西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Lab report

Name : ABID ALI
Student_no : 2019380141

Experiment 4

Experiment No:4

View and Index

Goal:

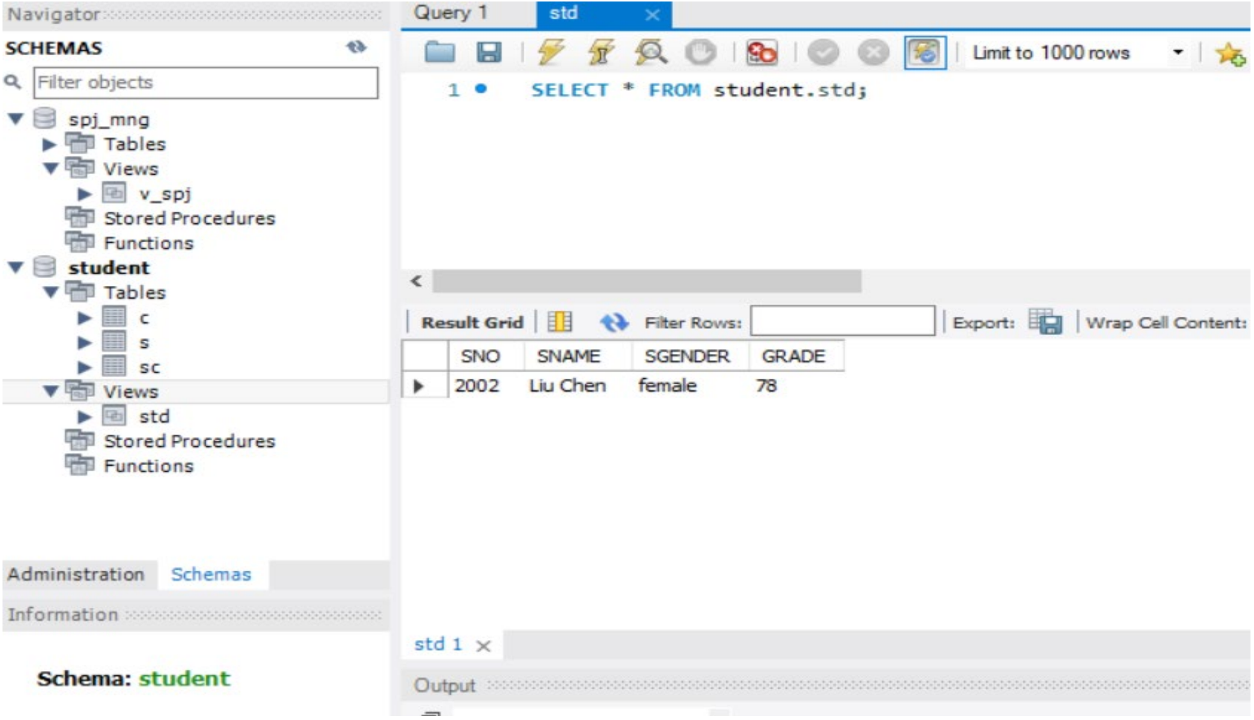
1. Be familiar with the use of GUI and SQL language to create, update and delete views.
2. Proficient in creating and deleting index using GUI and SQL language.
3. Understand and verify the role of index.

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.

Solution:

```
CREATE VIEW std AS
SELECT s.SNO, s.SNAME, s.SGENDER, sc.GRADE
FROM s
INNER JOIN sc
ON
s.sno=sc.sno
INNER JOIN c
ON
c.cno=sc.cno
WHERE
sc.cno LIKE '%2%' AND s.sbirth LIKE '%2001%';
```



```
SELECT * FROM student.std;
```

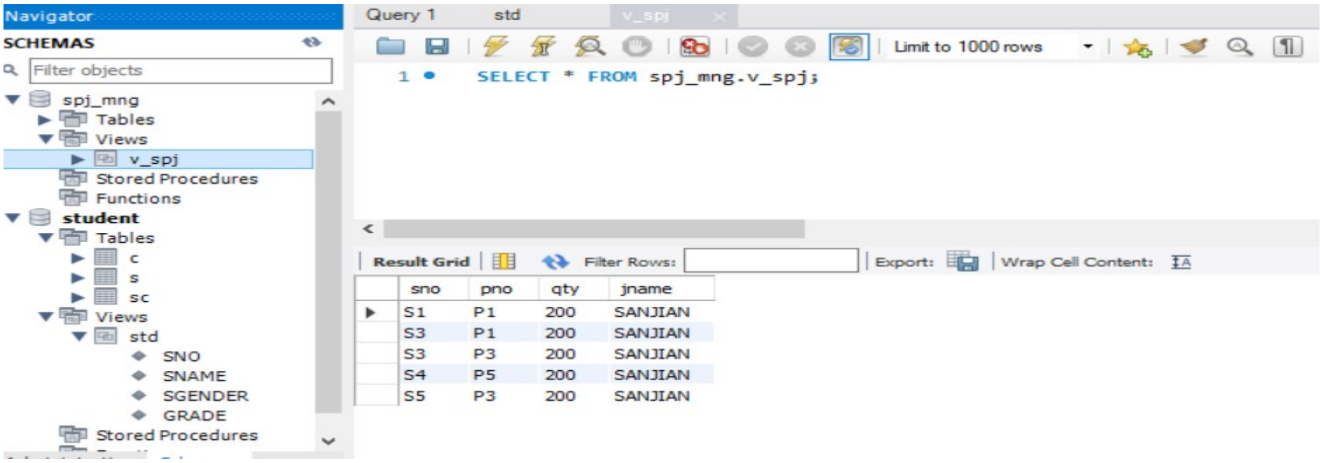
SNO	SNAME	SGENDER	GRADE
2002	Liu Chen	female	78

Schema: student

2 . 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)

Solution:

```
CREATE
VIEW spj_mng.v_spj AS
SELECT
spj_mng.spj.sno AS sno,
spj_mng.spj.pno AS pno,
spj_mng.spj.qty AS qty,
spj_mng.j.jname AS jname
FROM
(spj_mng.spj
JOIN spj_mng.j ON ((spj_mng.spj.jno = spj_mng.j.jno)))
WHERE
(spj_mng.j.jname = 'SANJIAN');
```



```
SELECT * FROM spj_mng.v_spj;
```

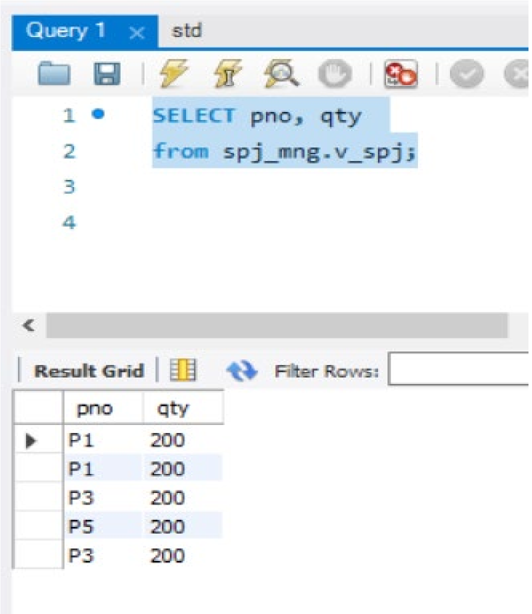
sno	pno	qty	jname
S1	P1	200	SANJIAN
S3	P1	200	SANJIAN
S3	P3	200	SANJIAN
S4	P5	200	SANJIAN
S5	P3	200	SANJIAN

3. Complete the following view query with SQL statement.

(1) Find out all the parts code and their quantity used by "SANJIAN" project.

Solution:

```
SELECT pno, qty
from spj_mng.v_spj;
```



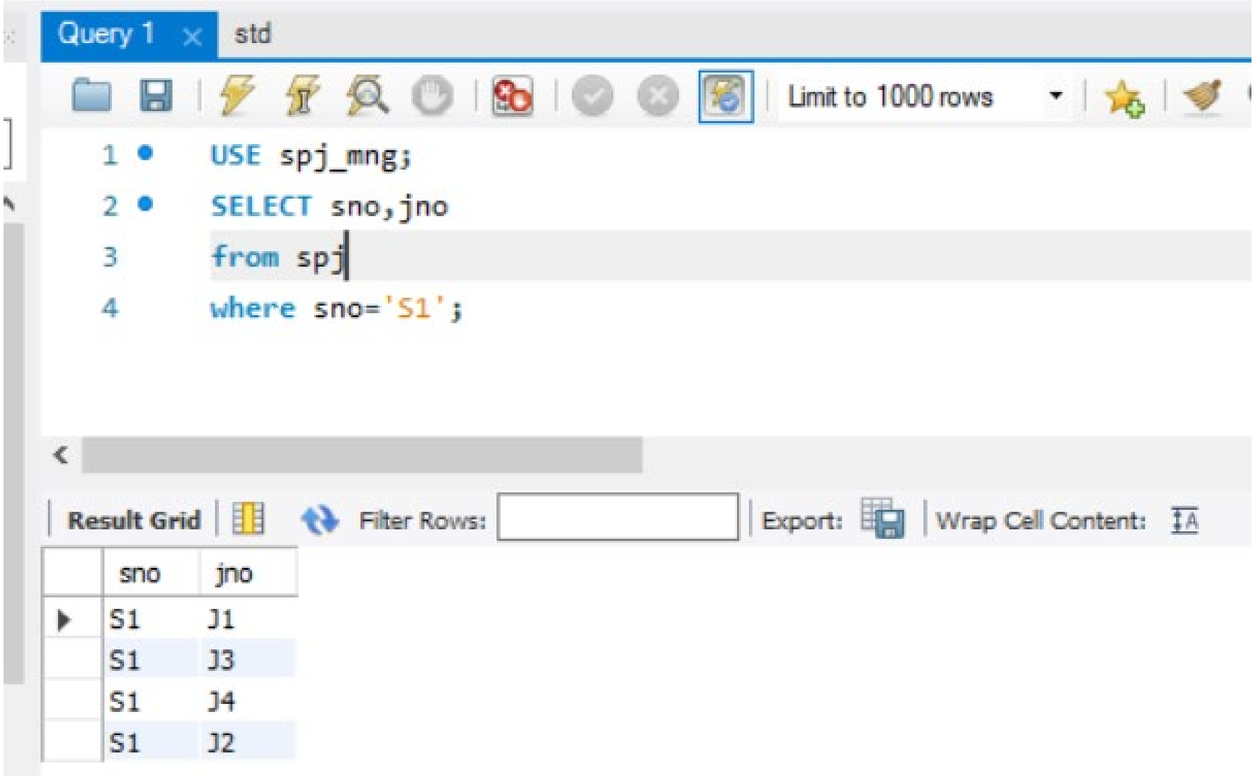
```
SELECT pno, qty
from spj_mng.v_spj;
```

pno	qty
P1	200
P1	200
P3	200
P5	200
P3	200

(2)Find out the supply situation of supplier S1.

