Name: Dikshya Kafle

Student Number: 2018380039

Experiment 4:

View and Index :

Goal

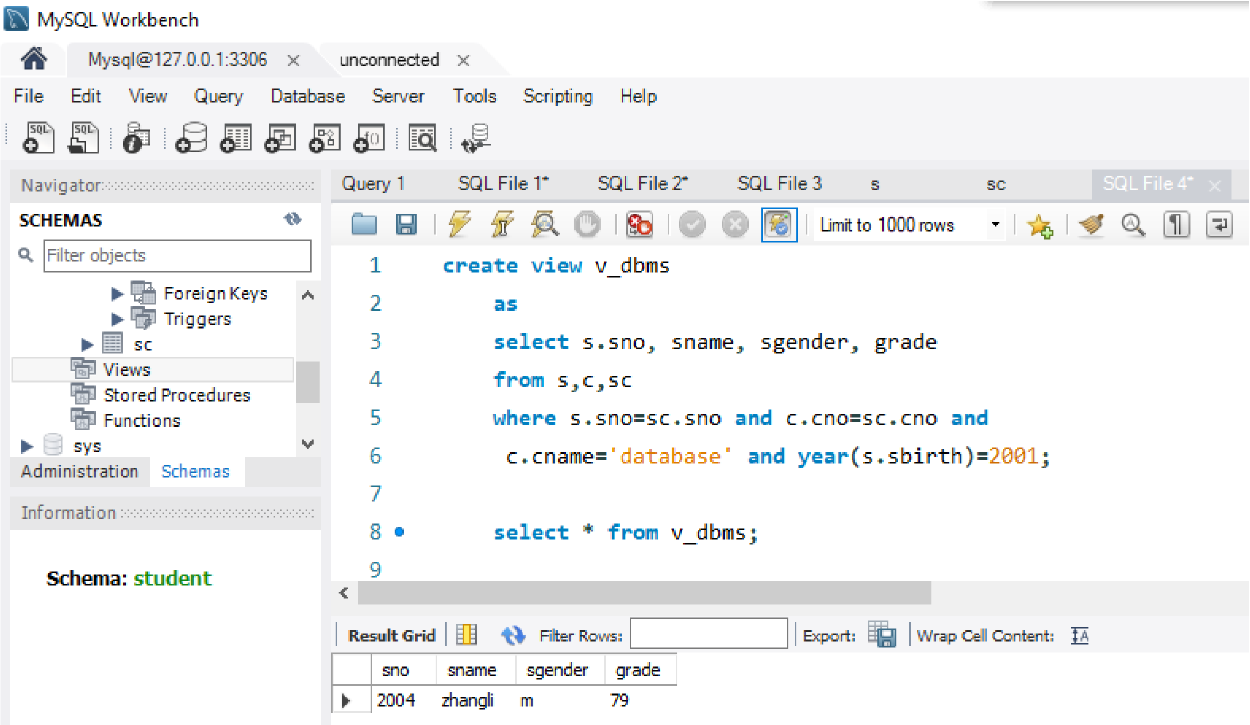
1. Be familiar with the use of GUI and SQL language to create, update and delete views.

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.

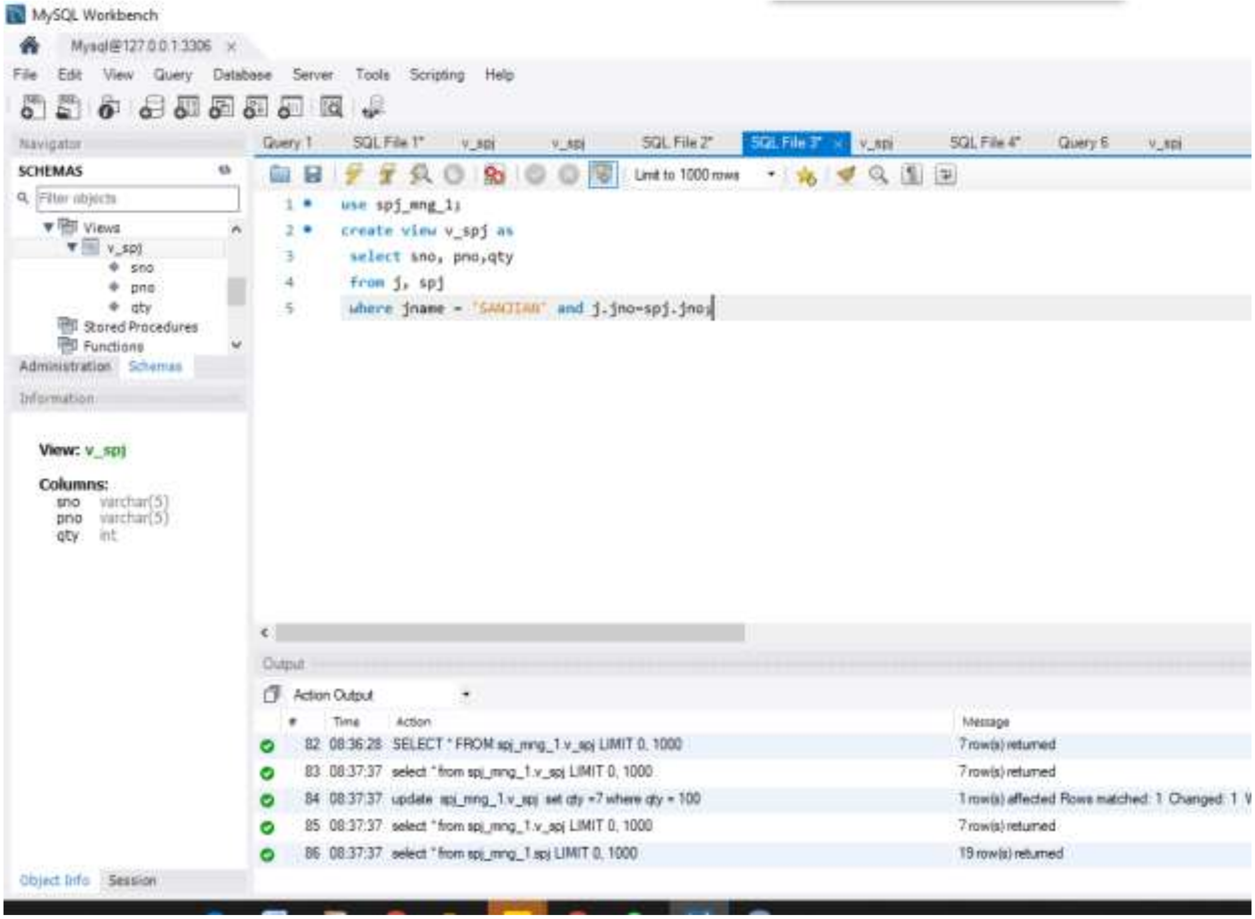


Schema: student

```
create view v_dbms
    as
    select s.sno, sname, sgender, grade
    from s,c,sc
    where s.sno=sc.sno and c.cno=sc.cno and
     c.cname='database' and year(s.sbirth)=2001;

    select * from v_dbms;
```

sno	sname	sgender	grade
2004	zhangli	m	79

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)



```
use spj_mng_1;
create view v_spj as
 select sno, pno,qty
 from j, spj
 where jname = 'SANJIAN' and j.jno=spj.jno;
```

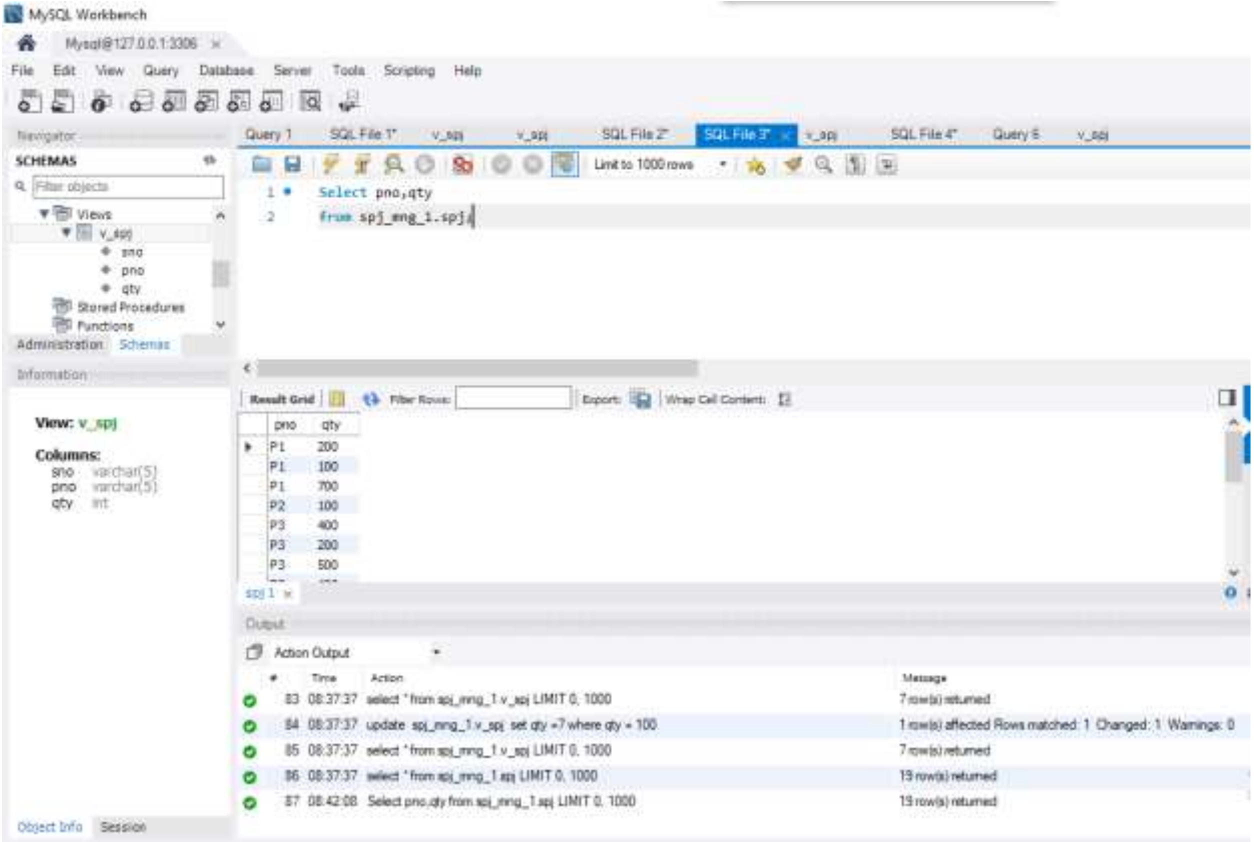
View: v_spj

Columns:
sno varchar(5)
pno varchar(5)
qty int

```
#   Time      Action                                              Message
82  08:36:28  SELECT * FROM spj_mng_1.v_spj LIMIT 0, 1000          7 row(s) returned
83  08:37:37  select *from spj_mng_1.v_spj LIMIT 0, 1000           7 row(s) returned
84  08:37:37  update  spj_mng_1.v_spj  set qty =7 where qty = 100  1 row(s) affected Rows matched: 1 Changed: 1 W
85  08:37:37  select *from spj_mng_1.v_spj LIMIT 0, 1000           7 row(s) returned
86  08:37:37  select *from spj_mng_1.spj LIMIT 0, 1000             19 row(s) returned
```

