# A. Title Page

Lewis University  
CPSC 50900: Database Systems   
Spring 2023 Term Project

Lewis University International Student Information

VidhyaSagar Janga, [vidhyasgarjanga@lewisu.edu](mailto:vidhyasgarjanga@lewisu.edu)

Bharath Kumar Tella, [bharathktella@lewisu.edu](mailto:bharathktella@lewisu.edu)

Work products stored in the Github repository:   
<https://github.com/VidhyaSagarJanga/Lewis-University-International-Student-Information>

<https://github.com/bharathtella/Lewis-University-International-Student-Information>

Table of Contents

[A. Title Page 1](#_Toc159428257)

[B. Initial Proposal 5](#_Toc159428258)

[C. Data Sources 6](#_Toc159428259)

[D. Alternative Ways to Store the Data 7](#_Toc159428260)

[First Alternative: 7](#_Toc159428261)

[Second Alternative: 8](#_Toc159428262)

[E. Relational Database Design Process 8](#_Toc159428263)

[Functional dependencies: 8](#_Toc159428264)

[Entities: 9](#_Toc159428265)

[Relationships between entities: 9](#_Toc159428266)

[Multi-valued attributes: 10](#_Toc159428267)

[F. Relational Database Design 10](#_Toc159428268)

[Normalization: 10](#_Toc159428269)

[Introducing Bridge Entities: 10](#_Toc159428270)

[Foreign keys: 10](#_Toc159428271)

[Surrogate keys: 11](#_Toc159428272)

[Database model from vertabello: 11](#_Toc159428273)

[G. Data Definition Language (DDL) Scripts 11](#_Toc159428274)

[SQL script used for creating tables: 11](#_Toc159428275)

[SQL script used for inserting data into tables: 12](#_Toc159428276)

[Excel spreadsheets created for all the tables after adding the foreign keys: 12](#_Toc159428277)

[Execution of Scripts by source command: 12](#_Toc159428278)

[Creating database and using it: 12](#_Toc159428279)

[Creating tables and adding foreign keys: 12](#_Toc159428280)

[Inserting data into the database: 13](#_Toc159428281)

[Queries ran for creating tables in different machine: 13](#_Toc159428282)

[Populating data into each table using insert queries in different machine: 15](#_Toc159428283)

[H. Data Manipulation Language Scripts 16](#_Toc159428284)

[DML Queries link: 16](#_Toc159428285)

[Insert Queries: 17](#_Toc159428286)

[Update Queries: 17](#_Toc159428287)

[Delete Query: 18](#_Toc159428288)

[Select Query: 18](#_Toc159428289)

[Join Queries: 19](#_Toc159428290)

[Summary Queries: 19](#_Toc159428291)

[Multi-table query: 20](#_Toc159428292)

[Query of choice: 20](#_Toc159428293)

[I. Indexes 21](#_Toc159428294)

[Index 1: 21](#_Toc159428295)

[Importance: 21](#_Toc159428296)

[SQL generated: 21](#_Toc159428297)

[Performance improvement: 21](#_Toc159428298)

[Index 2: 22](#_Toc159428299)

[Importance: 22](#_Toc159428300)

[SQL generated: 22](#_Toc159428301)

[Performance improvement: 22](#_Toc159428302)

[Index 3: 23](#_Toc159428303)

[Importance: 23](#_Toc159428304)

[SQL generated: 23](#_Toc159428305)

[Performance improvement: 23](#_Toc159428306)

[J. Views 24](#_Toc159428307)

[View 1: 24](#_Toc159428308)

[Git link: 24](#_Toc159428309)

[Code: 24](#_Toc159428310)

[Data contained: 25](#_Toc159428311)

[Importance for database: 25](#_Toc159428312)

[View 2: 25](#_Toc159428313)

[SQL created: 25](#_Toc159428314)

[Code: 25](#_Toc159428315)

[Data contained: 26](#_Toc159428316)

[Importance for database: 26](#_Toc159428317)

