2018380039\_Dikshya Kafle Lab report DBMS

**Name: Dikshya Kafle**

**Student Number: 2018380039**

**Experiment 4:**

**View and Index :**

**Goal**

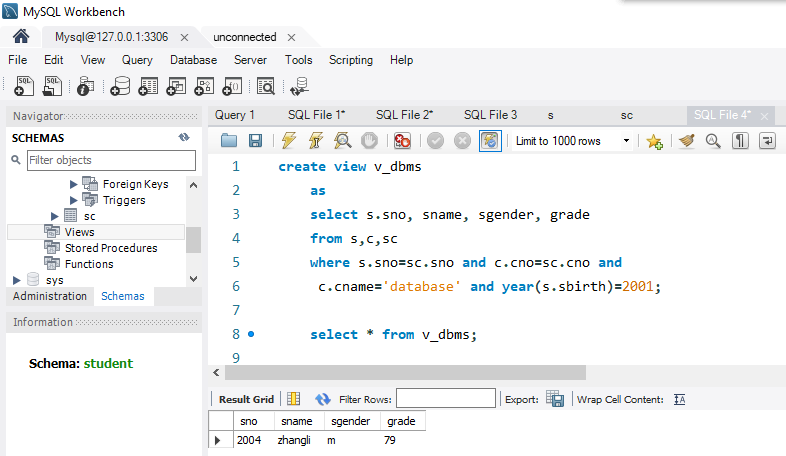
1. Be familiar with the use of GUI and SQL language to create, update and delete views.

2. Proficient in creating and deleting index using GUI and SQL language.

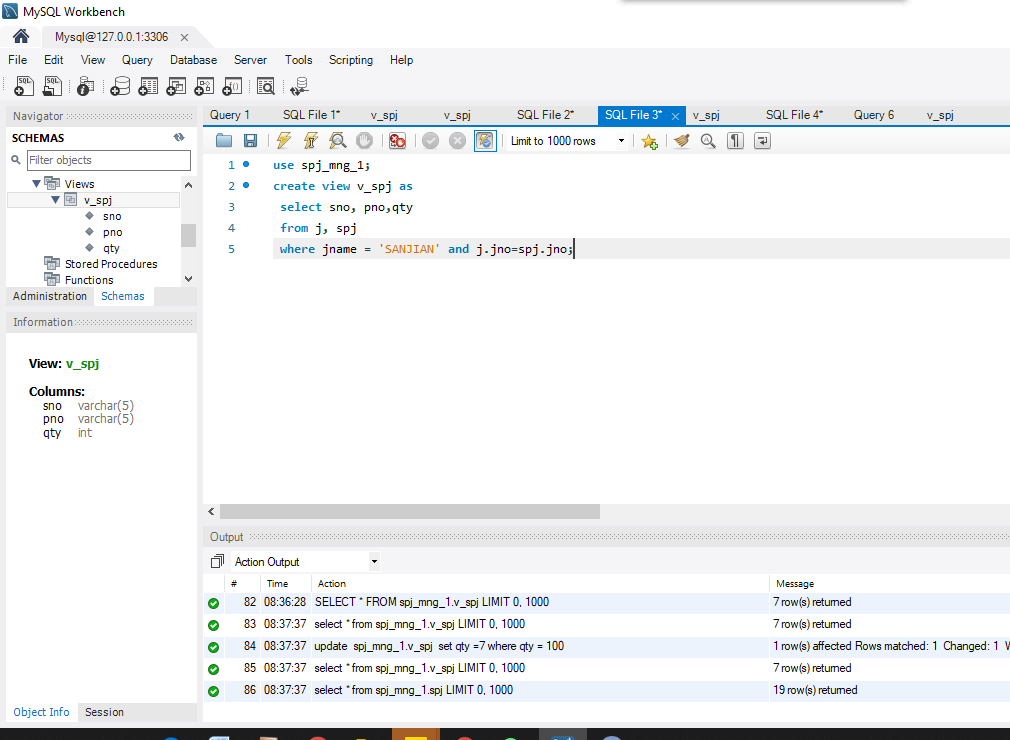
3. Understand and verify the role of index.

**Content**

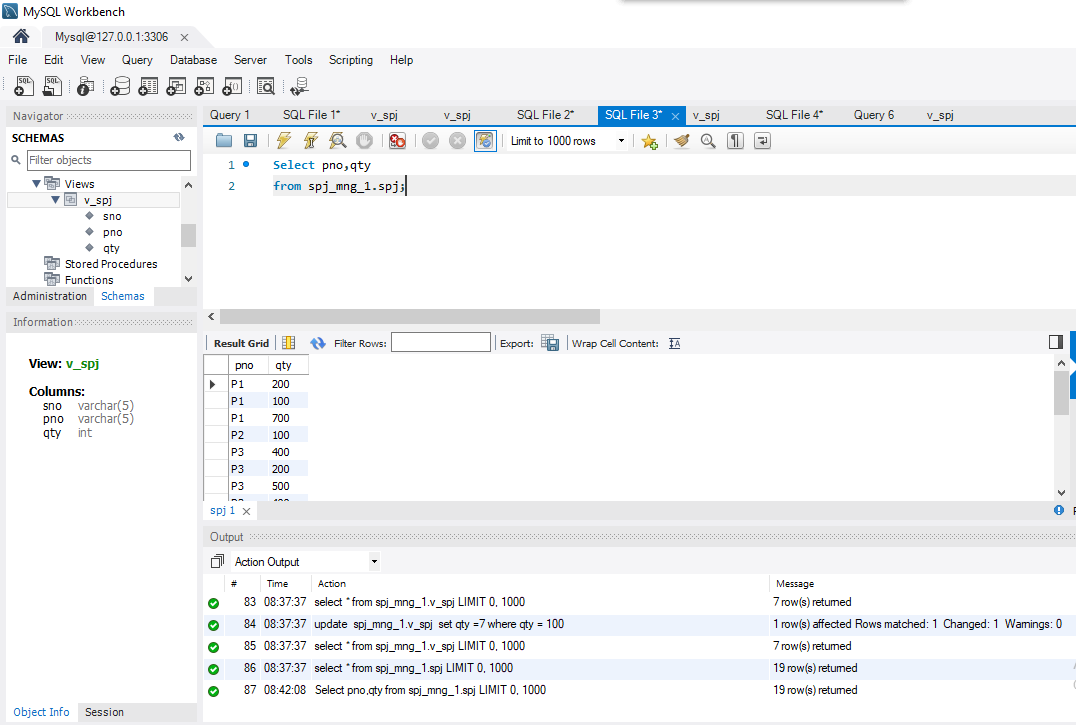
1. In the student database, use SQL statement to create a view of students who have taken the database course and are born in 2001. The view includes the information of student number, name, gender and grade.



