EC5070: Database Systems Lab 04

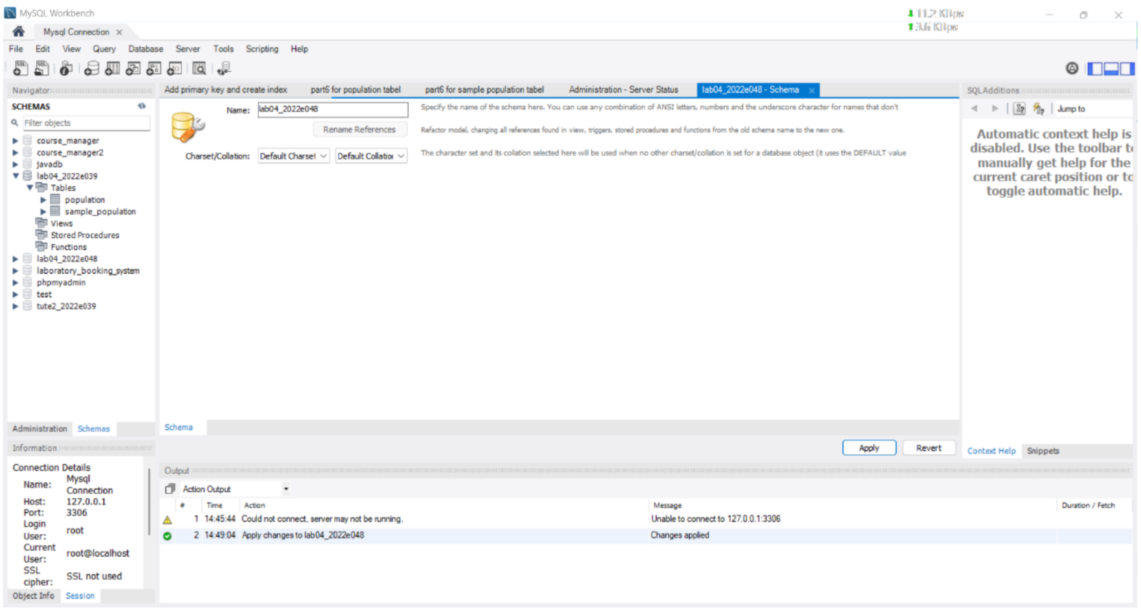
BANDARA H.G.T.D.

2022/E/048

06.06.2025

Attach the below screenshots for following scenarios.

Q1. Database with your registration number and lab number.


MySQL Workbench — Mysql Connection

SCHEMAS: course_manager, course_manager2, javadb, lab04_2022e039 (Tables: population, sample_population; Views; Stored Procedures; Functions), lab04_2022e048, laboratory_booking_system, phpmyadmin, test, tute2_2022e039

Tabs: Add primary key and create index | part6 for population tabel | part6 for sample population tabel | Administration - Server Status | lab04_2022e048 - Schema

Name: lab04_2022e048 — Specify the name of the schema here. You can use any combination of ANSI letters, numbers and the underscore character for names that don't

Rename References — Refactor model, changing all references found in view, triggers, stored procedures and functions from the old schema name to the new one.

Charset/Collation: Default Charset, Default Collation — The character set and its collation selected here will be used when no other charset/collation is set for a database object (it uses the DEFAULT value

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

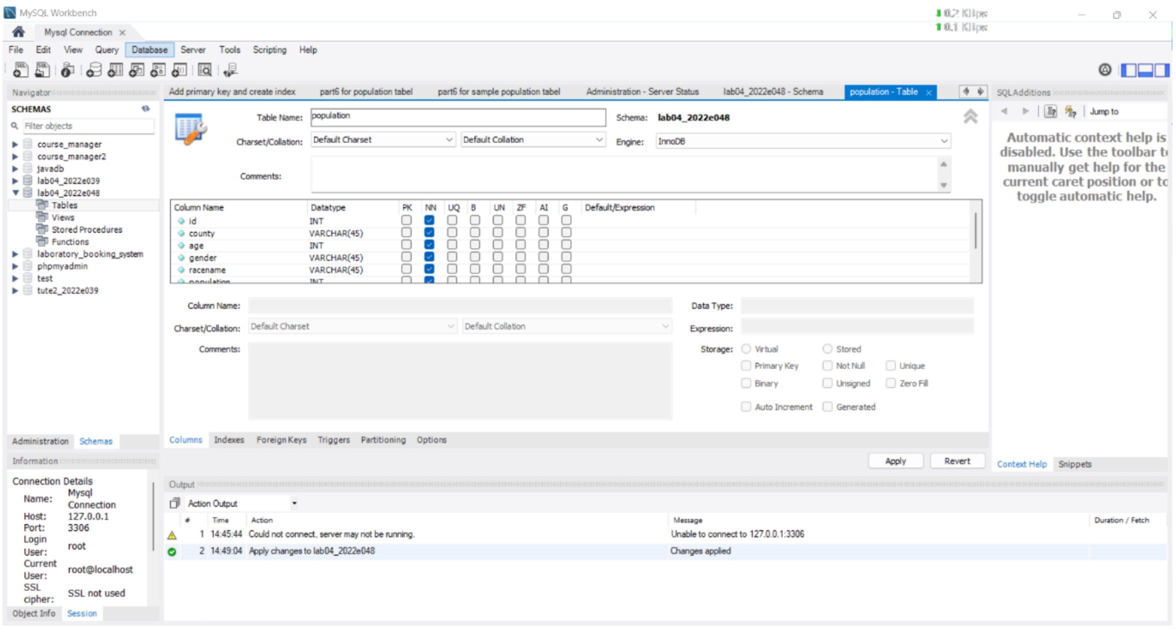
Connection Details
Name: Mysql Connection
Host: 127.0.0.1
Port: 3306
Login User: root
Current User: root@localhost
SSL cipher: SSL not used

Output — Action Output

```
#  Time      Action                                       Message                            Duration / Fetch
1  14:45:44  Could not connect, server may not be running. Unable to connect to 127.0.0.1:3306
2  14:49:04  Apply changes to lab04_2022e048              Changes applied
```

<u>FIGURE 01:CREATE DATABASE</u>

Q2. Tables with the name sample population and population.