Solution:

```
USE spj_mng;
SELECT sno,jno
from spj
where sno='S1';
```



```
USE spj_mng;
SELECT sno,jno
from spj
where sno='S1';
```

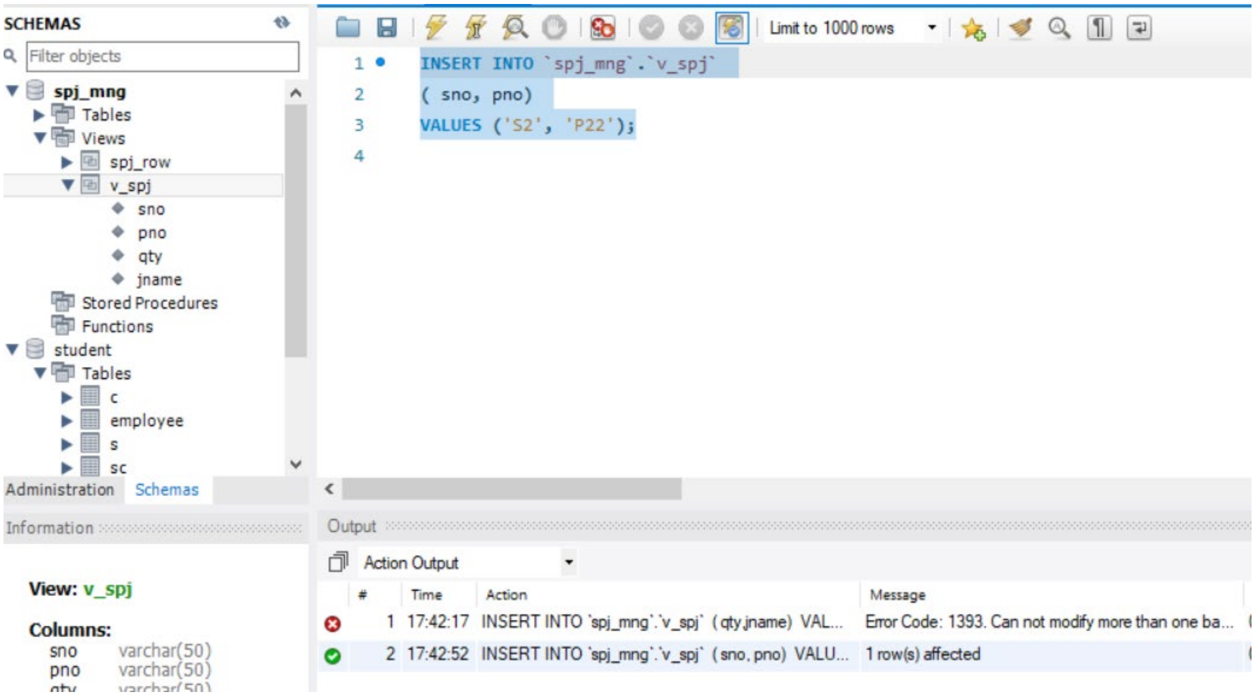
sno	jno
S1	J1
S1	J3
S1	J4
S1	J2

4. Update the data of views with SQL statement.(15 points)

(1) Insert a tuple into the view V_SPJ.

Solution:

INSERT INTO `spj_mng`.`v_spj`
(`sr`, `sno`, `pno`, `jno`, `qty`)
VALUES ('2', 'S2', 'P22', 'J222','2222');



```
INSERT INTO `spj_mng`.`v_spj`
( sno, pno)
VALUES ('S2', 'P22');
```

View: v_spj

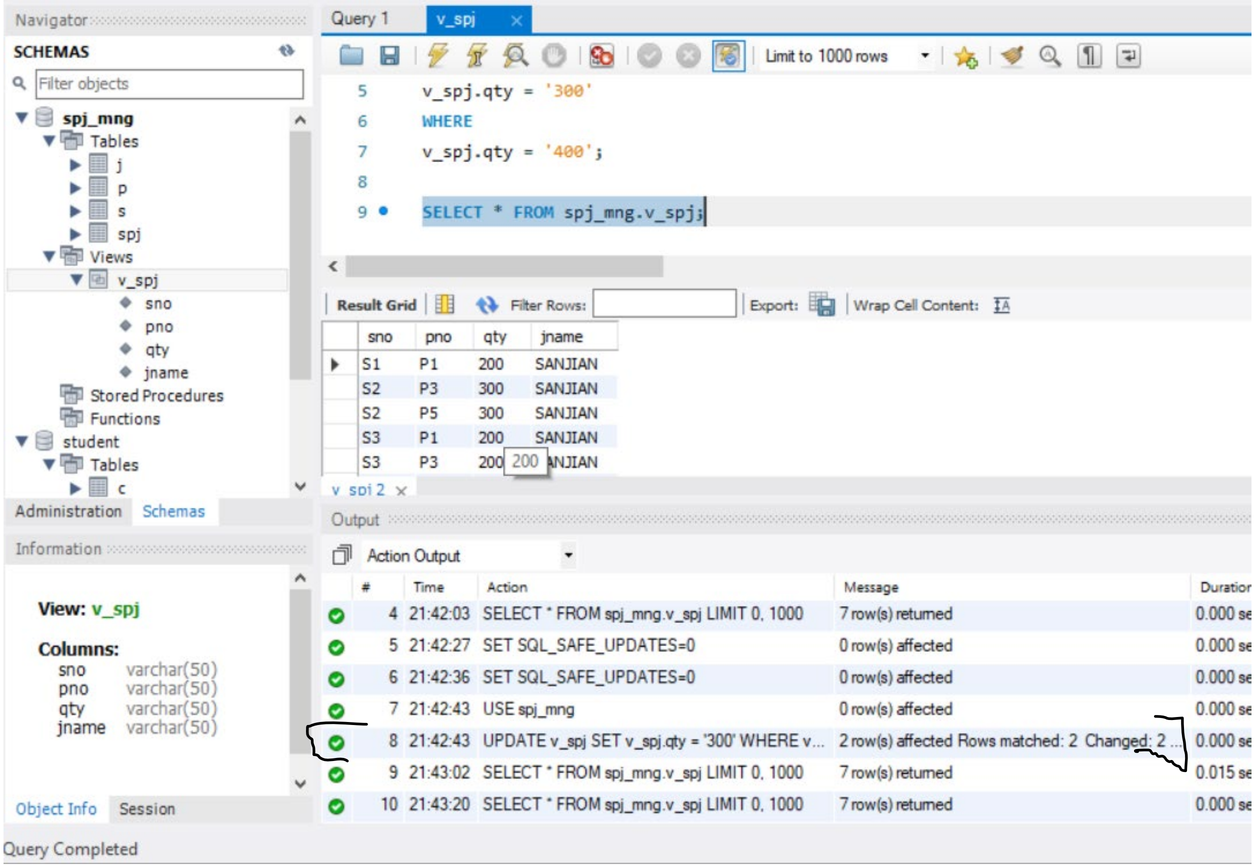
Columns:

```
sno     varchar(50)
pno     varchar(50)
qty     varchar(50)
```

```
#  Time      Action                                         Message
1  17:42:17  INSERT INTO `spj_mng`.`v_spj` (qty,jname) VAL...  Error Code: 1393. Can not modify more than one ba...
2  17:42:52  INSERT INTO `spj_mng`.`v_spj` (sno, pno) VALU...  1 row(s) affected
```

The key to do this problem is not to insert the attributes and column of primary key.We will insert other values except that then we can successfully input the values in the view.

(2)Modify the quantity value of any tuple in the view V_SPJ.



```
v_spj.qty = '300'
WHERE
v_spj.qty = '400';

SELECT * FROM spj_mng.v_spj;
```

sno	pno	qty	jname
S1	P1	200	SANJIAN
S2	P3	300	SANJIAN
S2	P5	300	SANJIAN
S3	P1	200	SANJIAN
S3	P3	200	SANJIAN

View: v_spj

Columns:

sno	varchar(50)
pno	varchar(50)
qty	varchar(50)
jname	varchar(50)

#	Time	Action	Message	Duration
4	21:42:03	SELECT * FROM spj_mng.v_spj LIMIT 0, 1000	7 row(s) returned	0.000 se
5	21:42:27	SET SQL_SAFE_UPDATES=0	0 row(s) affected	0.000 se
6	21:42:36	SET SQL_SAFE_UPDATES=0	0 row(s) affected	0.000 se
7	21:42:43	USE spj_mng	0 row(s) affected	0.000 se
8	21:42:43	UPDATE v_spj SET v_spj.qty = '300' WHERE v...	2 row(s) affected Rows matched: 2 Changed: 2 ...	0.000 se
9	21:43:02	SELECT * FROM spj_mng.v_spj LIMIT 0, 1000	7 row(s) returned	0.015 se
10	21:43:20	SELECT * FROM spj_mng.v_spj LIMIT 0, 1000	7 row(s) returned	0.000 se

Query Completed

Solution:

SET SQL_SAFE_UPDATES=0;
USE spj_mng;
UPDATE v_spj
SET
v_spj.qty = '300'
WHERE
v_spj.qty = '400';

SELECT * FROM spj_mng.v_spj;

We update in v_spj table in th column qty where 300 to 400
We can see that,we can see that ,we successfully did that.