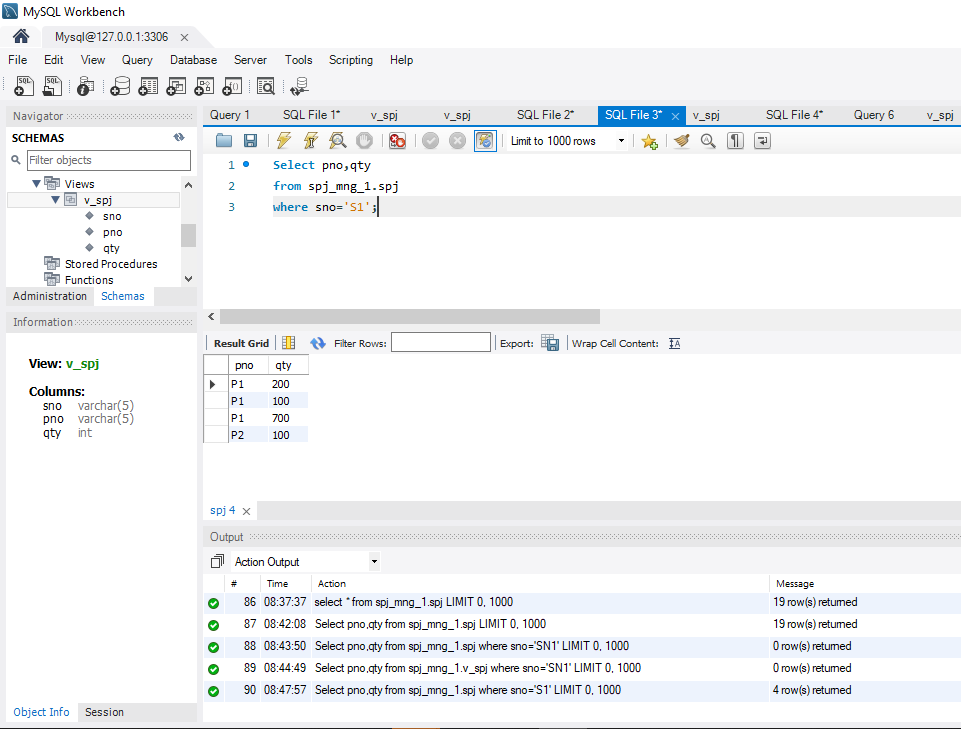
1. Create a view for the supply situation of “SANJIAN”project, including the attributes of supplier code (SNO), part code (PNO) and supply quantity (QTY), with two ways with SQL statement.(view name:V\_SPJ)



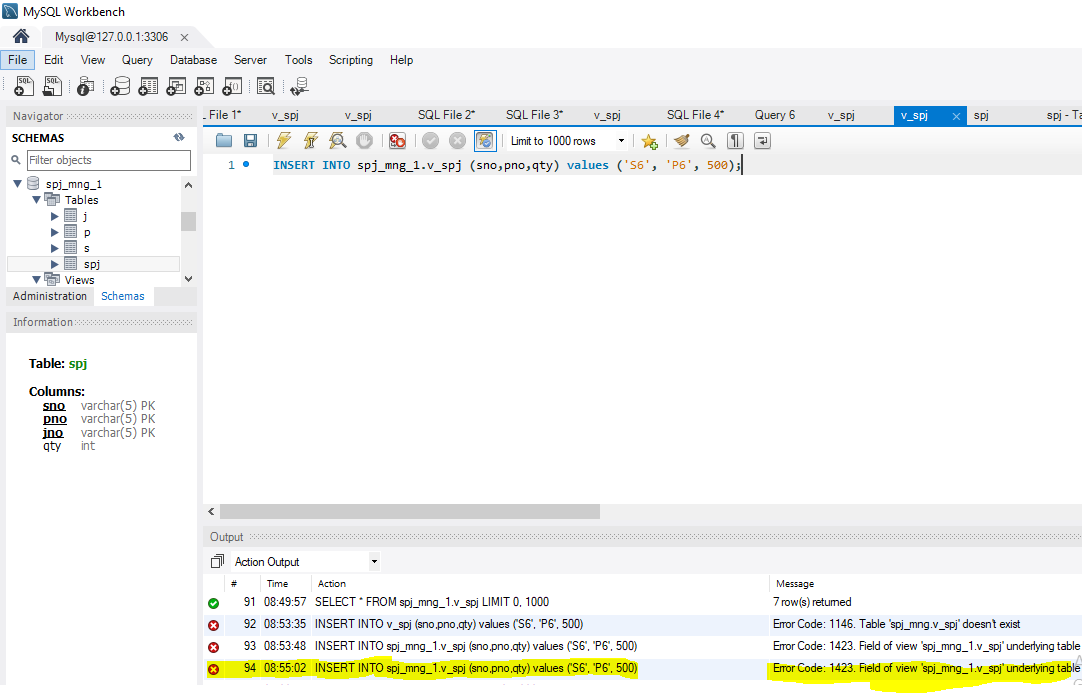
1. Complete the following view query with SQL statement.
2. Find out all the parts code and their quantity used by “SANJIAN” project



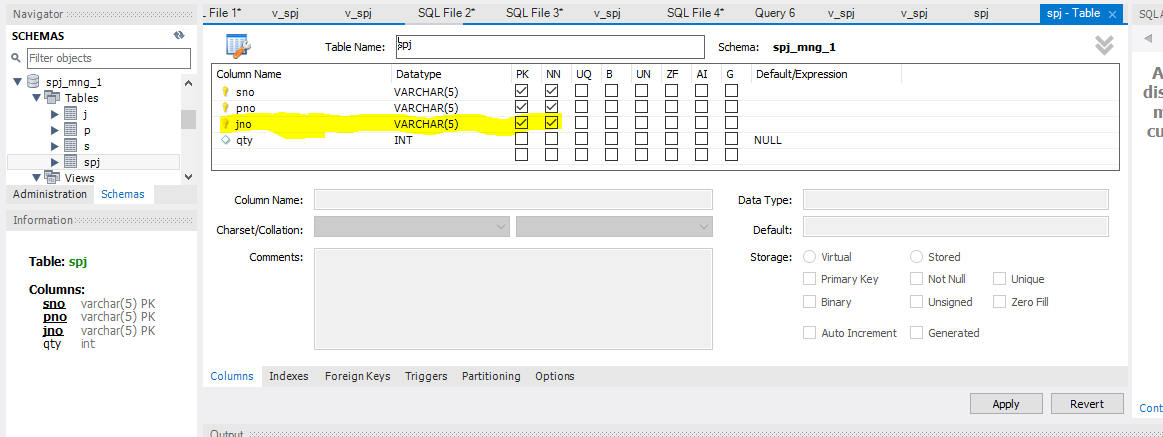
1. Find out the supply situation of supplier S1.



