# Faculty of Computing

**CS220: Database Systems**

**Class: BESE-13A**

# Lab 11: Views in MySQL

# Date: December 11, 2023

# Time:02:00-05:00

# Instructor: Dr. Shah Khalid

# Lab Engineer: Sundas Dawood

**Name:** Aimen Munawar

**Class:** BESE-13-A

**CMS ID:** 415867

**Lab 11: Views in MySQL**

**Introduction**

This lab will focus on the types of database objects other than tables. In this lab you will practice defining, modifying and using views.

**Objectives**

After completing this lab, you should be able to do the following:

* Create a view
* Alter/Update a view
* Alter a table

**Tools/Software Requirement**

* MySQL Community Server 5.6
* MySQL Workbench 6.1

**Helping Material**

<https://www.mysqltutorial.org/mysql-view-with-check-option/>

**Description**

Views (including updatable views) are stored queries that when invoked produce a result set. A view acts as a virtual table.

**CREATE VIEW/ ALTER VIEW**

**Syntax**

The syntax for the CREATE VIEW statement in MySQL is:

**CREATE [OR REPLACE] VIEW view\_name AS**

**SELECT columns**

**FROM tables**

**WHERE conditions**

**[WITH [CASCADED | LOCAL] CHECK OPTION];**

OR REPLACE is optional. If you do not specify this clause and the VIEW already exists, the CREATE VIEW statement will return an error.

The CREATE VIEW statement creates a new view, or replaces an existing view if the OR REPLACE clause is given. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW is the same as ALTER VIEW.

The ALTER VIEW statement does the same thing as CREATE OR REPLACE.

The full ALTER statement looks like this:

ALTER [<algorithm attributes>] VIEW [<database>.]< name> [(<columns>)] AS

<SELECT statement> [<check options>]

The select\_statement is a SELECT statement that provides the definition of the view. (Selecting from the view selects, in effect, using the SELECT statement.) The select\_statement can select from base tables or other views. The view definition is “frozen” at creation time. Changes to the underlying tables afterward do not affect the view definition. For example, if a view is defined as SELECT \* on a table, new columns added to the table later do not become part of the view.

The CREATE VIEW statement requires the CREATE VIEW privilege for the view, and some privilege for each column selected by the SELECT statement. For columns used elsewhere in the SELECT statement, you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view.

**Updatable and Insertable Views**

Some views are updatable and references to them can be used to specify tables to be updated in data change statements. That is, you can use them in statements such as [UPDATE](http://dev.mysql.com/doc/refman/5.7/en/update.html), [DELETE](http://dev.mysql.com/doc/refman/5.7/en/delete.html), or [INSERT](http://dev.mysql.com/doc/refman/5.7/en/insert.html) to update the contents of the underlying table. Derived tables can also be specified in multiple-table [UPDATE](http://dev.mysql.com/doc/refman/5.7/en/update.html) and [DELETE](http://dev.mysql.com/doc/refman/5.7/en/delete.html) statements, but can only be used for reading data to specify rows to be updated or deleted. Generally, the view references must be updatable, meaning that they may be merged and not materialized. Composite views have more complex rules.

For a view to be updatable there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

* Aggregate functions ([SUM()](http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html#function_sum), [MIN()](http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html#function_min), [MAX()](http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html#function_max), [COUNT()](http://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html#function_count), and so forth)
* DISTINCT
* GROUP BY
* HAVING
* [UNION](http://dev.mysql.com/doc/refman/5.7/en/union.html) or [UNION ALL](http://dev.mysql.com/doc/refman/5.7/en/union.html)
* Subquery in the select list (fails for [INSERT](http://dev.mysql.com/doc/refman/5.7/en/insert.html), okay for [UPDATE](http://dev.mysql.com/doc/refman/5.7/en/update.html), [DELETE](http://dev.mysql.com/doc/refman/5.7/en/delete.html))
* Certain joins (see additional join discussion later in this section)
* Reference to nonupdatable view in the FROM clause
* Subquery in the WHERE clause that refers to a table in the FROM clause
* Refers only to literal values (in this case, there is no underlying table to update)
* Multiple references to any column of a base table (fails for [INSERT](http://dev.mysql.com/doc/refman/5.7/en/insert.html), okay for [UPDATE](http://dev.mysql.com/doc/refman/5.7/en/update.html), [DELETE](http://dev.mysql.com/doc/refman/5.7/en/delete.html))

**The View WITH CHECK OPTION Clause**

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts to rows for which the WHERE clause in the select\_statement is not true. It also prevents updates to rows for which the WHERE clause is true but the update would cause it to be not true (in other words, it prevents visible rows from being updated to nonvisible rows).