MySQL Workbench — Mysql Connection

SCHEMAS: course_manager, course_manager2, javadb, lab04_2022e039, lab04_2022e048 (Tables, Views, Stored Procedures, Functions), laboratory_booking_system, phpmyadmin, test, tute2_2022e039

Tabs: Add primary key and create index | part6 for population tabel | part6 for sample population tabel | Administration - Server Status | lab04_2022e048 - Schema | population - Table

Table Name: population　Schema: lab04_2022e048
Charset/Collation: Default Charset, Default Collation　Engine: InnoDB
Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	☐	✓	☐	☐	☐	☐	☐	☐	
county	VARCHAR(45)	☐	✓	☐	☐	☐	☐	☐	☐	
age	INT	☐	✓	☐	☐	☐	☐	☐	☐	
gender	VARCHAR(45)	☐	✓	☐	☐	☐	☐	☐	☐	
racename	VARCHAR(45)	☐	✓	☐	☐	☐	☐	☐	☐	
population	INT	☐	✓	☐	☐	☐	☐	☐	☐	

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

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

Connection Details
Name: Mysql Connection
Host: 127.0.0.1
Port: 3306
Login User: root
Current User: root@localhost
SSL cipher: SSL not used

Output — Action Output

```
#  Time      Action                                       Message                            Duration / Fetch
1  14:45:44  Could not connect, server may not be running. Unable to connect to 127.0.0.1:3306
2  14:49:04  Apply changes to lab04_2022e048              Changes applied
```

<u>FIGURE 02:CREATE TABLE FOR POPULATION</u>

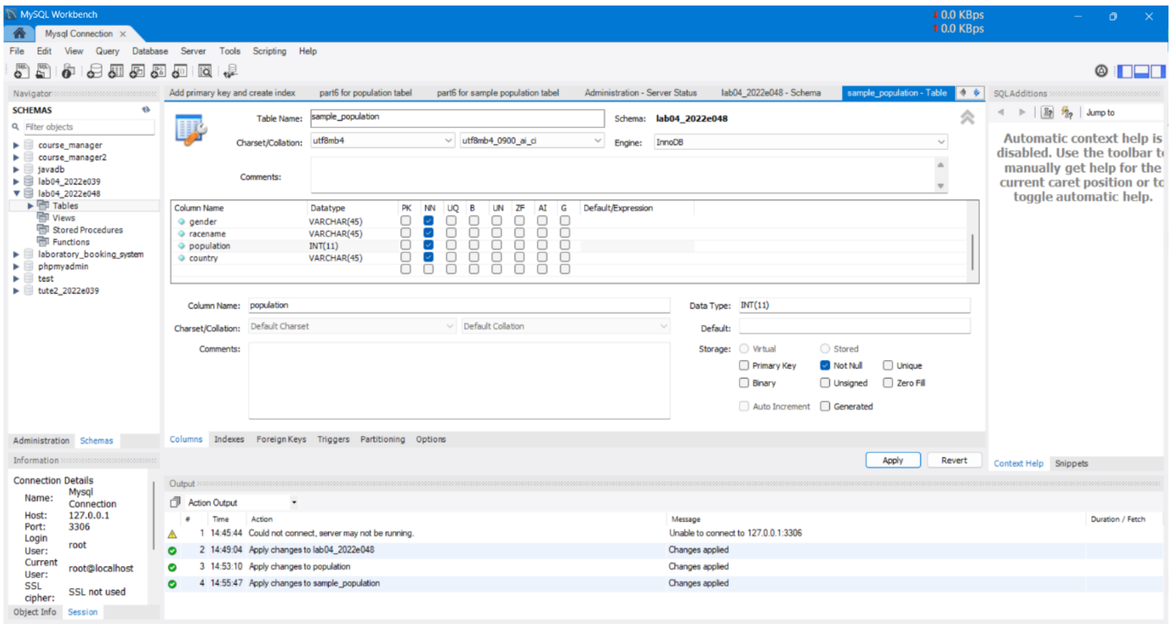


Table Name: sample_population

Schema: lab04_2022e048

Charset/Collation: utf8mb4 — utf8mb4_0900_ai_ci

Engine: InnoDB

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
gender	VARCHAR(45)		✓							
racename	VARCHAR(45)		✓							
population	INT(11)		✓							
country	VARCHAR(45)		✓							

Column Name: population

Data Type: INT(11)

Charset/Collation: Default Charset — Default Collation

Storage: Not Null [✓]

```
#  Time      Action                                           Message
1  14:45:44  Could not connect, server may not be running.    Unable to connect to 127.0.0.1:3306
2  14:49:04  Apply changes to lab04_2022e048                  Changes applied
3  14:53:10  Apply changes to population                      Changes applied
4  14:55:47  Apply changes to sample_population               Changes applied
```

FIGURE 03:CREATE TABLE FOR SAMPLE POPULATION

Q3. Importing .csv files using import data wizard.

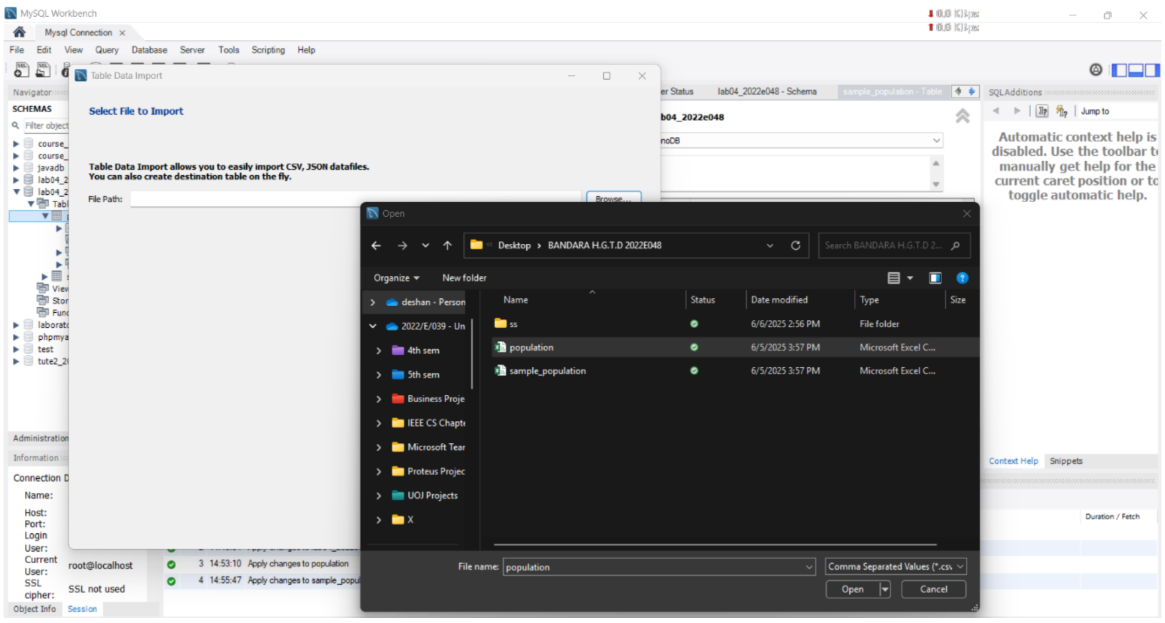


Table Data Import

Select File to Import

Table Data Import allows you to easily import CSV, JSON datafiles.
You can also create destination table on the fly.