(3) Delete one tuple from the view V_SPJ

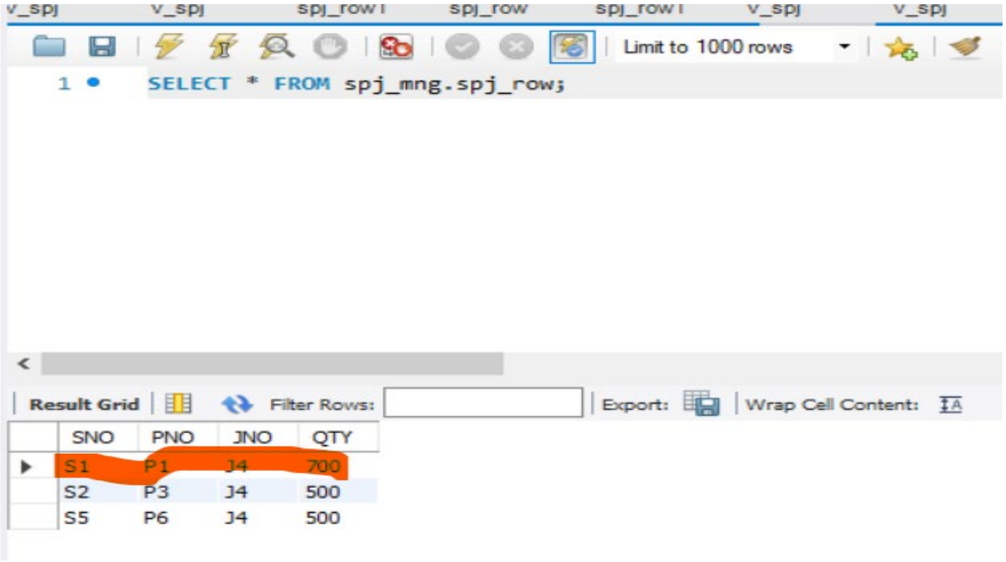
create view spj_row

as

select * from spj

where

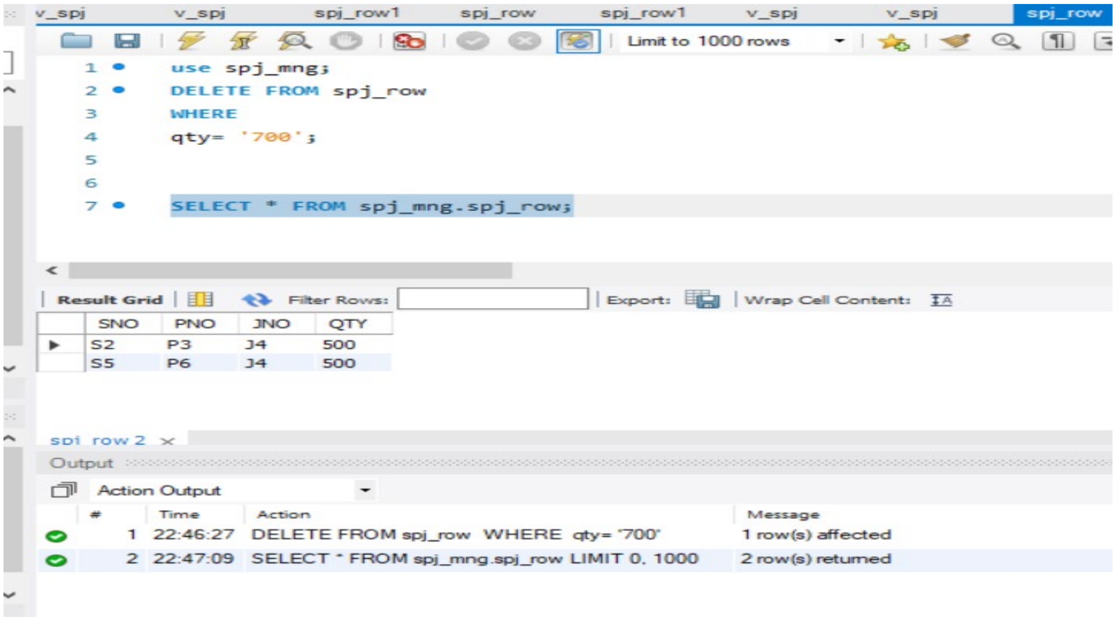
qty > '50';
SELECT * FROM spj_mng.v_spj;



```
SELECT * FROM spj_mng.spj_row;
```

	SNO	PNO	JNO	QTY
▶	S1	P1	J4	700
	S2	P3	J4	500
	S5	P6	J4	500

use spj_mng;
DELETE FROM spj_row
WHERE
qty= '700';



```
use spj_mng;
DELETE FROM spj_row
WHERE
qty= '700';


SELECT * FROM spj_mng.spj_row;
```

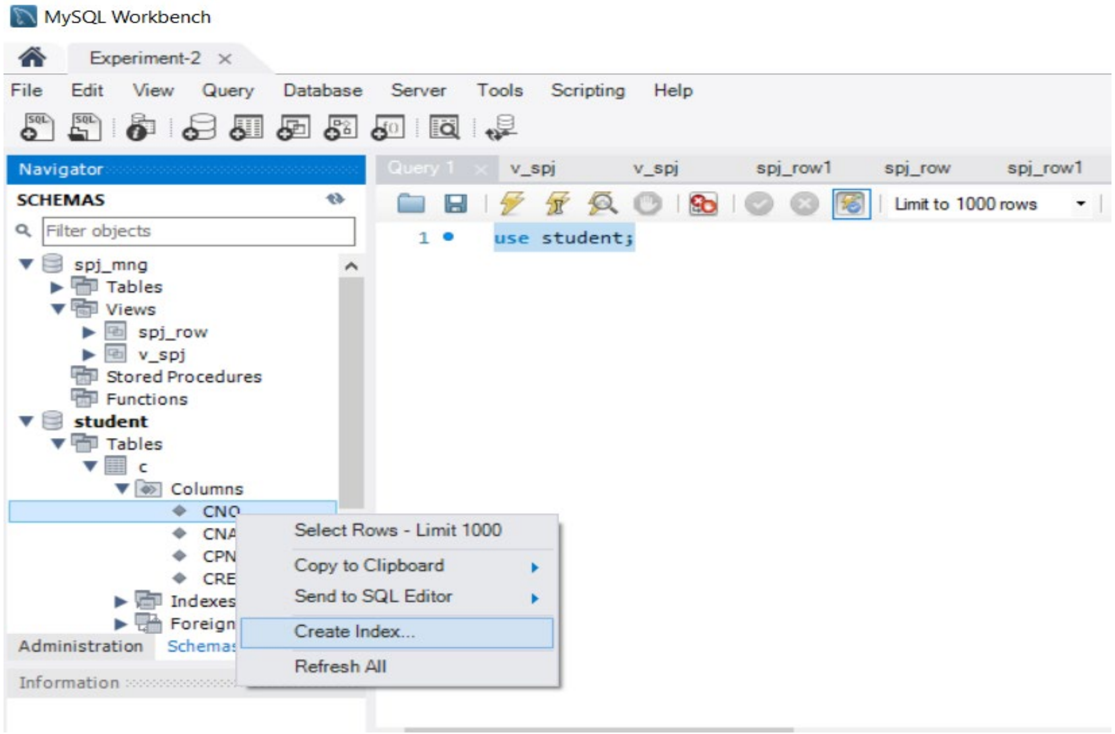
	SNO	PNO	JNO	QTY
▶	S2	P3	J4	500
	S5	P6	J4	500

	#	Time	Action	Message
✓	1	22:46:27	DELETE FROM spj_row WHERE qty= '700'	1 row(s) affected
✓	2	22:47:09	SELECT * FROM spj_mng.spj_row LIMIT 0, 1000	2 row(s) returned

We can see now that,SNO =S1 has been deleted from the table.

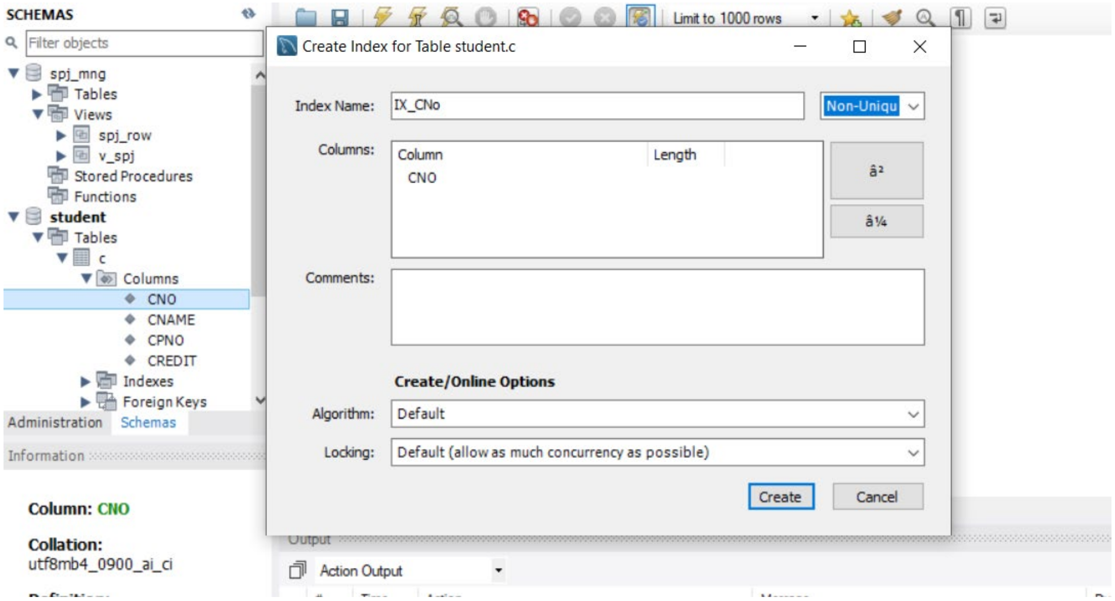
5. Create a descending index named IX_CNo for the CNO attribute of C table in student database by using GUI.

Answer No:5



```
use student;
```

Expand the database student and choose table “c” then expand the columns of table and right click on the “CNO” we will see the option of create index as per the figure.



