**The University Of Azad Jammu & Kashmir,**

**Muzaffarabad**

**Department of Software Engineering**

**LAB TASK 11**

**Database Systems**

**Course Code**: **CS-2204**

**Submitted To:**

Engr. Tahir Jahangir

**Submitted By:**

Ahmed Ali

**Roll No:**

2023-SE-41

**Session**: 2023-2027

Contents

[Lab Task – 11: SQL Server – Views & Indexes 3](#_Toc207376149)

[🎯 Objective: 3](#_Toc207376150)

[📘 Task 01: Database and Table Setup 3](#_Toc207376151)

[Instructions: 3](#_Toc207376152)

[EMPLOYEES 3](#_Toc207376153)

[DEPARTMENTS 3](#_Toc207376154)

[📊 Task 02: Creating and Querying Views 5](#_Toc207376155)

[Instructions: 5](#_Toc207376156)

[📁 Task 03: Update and Delete Operations Using Views 7](#_Toc207376157)

[Instructions: 7](#_Toc207376158)

[📗 Task 04: Index Creation and Analysis 8](#_Toc207376159)

[Instructions: 8](#_Toc207376160)

[📘 Task 05: Conceptual Reflection – Views vs. Indexes 10](#_Toc207376161)

[Instructions: 10](#_Toc207376162)

[2. How do views help in data abstraction and security? 10](#_Toc207376163)

[**Reflection on Challenges and Learning:** 11](#_Toc207376164)

# Lab Task – 11: SQL Server – Views & Indexes

# 🎯 Objective:

In this lab, students will explore the creation and usage of **Views** and **Indexes** in SQL Server to enhance data abstraction, security, and performance. This lab will reinforce understanding of database optimization and controlled access by engaging students in real-world data modeling and query execution without revealing physical table structures.

# 📘 Task 01: Database and Table Setup

🎯 **Objective**: Create a sample database and tables for practice with views and indexes.