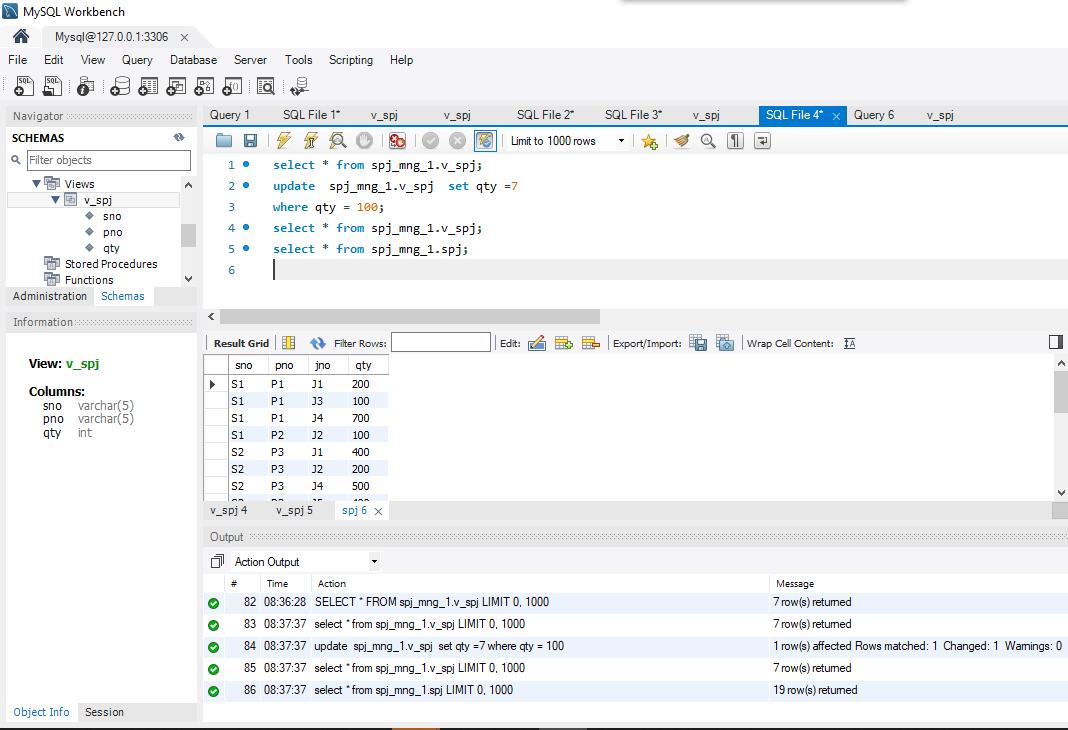
1. Update the data of views with SQL statement.
   * 1. Insert a tuple into the view V\_SPJ.

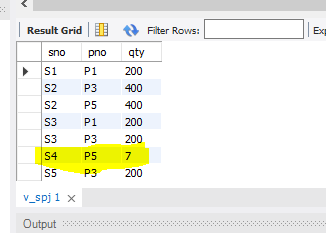


**JNO is set to Not Null**



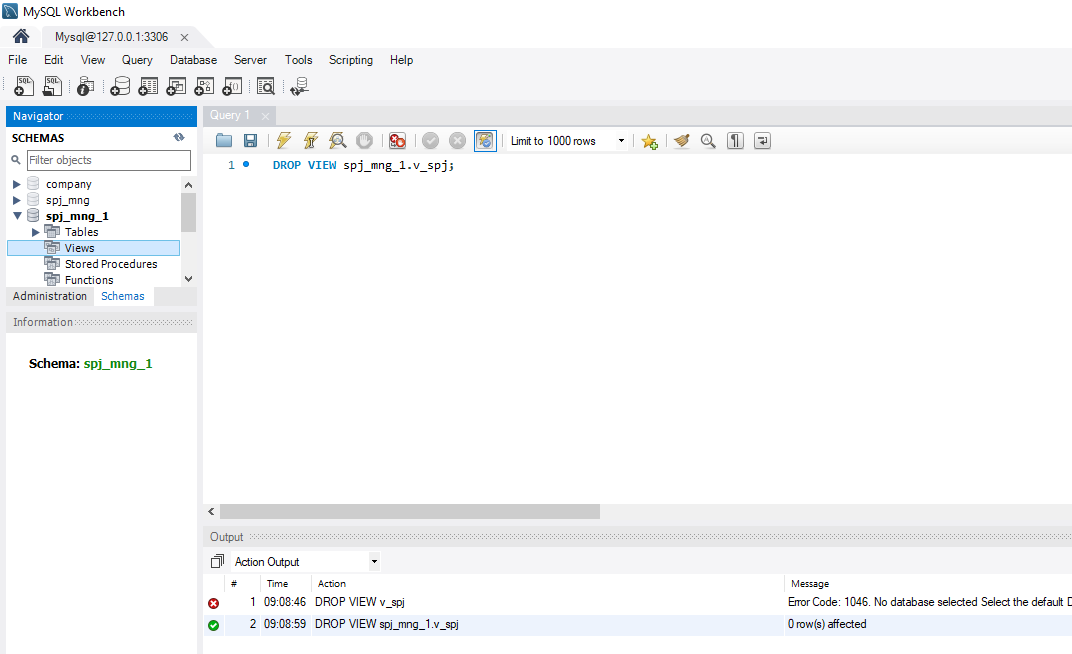
* + 1. Modify the quantity value of any tuple in the view V\_SPJ.