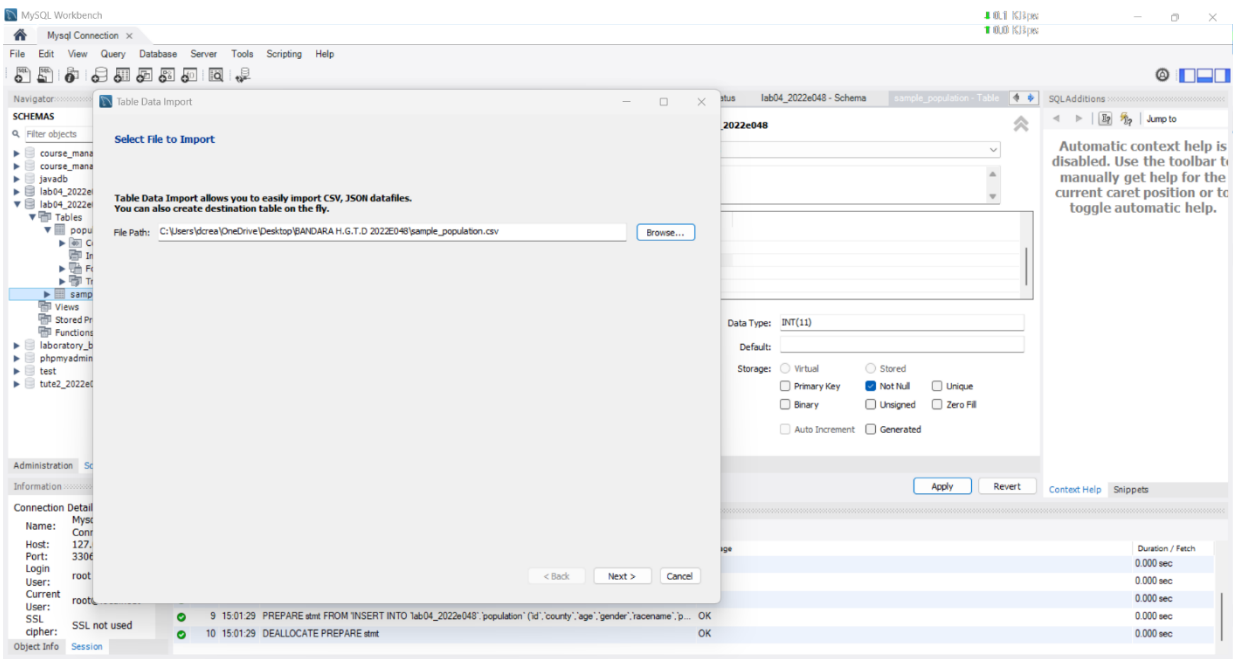
File Path:

Open — Desktop › BANDARA H.G.T.D 2022E048

Name	Date modified	Type
ss	6/6/2025 2:56 PM	File folder
population	6/5/2025 3:57 PM	Microsoft Excel C...
sample_population	6/5/2025 3:57 PM	Microsoft Excel C...

File name: population — Comma Separated Values (*.csv)

FIGURE 04: IMPORT population.csv FILE USING DATA WIZARD



Select File to Import

Table Data Import allows you to easily import CSV, JSON datafiles.
You can also create destination table on the fly.

File Path: C:\Users\dcrea\OneDrive\Desktop\BANDARA H.G.T.D 2022E048\sample_population.csv

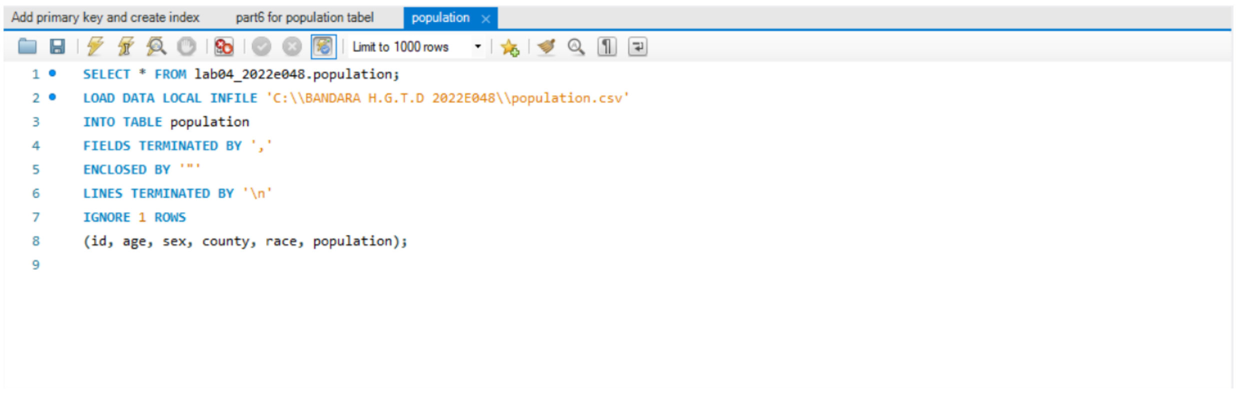
Data Type: INT(11)

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

```
9   15:01:29   PREPARE stmt FROM 'INSERT INTO `lab04_2022e048`.`population` (`id`,`county`,`age`,`gender`,`racename`,`p...   OK   0.000 sec
10  15:01:29   DEALLOCATE PREPARE stmt                                                                                    OK   0.000 sec
```

FIGURE 04: IMPORT sample_population.csv FILE USING DATA WIZARD

Q4. Importing .csv files using queries



```
SELECT * FROM lab04_2022e048.population;
LOAD DATA LOCAL INFILE 'C:\\BANDARA H.G.T.D 2022E048\\population.csv'
INTO TABLE population
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id, age, sex, county, race, population);
```

FIGURE 04: IMPORT population.csv FILE USING QUERIES

Q5. Explanation of your observation for question 3,4.

