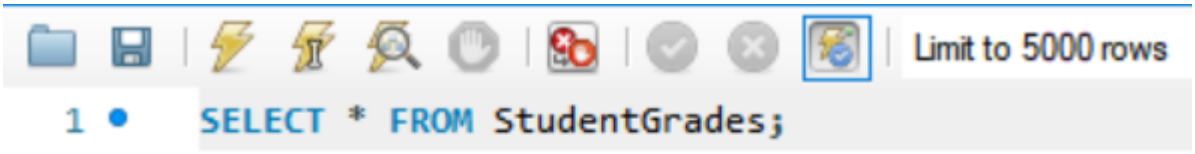
Creating Views:

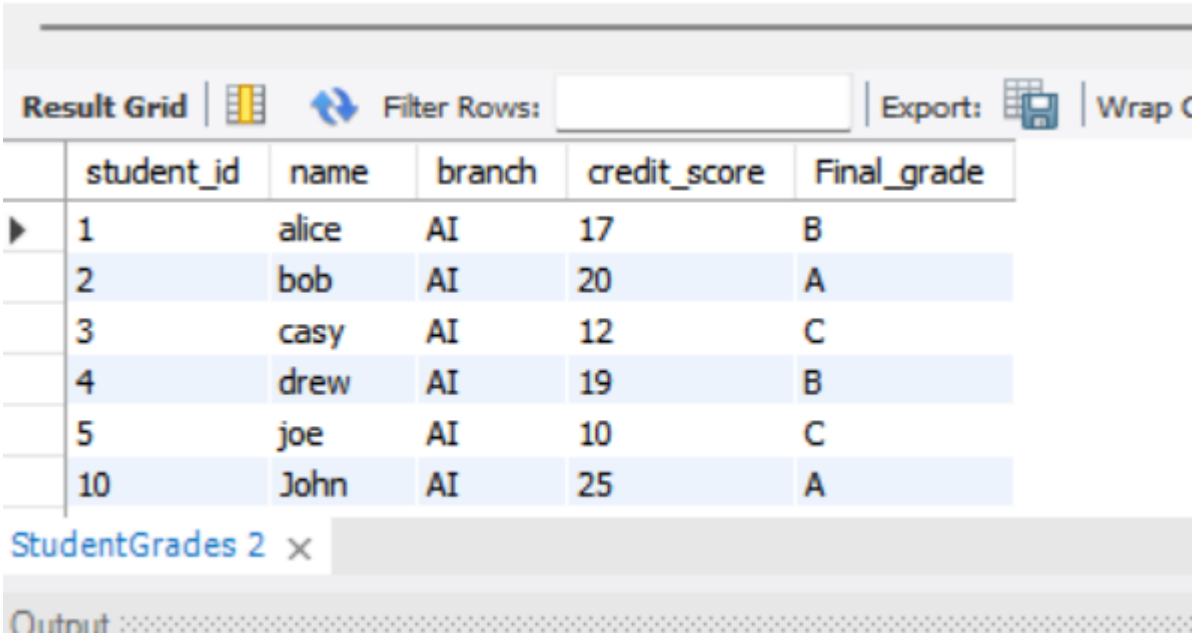
A view is helpful because it provides a virtual table representing the result of a stored query, allowing us to simplify complex queries and enhance data security by presenting only the necessary data to users without exposing the underlying table structures.

Query:



Accessing a view:

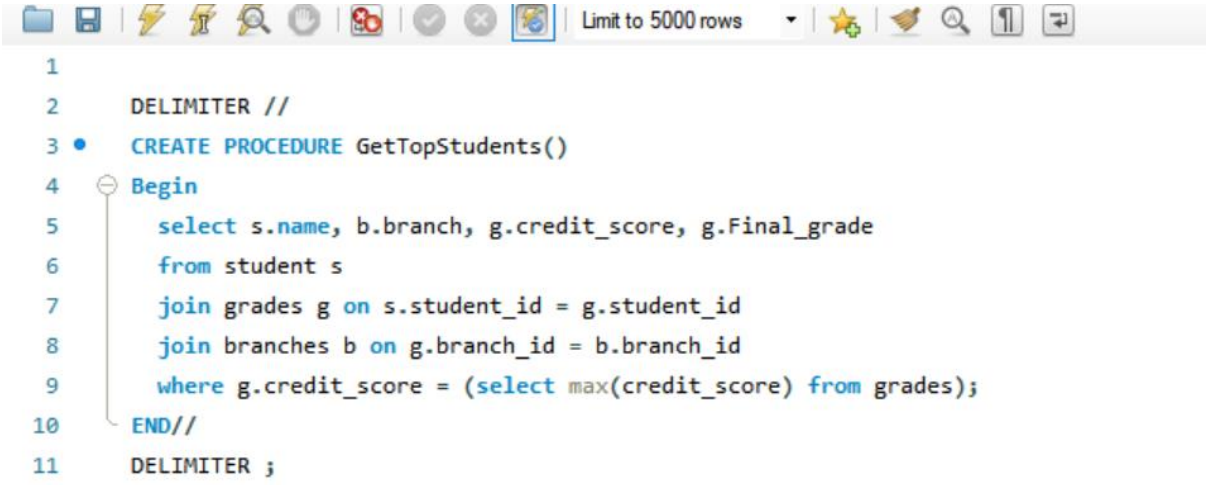




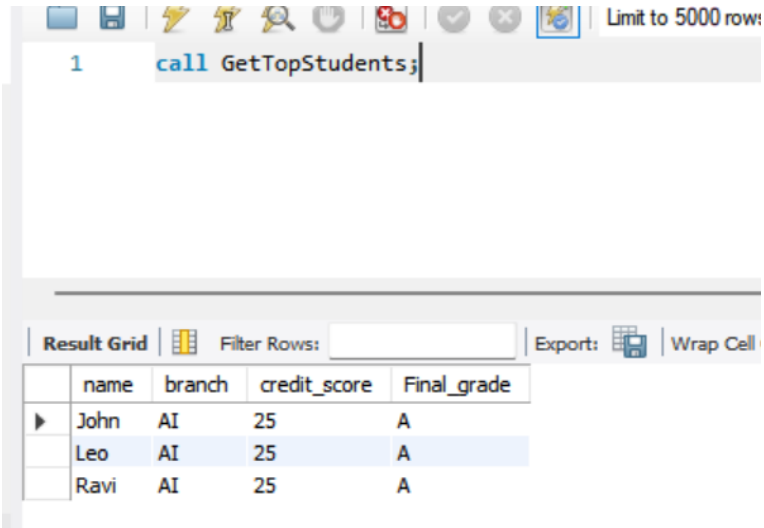
Creating a Procedure:

A procedure is helpful as it stores the logic in the form of a reusable query and we can access it by calling the stored procedure without needing to write the full code. This reduces the redundancy, code maintainability , and thereby enhancing the performance

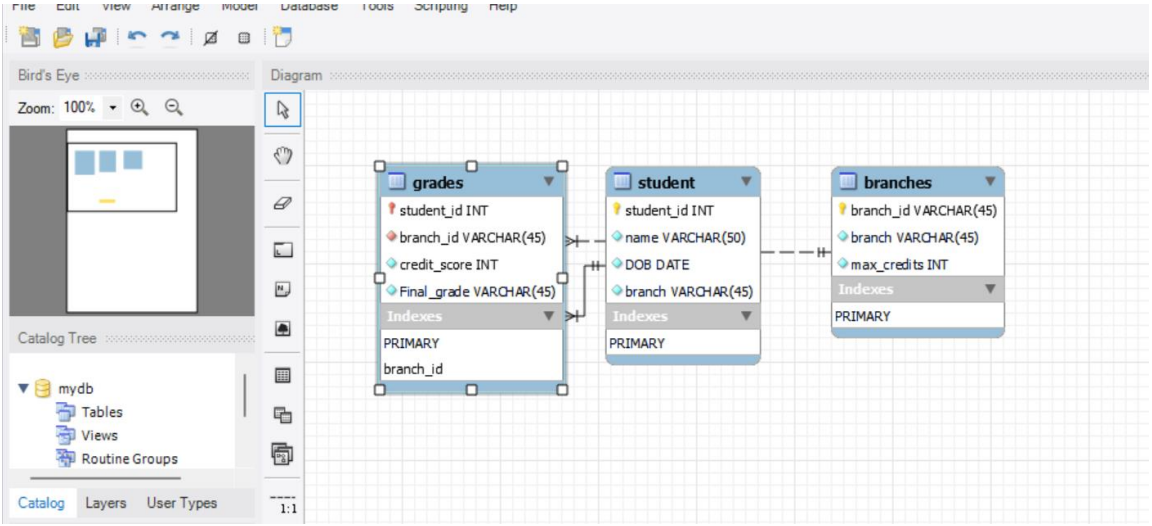
Query:



Result:



ER Diagram:



Description:

student table:

- student_id: Primary key, uniquely identifies each student.
- name: Stores the name of the student.
- DOB: Stores the date of birth of the student.
- branch: Stores the branch name (foreign key to branches table).

grades table:

- Composite Primary key: (student_id, branch_id)
- Foreign key :
  'student_id' referencing student(student_id)
  'branch_id' referencing branches(branch_id)

- credit_score: Stores the credit score of the student.
- final_grade: Stores the final grade of the student.

branches table:

- branch_id: Primary key, uniquely identifies each branch.
- branch: Stores the branch name.
- max_credits: Stores the maximum credits for the branch.

Describing the Relationships in the ER Diagram:

1. Relationship between student and grades tables:

Type: One-to-Many

Description:

Each student can have multiple grade entries, but each grade entry is associated with only one student. The student_id is foreign key in the grades table, which references the student_id primary key in the student table.

2. Relationship between grades and branches tables:

Type: Many-to-One

Description: Each grade entry is associated with one branch, but each branch can have multiple grade entries. The branch_id is the foreign key in the grades table, which refers to the branch_id primary key in the branches table.

3. Relationship between student and branches tables:

Type: Many-to-One

Description: Each student is associated with one branch, but each branch can have multiple students. ‘branch’ is the foreign key in the student table, which refers to the ‘branch’ primary key in the

branches table.

Normalization:

Normalization is essential for reducing data redundancy and ensuring data integrity in a database. It organizes data into related tables, which prevents duplicate data and maintains consistency and accuracy.

1NF (First Normal Form):

- ✓ Each table should have a primary key to ensure uniqueness.
- ✓ Each column should contain atomic (indivisible) values.
- ✓ Each column should contain values of a single type.

2NF (Second Normal Form):

Criteria:

- ✓ The table must be in 1NF.
- ✓ All non-key attributes must be fully functionally dependent on the whole primary key.

Description:

- **student table**:
  - Non-key attributes (name, DOB, branch) depend on the primary key student_id.
  - The table is in 2NF.
- **grades table**:
  - Composite primary key: (student_id, branch_id)

    - Non-key attributes (credit_score, final_grade) are dependent on both student_id and branch_id.
    - The table is in 2NF.
- **branches table**:
    - Non-key attributes (branch, max_credits) depend on the primary key branch_id.
    - The table is in 2NF.

Result: All tables are in 2NF.

3NF (Third Normal Form):

Criteria:

- ✓ The table must be in 2NF.
- ✓ All non-key attributes must be non-transitively dependent on the primary key (i.e., no transitive dependency).

Description:

- **student table**:
    - Non-key attributes (name, DOB, branch) depend directly on the primary key student_id.
    - No transitive dependencies exist.
    - The table is in 3NF.
- **grades table**:
    - Non-key attributes (credit_score, final_grade) depend directly on the composite primary key (student_id, branch_id).
    - No transitive dependencies exist.
    - The table is in 3NF.
- **branches table**:
    - Non-key attributes (branch, max_credits) depend directly on the primary key branch_id.
    - No transitive dependencies exist.
    - The table is in 3NF.

Conclusion:

In this project the main goal was to create a database which manages students records.

Throughout this project I achieved:

- ✓ Designing a database with relevant tables
- ✓ Obtaining an ER diagram with accurate relationships
- ✓ Populating the tables with data
- ✓ Testing out various SQL queries (Data retrieval, Join queries)
- ✓ Stored Procedure with example
- ✓ Manipulating Data
- ✓ Normalization