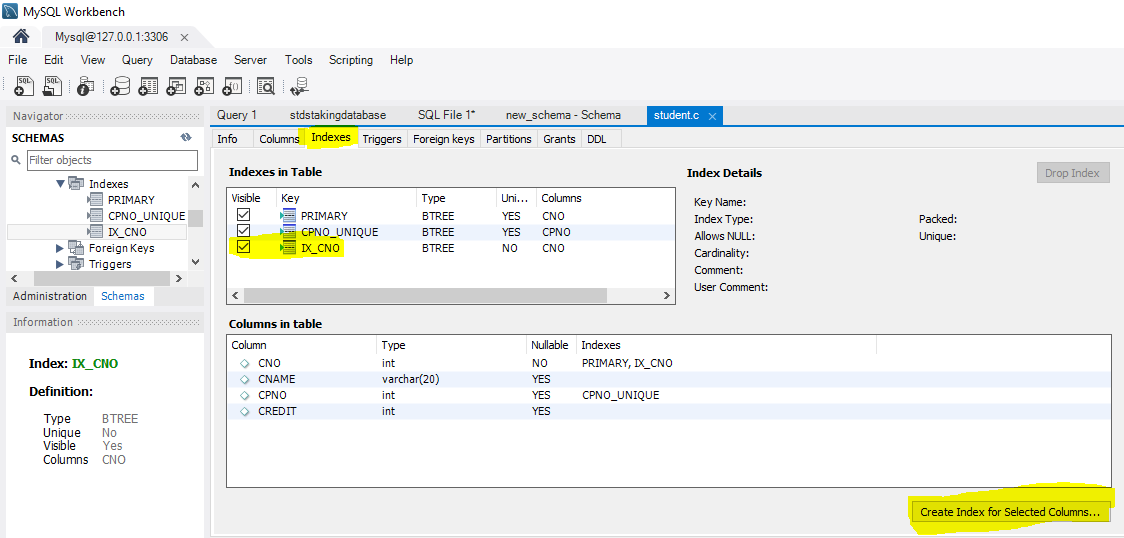


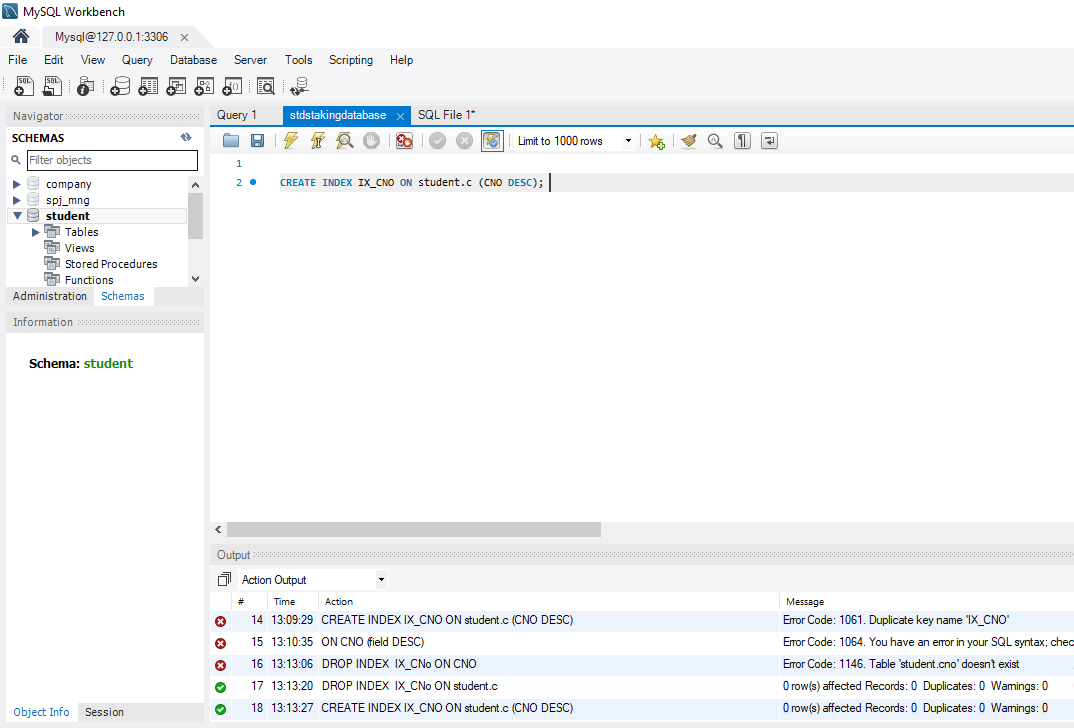
* + 1. Delete one tuple from the view V\_SPJ

Hint: Only when the created view can be resoluted, it can be deleted normally, otherwise it will fail to be deleted.

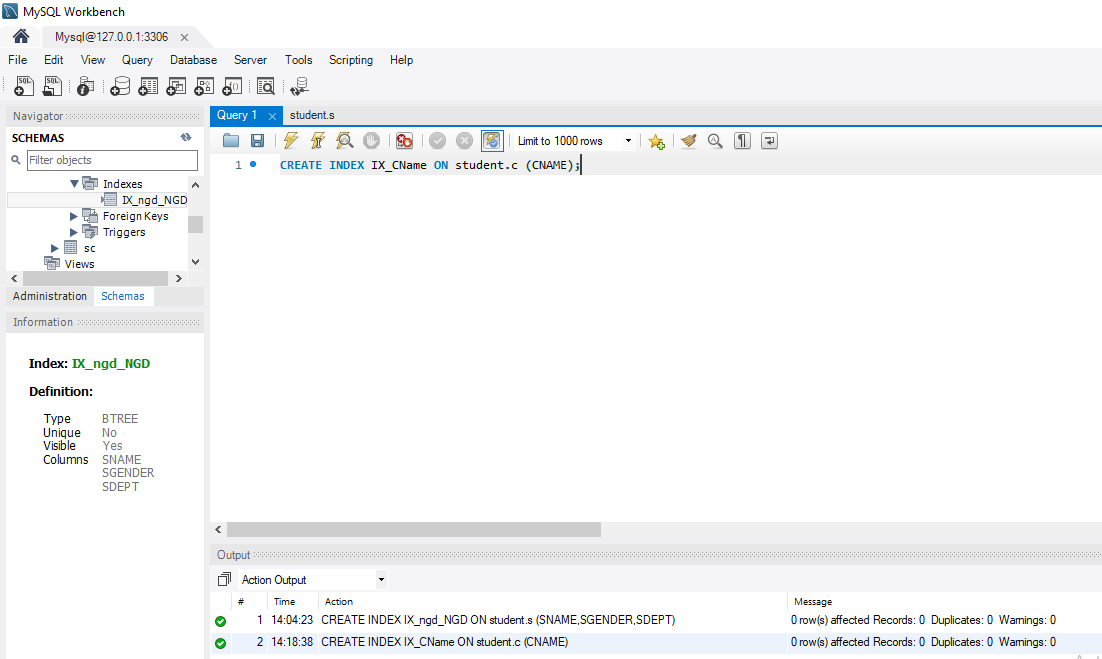


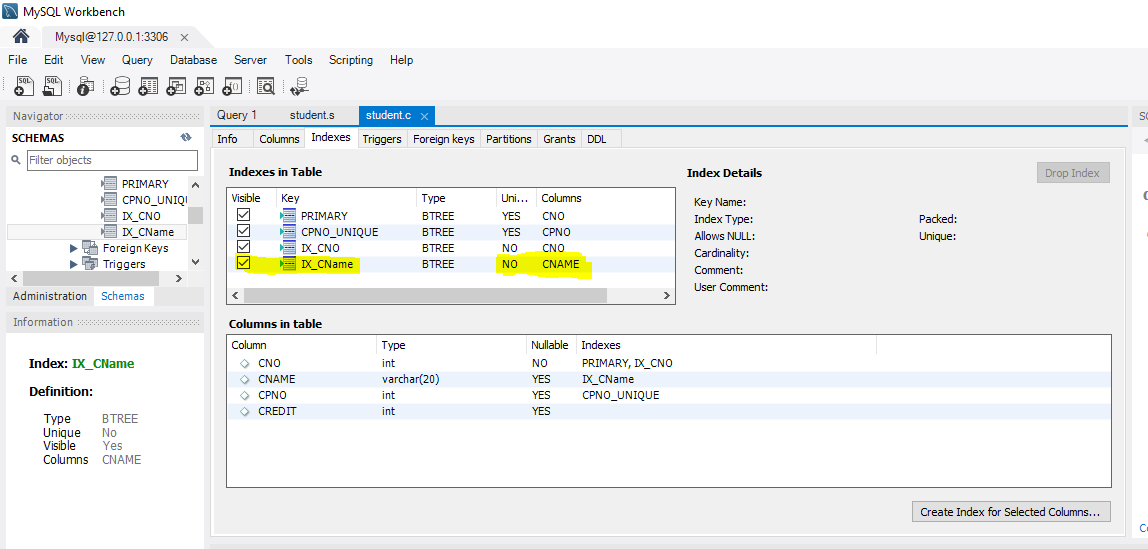
1. Create a descending index named IX\_CNo for the CNO attribute of C table in student database by using GUI.