3. Complete the following view query with SQL statement.

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



```
Select pno,qty
from spj_mng_1.spj;
```

pno	qty
P1	200
P1	100
P1	700
P2	100
P3	400
P3	200
P3	500

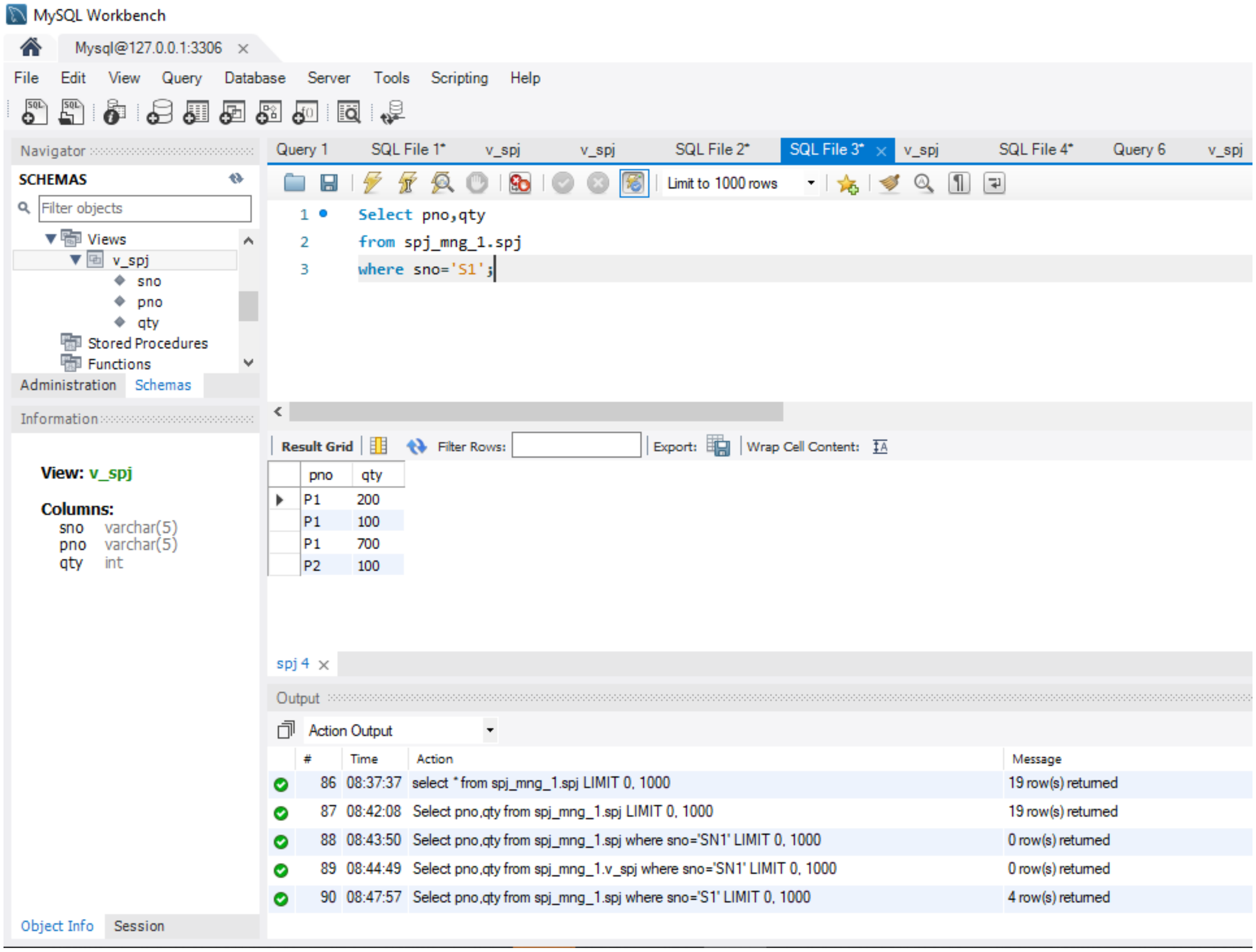
View: v_spj

Columns:

sno	varchar(5)
pno	varchar(5)
qty	int

```
#   Time      Action                                                   Message
83  08:37:37  select * from spj_mng_1.v_spj LIMIT 0, 1000                7 row(s) returned
84  08:37:37  update spj_mng_1.v_spj set qty =7 where qty = 100          1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
85  08:37:37  select * from spj_mng_1.v_spj LIMIT 0, 1000                7 row(s) returned
86  08:37:37  select * from spj_mng_1.spj LIMIT 0, 1000                  19 row(s) returned
87  08:42:08  Select pno,qty from spj_mng_1.spj LIMIT 0, 1000            19 row(s) returned
```

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



```
Select pno,qty
from spj_mng_1.spj
where sno='S1';
```

View: v_spj

Columns:

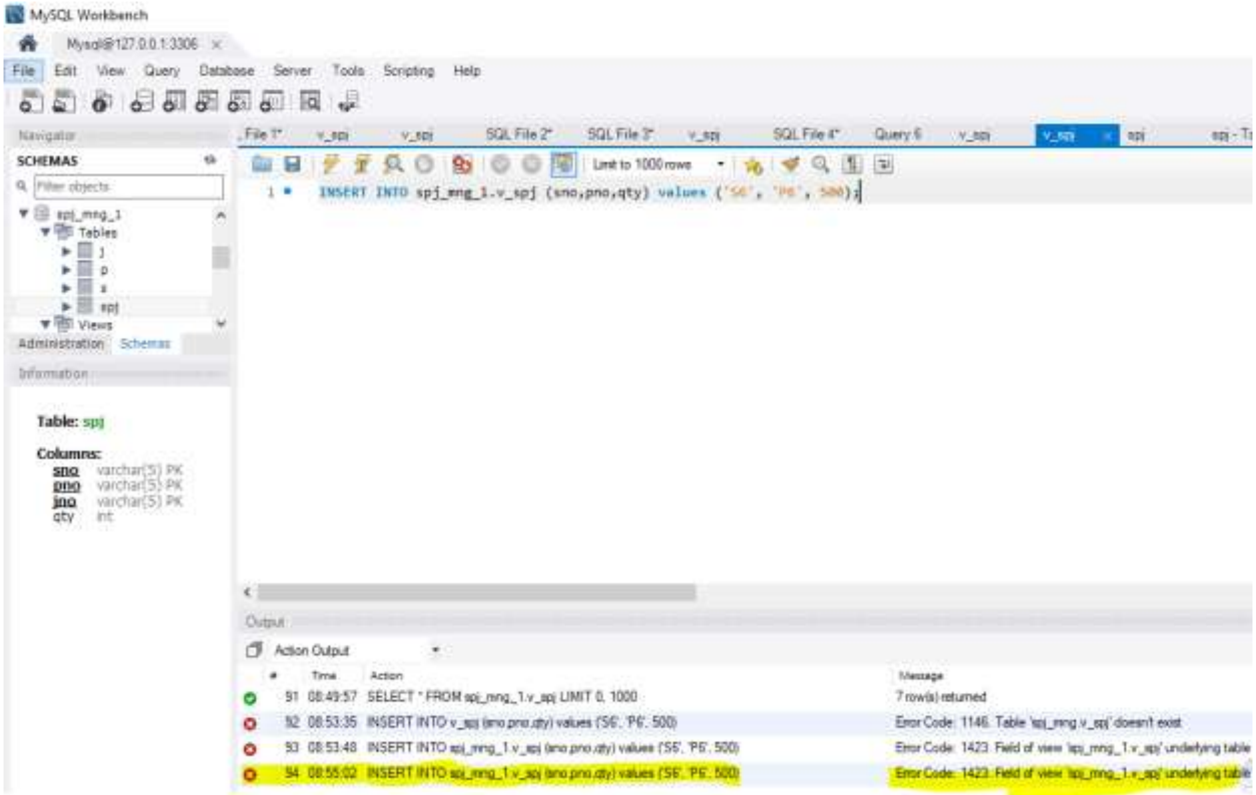
sno	varchar(5)
pno	varchar(5)
qty	int

pno	qty
P1	200
P1	100
P1	700
P2	100

#	Time	Action	Message
86	08:37:37	select * from spj_mng_1.spj LIMIT 0, 1000	19 row(s) returned
87	08:42:08	Select pno,qty from spj_mng_1.spj LIMIT 0, 1000	19 row(s) returned
88	08:43:50	Select pno,qty from spj_mng_1.spj where sno='SN1' LIMIT 0, 1000	0 row(s) returned
89	08:44:49	Select pno,qty from spj_mng_1.v_spj where sno='SN1' LIMIT 0, 1000	0 row(s) returned
90	08:47:57	Select pno,qty from spj_mng_1.spj where sno='S1' LIMIT 0, 1000	4 row(s) returned

4. Update the data of views with SQL statement.

(1) Insert a tuple into the view V_SPJ.



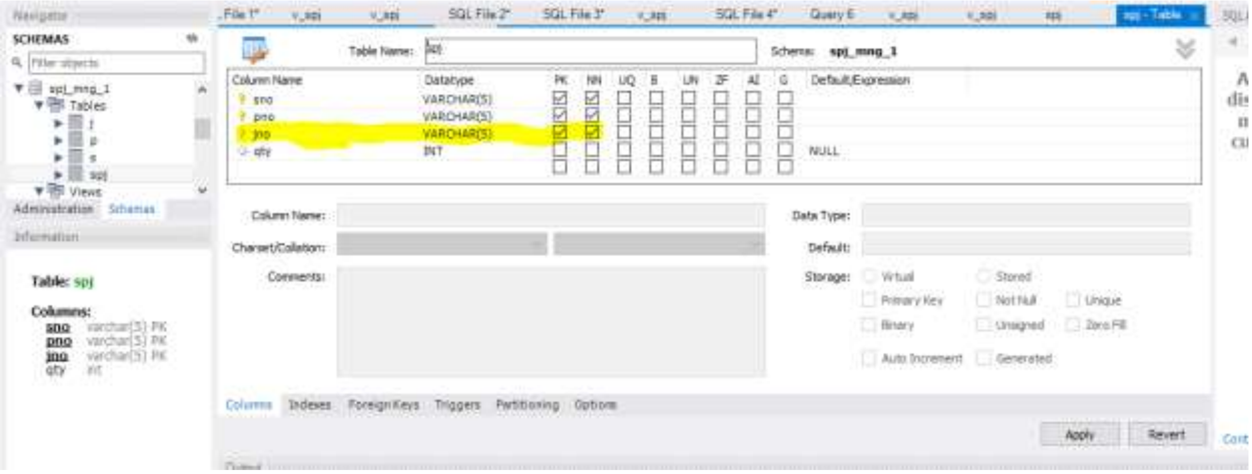
```
INSERT INTO spj_mng_1.v_spj (sno,pno,qty) values ('S6', 'P6', 500);
```

Table: spj

Columns:
sno varchar(5) PK
pno varchar(5) PK
jno varchar(5) PK
qty int

```
#   Time      Action                                                             Message
91  08:49:57  SELECT * FROM spj_mng_1.v_spj LIMIT 0, 1000                        7 row(s) returned
92  08:53:35  INSERT INTO v_spj (sno,pno,qty) values ('S6', 'P6', 500)            Error Code: 1146. Table 'spj_mng.v_spj' doesn't exist
93  08:53:48  INSERT INTO spj_mng_1.v_spj (sno,pno,qty) values ('S5', 'P6', 500)  Error Code: 1423. Field of view 'spj_mng_1.v_spj' underlying table
94  08:55:02  INSERT INTO spj_mng_1.v_spj (sno,pno,qty) values ('S6', 'P6', 500)  Error Code: 1423. Field of view 'spj_mng_1.v_spj' underlying table
```