**Lab Tasks**

Given the following **database schema**:

**Student (snum: integer, sname: char(30), major: char(25), level: char(2))**

**Faculty (fid: integer, fname: char(30), deptid: integer)**

**Class (cname: char(40), meets\_at: char(20), room: char(10), fid: integer | fid REFS Faculty.fid)**

**Enrolled (snum: integer, cname: char(40) | snum REFS student.snum, cname REFS class.name)**

1. Create a view named v1 which has the name of faculty members who do not teach any course.

**Code:**

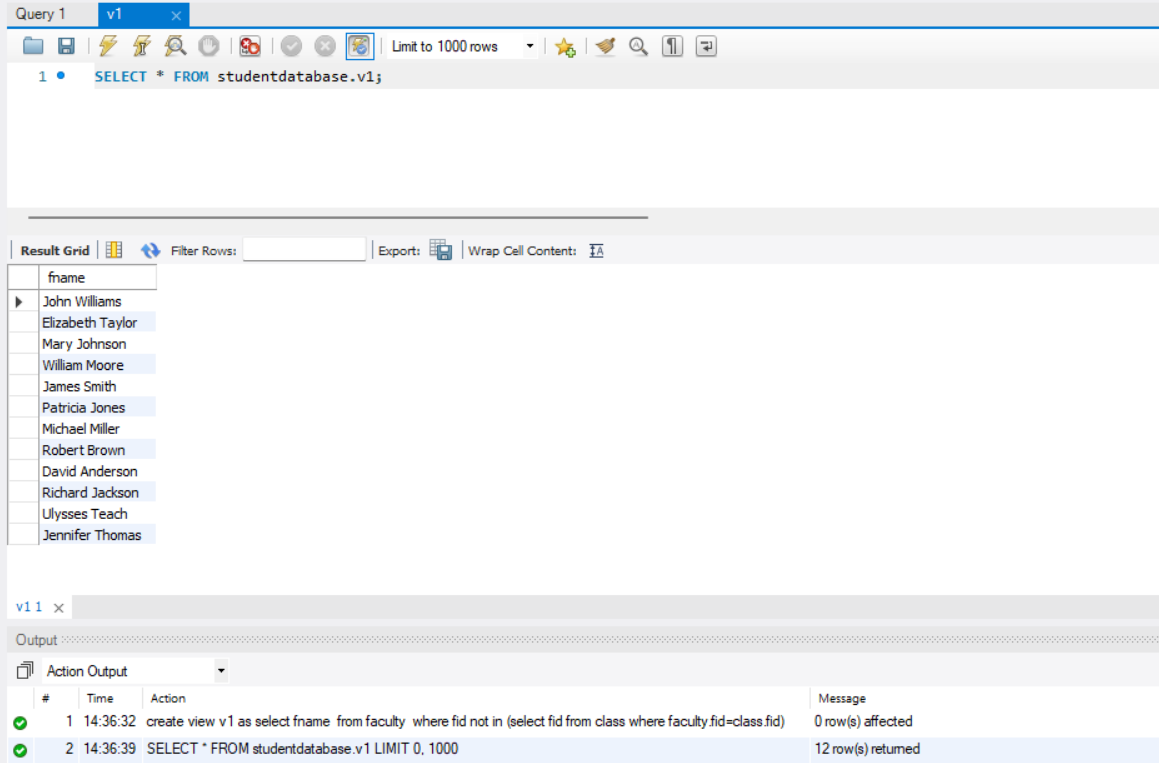
create view v1 as

select fname

from faculty

where fid not in (select fid from class where faculty.fid=class.fid);

**Output:**

****

1. Create another view named v2 which has the names of students who are enrolled in a course taught by faculty member “Ivana Teach”.