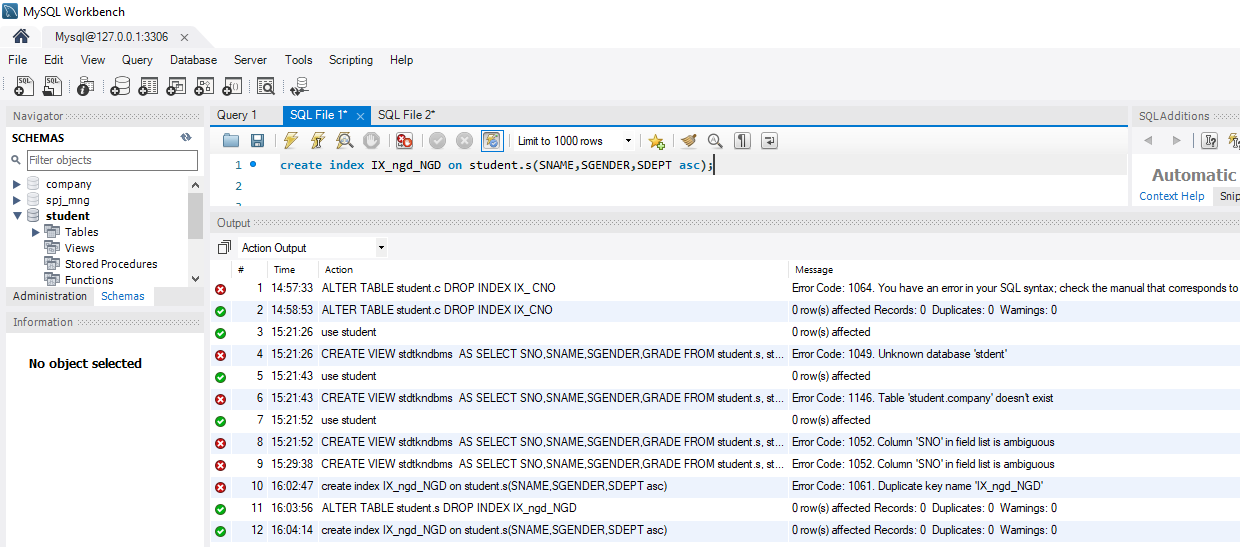


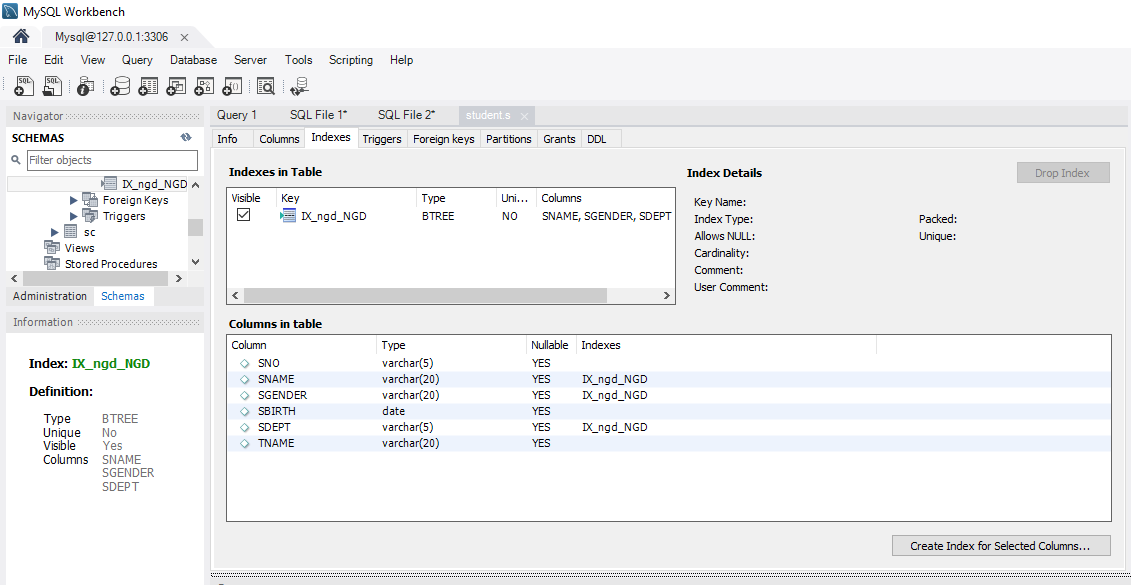
1. Use SQL statement to complete the following index operation on student database.
2. Create a non-unique index named IX\_CName on the CNAME attribute of table C.



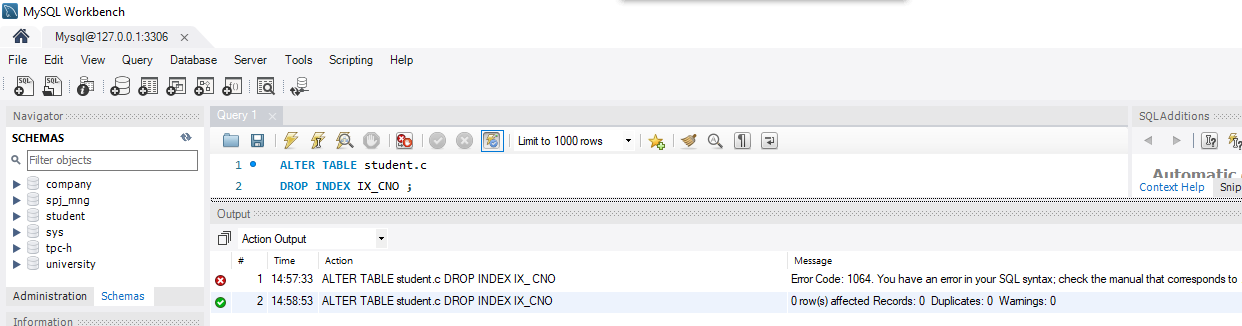


(2) Create a composite index named IX\_ngd\_NGD on the table S, which is an ascending index for sname, sgender and sdept attribute sets.