JNO is set to Not Null



Navigator
SCHEMAS
Filter objects
spj_mng_1
Tables
j
p
s
spj
Views
Administration Schemas
Information
File 1* v_spj v_spj SQL File 2* SQL File 3* v_spj SQL File 4* Query 6 v_spj v_spj spj spj - Table
Columns Indexes Foreign Keys Triggers Partitioning Options
Apply
Revert

Table Name: spj   Schema: spj_mng_1

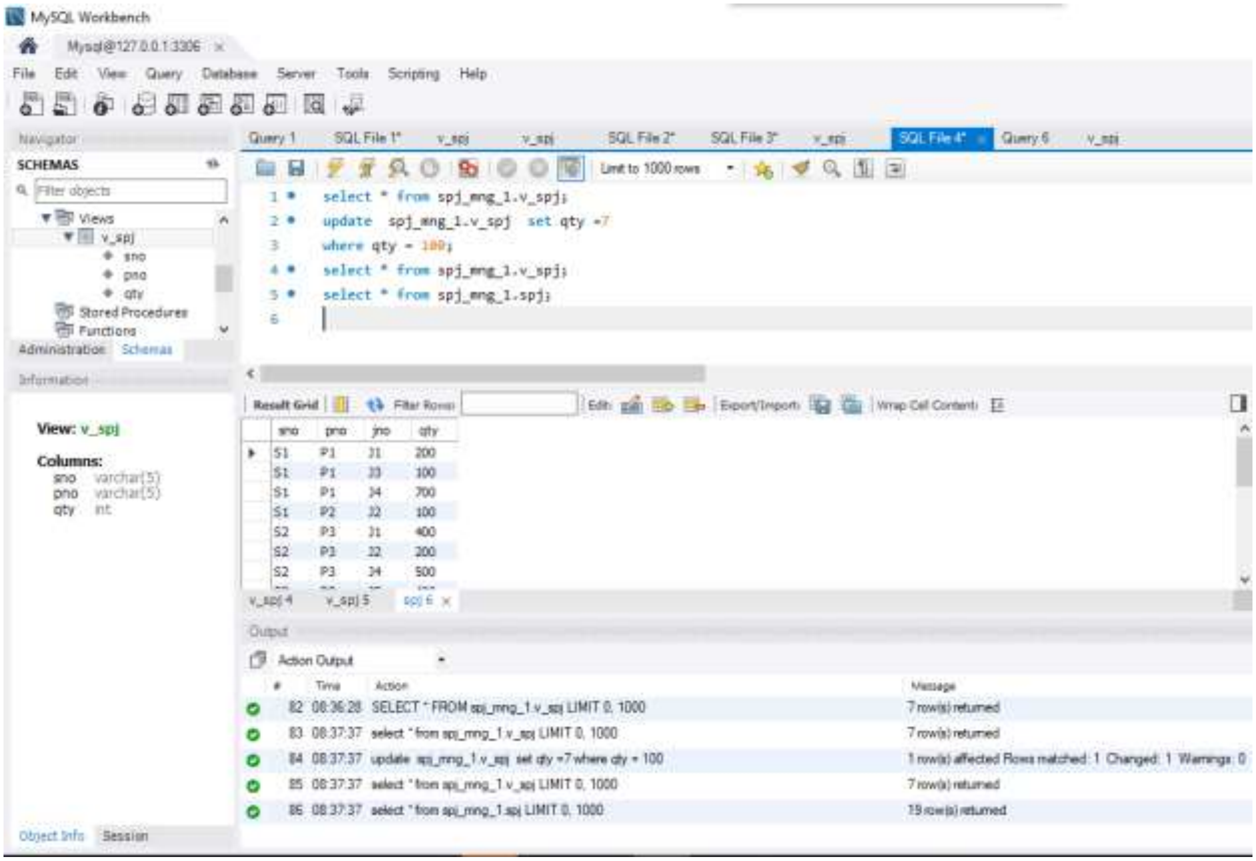
Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
sno	VARCHAR(5)	☑	☑	☐	☐	☐	☐	☐	☐	
pno	VARCHAR(5)	☑	☑	☐	☐	☐	☐	☐	☐	
jno	VARCHAR(5)	☑	☑	☐	☐	☐	☐	☐	☐	
qty	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
		☐	☐	☐	☐	☐	☐	☐	☐	

Column Name:
Charset/Collation:
Comments:
Data Type:
Default:
Storage: ○ Virtual ○ Stored
☐ Primary Key ☐ Not Null ☐ Unique
☐ Binary ☐ Unsigned ☐ Zero Fill
☐ Auto Increment ☐ Generated

Table: spj

Columns:
sno varchar(5) PK
pno varchar(5) PK
jno varchar(5) PK
qty int

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



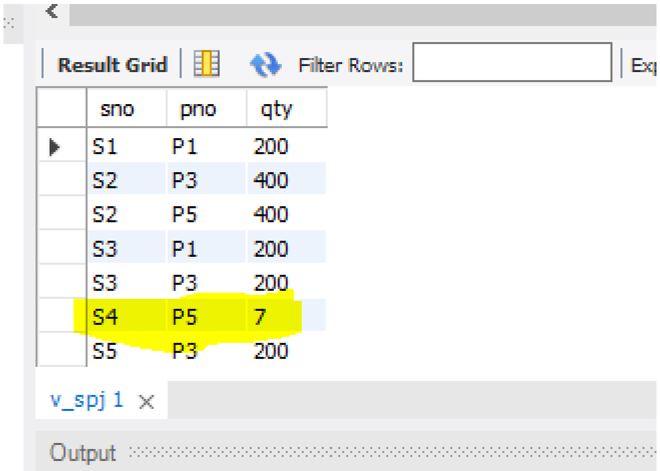
View: v_spj

Columns:
sno varchar(5)
pno varchar(5)
qty int

```
select * from spj_mng_1.v_spj;
update  spj_mng_1.v_spj  set qty =7
where qty = 100;
select * from spj_mng_1.v_spj;
select * from spj_mng_1.spj;
```

sno	pno	jno	qty
S1	P1	J1	200
S1	P1	J3	100
S1	P1	J4	700
S1	P2	J2	100
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J4	500

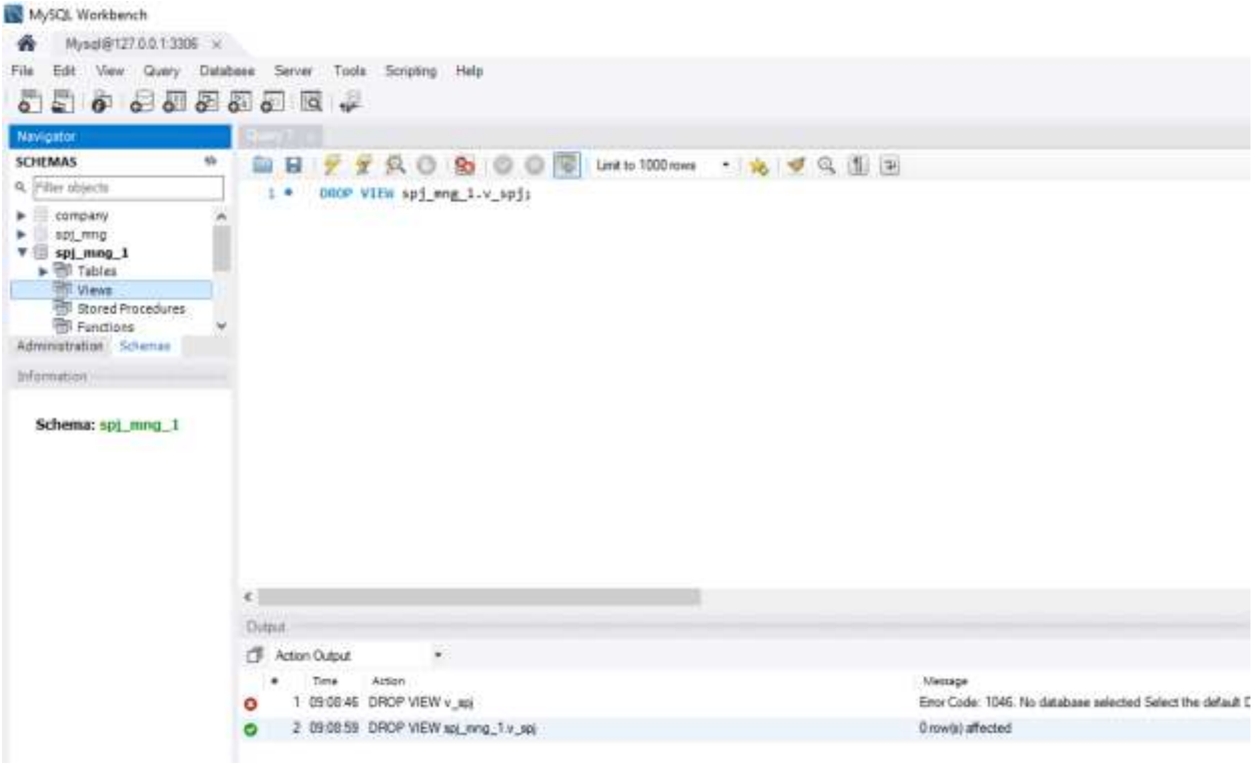
```
#   Time      Action                                                  Message
82  08:36:28  SELECT * FROM spj_mng_1.v_spj LIMIT 0, 1000              7 row(s) returned
83  08:37:37  select * from spj_mng_1.v_spj LIMIT 0, 1000              7 row(s) returned
84  08:37:37  update  spj_mng_1.v_spj  set qty =7 where qty = 100      1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
85  08:37:37  select * from spj_mng_1.v_spj LIMIT 0, 1000              7 row(s) returned
86  08:37:37  select * from spj_mng_1.spj LIMIT 0, 1000                19 row(s) returned
```



sno	pno	qty
S1	P1	200
S2	P3	400
S2	P5	400
S3	P1	200
S3	P3	200
S4	P5	7
S5	P3	200

(3) 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.