**Code:**

create view v2 as

select s.snum, s.sname

from Student s

where exists (

select 1

from Enrolled e1

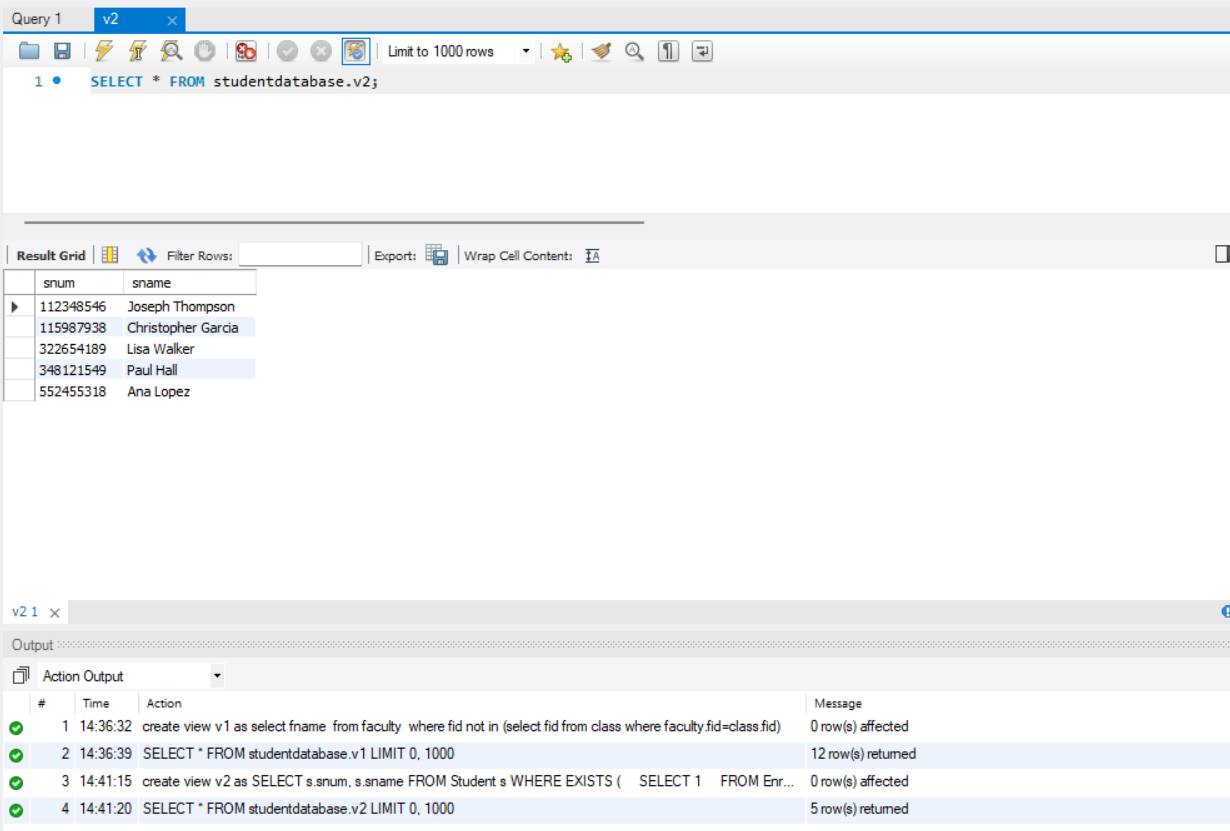
join Class c1 on e1.cname = c1.cname

join Faculty f1 on c1.fid = f1.fid

where s.snum = e1.snum and f1.fname = 'Ivana Teach'

);