[K. Triggers 27](#_Toc159428318)

[SQL for creating trigger: 27](#_Toc159428319)

[Git link: 27](#_Toc159428320)

[Code: 27](#_Toc159428321)

[Purpose: 27](#_Toc159428322)

[Screenshots of output: 28](#_Toc159428323)

[Table contents before creating trigger: 28](#_Toc159428324)

[Trigger created on database: 28](#_Toc159428325)

[Inserting into table without PK: 28](#_Toc159428326)

[Table contents after creating trigger and inserting values: 28](#_Toc159428327)

[L. Transactions 29](#_Toc159428328)

[Importance of transactions ensuring ACID behavior: 29](#_Toc159428329)

[Screenshots performing transactions on database: 29](#_Toc159428330)

[Operations used while doing transactions: 30](#_Toc159428331)

[Begin: 30](#_Toc159428332)

[Commit: 30](#_Toc159428333)

[Rollback: 30](#_Toc159428334)

[M. Database Security 30](#_Toc159428335)

[Various users on the database: 30](#_Toc159428336)

[GRANT statements that assign privileges to these different kinds of users 31](#_Toc159428337)

[Sql executed upload to git: 33](#_Toc159428338)

[Screenshots creating differrent users: 34](#_Toc159428339)

[Creating student user 34](#_Toc159428340)

[CREATING USERS: PROFESSORS: 34](#_Toc159428341)

[VEHICLEVENDORS: 35](#_Toc159428342)

[Employers: 35](#_Toc159428343)

[INSURANCE PROVIDERS: 36](#_Toc159428344)

[ADMINSTRATORS: 36](#_Toc159428345)

[N. Locking and Concurrent Access 36](#_Toc159428346)

[Purpose of locking tables: 36](#_Toc159428347)

[Screenshots performing locking using backup file: 37](#_Toc159428348)

[Performing locking on our database: 37](#_Toc159428349)

[O. Backing Up Your Database 38](#_Toc159428350)

[Steps taken to back up database: 38](#_Toc159428351)

[ Frequency of backup: 38](#_Toc159428352)

[ Storage location: 38](#_Toc159428353)

[ Security of backup file: 38](#_Toc159428354)

[ Automation of backup: 39](#_Toc159428355)

[Commands performed and backup file: 39](#_Toc159428356)

[Backup command performed on console: 39](#_Toc159428357)

[Automation and frequency of backup: 39](#_Toc159428358)

[Screenshot of backup file generated: 40](#_Toc159428359)

[P. Programming 41](#_Toc159428360)

[Link for python script: 41](#_Toc159428361)

[Python script: 41](#_Toc159428362)

[OUTPUT: 42](#_Toc159428363)

[Git Links for data exported from MySQL database: 43](#_Toc159428364)

[course\_data exported as excel file: 43](#_Toc159428365)

[tablesinfo exported as text file **:** 43](#_Toc159428366)

[R. Suggested Future Work 43](#_Toc159428367)

[Limitations of our database: 43](#_Toc159428368)

[Overcoming these shortcomings: 44](#_Toc159428369)

[Migration steps of database to cloud AWS: 44](#_Toc159428370)

[Advantages migrating to cloud: 44](#_Toc159428371)

[Advantages of using document-based *NoSQL format:* 44](#_Toc159428372)

[Disadvantages of using document-based *NoSQL format:* 45](#_Toc159428373)

# B. Initial Proposal

1. The "Lewis University International Student Information" is a comprehensive database designed to store and manage data related to students at Lewis University. The database includes various data such as

**Student Information**: StudentId, Name, Contact, Address, emergency contacts, LinkedIn Id etc.

**Course Table**: The course table stores information about the courses offered at the university, including the course code, title, description, and credit hours.

**Course registration details**: As student info and course table has many to many relation, this table is created.

**Course schedule Details**: This table contains information about the courses that a student is registered for, and the schedule of classes.