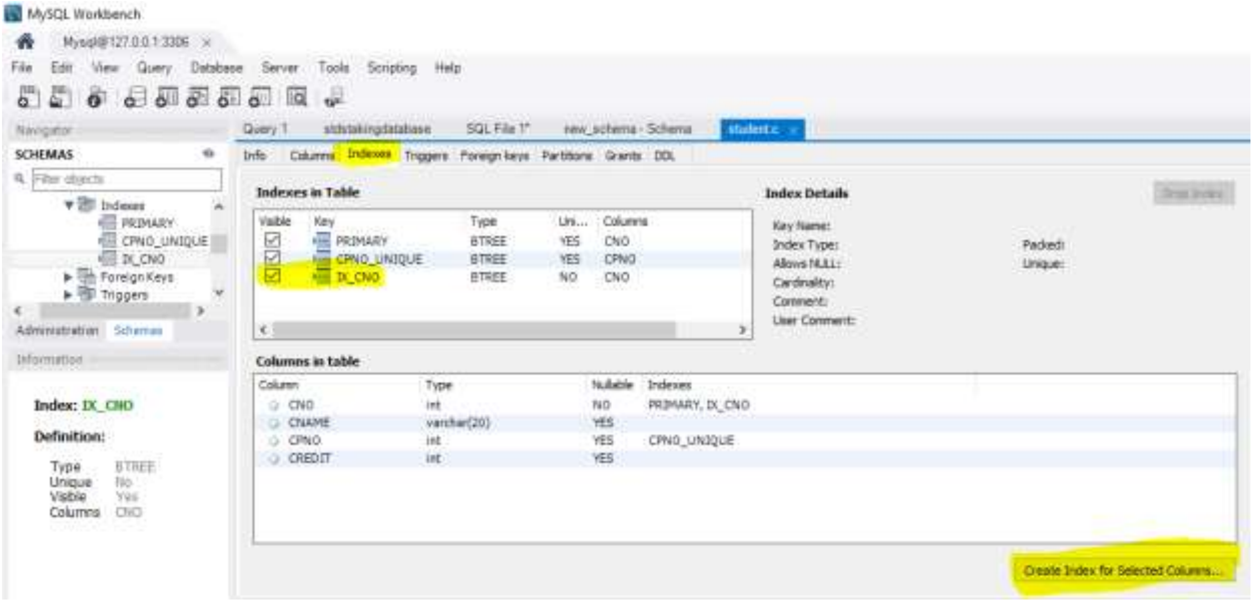


Schema: spj_mng_1

```
DROP VIEW spj_mng_1.v_spj;
```

```
    #  Time      Action                        Message
    1  09:08:46  DROP VIEW v_spj               Error Code: 1046. No database selected Select the default D
    2  09:08:59  DROP VIEW spj_mng_1.v_spj     0 row(s) affected
```

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



Index: IX_CNO

Definition:

Type	BTREE
Unique	No
Visible	Yes
Columns	CNO

Indexes in Table

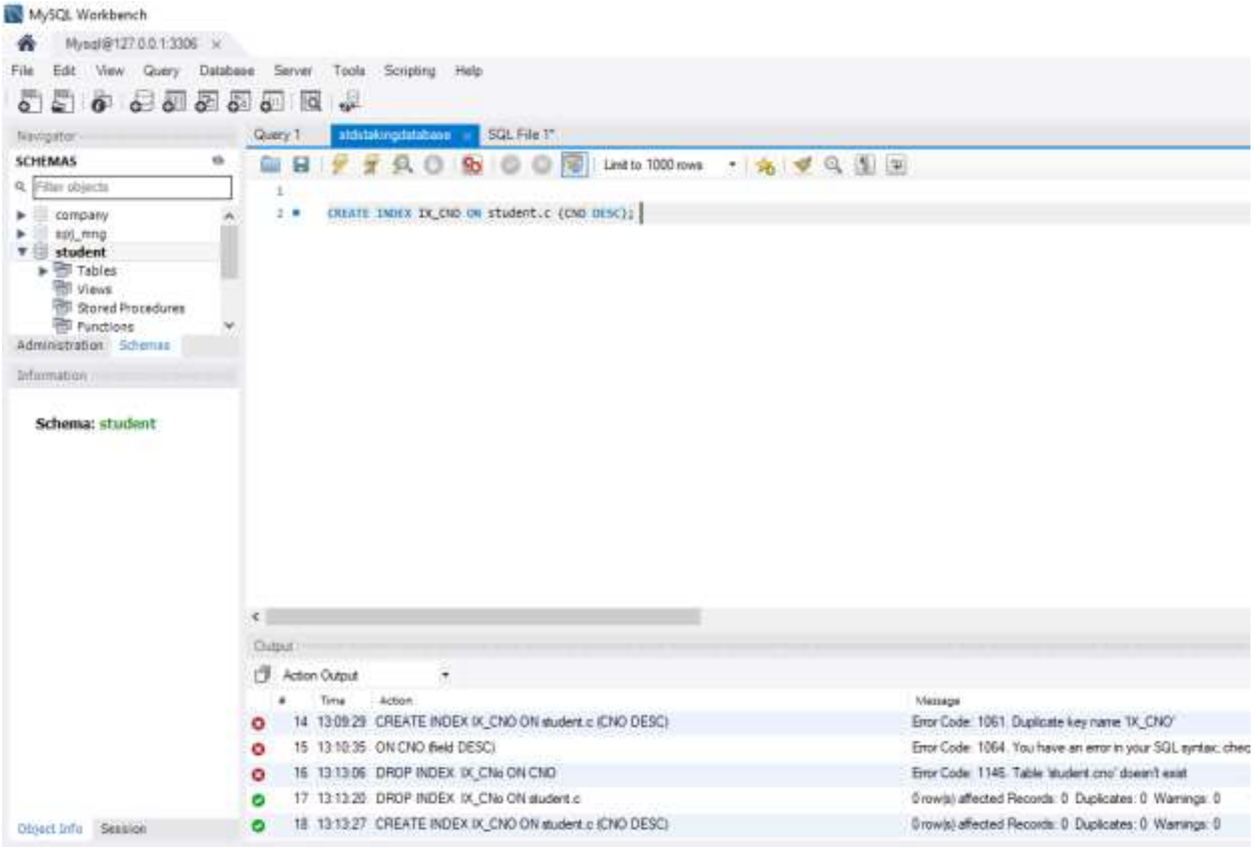
Visible	Key	Type	Uni...	Columns
☑	PRIMARY	BTREE	YES	CNO
☑	CPNO_UNIQUE	BTREE	YES	CPNO
☑	IX_CNO	BTREE	NO	CNO

Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:
Packed:
Unique:

Columns in table

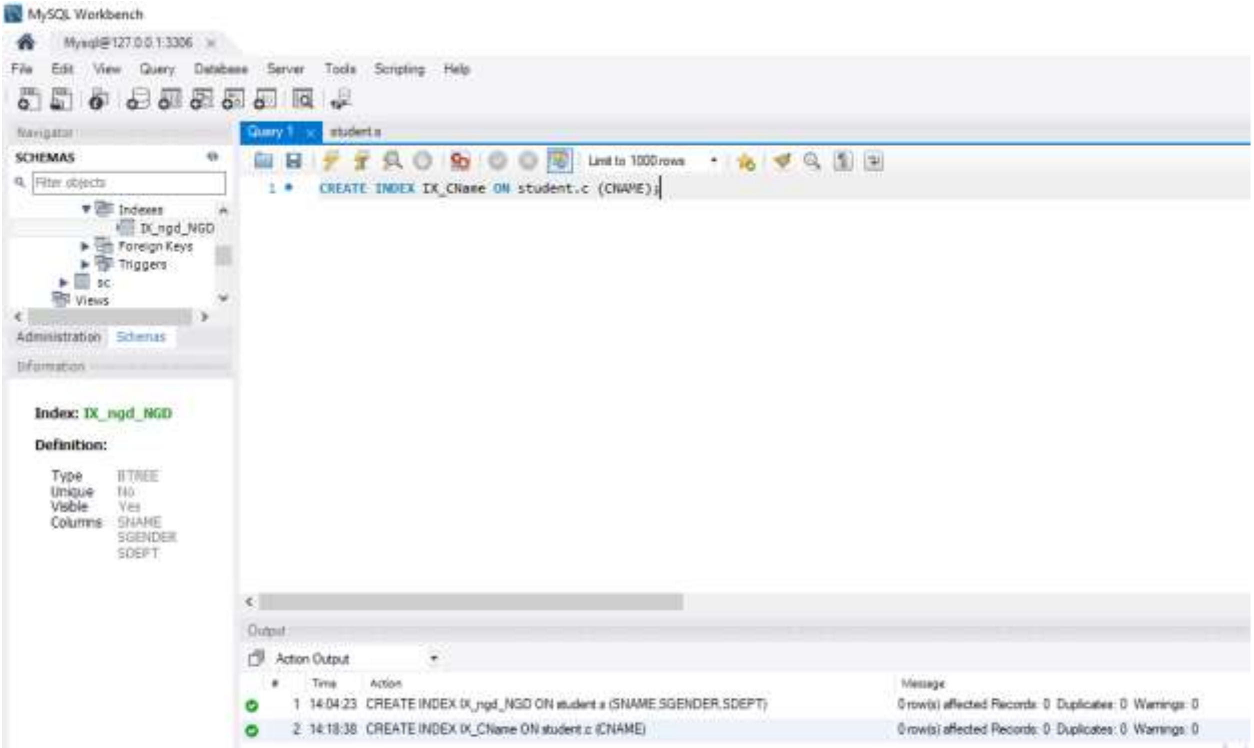
Column	Type	Nullable	Indexes
CNO	int	NO	PRIMARY, IX_CNO
CNAME	varchar(20)	YES	
CPNO	int	YES	CPNO_UNIQUE
CREDIT	int	YES	



```sql
CREATE INDEX IX_CNO ON student.c (CNO DESC);
```

```
 #   Time      Action                                        Message
14   13:09:29  CREATE INDEX IX_CNO ON student.c (CNO DESC)   Error Code: 1061. Duplicate key name 'IX_CNO'
15   13:10:35  ON CNO (field DESC)                           Error Code: 1064. You have an error in your SQL syntax; chec
16   13:13:06  DROP INDEX IX_CNo ON CNO                      Error Code: 1146. Table 'student.cno' doesn't exist
17   13:13:20  DROP INDEX IX_CNo ON student.c                0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
18   13:13:27  CREATE INDEX IX_CNO ON student.c (CNO DESC)   0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```

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.



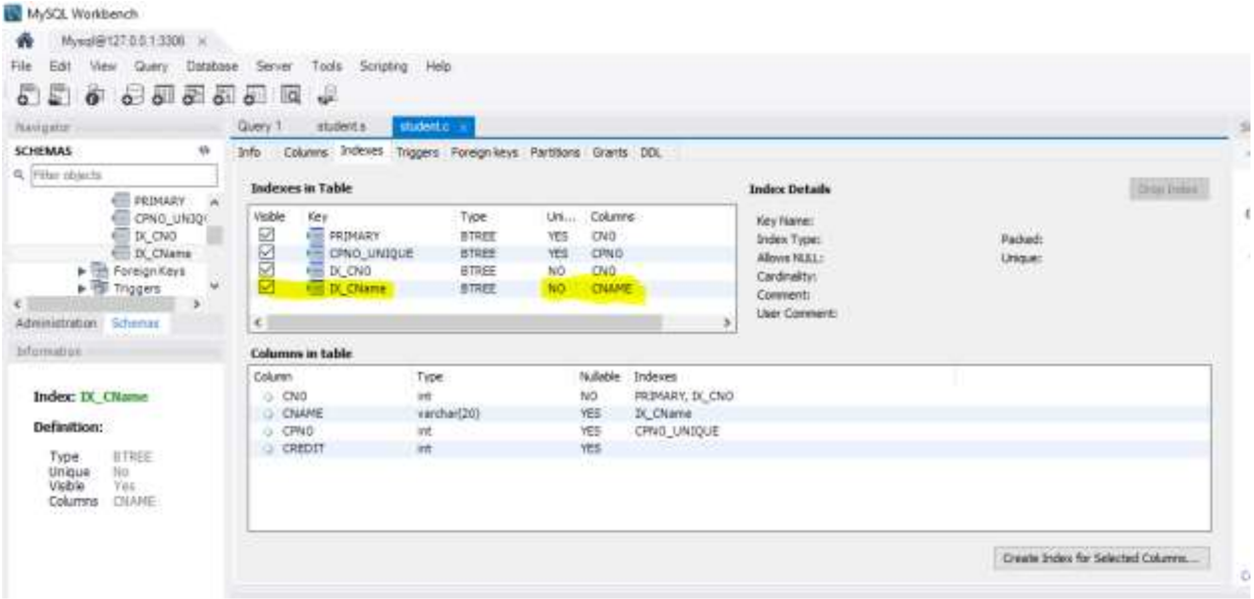
```
CREATE INDEX IX_CName ON student.c (CNAME);
```

Index: IX_ngd_NGD

Definition:

Type	BTREE
Unique	No
Visible	Yes
Columns	SNAME SGENDER SDEPT