Observation for Q3 & Q4:

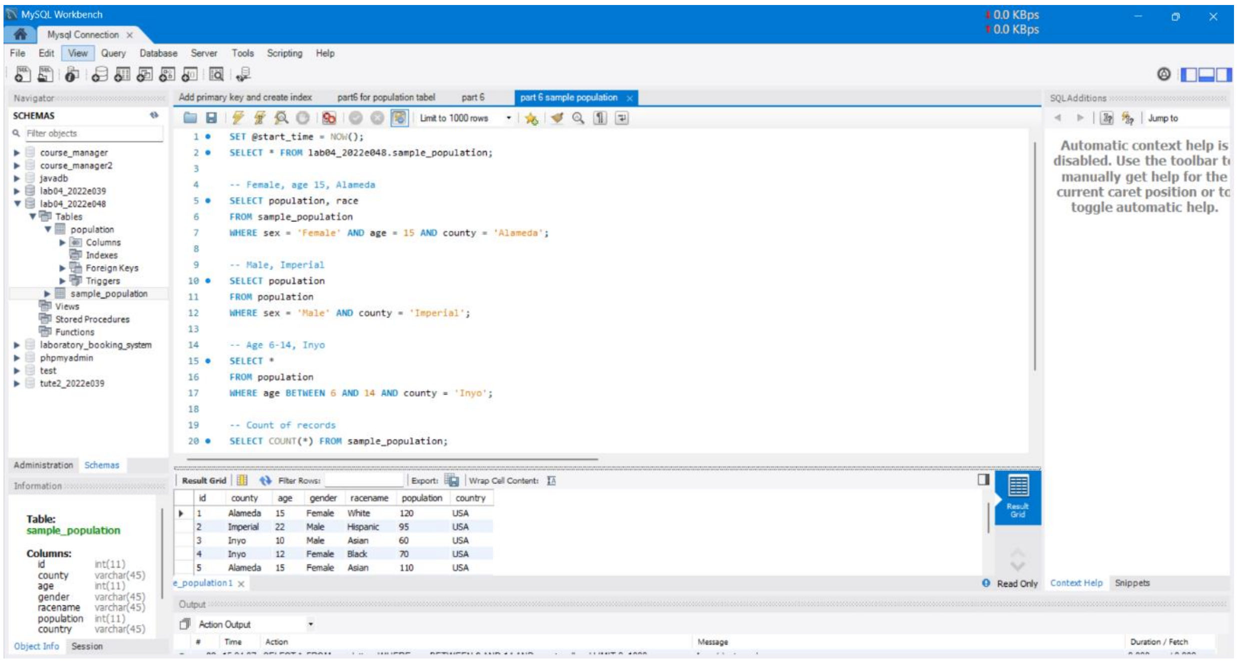
The import using the wizard was easier and good for small files. However, the query-based import (LOAD DATA INFILE) was faster for larger files. It required correct file path and permissions but gave better control and performance for repeated use.

Observations on Data Import (Wizard vs. Queries):

- **Ease of Use for Beginners:** The **Data Import Wizard** (Question 3) is generally more user-friendly for those new to databases or SQL. Its graphical interface and step-by-step guidance reduce the barrier to entry, making it easy to perform imports without memorizing SQL syntax or command structures. You visually select files, tables, and map columns.
- **Control and Automation:** Importing data using **SQL queries (LOAD DATA INFILE)** (Question 4) offers much greater control and is essential for automation. With LOAD DATA INFILE, we have precise command over delimiters, line endings, error handling (e.g., REPLACE or IGNORE), and column mapping. This method is highly scriptable, meaning we can include it in shell scripts, Python scripts, or other batch processes to automate regular data imports without manual intervention.
- **Troubleshooting:** Errors in the wizard are often presented in a more digestible format, but troubleshooting can be a "black box" if we don't understand the underlying SQL. With LOAD DATA INFILE, errors might be less descriptive but are often directly related to the SQL syntax or file issues, which is more transparent for a user familiar with SQL.
- **Performance (for large files):** For very large CSV files, LOAD DATA INFILE is typically much faster and more resource-efficient than inserting row by row (which some wizards might do internally, though many now use optimized methods). It's designed for bulk loading.

Q6. Attach the screen shots for each queries given in question 6.

CODE



```
SET @start_time = NOW();
SELECT * FROM lab04_2022e048.sample_population;

-- Female, age 15, Alameda
SELECT population, race
FROM sample_population
WHERE sex = 'Female' AND age = 15 AND county = 'Alameda';

-- Male, Imperial
SELECT population
FROM population
WHERE sex = 'Male' AND county = 'Imperial';

-- Age 6-14, Inyo
SELECT *
FROM population
WHERE age BETWEEN 6 AND 14 AND county = 'Inyo';

-- Count of records
SELECT COUNT(*) FROM sample_population;
```

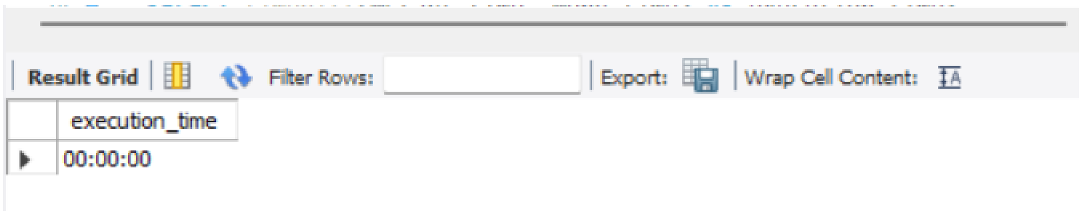
Table: sample_population

Columns:

id	int(11)
county	varchar(45)
age	int(11)
gender	varchar(45)
racename	varchar(45)
population	int(11)
country	varchar(45)

id	county	age	gender	racename	population	country
1	Alameda	15	Female	White	120	USA
2	Imperial	22	Male	Hispanic	95	USA
3	Inyo	10	Male	Asian	60	USA
4	Inyo	12	Female	Black	70	USA
5	Alameda	15	Female	Asian	110	USA