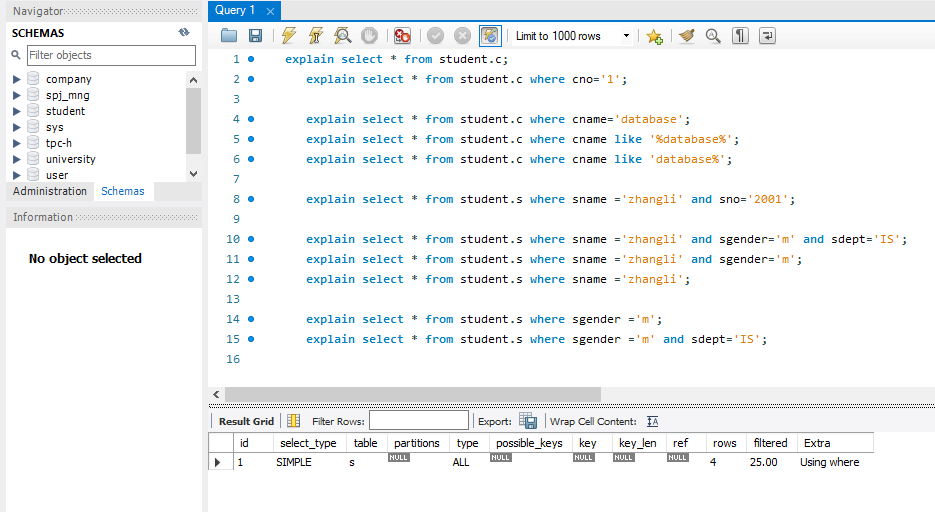


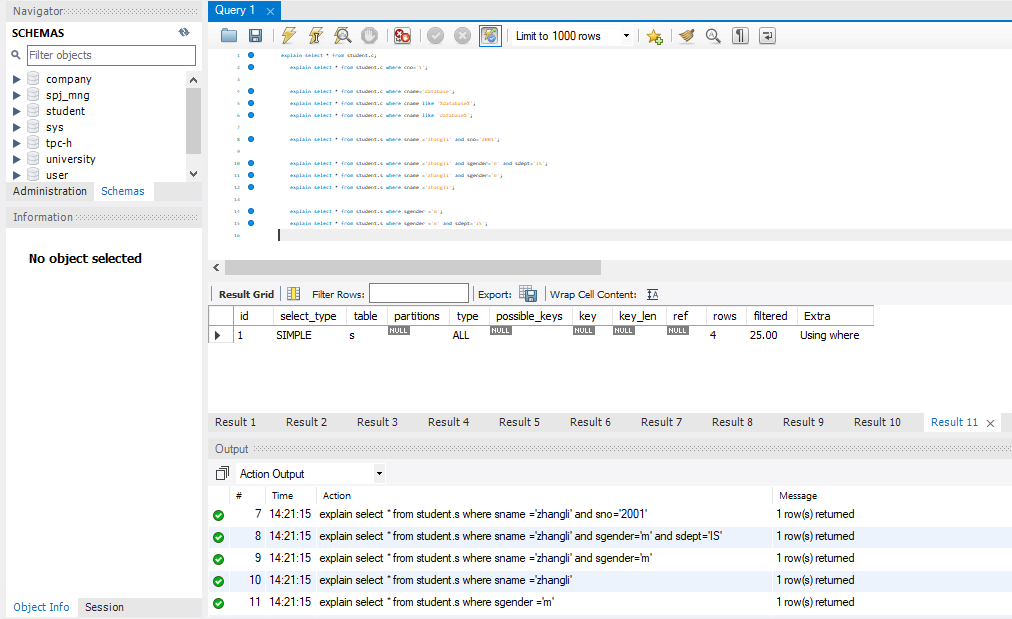


1. Delete index IX\_ CNO of table C。



(4) Based on the above indexes (table C: primary key index of CNO, general index of CNAME; table S: primary key index of SnO, IX\_ Nga composite index), use explain statement to obtain the query plan of each query statement, to observe the index usage in each query statement.

* + 1. explain select \* from c;
    2. explain select \* from c where cno = ‘1’;
    3. explain select \* from c where cname=’database’ ;
    4. explain select \* from c where cname like ‘%database%’;
    5. explain select \* from c where cname like ‘database%’;
    6. explain select \* from s where sname ='Zhangli' and sno='2001';
    7. explain select \* from s where sname ='Zhangli' and sgender='male' and sdept='IS';
    8. explain select \* from s where sname ='Zhangli' and sgender='male';
    9. explain select \* from s where sname ='Zhangli';
    10. explain select \* from s where sgender ='male';
    11. explain select \* from s where sgender ='male' and sdept='IS';
    12. 



1. Suppose there is a basic table userinfo as follows, design an experiment to verify the effect of index on database query efficiency.