```
#   Time      Action                                                        Message
1   14:04:23  CREATE INDEX IX_ngd_NGD ON student.s (SNAME,SGENDER,SDEPT)    0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
2   14:18:38  CREATE INDEX IX_CName ON student.c (CNAME)                    0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```



Indexes in Table

Visible	Key	Type	Uni...	Columns
☑	PRIMARY	BTREE	YES	CNO
☑	CPNO_UNIQUE	BTREE	YES	CPNO
☑	IX_CNO	BTREE	NO	CNO
☑	IX_CName	BTREE	NO	CNAME

Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:
Packed:
Unique:

Columns in table

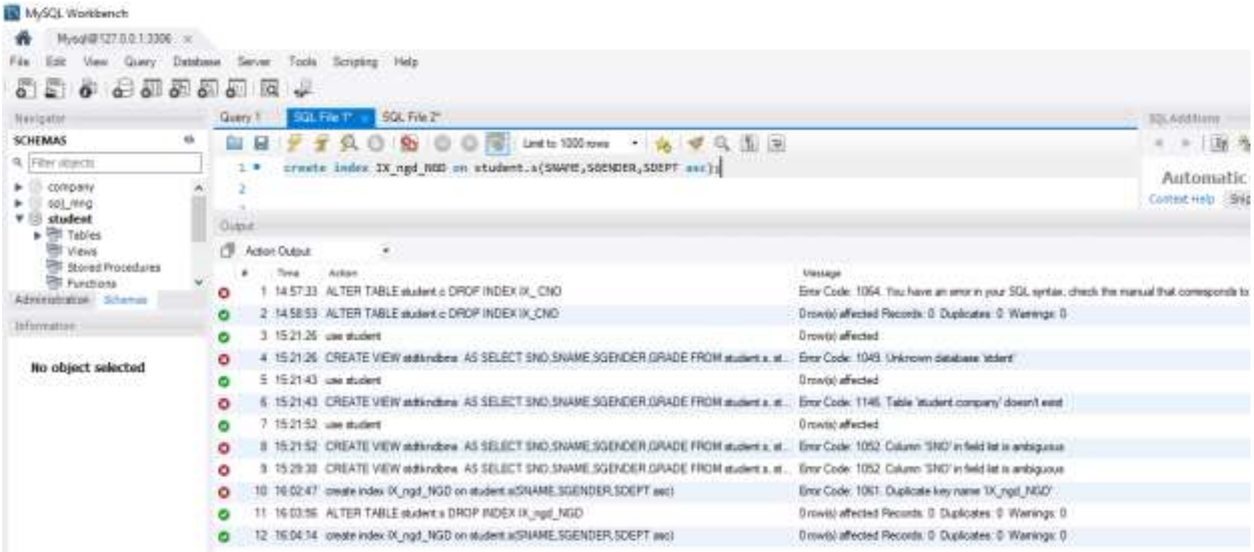
Column	Type	Nullable	Indexes
CNO	int	NO	PRIMARY, IX_CNO
CNAME	varchar(20)	YES	IX_CName
CPNO	int	YES	CPNO_UNIQUE
CREDIT	int	YES	

Index: IX_CName

Definition:

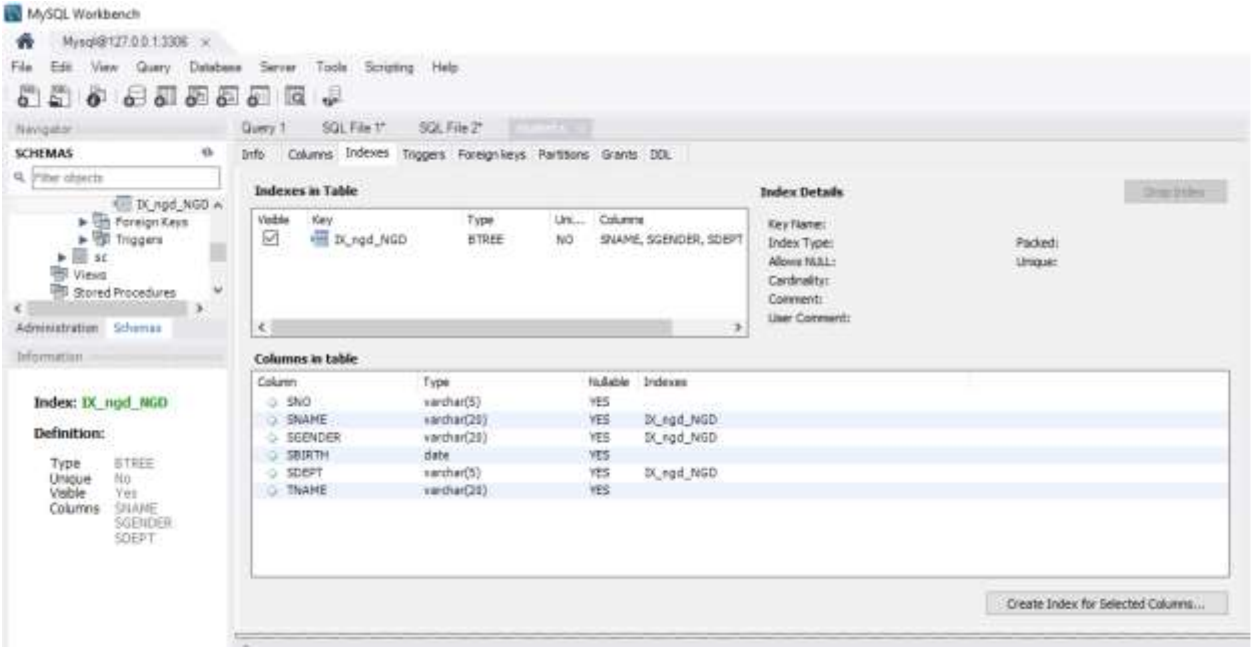
Type	BTREE
Unique	No
Visible	Yes
Columns	CNAME

(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.



```
create index IX_ngd_NGD on student.s(SNAME,SGENDER,SDEPT asc);
```

```
#   Time      Action                                                                      Message
1   14:57:33  ALTER TABLE student.c DROP INDEX IX_ CNO                                    Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to
2   14:58:53  ALTER TABLE student.c DROP INDEX IX_CNO                                     0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
3   15:21:26  use student                                                                 0 row(s) affected
4   15:21:26  CREATE VIEW stdkndbma  AS SELECT SNO,SNAME,SGENDER,GRADE FROM student.s, st...  Error Code: 1049. Unknown database 'stdent'
5   15:21:43  use student                                                                 0 row(s) affected
6   15:21:43  CREATE VIEW stdkndbma  AS SELECT SNO,SNAME,SGENDER,GRADE FROM student.s, st...  Error Code: 1146. Table 'student.company' doesn't exist
7   15:21:52  use student                                                                 0 row(s) affected
8   15:21:52  CREATE VIEW stdkndbma  AS SELECT SNO,SNAME,SGENDER,GRADE FROM student.s, st...  Error Code: 1052. Column 'SNO' in field list is ambiguous
9   15:29:38  CREATE VIEW stdkndbma  AS SELECT SNO,SNAME,SGENDER,GRADE FROM student.s, st...  Error Code: 1052. Column 'SNO' in field list is ambiguous
10  16:02:47  create index IX_ngd_NGD on student.s(SNAME,SGENDER,SDEPT asc)                Error Code: 1061. Duplicate key name 'IX_ngd_NGD'
11  16:03:56  ALTER TABLE student.s DROP INDEX IX_ngd_NGD                                 0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
12  16:04:14  create index IX_ngd_NGD on student.s(SNAME,SGENDER,SDEPT asc)                0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```



Index: IX_ngd_NGD

Definition:

Type BTREE
Unique No
Visible Yes
Columns SNAME
SGENDER
SDEPT

Indexes in Table

Visible	Key	Type	Uni...	Columns
☑	IX_ngd_NGD	BTREE	NO	SNAME, SGENDER, SDEPT

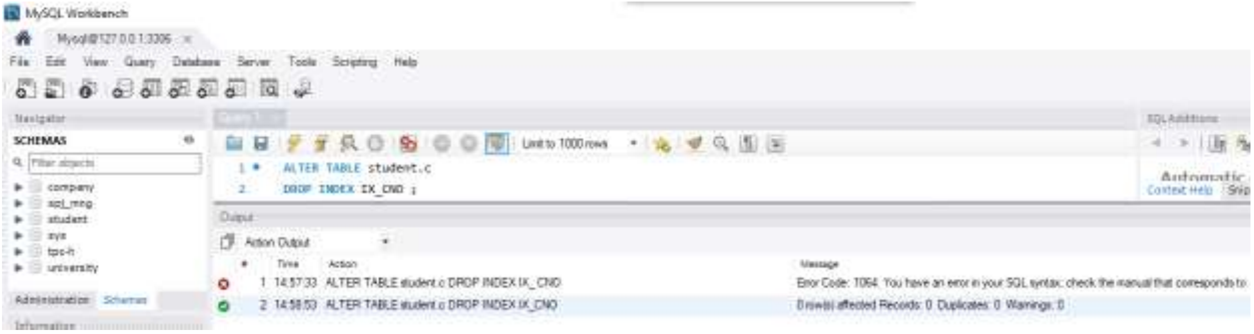
Index Details

Key Name:
Index Type:
Allows NULL:
Cardinality:
Comment:
User Comment:
Packed:
Unique:

Columns in table

Column	Type	Nullable	Indexes
SNO	varchar(5)	YES	
SNAME	varchar(20)	YES	IX_ngd_NGD
SGENDER	varchar(20)	YES	IX_ngd_NGD
SBIRTH	date	YES	
SDEPT	varchar(5)	YES	IX_ngd_NGD
TNAME	varchar(20)	YES	

(2) Delete index IX_ CNO of table C。

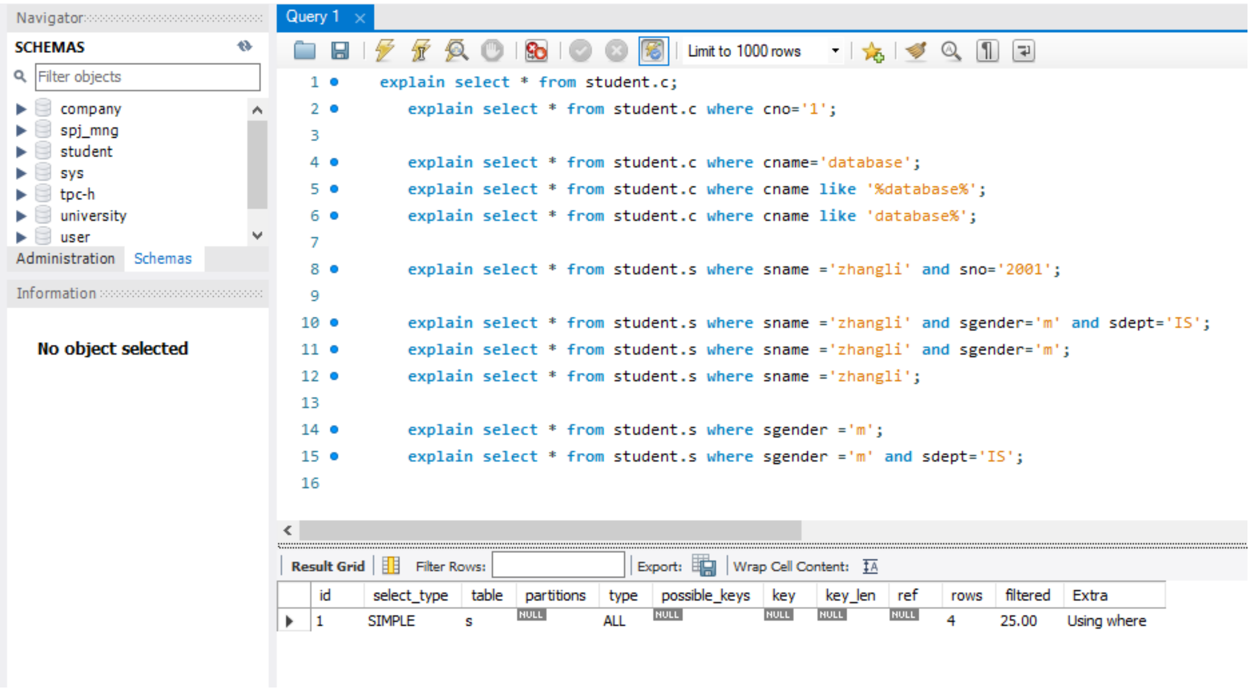


```
ALTER TABLE student.c
DROP INDEX IX_CNO ;
```

```
#  Time      Action                                    Message
1  14:57:33  ALTER TABLE student.c DROP INDEX IX_ CNO  Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to
2  14:58:53  ALTER TABLE student.c DROP INDEX IX_CNO   0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```

(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.