Index Name: IX_CNo Non-Uniqu

Columns:

Column	Length
CNO	

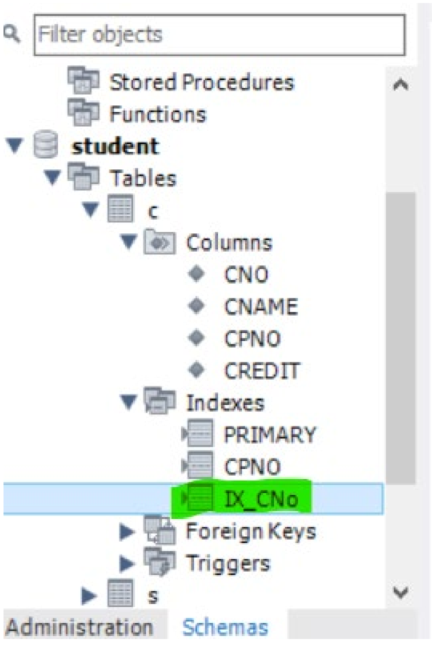
Comments:

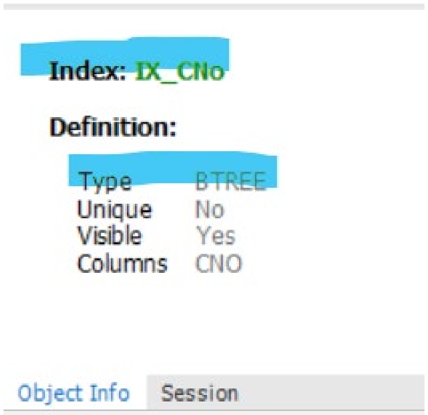
Algorithm: Default

Locking: Default (allow as much concurrency as possible)

Column: CNO

Collation: utf8mb4_0900_ai_ci





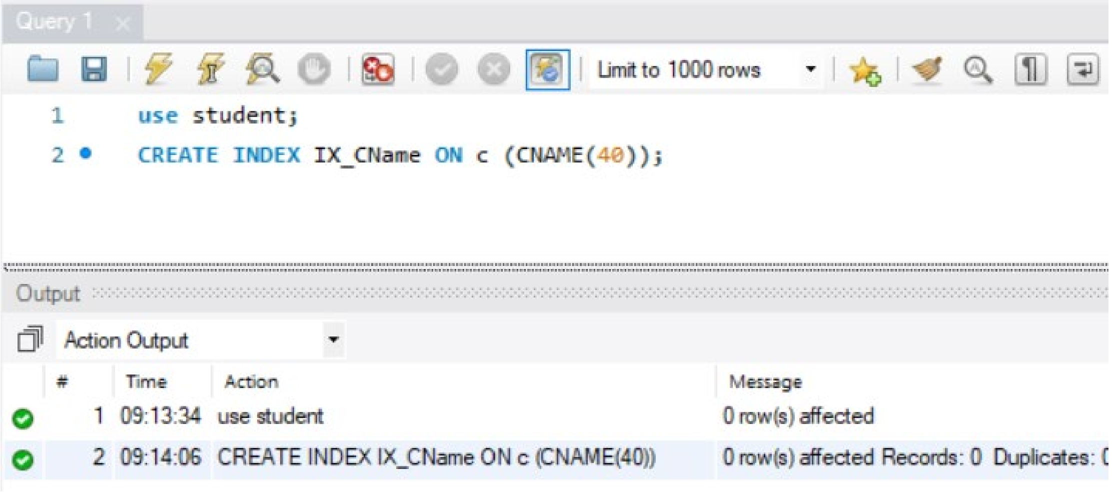
Index: IX_CNo

Definition:

Type	BTREE
Unique	No
Visible	Yes
Columns	CNO

6. Use SQL statement to complete the following index operation on student database.

(1) Create a non-unique index named IX_CNAME on the CNAME attribute of table C.



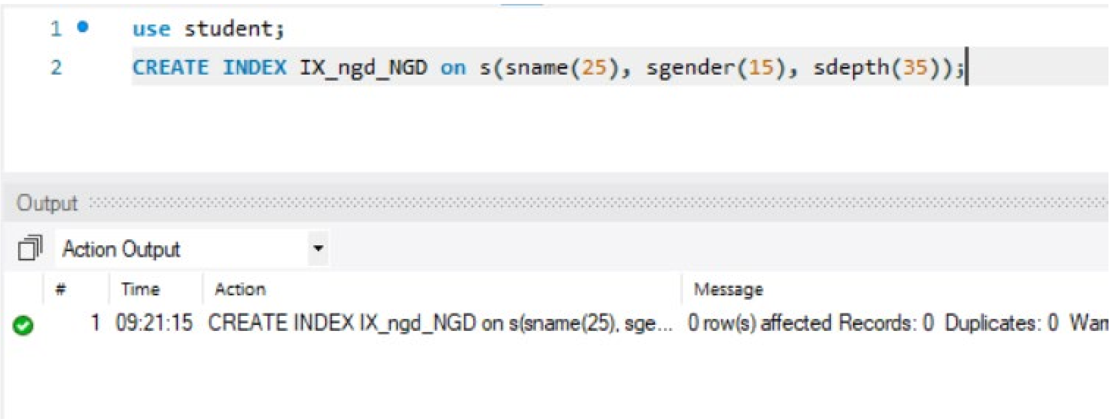
```
use student;
CREATE INDEX IX_CName ON c (CNAME(40));
```

```
#   Time      Action                                   Message
1   09:13:34  use student                              0 row(s) affected
2   09:14:06  CREATE INDEX IX_CName ON c (CNAME(40))   0 row(s) affected Records: 0  Duplicates: (
```

Whenever we don't use the keyword 'UNIQUE"then non-unique index will be created.

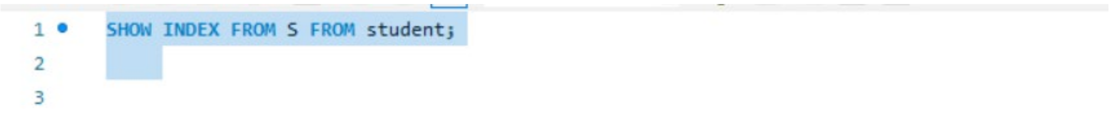
We have added prefix 40 as length of index but it can be upto 1000.

(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.



```
use student;
CREATE INDEX IX_ngd_NGD on s(sname(25), sgender(15), sdepth(35));
```

```
#   Time       Action                                          Message
1   09:21:15   CREATE INDEX IX_ngd_NGD on s(sname(25), sge...  0 row(s) affected Records: 0  Duplicates: 0  Warn
```



```
SHOW INDEX FROM S FROM student;
```

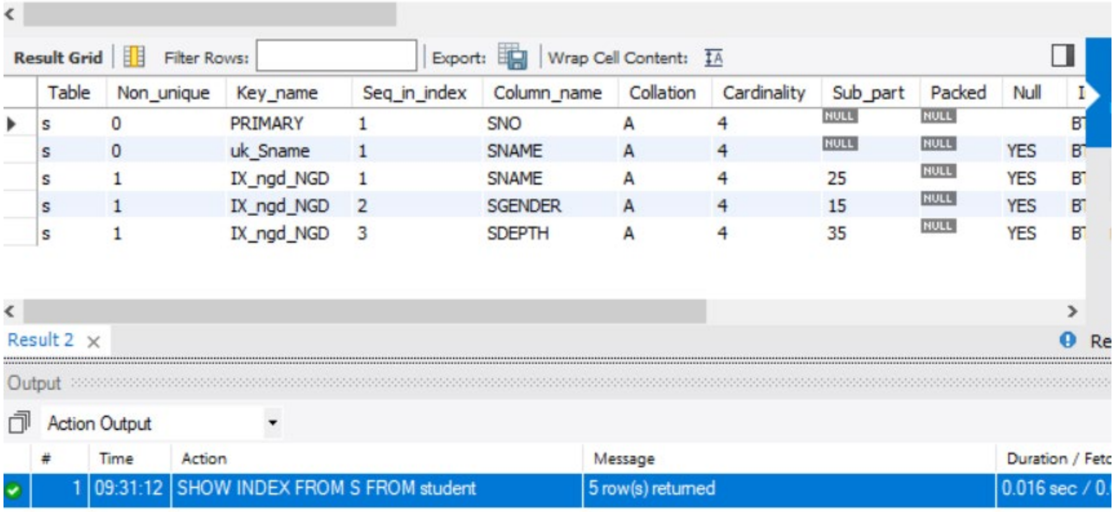


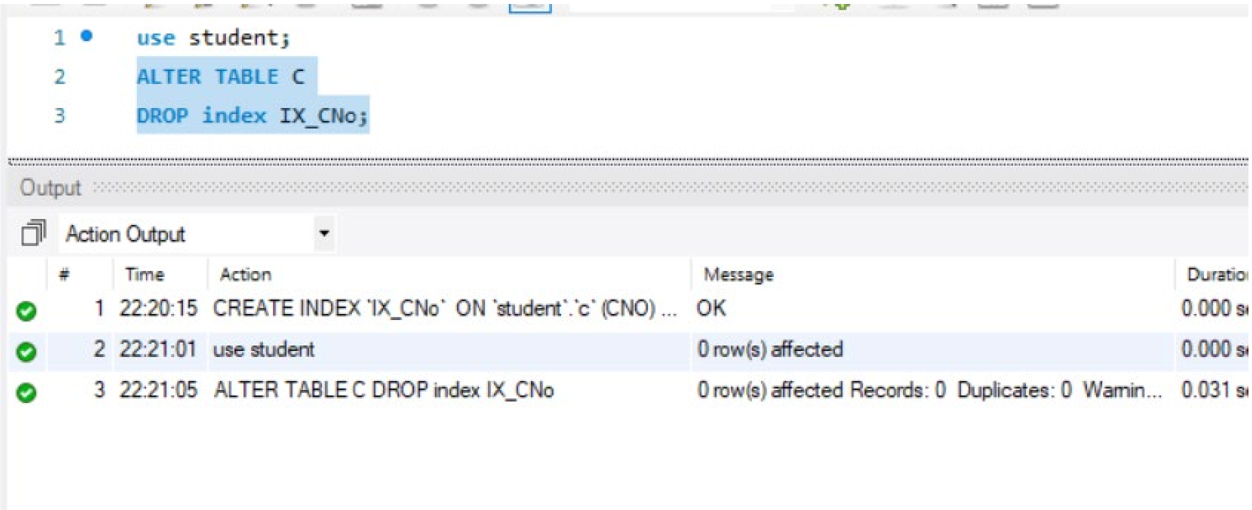
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	I
s	0	PRIMARY	1	SNO	A	4	NULL	NULL		BT
s	0	uk_Sname	1	SNAME	A	4	NULL	NULL	YES	BT
s	1	IX_ngd_NGD	1	SNAME	A	4	25	NULL	YES	BT
s	1	IX_ngd_NGD	2	SGENDER	A	4	15	NULL	YES	BT
s	1	IX_ngd_NGD	3	SDEPTH	A	4	35	NULL	YES	BT

```
#   Time       Action                           Message              Duration / Fetc
1   09:31:12   SHOW INDEX FROM S FROM student   5 row(s) returned    0.016 sec / 0.
```