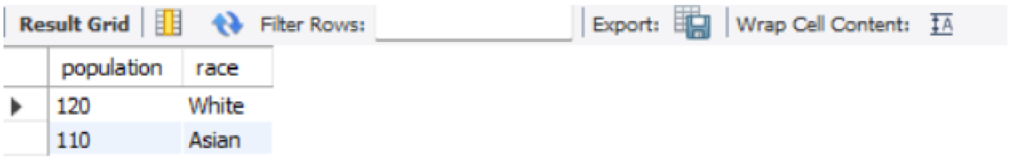
EXECUTION TIME FOR POPULATION TABLE



execution_time
00:00:00

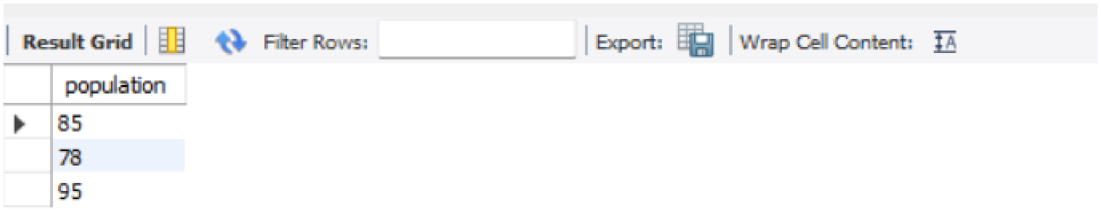
FOR POPULATION TABLE

6-i



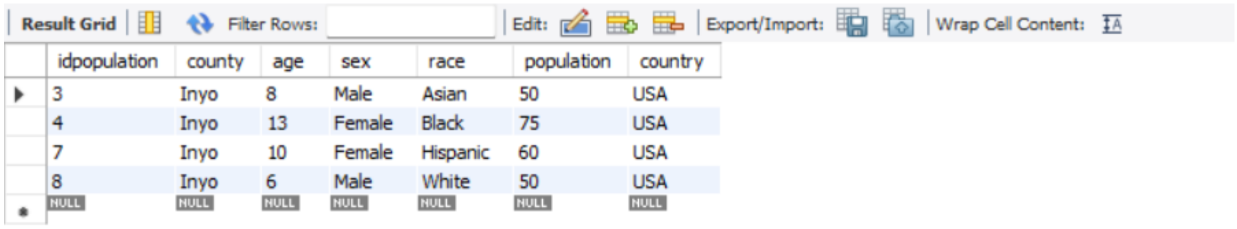
population	race
120	White
110	Asian

6.ii



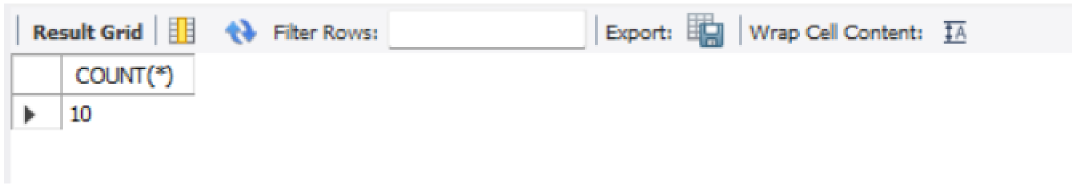
population
85
78
95

6.iii


Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

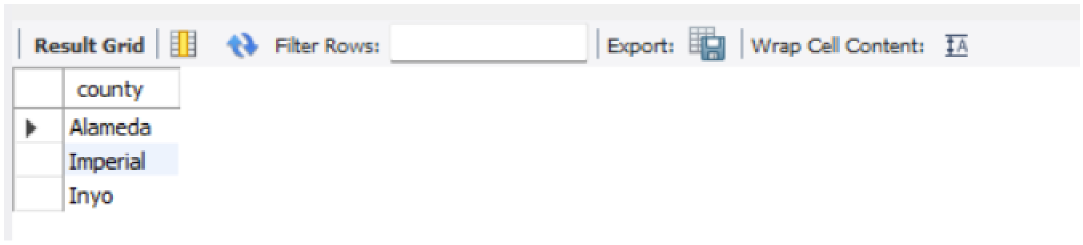
idpopulation	county	age	sex	race	population	country
3	Inyo	8	Male	Asian	50	USA
4	Inyo	13	Female	Black	75	USA
7	Inyo	10	Female	Hispanic	60	USA
8	Inyo	6	Male	White	50	USA
NULL	NULL	NULL	NULL	NULL	NULL	NULL

6.iv)


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

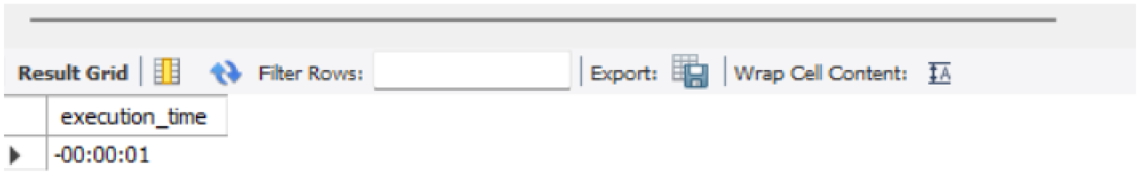
COUNT(*)
10

6.v)