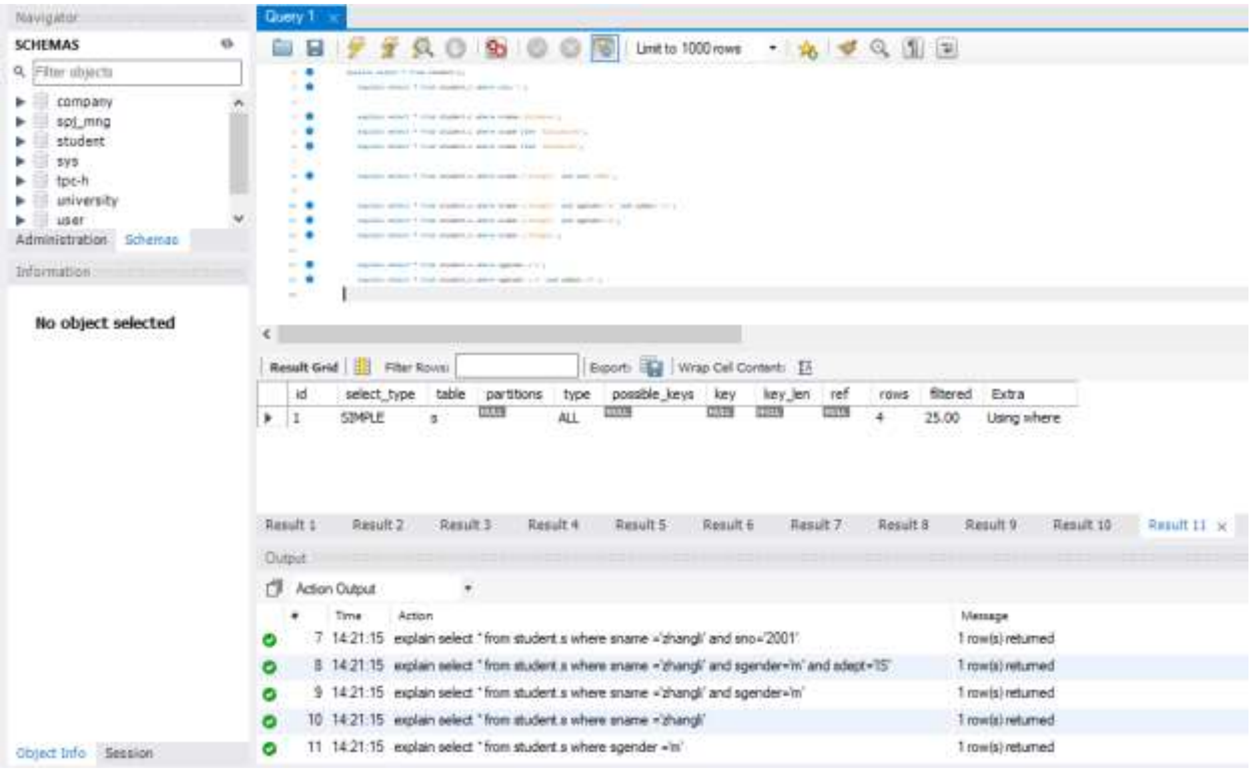
① explain select * from c;
② explain select * from c where cno = ‘1’;
③ explain select * from c where cname=’database’ ;
④ explain select * from c where cname like ‘%database%’;
⑤ explain select * from c where cname like ‘database%’;
⑥ explain select * from s where sname ='Zhangli' and sno='2001';
⑦ 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';



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

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

   explain select * from student.s where sname ='zhangli' and sno='2001';

   explain select * from student.s where sname ='zhangli' and sgender='m' and sdept='IS';
   explain select * from student.s where sname ='zhangli' and sgender='m';
   explain select * from student.s where sname ='zhangli';

   explain select * from student.s where sgender ='m';
   explain select * from student.s where sgender ='m' 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

⑫



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

```
 #  Time      Action                                                                          Message
 7  14:21:15  explain select * from student s where sname ='zhangli' and sno='2001'              1 row(s) returned
 8  14:21:15  explain select * from student s where sname ='zhangli' and sgender='m' and sdept='IS'  1 row(s) returned
 9  14:21:15  explain select * from student s where sname ='zhangli' and sgender='m'              1 row(s) returned
10  14:21:15  explain select * from student s where sname ='zhangli'                              1 row(s) returned
11  14:21:15  explain select * from student s where sgender ='m'                                  1 row(s) returned
```

7. 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

)
```

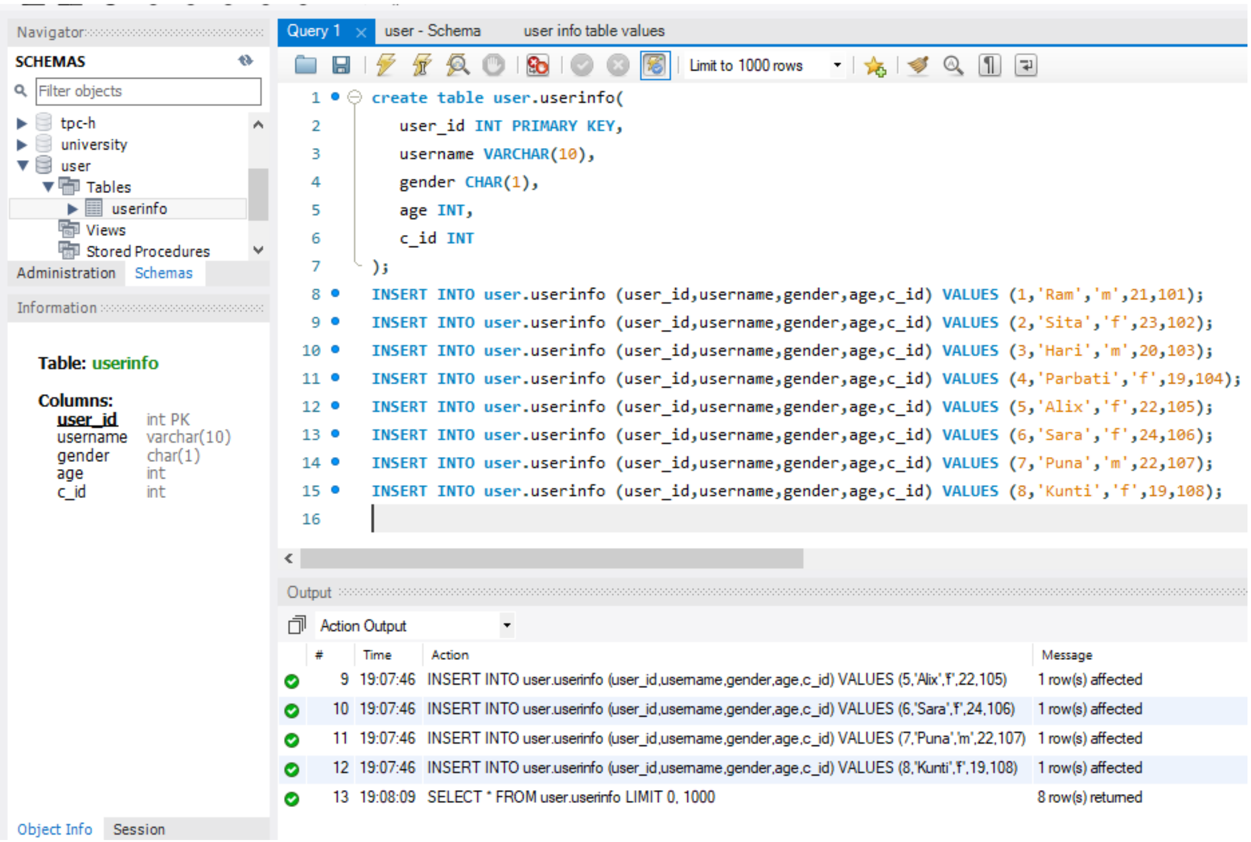


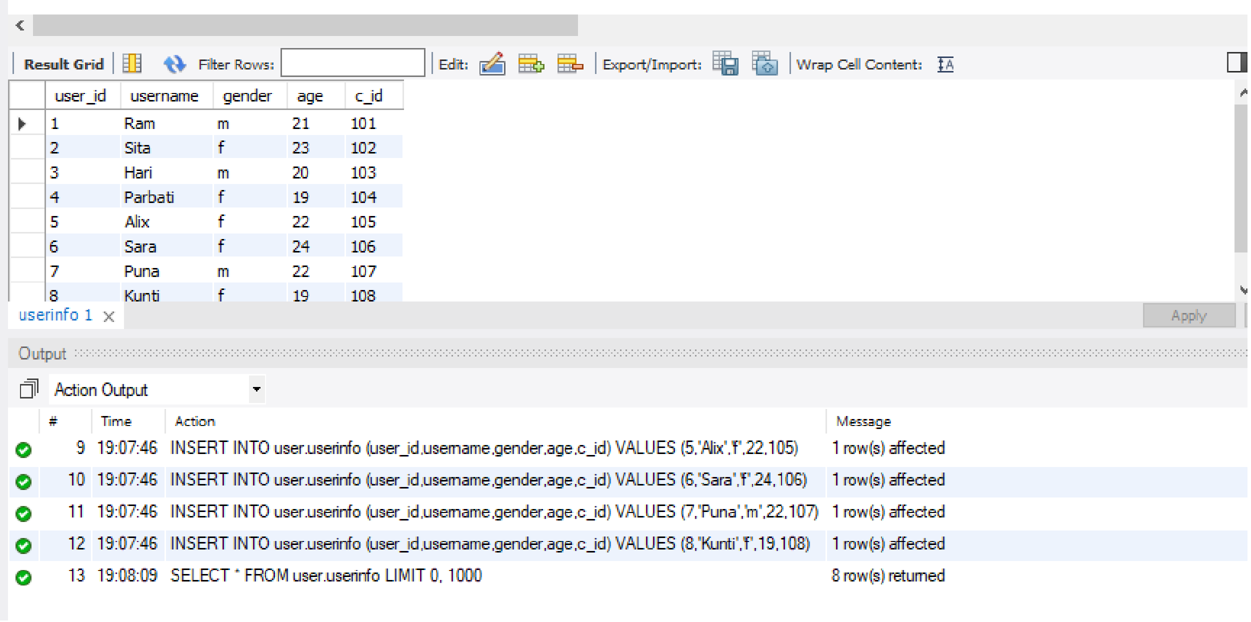
Table: userinfo

Columns:

user_id	int PK
username	varchar(10)
gender	char(1)
age	int
c_id	int