(3) Delete index IX_ CNo of table C。

Code:

use student;
ALTER TABLE C
DROP index IX_CNo;



```
use student;
ALTER TABLE C
DROP index IX_CNo;
```

Output

Action Output

```
    #  Time      Action                                          Message                                                    Duratio
✓   1  22:20:15  CREATE INDEX `IX_CNo` ON `student`.`c` (CNO) ...  OK                                                         0.000 s
✓   2  22:21:01  use student                                     0 row(s) affected                                          0.000 s
✓   3  22:21:05  ALTER TABLE C DROP index IX_CNo                 0 row(s) affected Records: 0  Duplicates: 0  Warnin...     0.031 s
```

(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';

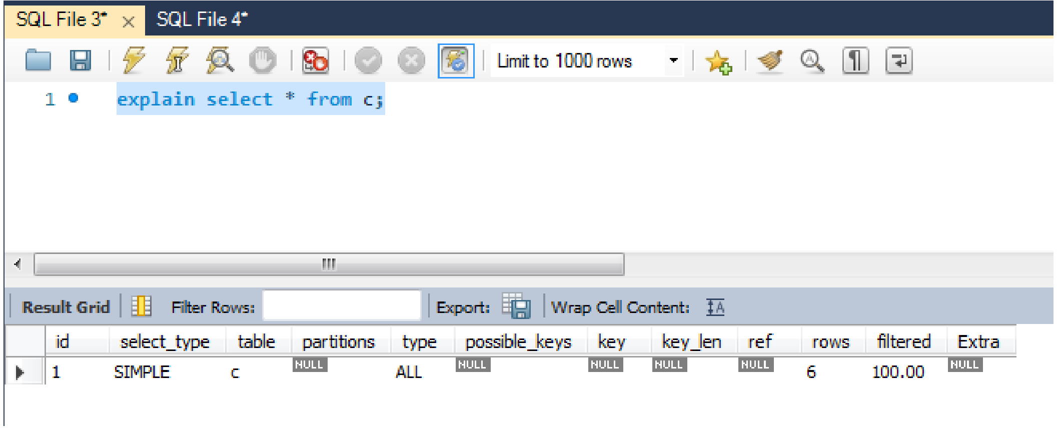
⑪ explain select * from s where sgender ='male' and sdept='IS';

In SQL the **EXPLAIN** keyword is used to get the information about the execution of query in a SQL Database. EXPLAIN can used with before SELECT, DELETE, INSERT, REPLACE, and UPDATE**.**

By using Explain we can find out the problem and its result contains the following columns which have special meaning listed below:

□ ID (query ID)
□ Select Type (type of statement)
□ Table (table referenced)
□ Type (Type of Join)
□ Possible Keys (will list the keys which can be used)
□ Key (Actual used key)
□ Key Length (Will list length of used key)
□ Ref (Will list index to be compared)
□ Rows (No of rows searched)
□ Extra (Additional Info)

1) **explain select * from c;**



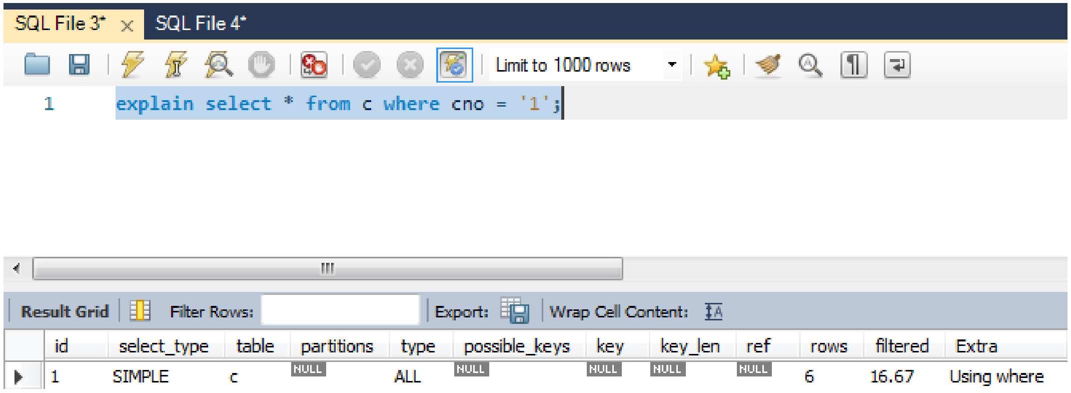
```
explain select * from c;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ALL	NULL	NULL	NULL	NULL	6	100.00	NULL

We can see that, data of c table hasn’t been showed .

2)

explain select * from c where cno = ‘1’;



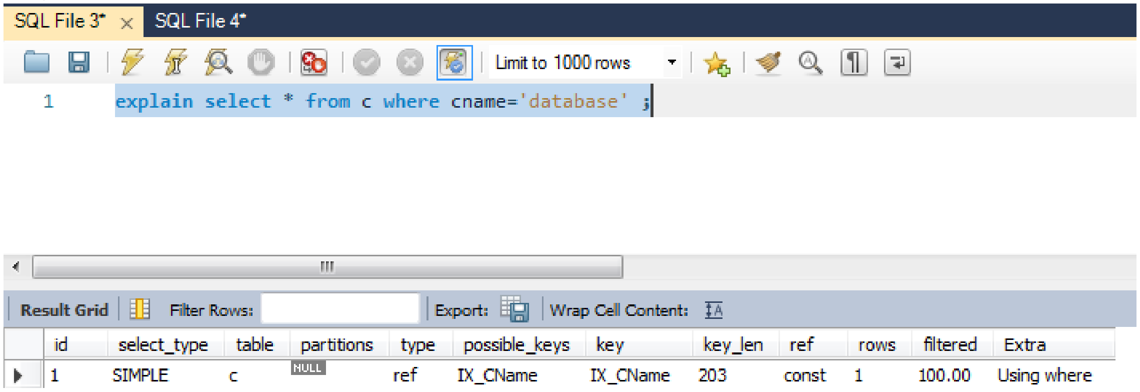
```
explain select * from c where cno = '1';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ALL	NULL	NULL	NULL	NULL	6	16.67	Using where

Here we can that we are using where clause and its showing this in **Extra** coloum but in previous it was not showing that.

3)

explain select * from c where cname=’database’ ;



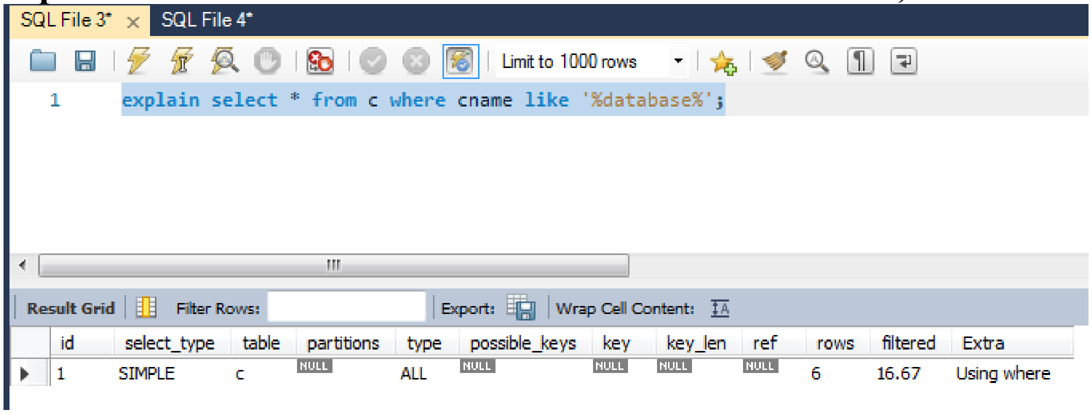
```
explain select * from c where cname='database' ;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ref	IX_CName	IX_CName	203	const	1	100.00	Using where

Its showing Possible Keys because we have index on CNAME coloum named as IX_CName.

4)

explain select * from c where cname like '%database%';



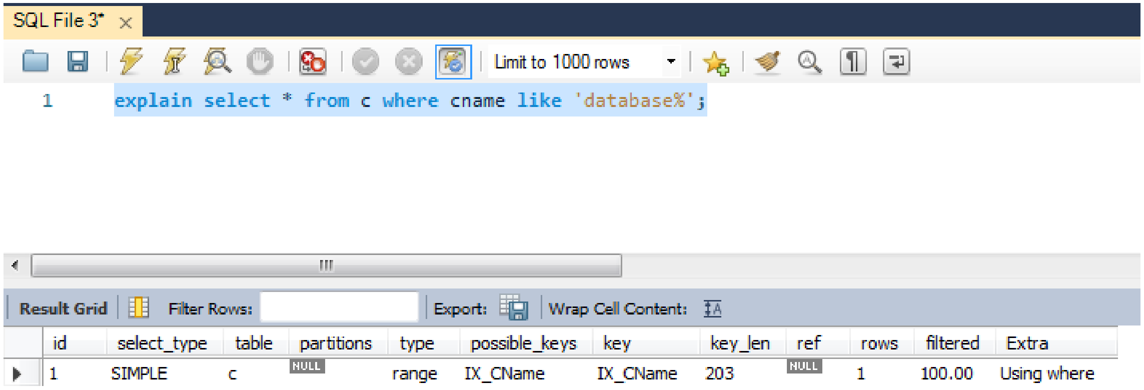
```
explain select * from c where cname like '%database%';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	ALL	NULL	NULL	NULL	NULL	6	16.67	Using where

We use wildcard here.Match with the word database.