**Output:**

****

1. Create a view stdVu that is based on the student relation.

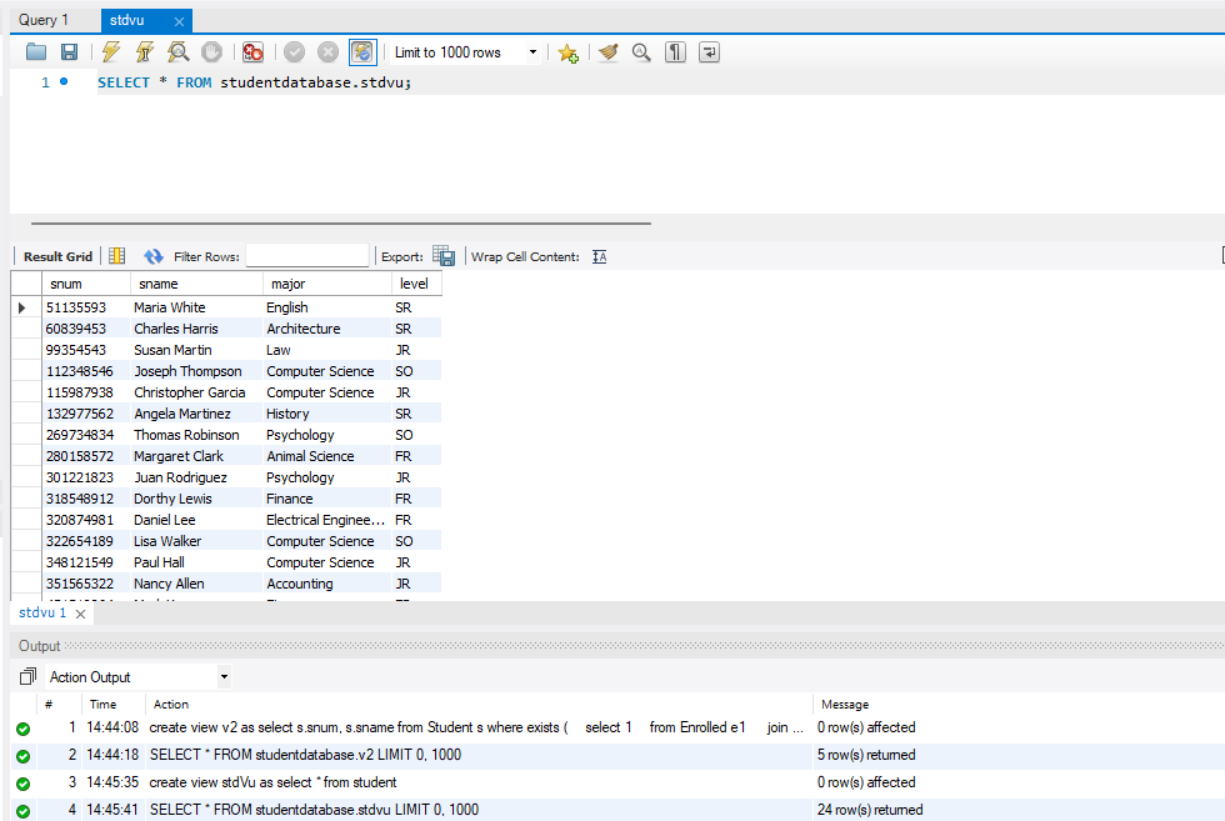
**Code:**

create view stdVu as

select \*

from student;

**Output:**

****

1. Alter the definition of student table. Add a column course to the student’s relation.

**Code:**

alter table student

add column course varchar(200);

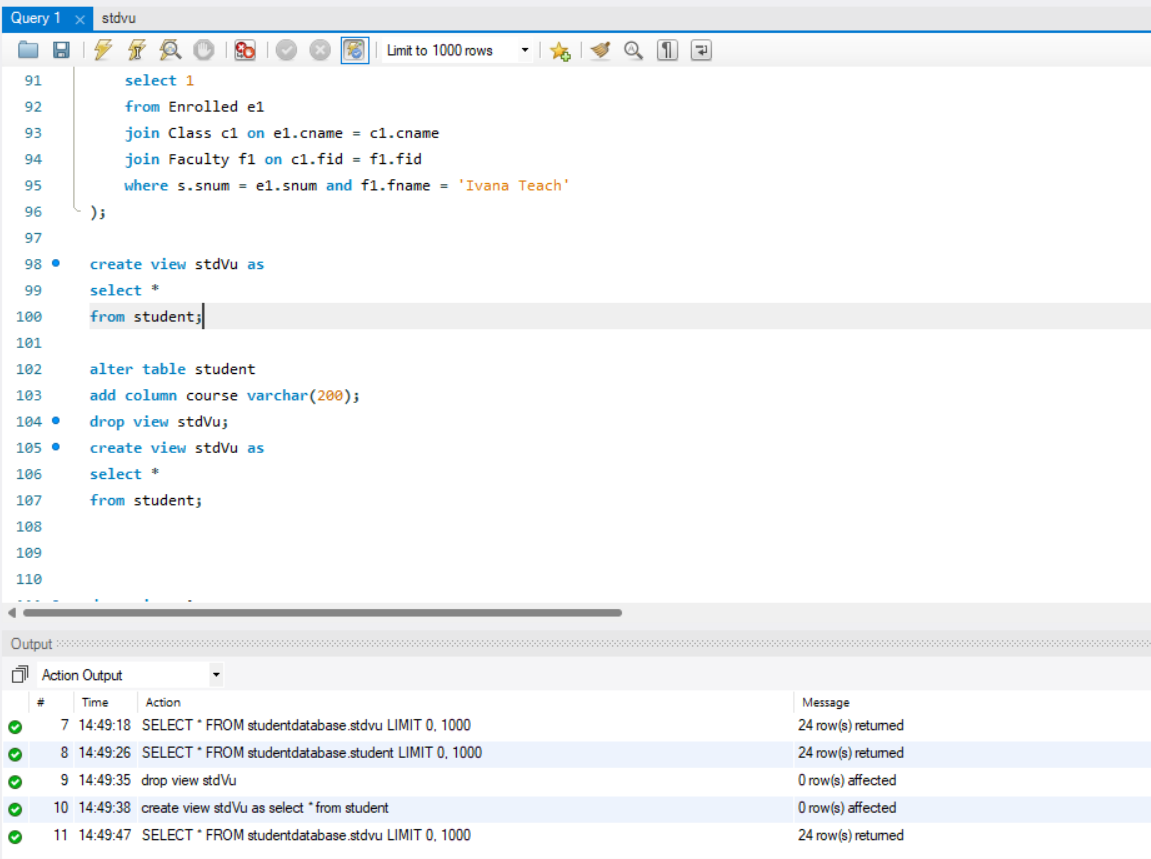
drop view stdVu;

create view stdVu as

select \*

from student;