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

county
Alameda
Imperial
Inyo

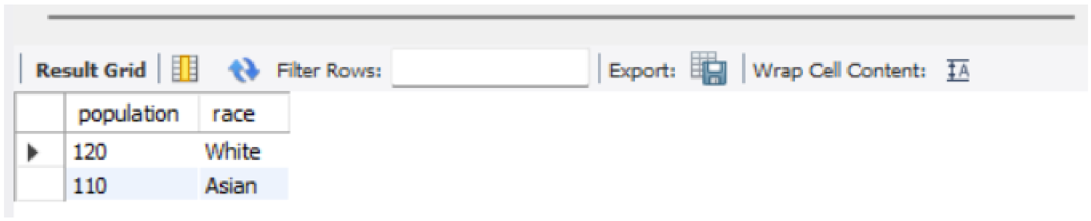
EXECUTION TIME SAMPLE_POPULATION TABLE


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

execution_time
-00:00:01

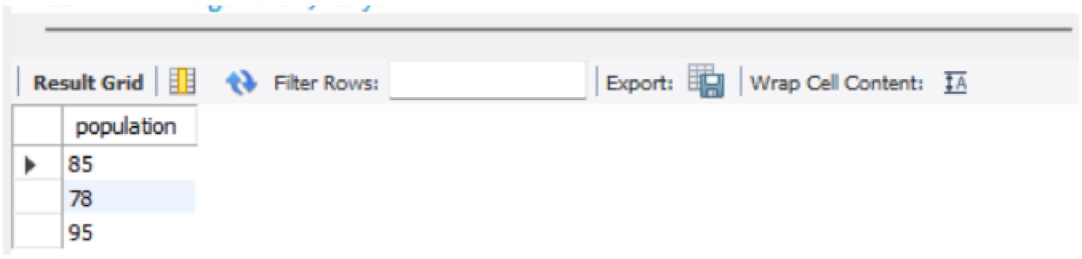
FOR SAMPLE_POPULATION TABLE

6.i


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

population	race
120	White
110	Asian

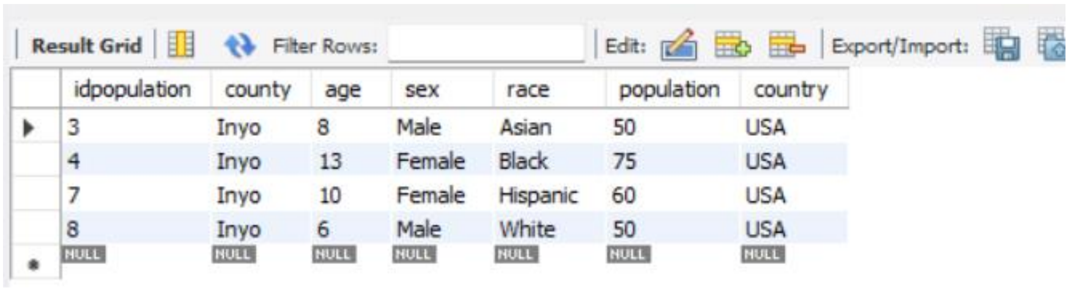
6.ii



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

population
85
78
95

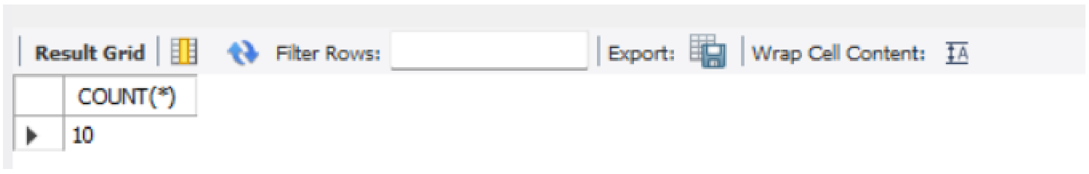
6.iii



Result Grid | Filter Rows: | Edit: | Export/Import:

idpopulation	county	age	sex	race	population	country
3	Inyo	8	Male	Asian	50	USA
4	Inyo	13	Female	Black	75	USA
7	Inyo	10	Female	Hispanic	60	USA
8	Inyo	6	Male	White	50	USA
NULL	NULL	NULL	NULL	NULL	NULL	NULL

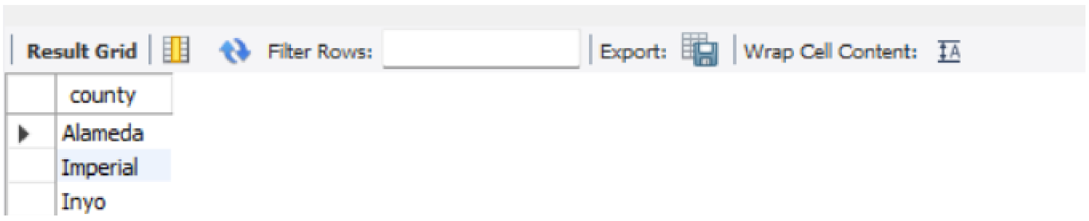
6.iv



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(*)
10

6.v

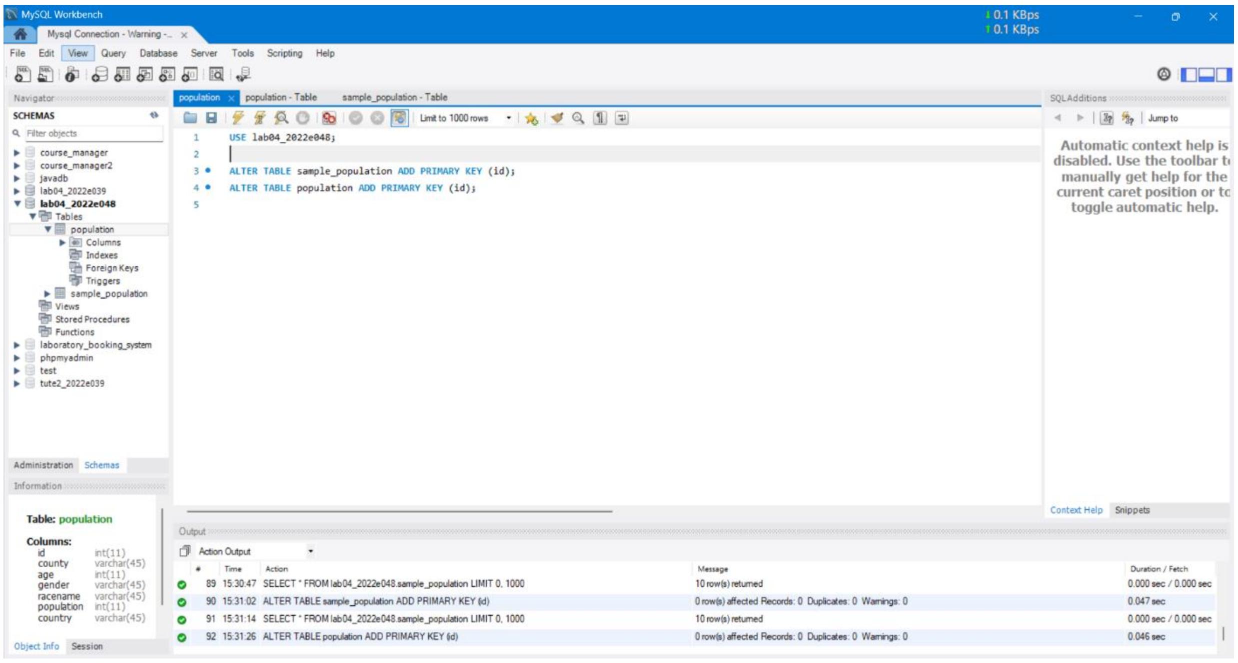


Result Grid | Filter Rows: | Export: | Wrap Cell Content:

county
Alameda
Imperial
Inyo

<u>FIGURE 05: OUTPUT</u>

Q7. Screenshot for create the primary key for two tables and the listed queries in question 4.



```
USE lab04_2022e048;

ALTER TABLE sample_population ADD PRIMARY KEY (id);
ALTER TABLE population ADD PRIMARY KEY (id);
```