**Housing information**: on-campus or off-campus housing details

**Working details**: the student got any on campus job or working on CPT/OPT

**Insurance details:** details about Insurance provider, Start date, Expiry date, Maximum benefit amount.

**Vehicle Vendor Database:** details about Vendor company name, contact information, Insurance details who rents or sells the vehicle to students.

1. The main objective of this project is to centralize information about resources that can help students studying at Lewis University, especially those who come from different parts of the world. For instance, as an international student arriving in the US, we have faced difficulties in finding suitable accommodation. Despite searching for apartments online, we ended up taking a lease that was quite expensive. However, we later discovered that we could have found much cheaper accommodation if we had contacted senior students for advice. Similarly, as there is no public transportation in Romeoville, we had to rely on expensive cab rides to travel to the university until we were able to rent a car through friends in my class. By centralizing information about housing, transportation, employment opportunities, and other resources, this system can help students overcome such challenges and find the support they need to succeed at Lewis University.
2. This is important because it provides a centralized platform to store and manage data related to students of Lewis University. This database can help upcoming students, especially international students, who may have difficulty finding accommodation, transportation, health insurance, and other resources in an unfamiliar place. By providing easy access to information about these resources, the database can help students save time and money and make informed decisions. Additionally, the database can help build a sense of community among students by connecting them with each other and facilitating communication and collaboration. Overall, the project aims to enhance the student experience and support the academic success of students at Lewis University.
3. We feel that the best way to get data for the above tables is by collecting it from the students themselves. This can be done through online forms, surveys, or questionnaires that students can fill out voluntarily. Other sources include by using online directories, for example we can get the insurance details by using online directories such as Google Business or Yelp.
4. This project can be used by a variety of users. For instance, students who are seeking accommodations or transportation services can access the database to find suitable options. Additionally, the database can assist students in connecting with other students who share the same classes or schedules. Moreover, it can be helpful for new students to get in touch with senior students who can provide useful tips and advice about the university, the village, and surrounding areas. Overall, the database is a valuable resource for anyone who is part of the Lewis University community.
5. We can build a comprehensive platform for students to access all the information they need in one place, making it easier for them to manage their academic and personal lives. The application could have different sections or modules for each aspect of student life, such as housing, transportation, course registration, and extracurricular activities. Users could log in and access the relevant information for their specific needs, such as finding a roommate or carpooling to class. The application could also allow students to connect with one another, such as through a messaging system or forum, to share tips and advice. Overall, a combined application would streamline the student experience and make it easier for them to navigate life at Lewis University.

# C. Data Sources

The sample data created is updated in sheets in the excel uploaded to the GitHub repository.

The normalization of data is not done accurately, improvement and changes to the project might be done going forward.

**• Student Information Table:** The data for this table will be collected through interviews with students in our community, including their name, contact information, address, emergency contacts, and LinkedIn ID.

**• Course Table:** The data for this table was gathered from the official Lewis University website and includes details such as course names, course IDs, credit hours and course description.

**• Course Registration Details Table:** We collected data for this table through interviews with students in our community, including the classes they registered for, and their course schedule Id.

**• Course Schedule Table**: The data for this table was gathered by checking the MyLewis application, including class times, locations, and instructors.

**• Housing Information Table:** We gathered data for this table from the place we live and entered it as sample data. This includes details such as housing type, address, rental rates.

**• Working Details Table**: The data for this table will be collected by interviewing students in our community and contacting the university international services office. This includes details such as whether the student has on-campus employment or is working off-campus through CPT/OPT programs. For now, we entered some sample data created by us.

**• Insurance Details Table:** This includes details such as the insurance provider name, start date, expiry date, maximum benefit amount.

**• Vehicle Vendors Table:** The data for this table can be gathered by contacting local vehicle vendors, including their name, contact information, any discounts, or promotions available for students. Additionally, we gathered some data from students who have already purchased or rented vehicles from these vendors.

The tables are already created in excel as different sheets. This should be improved in many ways including normalization and attributes addition.