**Output:**



A screenshot of a computer

Description automatically generated

1. Notice the changes in views which are based on student relation. Comment what happens to view data if the base table is modified.

**When the structure of the underlying `Student` table is modified, views based on that table do not automatically reflect those changes. Therefore, to ensure that the view accurately represents the updated table structure, it is necessary to drop and recreate the view. In Task 4, the `stdVu` view is dropped and recreated after adding a new column (“course”) to the “Student” table, ensuring that the view aligns with the modified table structure and includes the latest changes.**

1. Alter view v2 based on the definition: It has the names of all juniors (Level = JR) who are enrolled in a class taught by ‘Ivana Teach’.

**Code:**

alter view v2 as

select s.snum, s.sname

from Student s

where s.level = 'JR'

and exists (

select 1

from Enrolled e1

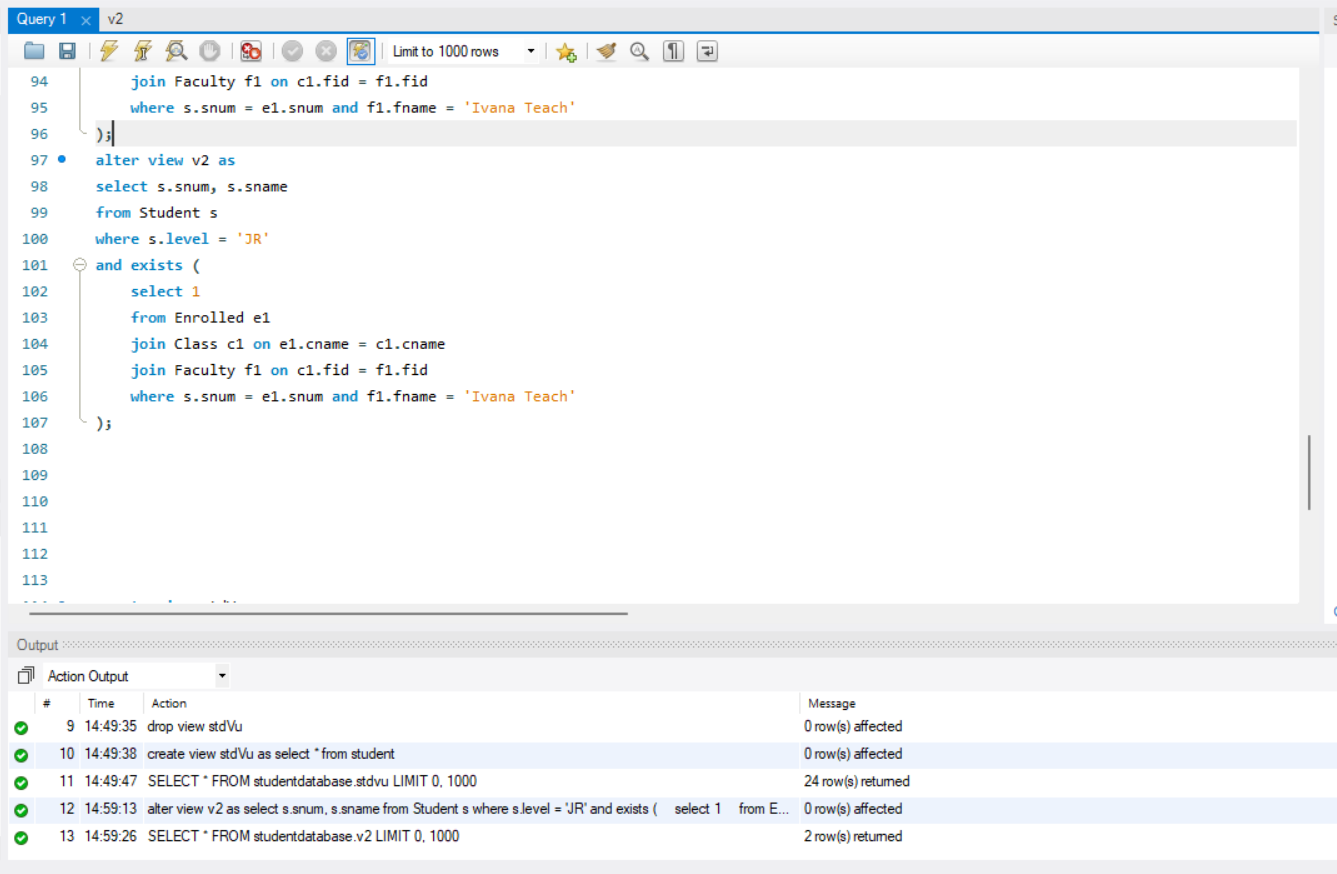
join Class c1 on e1.cname = c1.cname

join Faculty f1 on c1.fid = f1.fid

where s.snum = e1.snum and f1.fname = 'Ivana Teach'

);

**Output:**



A screenshot of a computer

Description automatically generated

1. Create views based on the following queries also:
   1. The names of students majoring in ‘Computer Science’.

**Code:**

create view v7 as

select \*

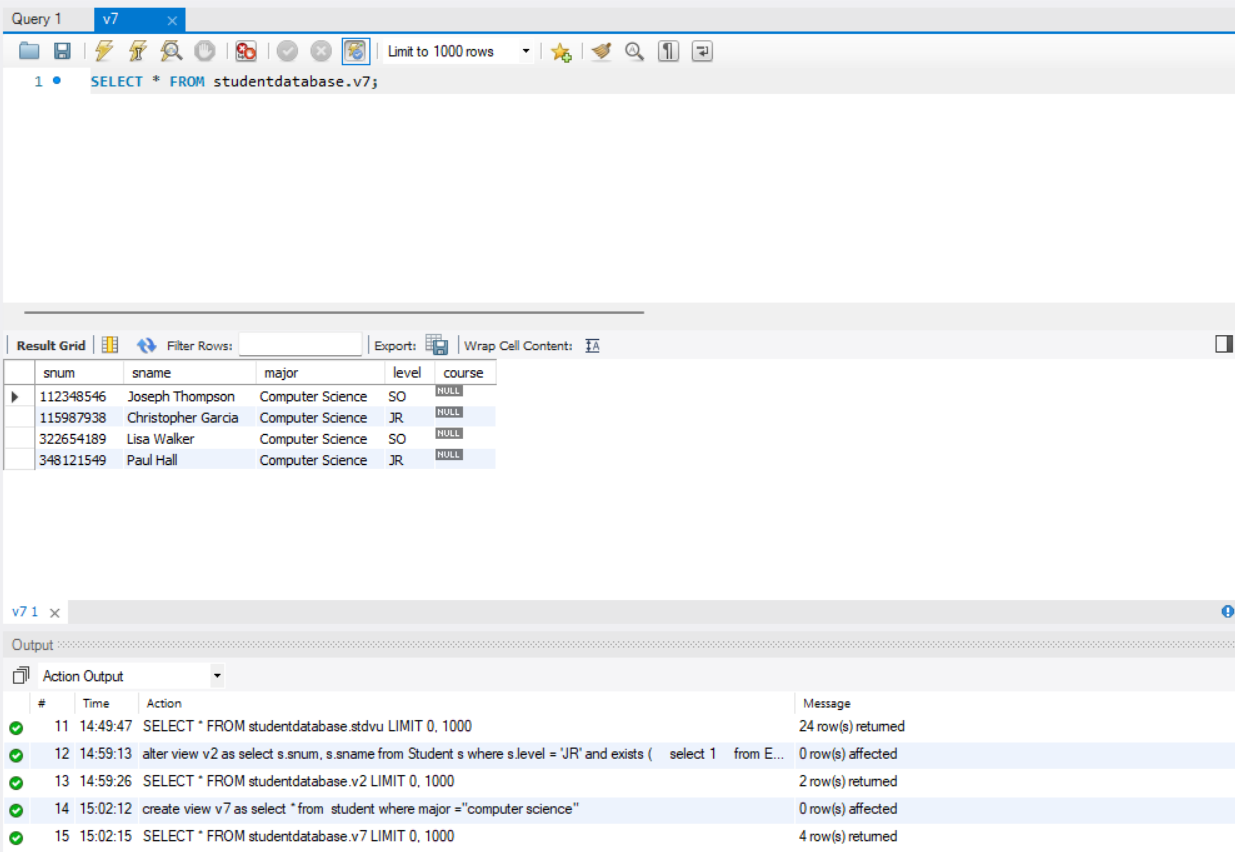
from student

where major ="computer science";

**Output:**

**A screenshot of a computer

Description automatically generated**

****

* 1. The names of classes taught by ‘John Williams’ in dept # 68.