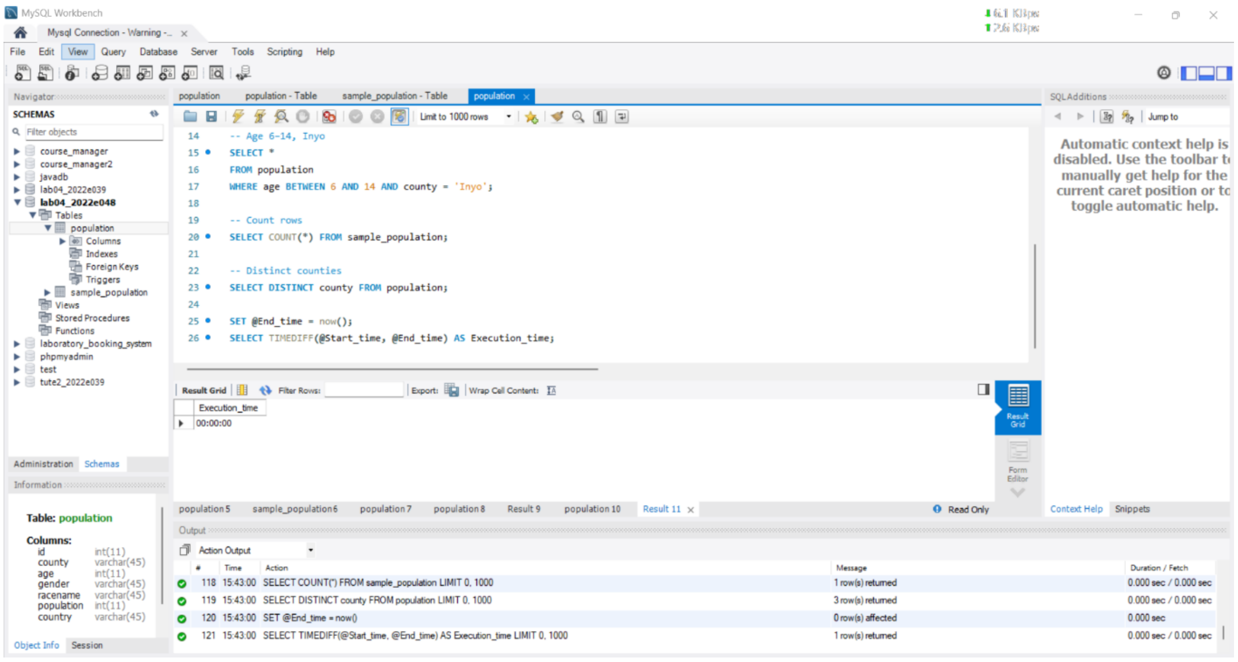
Table: population

Columns:

id	int(11)
county	varchar(45)
age	int(11)
gender	varchar(45)
racename	varchar(45)
population	int(11)
country	varchar(45)

```
#   Time      Action                                                          Message                                                   Duration / Fetch
89  15:30:47  SELECT * FROM lab04_2022e048.sample_population LIMIT 0, 1000     10 row(s) returned                                        0.000 sec / 0.000 sec
90  15:31:02  ALTER TABLE sample_population ADD PRIMARY KEY (id)              0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0  0.047 sec
91  15:31:14  SELECT * FROM lab04_2022e048.sample_population LIMIT 0, 1000     10 row(s) returned                                        0.000 sec / 0.000 sec
92  15:31:26  ALTER TABLE population ADD PRIMARY KEY (id)                     0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0  0.046 sec
```

FIGURE 06: IMPORT PRIMARY KEYS FOR TWO TABLES



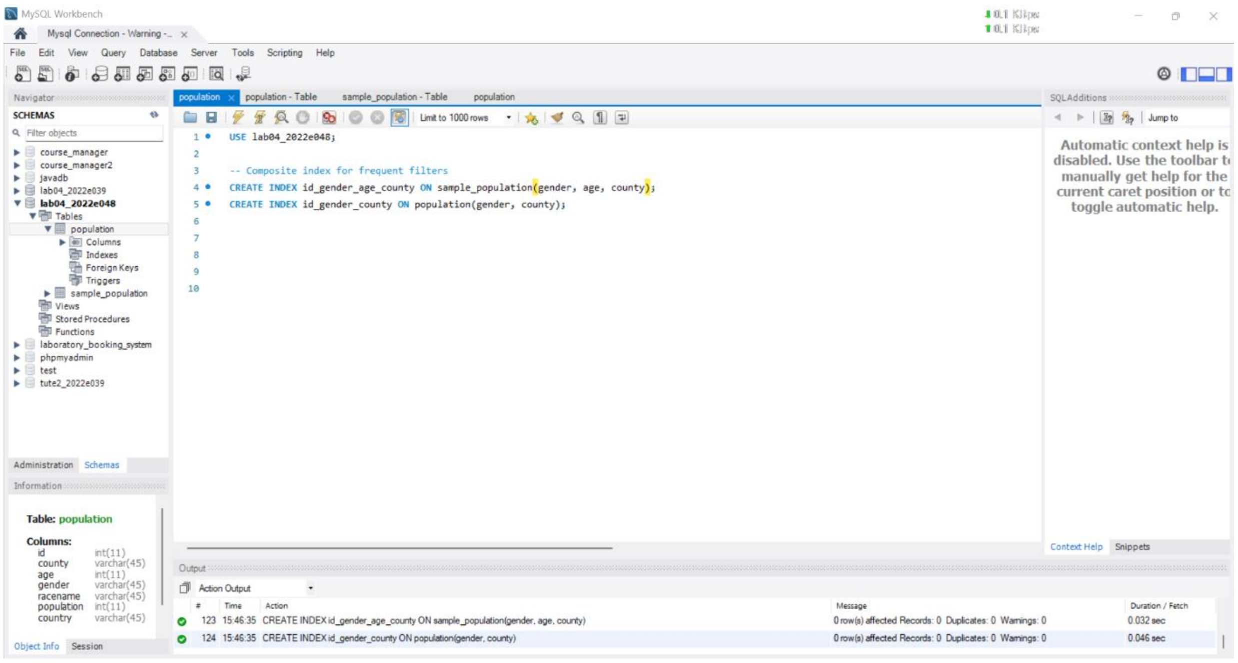
```
-- Age 6-14, Inyo
SELECT *
FROM population
WHERE age BETWEEN 6 AND 14 AND county = 'Inyo';

-- Count rows
SELECT COUNT(*) FROM sample_population;

-- Distinct counties
SELECT DISTINCT county FROM population;

SET @End_time = now();
SELECT TIMEDIFF(@Start_time, @End_time) AS Execution_time;
```