5)

explain select * from c where cname like 'database%';



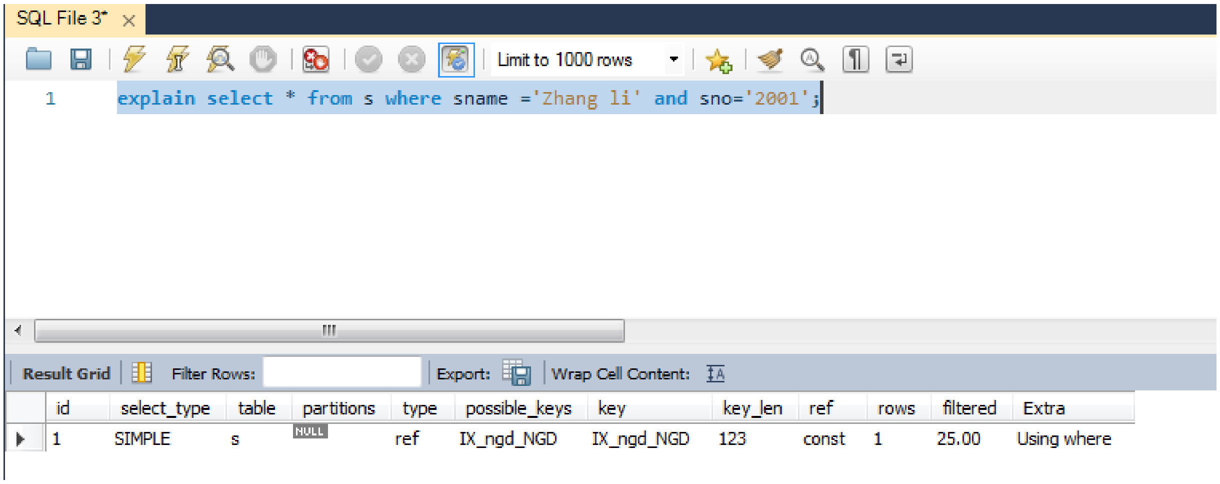
```
explain select * from c where cname like 'database%';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	c	NULL	range	IX_CName	IX_CName	203	NULL	1	100.00	Using where

We can understand from the query that explains that it will check range as mentioned in the type column.

6)

explain select * from s where sname ='Zhang li' and sno='2001';



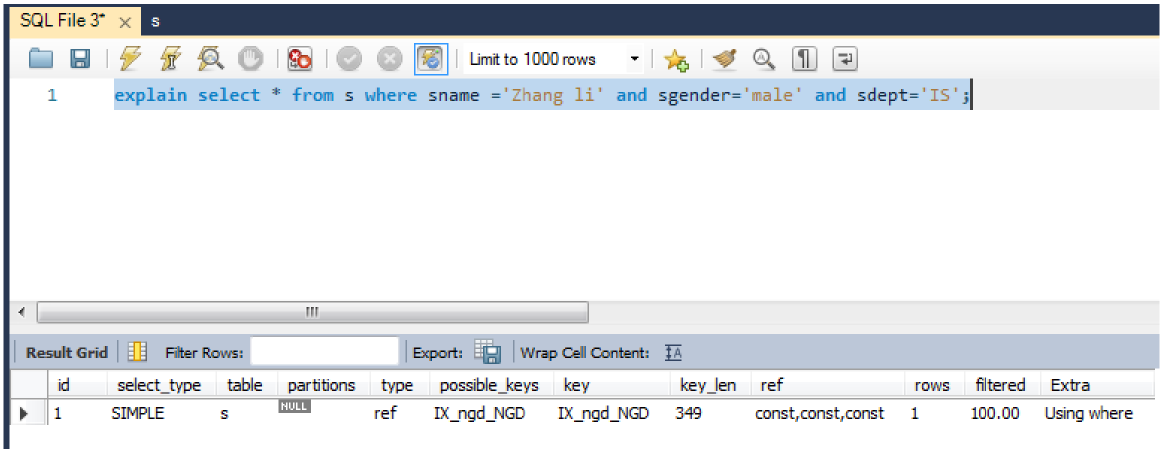
```sql
explain select * from s where sname ='Zhang li' and sno='2001';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ref	IX_ngd_NGD	IX_ngd_NGD	123	const	1	25.00	Using where

In this query it explains the check of two condition of **sname** and **sno**,we also get the information about possible key value.

7)

explain select * from s where sname ='Zhangli' and sgender='male' and sdept='IS';

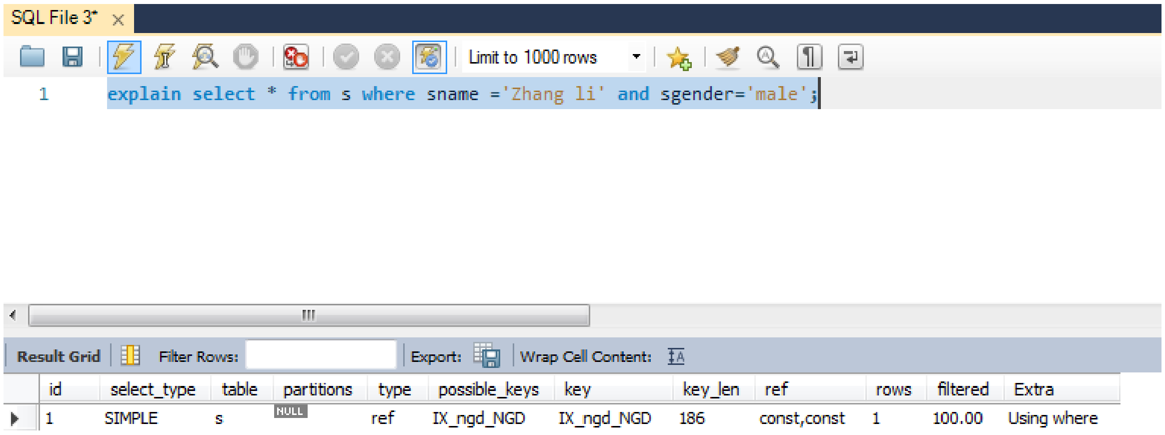


```sql
explain select * from s where sname ='Zhang li' and sgender='male' and sdept='IS';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ref	IX_ngd_NGD	IX_ngd_NGD	349	const,const,const	1	100.00	Using where

8)

explain select * from s where sname ='Zhang li' and sgender='male';

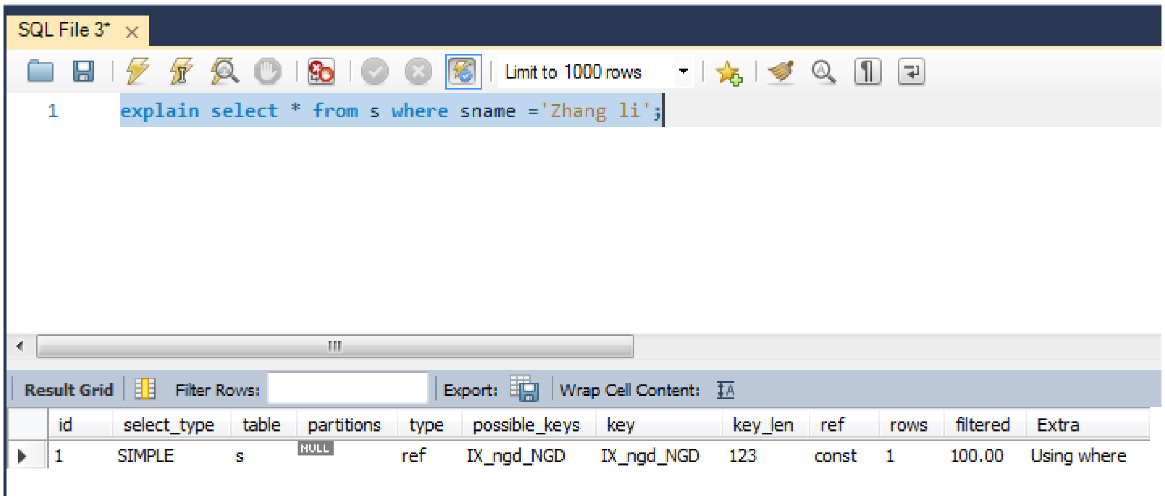


```
1   explain select * from s where sname ='Zhang li' and sgender='male';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ref	IX_ngd_NGD	IX_ngd_NGD	186	const,const	1	100.00	Using where

9)

explain select * from s where sname ='Zhang li';

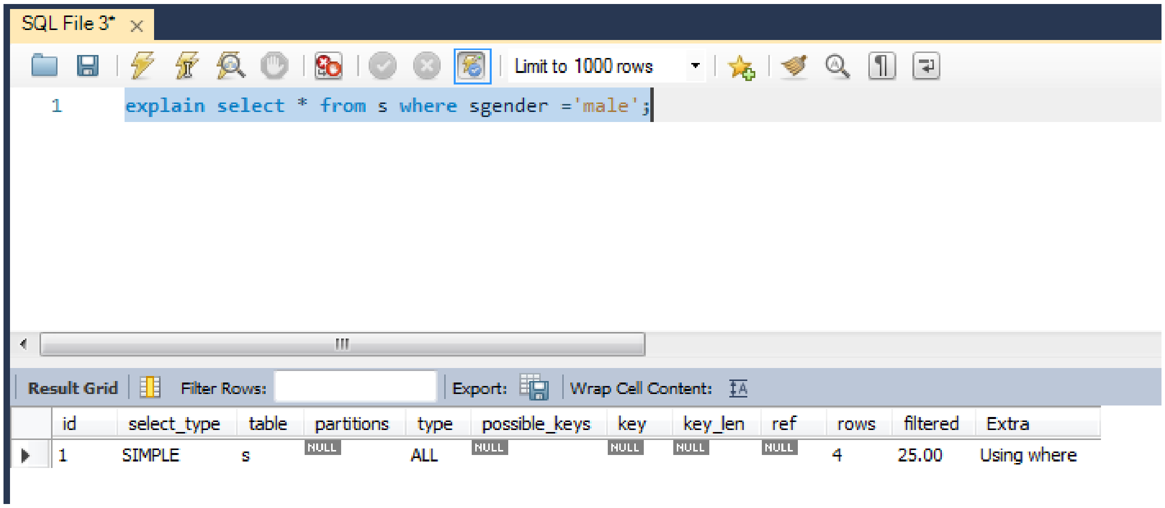


```
1   explain select * from s where sname ='Zhang li';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ref	IX_ngd_NGD	IX_ngd_NGD	123	const	1	100.00	Using where

10)

explain select * from s where sgender ='male';

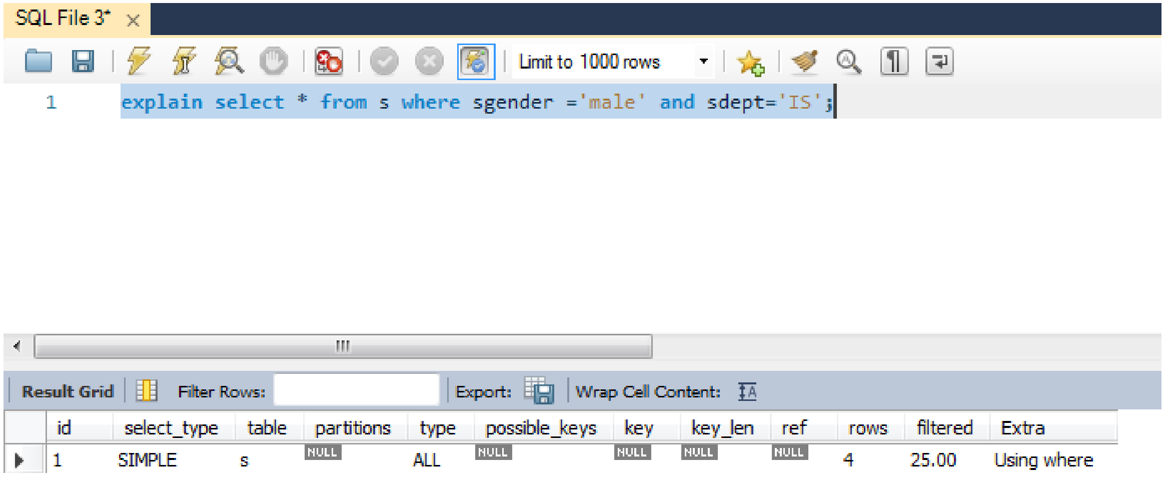