## Instructions:

1. Create a new database using the format:

your\_full\_name\_lab\_task\_11

*Example*: muhammad\_asim\_lab\_task\_11

1. Design the following two tables with appropriate attributes and constraints:

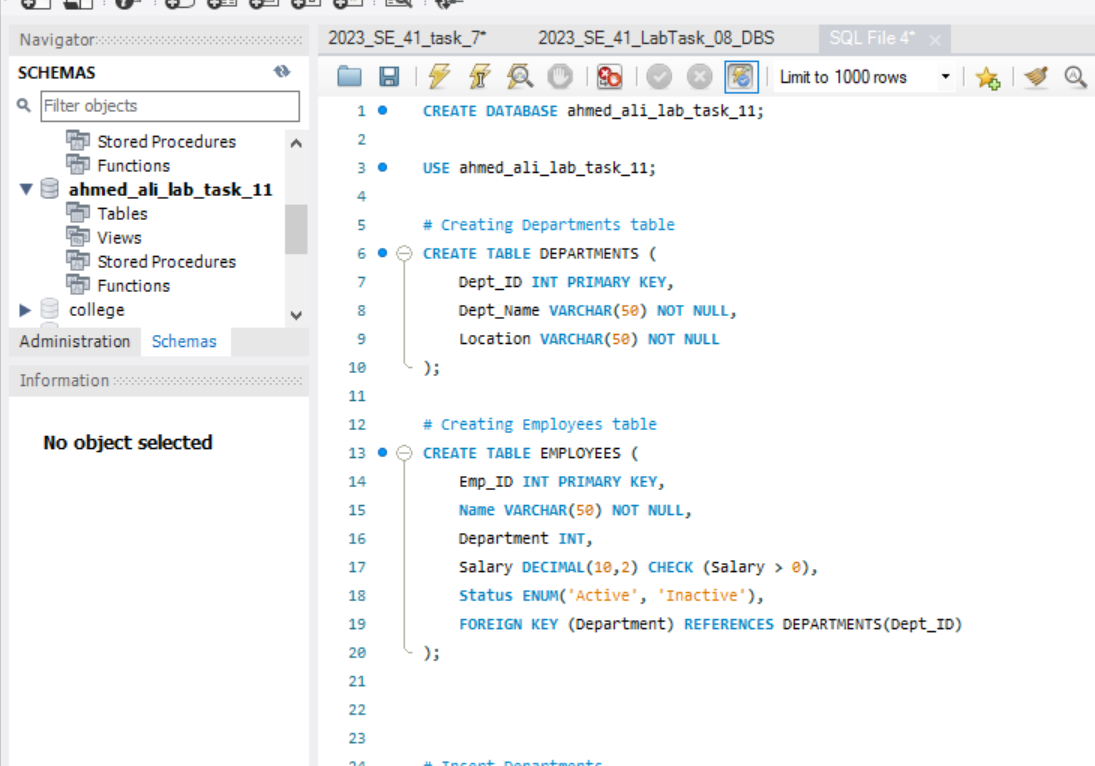
## EMPLOYEES

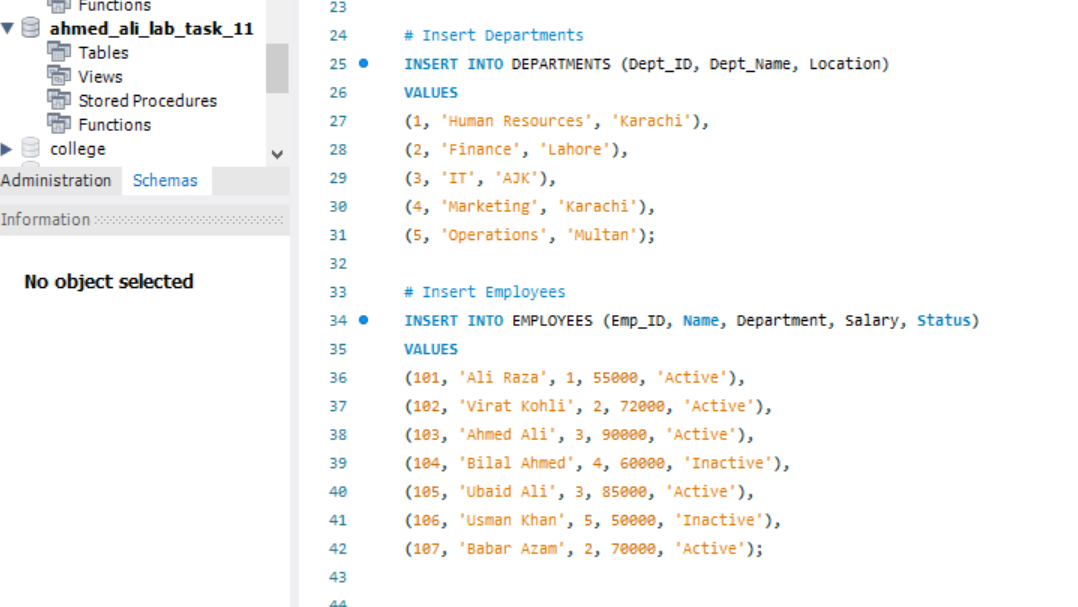
o Emp\_ID (Primary Key) o Name o Department o Salary o Status (Active/Inactive)