create table userinfo

(

user\_id int primary key, //USER ID

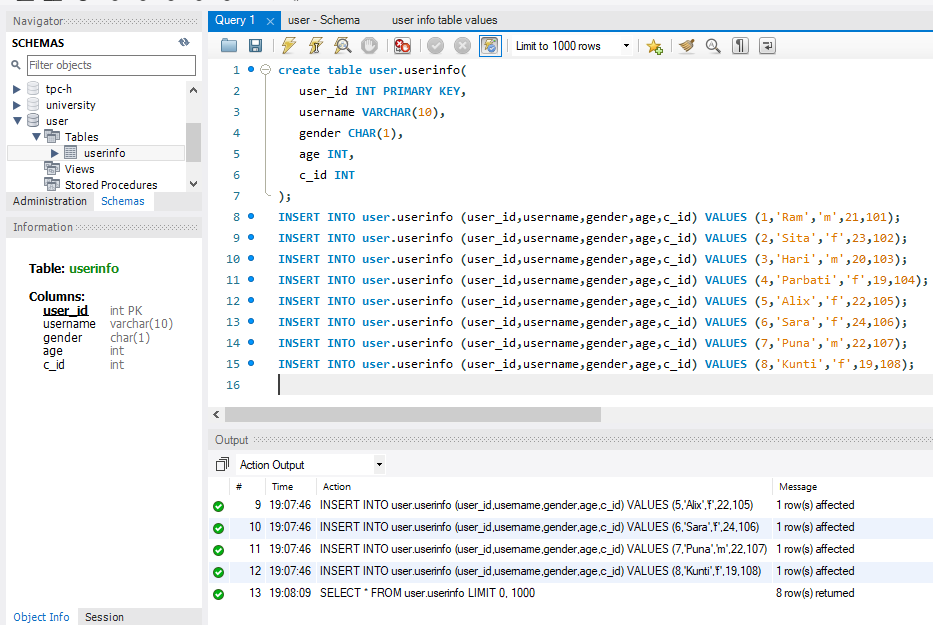
username varchar(10), //USERNAME

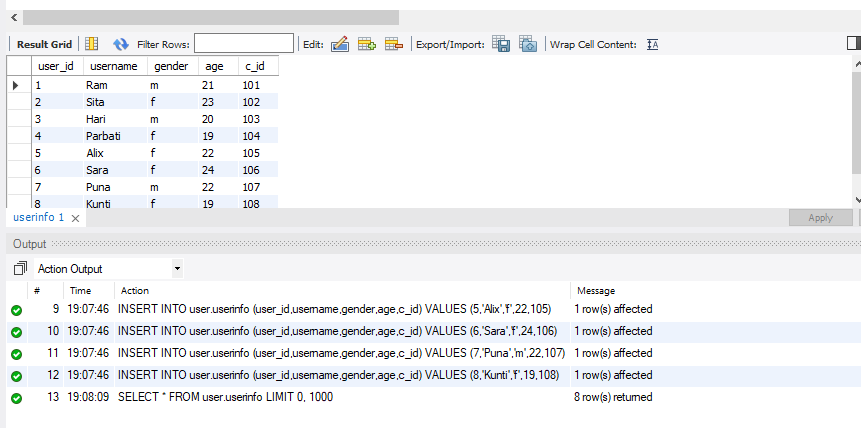
gender char(1), //GENDER

age int, //AGE

c\_id int //NO OF COLLEGE

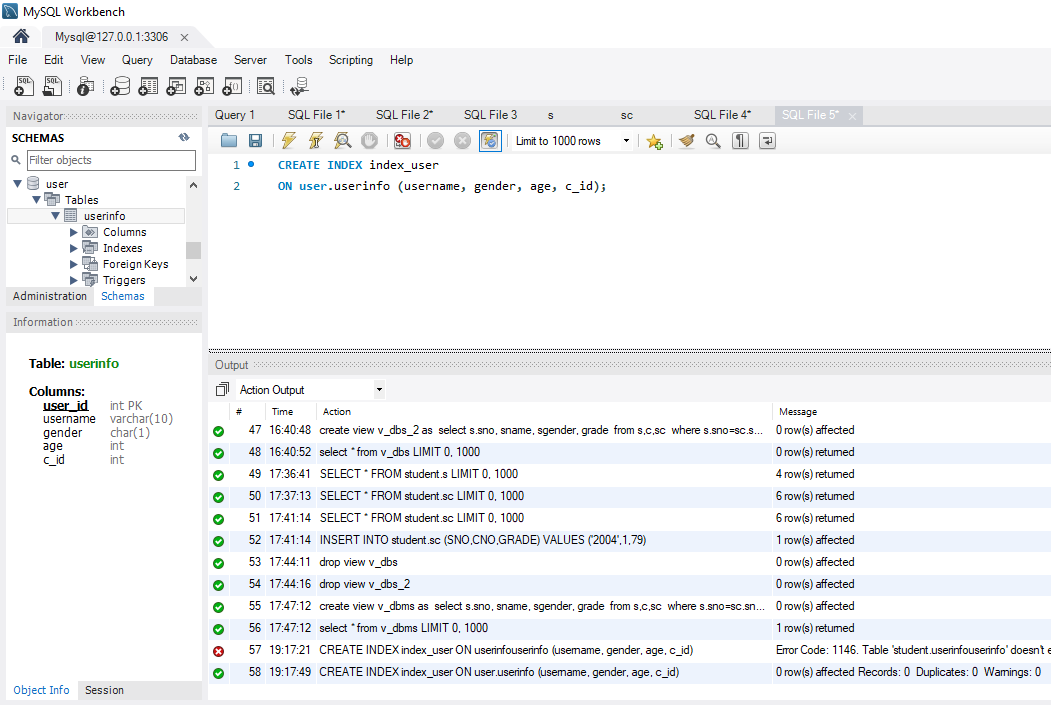
)



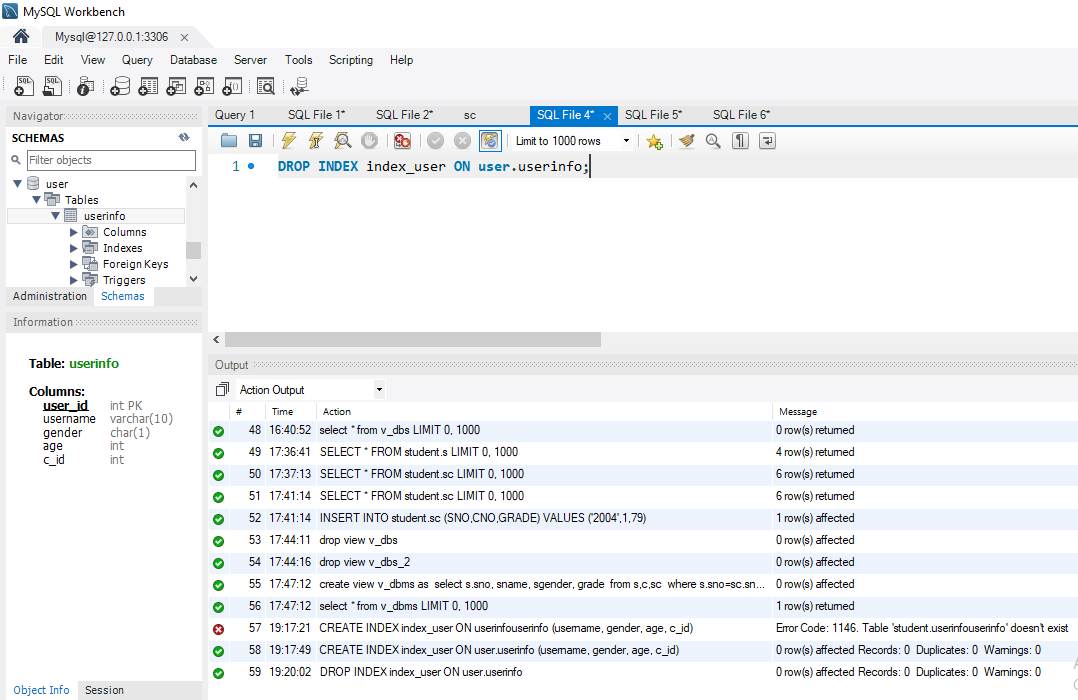


* + - 1. Verify the efficiency difference between indexed and non\_indexed queries.