```
explain select * from s where sgender ='male';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ALL	NULL	NULL	NULL	NULL	4	25.00	Using where

11)

explain select * from s where sgender ='male' and sdept='IS';



```
explain select * from s where sgender ='male' and sdept='IS';
```

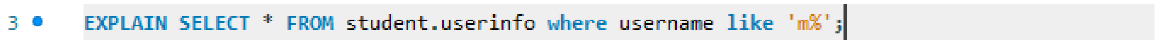
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	ALL	NULL	NULL	NULL	NULL	4	25.00	Using where

7.Suppose there is a basic table userinfo as follows, design an experiment to verify the effect of index on database query efficiency. (40 points) create table userinfo

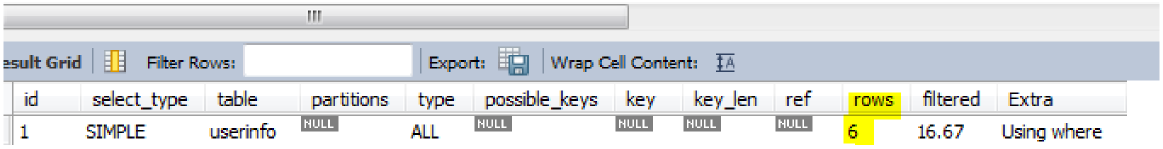
```
    (
        user_id int primary key,  //USER ID
        username varchar(10),   //USERNAME
        gender char(1),              //GENDER
        age int,                  //AGE
       c_id int                   //NO OF COLLEGE
    )
```

(1) Verify the efficiency difference between indexed and non-indexed queries.

Executing query without index you can sql has to check all the 6 rows to find out the username starting with m as shown in figure.

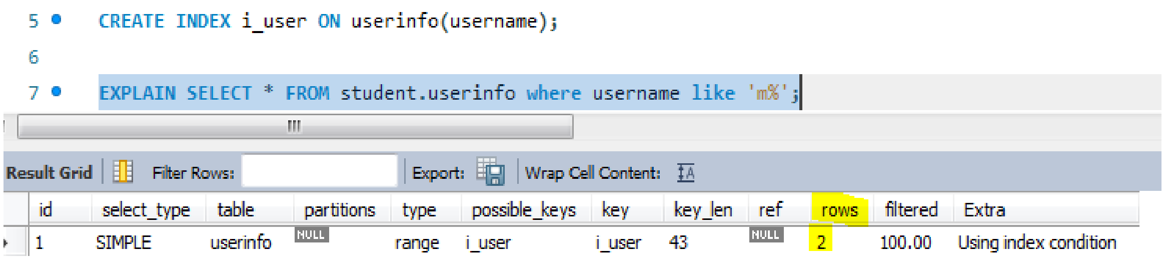


```
EXPLAIN SELECT * FROM student.userinfo where username like 'm%';
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	userinfo	NULL	ALL	NULL	NULL	NULL	NULL	6	16.67	Using where

Now executing same query after creating index **i_user** of table.



```
CREATE INDEX i_user ON userinfo(username);

EXPLAIN SELECT * FROM student.userinfo where username like 'm%';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	userinfo	NULL	range	i_user	i_user	43	NULL	2	100.00	Using index condition

You can see the efficiency of execution of query it searched for only 2 rows as shown in figure after creating index but before that it was searching all over the table to match the values.

(2) 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.

To create three single field indexes on a table **test**:

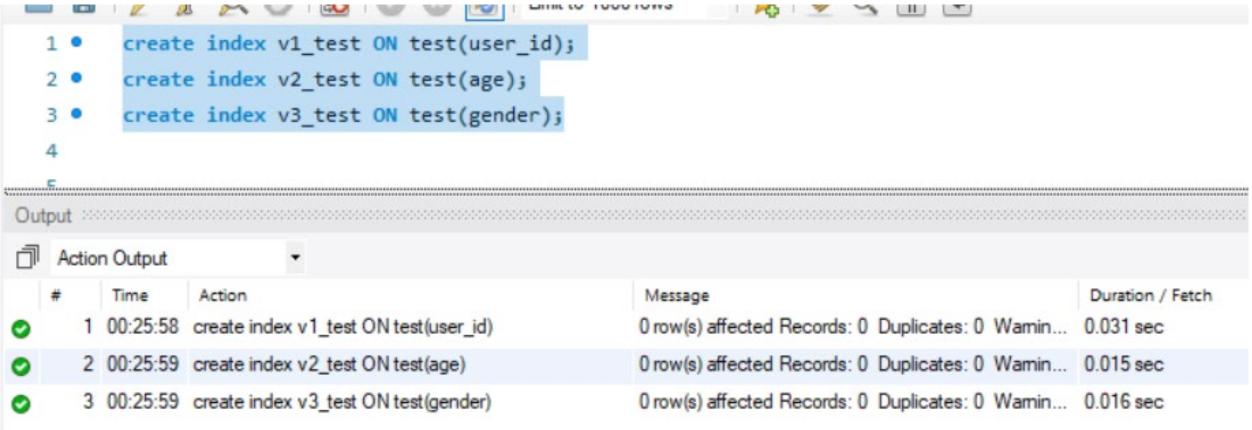
Code:

create index v1_test ON test(user_id);

create index v2_test ON test(age);

create index v3_test ON test(gender);

SELECT * FROM student.test;



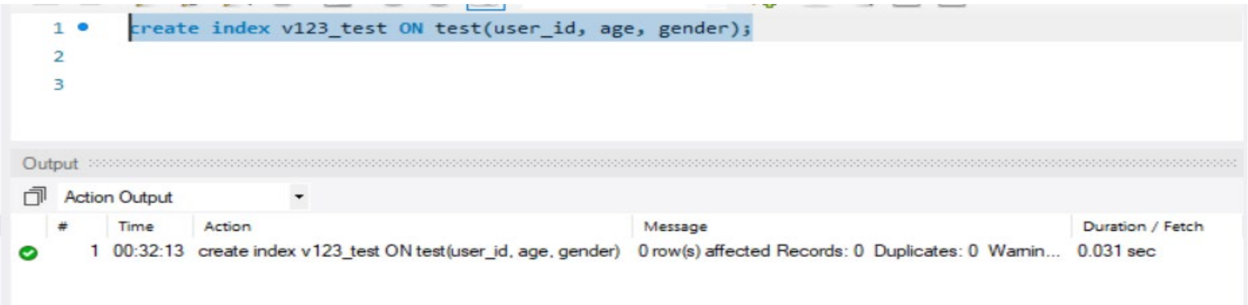
```
create index v1_test ON test(user_id);
create index v2_test ON test(age);
create index v3_test ON test(gender);
```

#	Time	Action	Message	Duration / Fetch
1	00:25:58	create index v1_test ON test(user_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.031 sec
2	00:25:59	create index v2_test ON test(age)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.015 sec
3	00:25:59	create index v3_test ON test(gender)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.016 sec

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 student.test where age='10' and user_id='234';

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

create index v123_test ON test(user_id, age, gender);

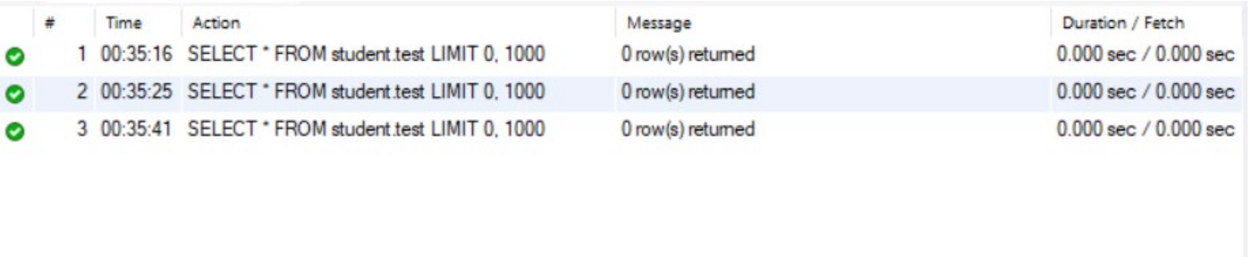


```
create index v123_test ON test(user_id, age, gender);
```

	#	Time	Action	Message	Duration / Fetch
✓	1	00:32:13	create index v123_test ON test(user_id, age, gender)	0 row(s) affected Records: 0 Duplicates: 0 Warnin...	0.031 sec

Testing:

After executing command with 3 or 3 conditions having 1 multi field index is more efficient than the narrow single indexes because it takes less time. SELECT * FROM student.test;



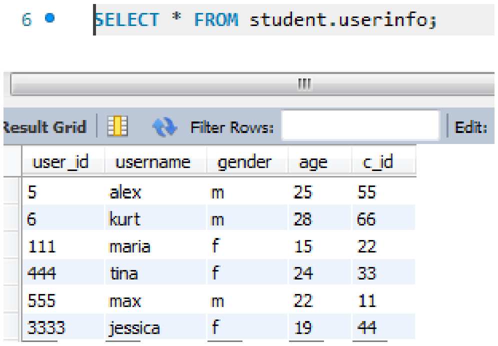
	#	Time	Action	Message	Duration / Fetch
✓	1	00:35:16	SELECT * FROM student.test LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
✓	2	00:35:25	SELECT * FROM student.test LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
✓	3	00:35:41	SELECT * FROM student.test LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

In conclusion ,we can say that, multi field index is better than other because it takes less after few execution.