## DEPARTMENTS

o Dept\_ID (Primary Key) o Dept\_Name o Location

3. Insert at least **5–7 sample records** in each table, ensuring a mix of departments and statuses for variety.



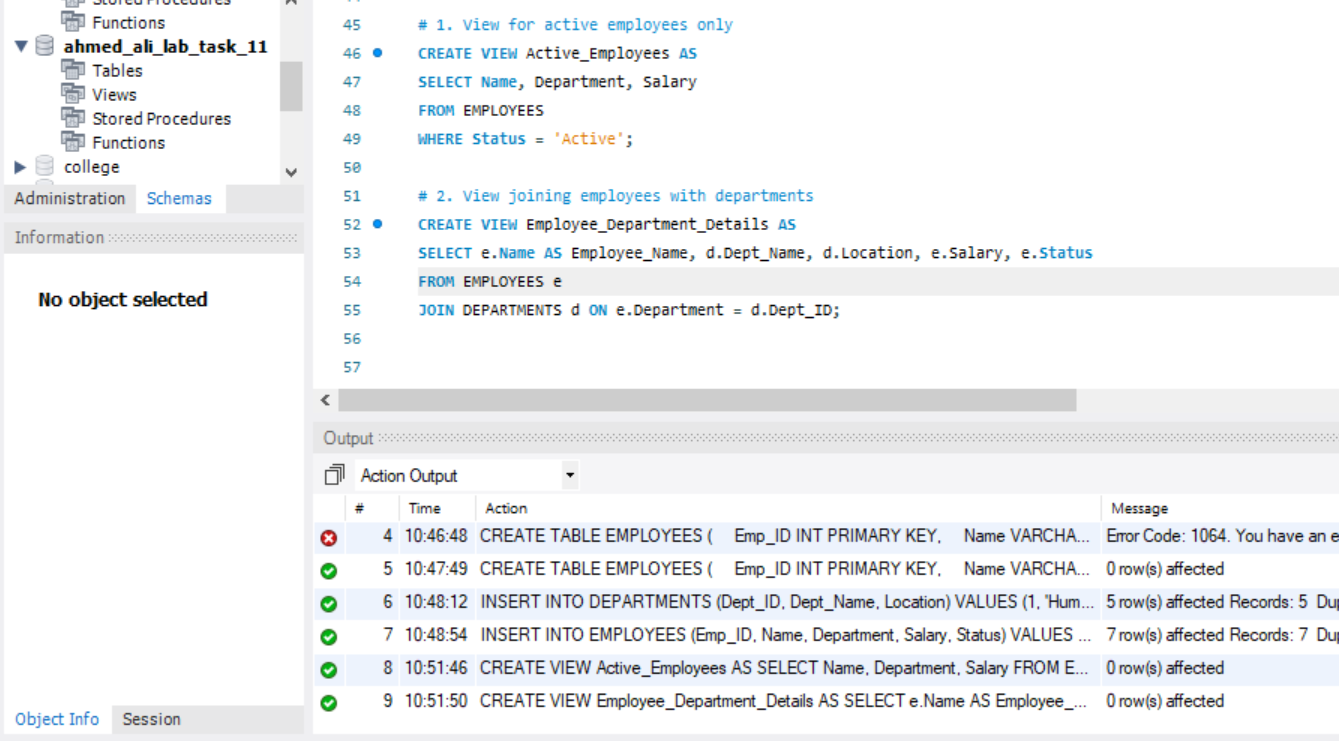


# 📊 Task 02: Creating and Querying Views

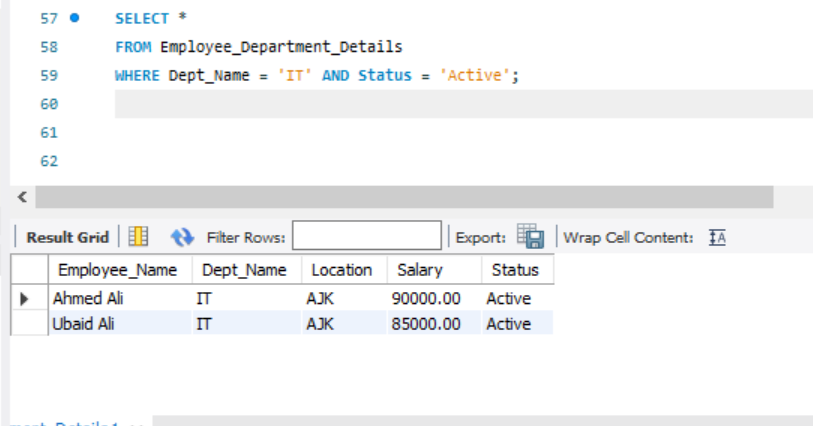
🎯 **Objective**: Learn to create and use views for simplifying complex queries and enforcing restricted access.