**Indexed**

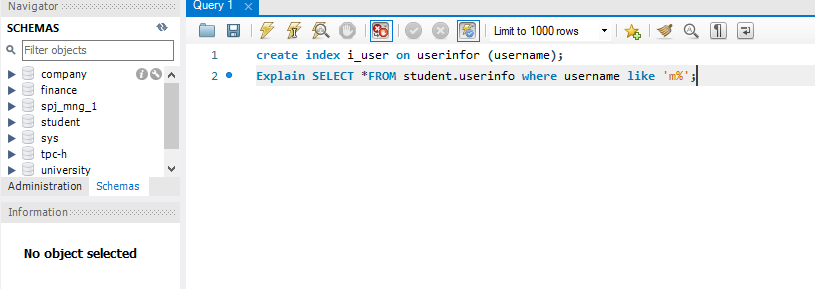


Drop index:











You can see the query execution efficiency that it searched for only 2 rows as shown in the figure after index formation, but before that it searched to fit the values in the table.

When we run the query, we check the CPU time (time spent on the query by CPU):

Note that elapsed time does not necessarily give the time of the query as the CPU might be busy with other processes and the query will be just waiting for the CPU so we just use the CPU time as the performance metric.

1. Verify the query efficiency of single field narrow index and multi field wide index, pay attention to understand the left most matching principle in wide index.

**SQL Query:**

To create three single field indexes on a table **test**:  
SELECT \* FROM student.test;

create index v1\_test ON test(sr);

create index v2\_test ON test(level);

create index v3\_test ON test(value);  
By using multiple operation on different indexes efficiency will be decreased because DATABASE will fetch and manipulate data from 3 different views as compare to multi field index.  
SELECT \* from sudent.test where level='10' and value='233';

**SQL Query:**  
To create one multi field wide index on a table **test**:

create index v123\_test ON test(sr,level,value);  
**Testing:**

It is more effective than the narrow single indexes after executing commands with 3 or 3 conditions that have 1 multi field index since it takes less time.

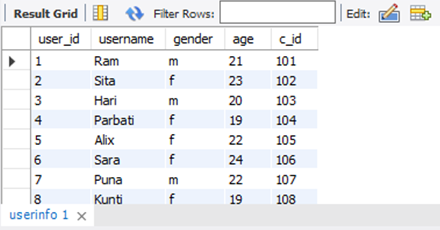
SELECT \* from test where level='0' and value='234';



3.Verify the difference of query efficiency between clustered index (primary key index) and secondary index: build clustered index and non-clustered index on the same field to compare query efficiency. (optional)

**PRIMARY/CLUSTERED INDEX:**

As we know by default in mysql PRIMARY index is clustered and all others are NON-CLUSTERED implicitly. Accessing a row using the clustered index is quick because its leads directly to the page with all data of row. If a table is large, it also helps often to saves a disk I/O operation.  
I have updated user ids but due to PRIMARY index its automatically updating the values to sort out the values of **user\_id:**



**SECONDARY INDEX**:  
Many of the indexes are called secondary indexes, except the primary one. In order to search the data in rows, each secondary index contains the primary key column, we can pick a small primary key to reduce the execution time and improve execution efficiency.

4.At present, only memory engine of MySQL supports both b-tree index and hash index. Create a basic table based on memory storage engine in mysql, and verify the query efficiency difference between b-tree index and hash index based on this table. (optional)

**B-TREE:**

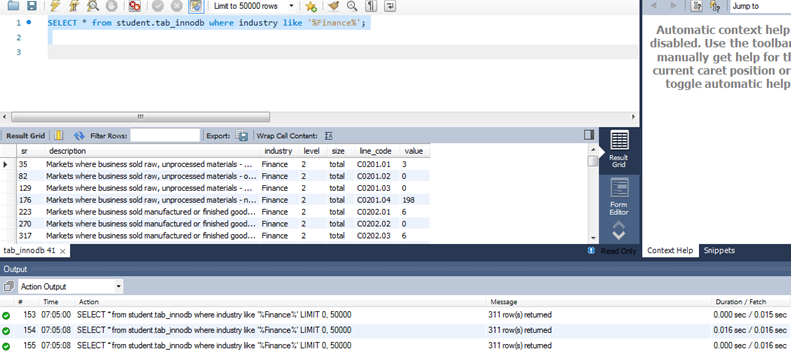
By default, when we create an index, its form is set to b-tree, which gives the best sort and helps to search for data such as binary search and others.

In the screenshot, you can see how to build the B-Tree Index and test the question after that:

**SQL QUERY:**

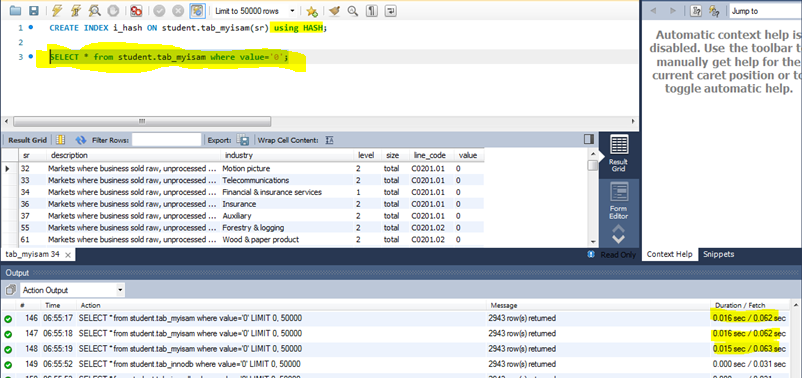
CREATE INDEX i\_btree ON student.tab\_innodb(sr);

Fetching data on base of range query help better and works efficiently with



**HASH INDEX:**Hash indexing is used for data fetching when equality operators are used, this indexing is best for such operations because they provide fast speed in data fetching.  
**SQL QUERY:**

CREATE INDEX i\_hash ON student.tab\_myisam(sr) using HASH;



In general, we can say that for point queries, the hash index is best and the btree index works for range queries.

**Conclusion:**

We how the CPU time decreased when we added the index key to the appropriate attribute. This makes us able to conclude that indexes will make queries run faster and in fact, the difference is clearer as much as the table has a big number of rows (if we only deal with a small number of rows, adding an index will not have that big impact on the query performance)

**Problem:**

Especially, for the view and index experiment we faced a lot of technical failures. For instance, a simple code typing error led me to redo the whole database again. Also some of the questions were very challenging**. I** had problems in adding millions of data. At first I entered few data manually but it would take a lot of time.Adding an index does not result in reducing the cpu time.

**Solution:**

With, Prof. Xiaonan Zhao’s very helpful guidance and assistance, I was able to solve most portions of our errors. I also used the internet and our textbook to find our remaining mistakes that led to fatal errors, it also provided us some SQL codes we haven’t practiced.

**Summary:**

We practiced more on views and indexes. We created several views and tested how it is like to use the view by another user and checked the problems of updating a table through the view. After that, we practiced creating indexes with different properties using both GUI and SQL. Finally, we made an experiment to see how creating an index on an appropriate attribute can improve query performance in term of CPU time.