```sql
create table user.userinfo(
    user_id INT PRIMARY KEY,
    username VARCHAR(10),
    gender CHAR(1),
    age INT,
    c_id INT
);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (1,'Ram','m',21,101);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (2,'Sita','f',23,102);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (3,'Hari','m',20,103);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (4,'Parbati','f',19,104);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (5,'Alix','f',22,105);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (6,'Sara','f',24,106);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (7,'Puna','m',22,107);
INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (8,'Kunti','f',19,108);
```

```
#   Time      Action                                                                                    Message
9   19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (5,'Alix','f',22,105)     1 row(s) affected
10  19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (6,'Sara','f',24,106)     1 row(s) affected
11  19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (7,'Puna','m',22,107)     1 row(s) affected
12  19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (8,'Kunti','f',19,108)    1 row(s) affected
13  19:08:09  SELECT * FROM user.userinfo LIMIT 0, 1000                                                     8 row(s) returned
```

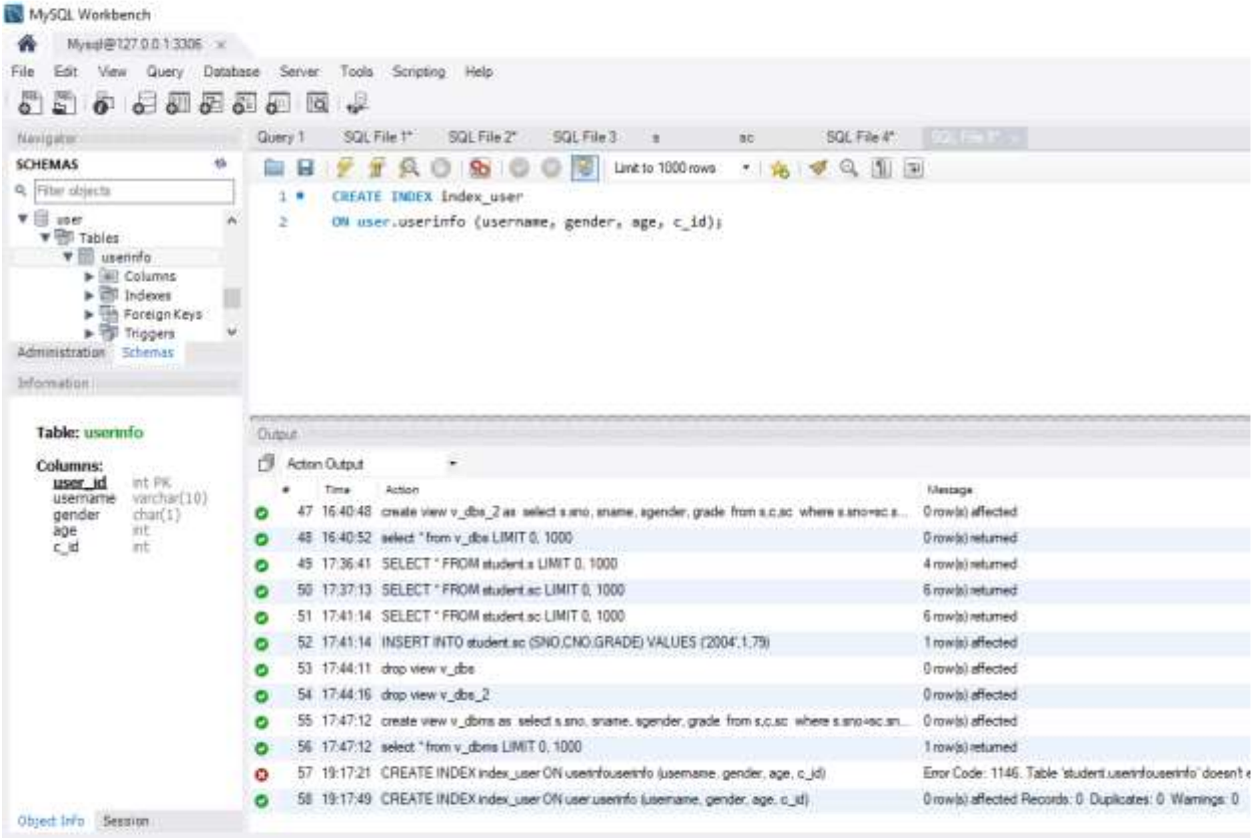


user_id	username	gender	age	c_id
1	Ram	m	21	101
2	Sita	f	23	102
3	Hari	m	20	103
4	Parbati	f	19	104
5	Alix	f	22	105
6	Sara	f	24	106
7	Puna	m	22	107
8	Kunti	f	19	108

```
#   Time      Action                                                                                    Message
9   19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (5,'Alix','f',22,105)     1 row(s) affected
10  19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (6,'Sara','f',24,106)     1 row(s) affected
11  19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (7,'Puna','m',22,107)     1 row(s) affected
12  19:07:46  INSERT INTO user.userinfo (user_id,username,gender,age,c_id) VALUES (8,'Kunti','f',19,108)    1 row(s) affected
13  19:08:09  SELECT * FROM user.userinfo LIMIT 0, 1000                                                     8 row(s) returned
```

1. Verify the efficiency difference between indexed and non_indexed queries.

Indexed



```
CREATE INDEX index_user
ON user.userinfo (username, gender, age, c_id);
```

Table: userinfo

Columns:

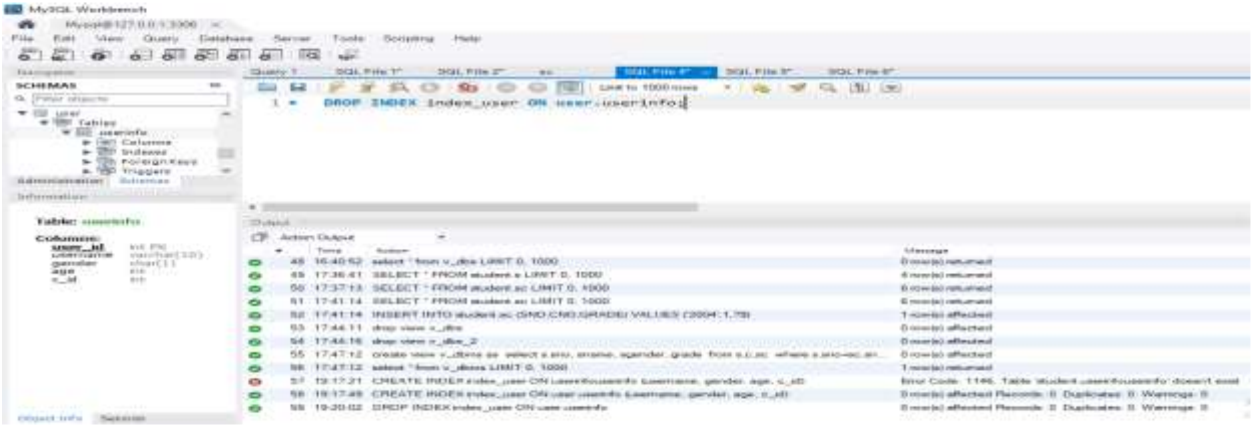
```
user_id    int PK
username   varchar(10)
gender     char(1)
age        int
c_id       int
```

Output

Action Output

```
#   Time      Action                                                                      Message
47  16:40:48  create view v_dbs_2 as  select s.sno, sname, sgender, grade  from s,c,sc  where s.sno=sc.s...  0 row(s) affected
48  16:40:52  select * from v_dbs LIMIT 0, 1000                                           0 row(s) returned
49  17:36:41  SELECT * FROM student.s LIMIT 0, 1000                                       4 row(s) returned
50  17:37:13  SELECT * FROM student.sc LIMIT 0, 1000                                      6 row(s) returned
51  17:41:14  SELECT * FROM student.sc LIMIT 0, 1000                                      6 row(s) returned
52  17:41:14  INSERT INTO student.sc (SNO,CNO,GRADE) VALUES ('2004',1,79)                 1 row(s) affected
53  17:44:11  drop view v_dbs                                                             0 row(s) affected
54  17:44:16  drop view v_dbs_2                                                           0 row(s) affected
55  17:47:12  create view v_dbms as  select s.sno, sname, sgender, grade  from s,c,sc  where s.sno=sc.sn...  0 row(s) affected
56  17:47:12  select * from v_dbms LIMIT 0, 1000                                          1 row(s) returned
57  19:17:21  CREATE INDEX index_user ON userinfouserinfo (username, gender, age, c_id)   Error Code: 1146. Table 'student.userinfouserinfo' doesn't e
58  19:17:49  CREATE INDEX index_user ON user.userinfo (username, gender, age, c_id)      0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```

Drop index:



```
DROP INDEX index_user ON user.userinfo;
```

Table: userinfo

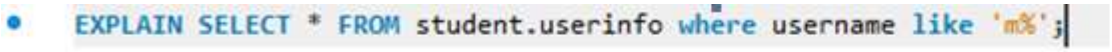
Columns:

```
user_id    int PK
username   varchar(10)
gender     char(1)
age        int
c_id       int
```

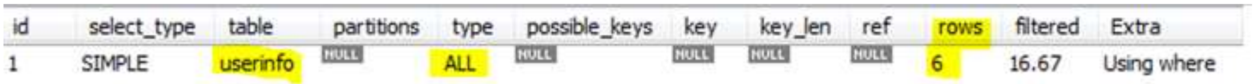
Output

Action Output

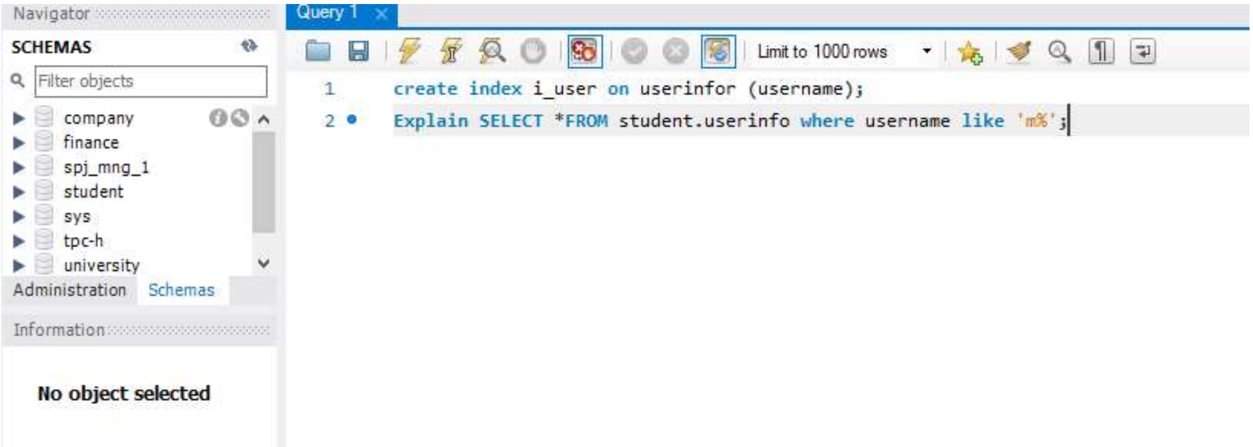
```
#   Time      Action                                                                      Message
48  16:40:52  select * from v_dbs LIMIT 0, 1000                                           0 row(s) returned
49  17:36:41  SELECT * FROM student.s LIMIT 0, 1000                                       4 row(s) returned
50  17:37:13  SELECT * FROM student.sc LIMIT 0, 1000                                      6 row(s) returned
51  17:41:14  SELECT * FROM student.sc LIMIT 0, 1000                                      6 row(s) returned
52  17:41:14  INSERT INTO student.sc (SNO,CNO,GRADE) VALUES ('2004',1,79)                 1 row(s) affected
53  17:44:11  drop view v_dbs                                                             0 row(s) affected
54  17:44:16  drop view v_dbs_2                                                           0 row(s) affected
55  17:47:12  create view v_dbms as  select s.sno, sname, sgender, grade  from s,c,sc  where s.sno=sc.sn...  0 row(s) affected
56  17:47:12  select * from v_dbms LIMIT 0, 1000                                          1 row(s) returned
57  19:17:21  CREATE INDEX index_user ON userinfouserinfo (username, gender, age, c_id)   Error Code: 1146. Table 'student.userinfouserinfo' doesn't exist
58  19:17:49  CREATE INDEX index_user ON user.userinfo (username, gender, age, c_id)      0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
59  19:20:02  DROP INDEX index_user ON user.userinfo                                      0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```



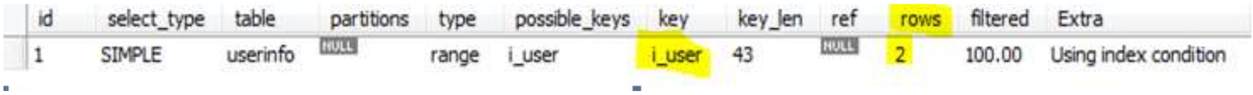
```
EXPLAIN SELECT * FROM student.userinfo where username like 'm%';
```



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



```
create index i_user on userinfor (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 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.