## Instructions:

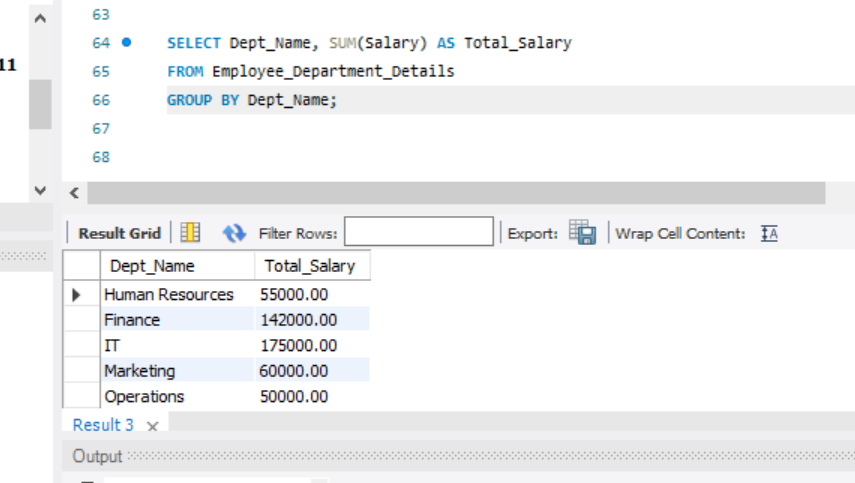
1. Create a view that displays only **active employees** with their names, departments, and salary.
2. Create a view that joins EMPLOYEES and DEPARTMENTS to show employee name, department name, and location.

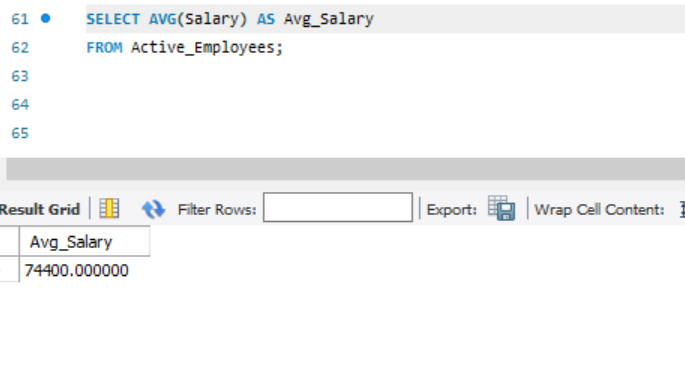


1. Use the views you created to run SELECT queries that demonstrate: o Filtering data (e.g., employees in a specific department).



* Aggregating salary information from a view.