**Code:**

create view v7b as

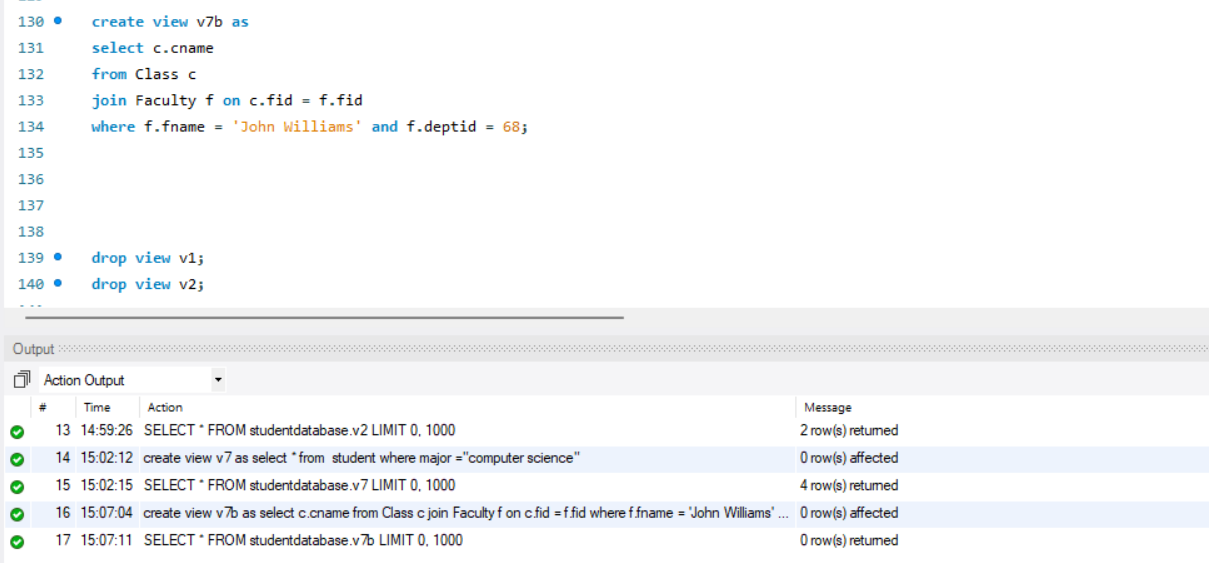
select c.cname

from Class c

join Faculty f on c.fid = f.fid

where f.fname = 'John Williams' and f.deptid = 68;

**Output:**

****

**A screenshot of a computer

Description automatically generated**

* 1. The distinct student level in ‘Database Systems’ class in descending order.