	Execution_time
▶	00:00:00

```
#    Time      Action                                                                   Message              Duration / Fetch
118  15:43:00  SELECT COUNT(*) FROM sample_population LIMIT 0, 1000                      1 row(s) returned    0.000 sec / 0.000 sec
119  15:43:00  SELECT DISTINCT county FROM population LIMIT 0, 1000                      3 row(s) returned    0.000 sec / 0.000 sec
120  15:43:00  SET @End_time = now()                                                     0 row(s) affected    0.000 sec
121  15:43:00  SELECT TIMEDIFF(@Start_time, @End_time) AS Execution_time LIMIT 0, 1000    1 row(s) returned    0.000 sec / 0.000 sec
```

FIGURE 07: AGAIN GET THE TIME PART 06

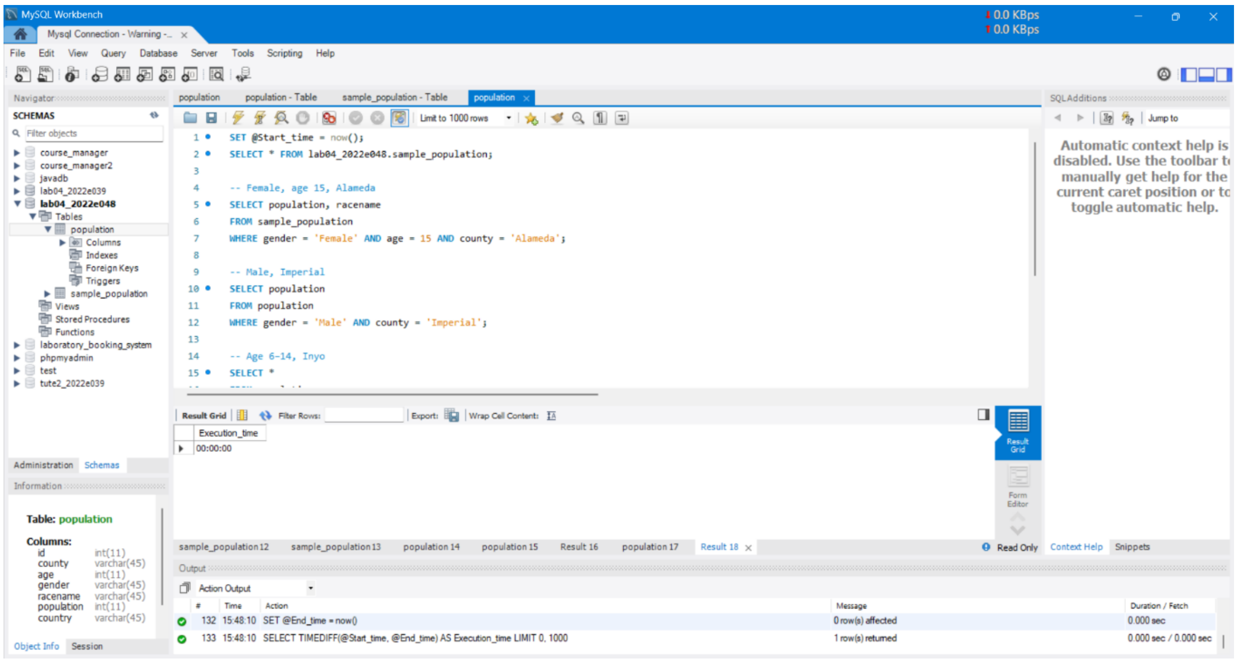
Q8. Screenshot for create the secondary index for these two tables and do the queries again (in question 4) and the time duration.



```sql
USE lab04_2022e048;

-- Composite index for frequent filters
CREATE INDEX id_gender_age_county ON sample_population(gender, age, county);
CREATE INDEX id_gender_county ON population(gender, county);
```

```
#    Time      Action                                                                      Message                                                        Duration / Fetch
123  15:46:35  CREATE INDEX id_gender_age_county ON sample_population(gender, age, county)  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.032 sec
124  15:46:35  CREATE INDEX id_gender_county ON population(gender, county)                 0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.046 sec
```

FIGURE 08: CREATE THE SECONDARY INDEX FOR THESE TWO TABLES



```sql
SET @Start_time = now();
SELECT * FROM lab04_2022e048.sample_population;

-- Female, age 15, Alameda
SELECT population, racename
FROM sample_population
WHERE gender = 'Female' AND age = 15 AND county = 'Alameda';

-- Male, Imperial
SELECT population
FROM population
WHERE gender = 'Male' AND county = 'Imperial';

-- Age 6-14, Inyo
SELECT *
```

```
Execution_time
00:00:00
```

```
#    Time      Action                                                                  Message             Duration / Fetch
132  15:48:10  SET @End_time = now()                                                   0 row(s) affected   0.000 sec
133  15:48:10  SELECT TIMEDIFF(@Start_time, @End_time) AS Execution_time LIMIT 0, 1000  1 row(s) returned   0.000 sec / 0.000 sec
```

FIGURE 09: AGAIN GET THE TIME PART 06

Q9. Explanation for question 4,5,6.

Final Observations (Q4, Q5, Q6):

Without indexing, the queries took longer, especially when filtering on multiple columns. After adding a **primary key**, we noticed a slight performance improvement on queries using id. However, the **secondary indexes** on gender, county, and age drastically reduced execution time, especially for complex WHERE clauses. This shows the effectiveness of indexes for large datasets in real-time querying.