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

PRIMARY/CLUSTERED INDEX:

A primary key is a unique index that is clustered by default. By default means that when you create a primary key, if the table is not clustered yet, the primary key will be created as a clustered unique index. Unless you explicitly specify the nonclustered option.



```
SELECT * FROM student.userinfo;
```

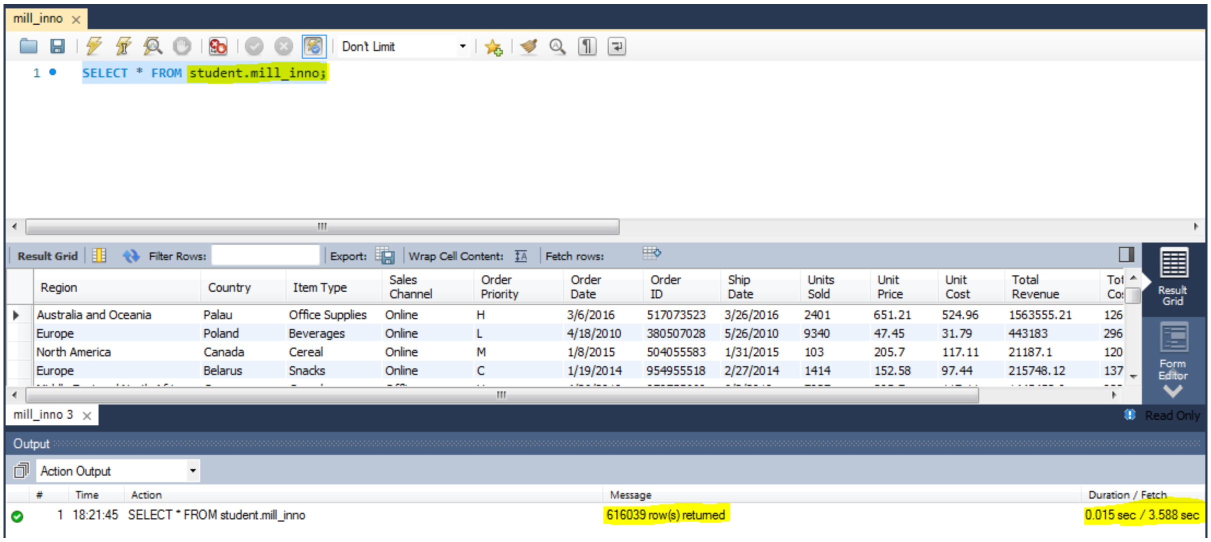
user_id	username	gender	age	c_id
5	alex	m	25	55
6	kurt	m	28	66
111	maria	f	15	22
444	tina	f	24	33
555	max	m	22	11
3333	jessica	f	19	44

SECONDARY INDEX:

Secondary indexes are indexes **that process a segment type in a sequence other than the one** that is defined by the segment's key. A secondary index can also process a segment type based on a qualification in a dependent segment.

(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: When we create an index by default its type will be set as b-tree, which gives best sort and helps in searching data like binary search and other, You can see in the screenshot how to create B-Tree Index and evaluation of query after this: **SQL QUERY:**
CREATE INDEX i_btree ON student.tab_innodb(sr);
Fetching data on base of range query help better and works efficiently with



```
1   SELECT * FROM student.mill_inno;
```

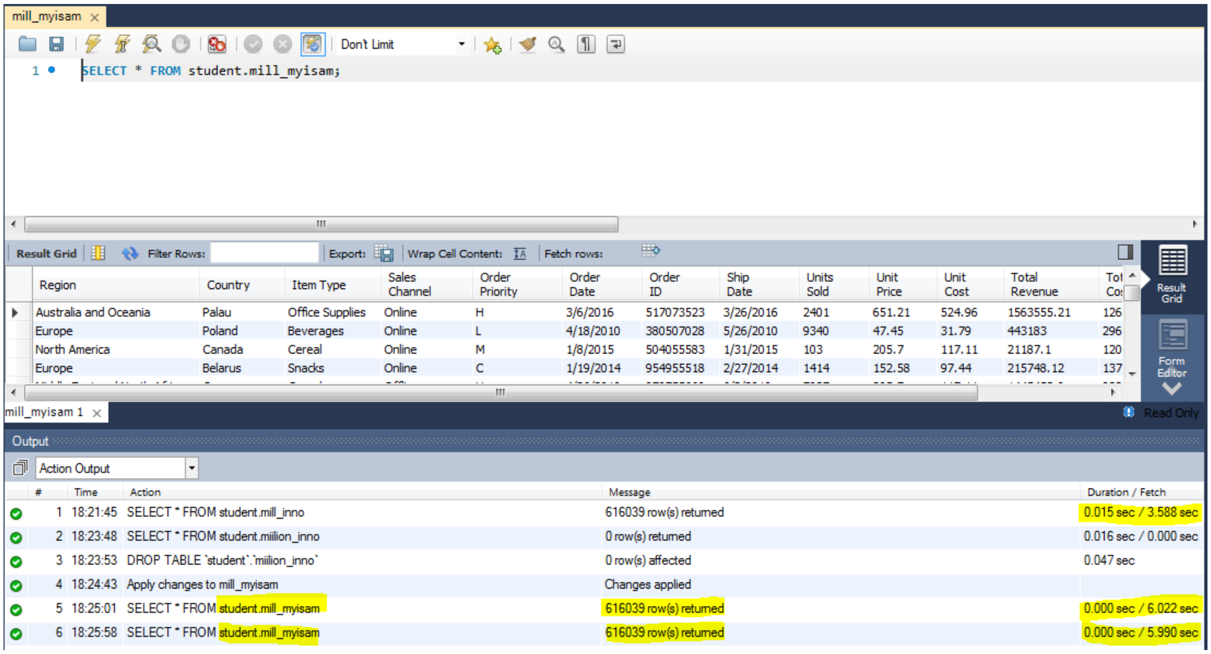
Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Tot Co:
Australia and Oceania	Palau	Office Supplies	Online	H	3/6/2016	517073523	3/26/2016	2401	651.21	524.96	1563555.21	126
Europe	Poland	Beverages	Online	L	4/18/2010	380507028	5/26/2010	9340	47.45	31.79	443183	296
North America	Canada	Cereal	Online	M	1/8/2015	504055583	1/31/2015	103	205.7	117.11	21187.1	120
Europe	Belarus	Snacks	Online	C	1/19/2014	954955518	2/27/2014	1414	152.58	97.44	215748.12	137

#	Time	Action	Message	Duration / Fetch
1	18:21:45	SELECT * FROM student.mill_inno	616039 row(s) returned	0.015 sec / 3.588 sec

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. As you can see in the below snap hash indexing by using MYISAM Engine took double time to fetch data instead of Innodb or B-Tree indexing.

SQL QUERY:
CREATE INDEX i_hash ON student.tab_myisam(sr) using HASH;



```
SELECT * FROM student.mill_myisam;
```

Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Tot Co:
Australia and Oceania	Palau	Office Supplies	Online	H	3/6/2016	517073523	3/26/2016	2401	651.21	524.96	1563555.21	126
Europe	Poland	Beverages	Online	L	4/18/2010	380507028	5/26/2010	9340	47.45	31.79	443183	296
North America	Canada	Cereal	Online	M	1/8/2015	504055583	1/31/2015	103	205.7	117.11	21187.1	120
Europe	Belarus	Snacks	Online	C	1/19/2014	954955518	2/27/2014	1414	152.58	97.44	215748.12	137

#	Time	Action	Message	Duration / Fetch
1	18:21:45	SELECT * FROM student.mill_inno	616039 row(s) returned	0.015 sec / 3.588 sec
2	18:23:48	SELECT * FROM student.miilion_inno	0 row(s) returned	0.016 sec / 0.000 sec
3	18:23:53	DROP TABLE `student`.`miilion_inno`	0 row(s) affected	0.047 sec
4	18:24:43	Apply changes to mill_myisam	Changes applied	
5	18:25:01	SELECT * FROM student.mill_myisam	616039 row(s) returned	0.000 sec / 6.022 sec
6	18:25:58	SELECT * FROM student.mill_myisam	616039 row(s) returned	0.000 sec / 5.990 sec

In general, we can say that hash index is best for point queries and the btree indexing fits for range queries.This is the final conclusion ,we can get.

Problems:
At the beginning I had some GUI errors and many syntax errors because It felt very complicated at the beginning ,shifting from GUI to Sql statements.

Solutions:

To solve these problems which I faced during doing this practical, I took help from internet especially YouTube,StackOverflow and W3school to get information about these errors for the solution. I also asked the teacher to help me understand them. And provided instructions helped to solve some of my errors during the experiment.

Summary:

From this experiment I have learned use of GUI and SQL language to create, update and delete views. Understand the role of index and how to add and delete operation in column.It was challenging experiment.I was able to successfully complete the experiment.

Attachments:
1) DB4_2019380141_ABID ALI.docx
2) DB4_2019380141_ABID ALI.pdf

References:

1) https://www.w3schools.com/

2) https://stackoverflow.com/

3) https://youtube.com/

4)https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server/

5)https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html#:~:text=Indexes%20are%20used%20to%20find,table%2C%20the%20more%20this%20costs.