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.

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:

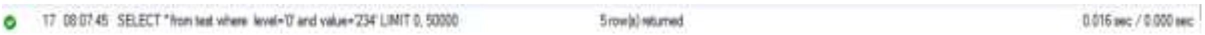
To create one multi field wide index on a table **test**:

```
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';
```



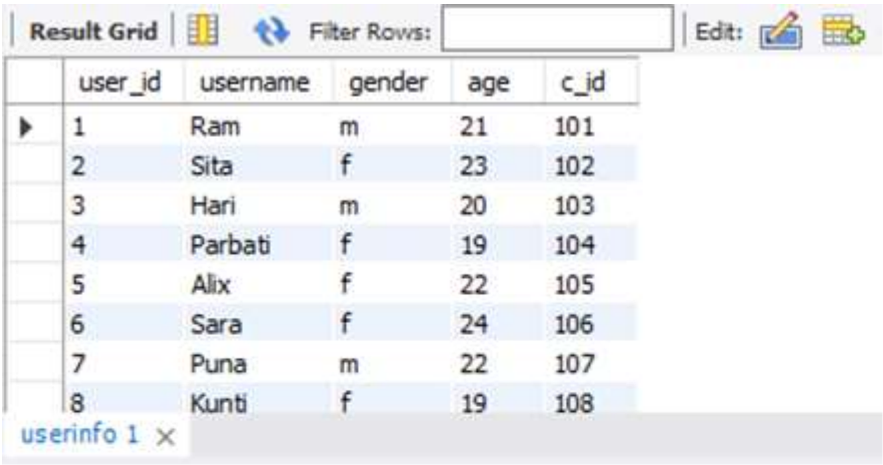
```
17  08:07:45  SELECT * from test where  level='0' and value='234' LIMIT 0, 50000        5 row(s) returned        0.016 sec / 0.000 sec
```

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:**



user_id	username	gender	age	c_id
1	Ram	m	21	101
2	Sita	f	23	102
3	Hari	m	20	103
4	Parbati	f	19	104
5	Alix	f	22	105
6	Sara	f	24	106
7	Puna	m	22	107
8	Kunti	f	19	108

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:

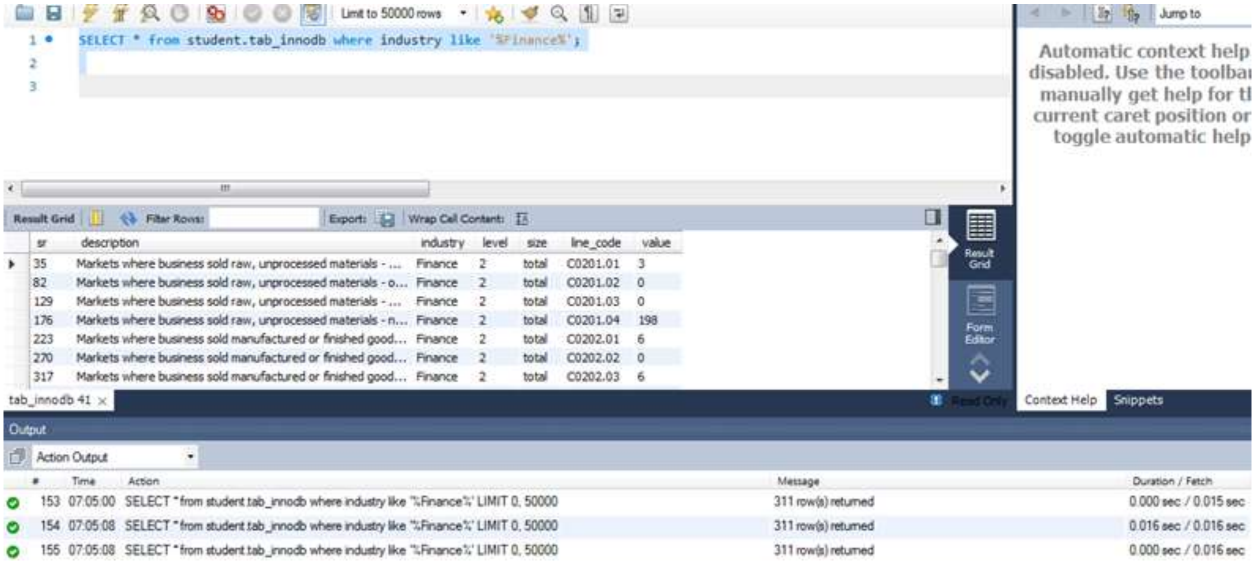
By default, when we create an index, its form is set to b-tree, which gives the best sort and helps to search for data such as binary search and others.

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



```
SELECT * from student.tab_innodb where industry like '%Finance%';
```

Automatic context help
disabled. Use the toolbar
manually get help for th
current caret position or
toggle automatic help

sr	description	industry	level	size	line_code	value
35	Markets where business sold raw, unprocessed materials - ...	Finance	2	total	C0201.01	3
82	Markets where business sold raw, unprocessed materials - o...	Finance	2	total	C0201.02	0
129	Markets where business sold raw, unprocessed materials - ...	Finance	2	total	C0201.03	0
176	Markets where business sold raw, unprocessed materials - n...	Finance	2	total	C0201.04	198
223	Markets where business sold manufactured or finished good...	Finance	2	total	C0202.01	6
270	Markets where business sold manufactured or finished good...	Finance	2	total	C0202.02	0
317	Markets where business sold manufactured or finished good...	Finance	2	total	C0202.03	6

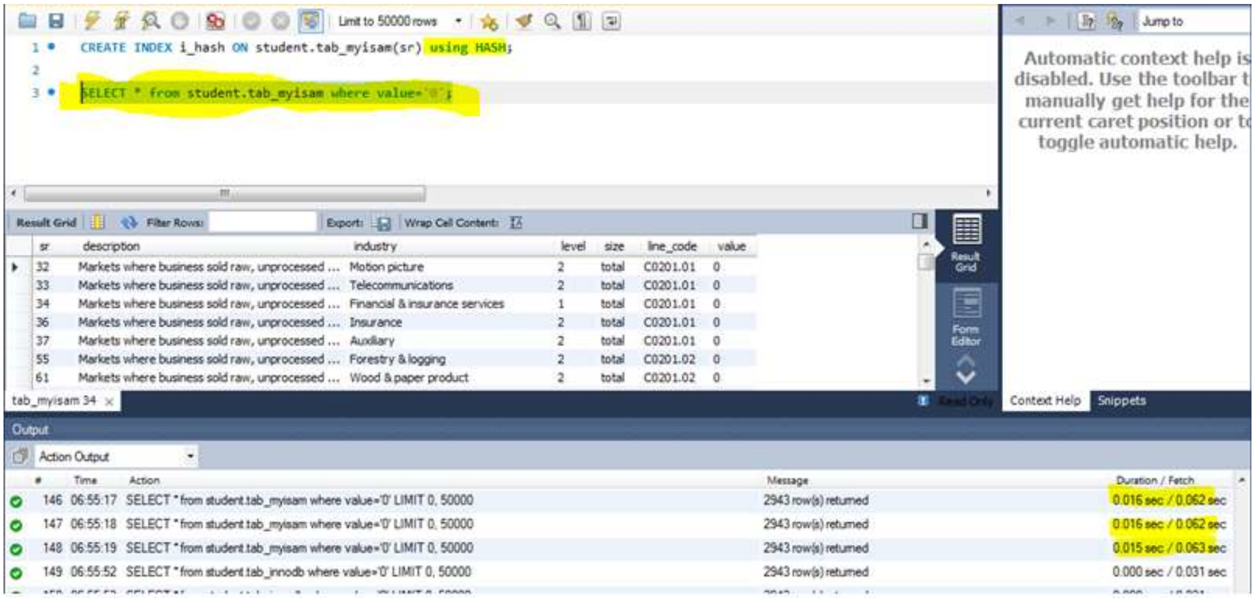
```
#    Time      Action                                                                   Message               Duration / Fetch
153  07:05:00  SELECT * from student.tab_innodb where industry like '%Finance%' LIMIT 0, 50000   311 row(s) returned   0.000 sec / 0.015 sec
154  07:05:08  SELECT * from student.tab_innodb where industry like '%Finance%' LIMIT 0, 50000   311 row(s) returned   0.016 sec / 0.016 sec
155  07:05:08  SELECT * from student.tab_innodb where industry like '%Finance%' LIMIT 0, 50000   311 row(s) returned   0.000 sec / 0.016 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.

SQL QUERY:

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



```
CREATE INDEX i_hash ON student.tab_myisam(sr) using HASH;

SELECT * from student.tab_myisam where value='0';
```

sr	description	industry	level	size	line_code	value
32	Markets where business sold raw, unprocessed ...	Motion picture	2	total	C0201.01	0
33	Markets where business sold raw, unprocessed ...	Telecommunications	2	total	C0201.01	0
34	Markets where business sold raw, unprocessed ...	Financial & insurance services	1	total	C0201.01	0
36	Markets where business sold raw, unprocessed ...	Insurance	2	total	C0201.01	0
37	Markets where business sold raw, unprocessed ...	Auxliary	2	total	C0201.01	0
55	Markets where business sold raw, unprocessed ...	Forestry & logging	2	total	C0201.02	0
61	Markets where business sold raw, unprocessed ...	Wood & paper product	2	total	C0201.02	0

```
 #   Time      Action                                                        Message               Duration / Fetch
146  06:55:17  SELECT * from student.tab_myisam where value='0' LIMIT 0, 50000  2943 row(s) returned  0.016 sec / 0.062 sec
147  06:55:18  SELECT * from student.tab_myisam where value='0' LIMIT 0, 50000  2943 row(s) returned  0.016 sec / 0.062 sec
148  06:55:19  SELECT * from student.tab_myisam where value='0' LIMIT 0, 50000  2943 row(s) returned  0.015 sec / 0.063 sec
149  06:55:52  SELECT * from student.tab_innodb where value='0' LIMIT 0, 50000  2943 row(s) returned  0.000 sec / 0.031 sec
```

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.