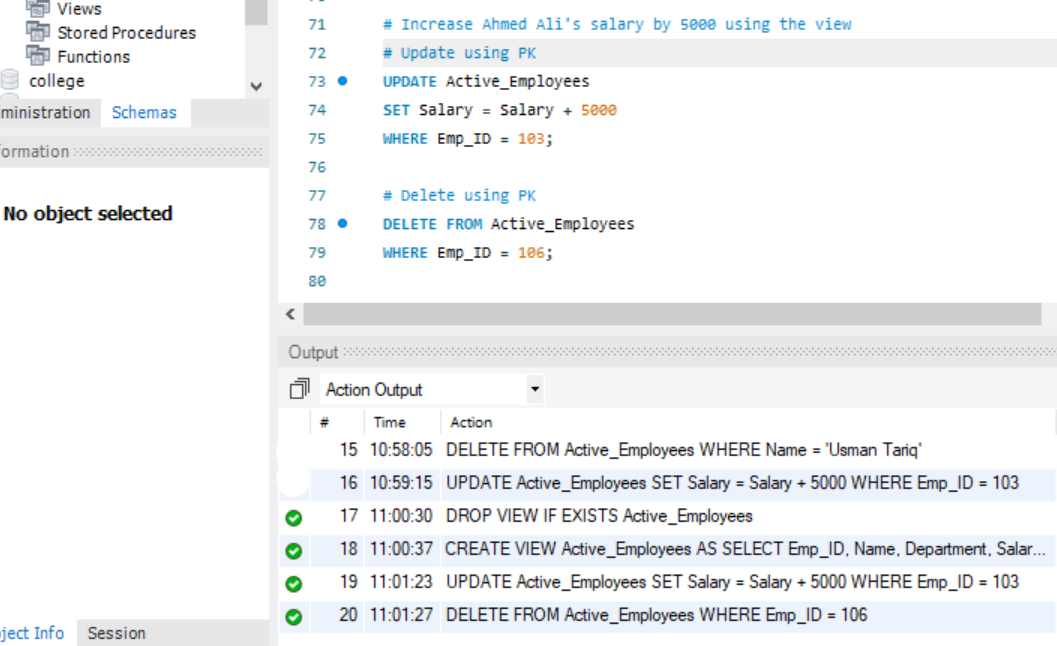


# 📁 Task 03: Update and Delete Operations Using Views

🎯 **Objective**: Understand the limitations and capabilities of performing DML operations through views.

## Instructions:

1. Update the salary of a specific employee through the appropriate view.
2. Try to delete an employee record using a view (document whether it worked and why/why not).



1. **Reflect on which operations were successful or restricted, and provide reasons for those outcomes.**

During this task, I observed that update and delete operations through views depend on whether the view is updatable. Using the **Active\_Employees view**, I was able to successfully update and delete records once the Emp\_ID (primary key) was included in the view. This worked because the view is directly based on a single table (EMPLOYEES) without joins or aggregates, so MySQL can map the changes back to the base table. However, when I tried to perform these operations using non-key columns (like Name), MySQL Workbench blocked them due to **Safe Update Mode**, which requires a primary key condition to prevent accidental bulk changes. On the other hand, attempts to delete from the **Employee\_Department\_Details view** failed completely because it is a **join view**, and MySQL does not allow updates or deletions on such views. This shows that simple views are updatable, while join or complex views are restricted.

# 📗 Task 04: Index Creation and Analysis

🎯 **Objective**: Improve query performance through indexing and analyze its impact.

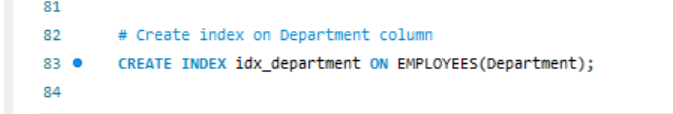
## Instructions:

1. Identify a column in the EMPLOYEES table that would benefit from indexing (e.g., Department or Status).

In the **EMPLOYEES** table, a good candidate is **Department**, because:

* Many queries filter employees by department (WHERE Department = …).
* This column is frequently used in **joins** with the DEPARTMENTS table.
* Indexing it will speed up lookups and joins.

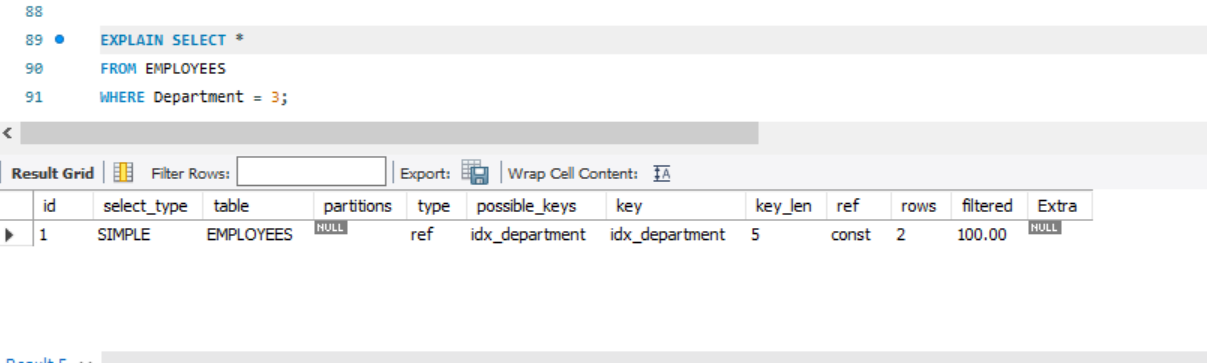
1. Create an index on the selected column.