**Code:**

create view v7c as

select distinct s.level

from Student s

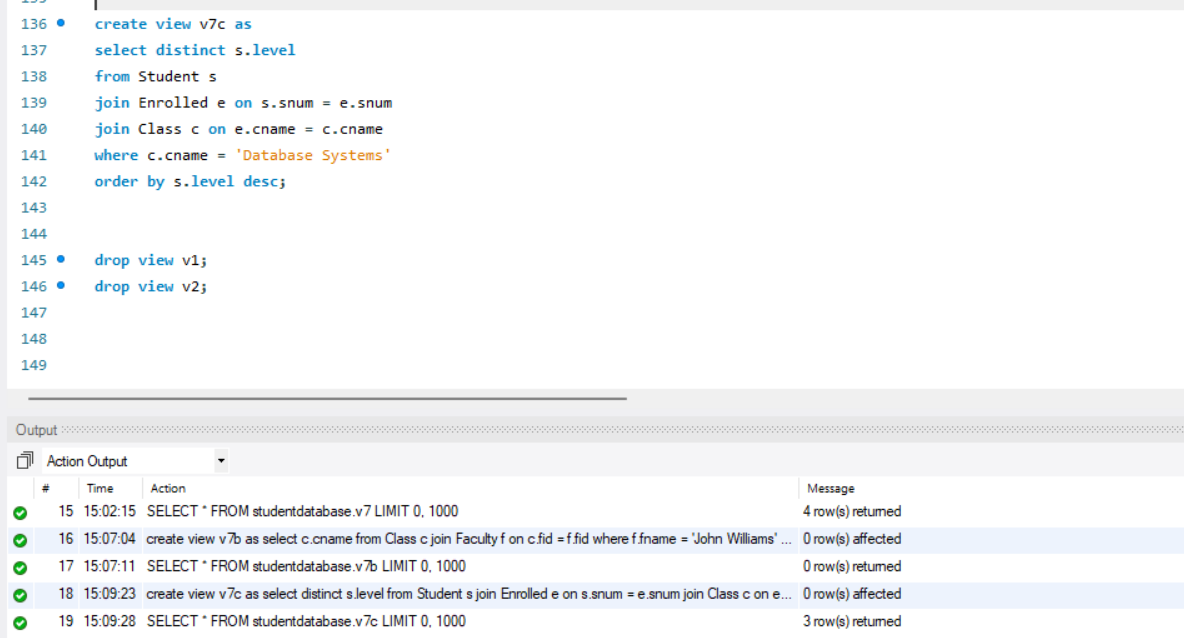
join Enrolled e on s.snum = e.snum

join Class c on e.cname = c.cname

where c.cname = 'Database Systems'

order by s.level desc;

**Output:**



A screenshot of a computer

Description automatically generated

* 1. The name of ‘Christopher Garcia’s teachers.

**Code:**

create view v7d as

select distinct f.fname

from Faculty f

join Class c on f.fid = c.fid

join Enrolled e on c.cname = e.cname

join Student s on e.snum = s.snum

where s.sname = 'Christopher Garcia';

**Output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

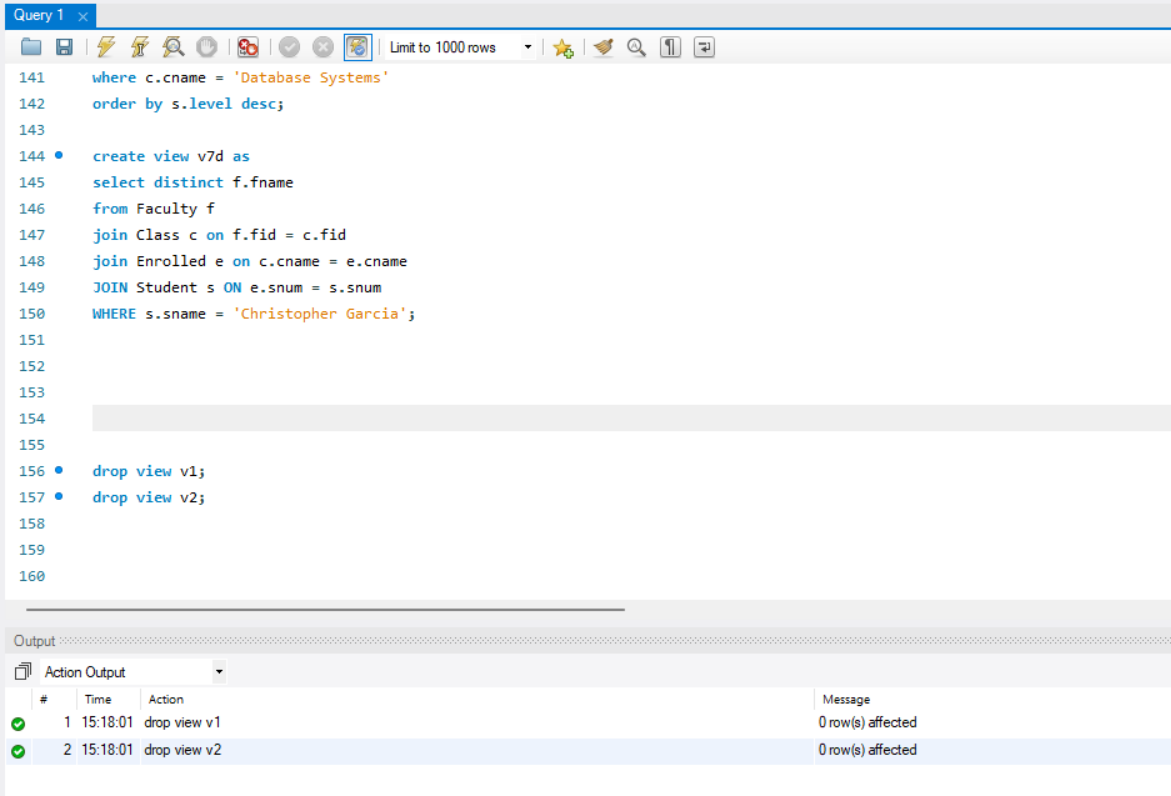
1. Drop view v1, v2.

**Code:**

drop view v1;

drop view v2;

**Output:**

****

# Deliverables

Complete your lab tasks in SQL workbench and submit a word file with queries and the screenshots of the results to all the questions attempted. Upload it on LMS. Late submissions will not be accepted.