1. Run a query before and after creating the index and **observe execution speed or behavior**.

**Before indexing:**  
In this case, the Department column was already indexed automatically because it is part of a foreign key constraint. Therefore, there was no state where the column existed without an index. If we had queried without an index, MySQL would have performed a full table scan (type = ALL in EXPLAIN).

**After indexing:**  
When running EXPLAIN on the query with WHERE Department = 3, the execution plan shows type = ref and key = idx\_department. This confirms that MySQL is using the index for filtering, which improves performance by avoiding a full table scan.



1. Document:

**Why that column was selected**:

The Department column is frequently used in queries for filtering employees by department and also appears in joins with the DEPARTMENTS table. Indexing this column improves lookup efficiency and reduces full table scans.

**What impact the index had (did query speed improve?)**

Before indexing, queries filtering by department required scanning the entire EMPLOYEES table (full table scan). After creating the index, the execution plan showed that MySQL used the index, resulting in faster access to rows that matched the department condition.

**Whether indexing all columns is a good idea — justify.**

Indexing all columns is **not recommended**. While indexes speed up read operations, they also consume extra storage and slow down write operations (INSERT, UPDATE, DELETE), since the index must be updated as well. Therefore, only columns that are frequently used in filtering, searching, or joining should be indexed.

# 📘 Task 05: Conceptual Reflection – Views vs. Indexes

🎯 **Objective**: Demonstrate theoretical understanding of core concepts.

## Instructions:

Answer the following questions in your lab report:

**1. What are the key differences between a view and an index?**

A **view** is a virtual table created from a query that provides a different way of looking at the data, while an **index** is a database object that improves the speed of searching and filtering. Views are mainly for abstraction, security, and simplifying queries, whereas indexes are for improving query performance. For example, in my lab, Active\_Employees was a view to show only active employees, but idx\_status was an index created to make filtering by status faster.

## How do views help in data abstraction and security?

Views hide unnecessary details and expose only the required data. They can be used to restrict access by showing only selected rows and columns instead of the full table. For instance, my Active\_Employees view only displayed employees who are active, preventing access to inactive ones, while Employee\_Department\_Details combined multiple tables for simplified access. This provides both abstraction and controlled access to data.

1. **How do indexes enhance performance, and what are their trade-offs?**

Indexes speed up queries by reducing the number of rows scanned. For example, filtering by Status or Department was much faster after creating an index because MySQL avoided full table scans. However, indexes also take up extra storage and slow down write operations (INSERT, UPDATE, DELETE), since every change to the data also updates the index.

1. **Can all views be updated? Why or why not? Give examples from your lab tasks.**

No, not all views are updatable. Simple views based on one table without joins or aggregates are usually updatable, but complex views (with joins, GROUP BY, aggregates, DISTINCT) are not. In my lab, I was able to update and delete records through Active\_Employees (a simple view), but I could not delete from Employee\_Department\_Details (a join view), because MySQL does not allow updates/deletes on non-updatable views.

1. **Why might a database administrator avoid overusing indexes?**

While indexes improve read performance, too many indexes can hurt overall performance. Each index uses storage space and must be updated whenever data changes, which slows down insert, update, and delete operations. A database administrator will carefully choose which columns to index, usually those used often in filtering or joining, rather than indexing everything.

### **Reflection on Challenges and Learning:**

While completing this lab, I faced several challenges such as dealing with errors in MySQL syntax, safe update mode restrictions, and understanding why some operations (like deleting from a join view) were not allowed. I initially struggled with updating and deleting through views because MySQL required the use of primary key columns and safe update mode blocked non-key updates. Another challenge was testing indexes on the Department column, since it was already indexed automatically due to the foreign key constraint. To overcome this, I practiced with the Status column to clearly observe the effect of indexing. Through these tasks, I learned the practical differences between views and indexes, how views simplify queries and enforce restrictions, and how indexes improve performance but also come with trade-offs. Overall, this lab improved my understanding of **data abstraction, security, performance tuning, and MySQL limitations in practice**.