# D. Alternative Ways to Store the Data

## First Alternative:

* **NoSQL databases**, which are non-relational databases that do not use the traditional tabular schema of relational databases would be our first alternative.
* Example of a NoSQL, **DynamoDB in AWS** is a key-value NoSQL database that uses its own proprietary **JSON-based query API**.

Step by step process of storing data / Implementation of NoSQL:

* Choosing the data model like key-value, graph, etc. that better fits the application needs.
* Design the schema if required and create a new database instance to store data.
* Insert the data using appropriate data structure and API. This involves using functions or methods in the application.
* Query the data using required queries and indexes based on the data model used.

Advantages when compared to relational databases:

* Cost-effective as less cost compared to SQL or relational databases.
* High Scalability such that it can handle high levels of traffic.
* For larger datasets, NoSQL can provide faster read and write operations.
* Don’t require predefined schema, which makes it easier to store unstructured and semi-structured data.

Disadvantages when compared to relational databases:

* Complex to maintain and perform queries.
* Comparatively weaker security
* As it is a standard query language, migration to other databases / integration with application would become difficult.
* Limited tools for data analysis when compared.

## Second Alternative:

* **Object Oriented database**, an object-oriented data model which is often used in case of object-oriented programming languages like Java and Python are used in the application layer.
* An example of an object-oriented database is db4o, an open-source Object Oriented database that can be used for storing objects, which are instances of classes that contain both data and methods to manipulate that data.

Step by step process of storing data / Implementation of NoSQL:

* Defining the data model that needs to be used. This depends on the programming language that will be used in the application layer.
* Designing the schema which includes creating tables / collections that store objects.
* Mapping the objects in the code to corresponding tables in the database. This can be achieved by using an ORM layer.
* Manipulation of data is then enabled, either by designing a new query language or integrating it with an existing one.

Advantages when compared to relational databases:

* This is a better choice than relational databases when used with OOP languages.
* Faster compared as the mapping of objects with the tables
* Lower maintenance required when compared with relational databases.
* Security of data can be improved by restricting access to sensitive data.

Disadvantages when compared to relational databases:

* Limited tool support comparatively.
* Highly expensive and limited scalability especially for large-scale data deployments.
* As OODBs are tied to a specific programming language or platform, portability / migration to other databases would become difficult.
* Compared to relational databases, limited community of developers makes harder to find support and resources.

# E. Relational Database Design Process

## Functional dependencies:

* StudentInfo:
  + StudentId --> Name, Gender, Major, ContactNo, EmergencyContact, Address, LinkedInId
* CourseInfo:
  + CourseId --> CourseName, CreditHours, CourseDescription, Grade
* CourseScheduleInfo:
  + CRNId --> Day, StartDate, EndDate, ClassTime, Location, Instructor
* HousingInfo:
  + AddressId --> FlatNo, AptNo, AptType, Street, Zipcode, City
* InsuranceDetails:
  + InsuranceId --> Provider, StartDate, ExpiryDate, MaxBenefit
* VehicleVendors:
  + VendorId --> VendorName, Contact, Email
* WorkingDetails:
  + WorkId --> JobType, Employer, Jobtitle, SalaryPerHr

## Entities:

* Student
* Course
* Schedule
* Insurance
* Working
* Address
* Vehicle Vendors

## Relationships between entities:

Student – course relationship:

* A student can take many courses.
* A course can be taken by many students.

Student – Address relationship:

* A student lives in only one apartment.
* An apartment can be rented by many students.

Student – Working relationship:

* A student can do many jobs.
* One job can be done by only one student.

Student – Insurance relationship:

* A student can have many insurances (health, vehicle)
* Insurance can apply to only one student.

Student – Vehicle Vendor relationship:

* A student can own a vehicle from only one vendor.
* One vendor can sell vehicles to many students.

Course – Schedule relationship:

* A course can have many schedules.
* One schedule applies only to one course.

## Multi-valued attributes:

Introducing the new entities for below multivalued attributes:

* Address: aptno, city, state, etc. new entity added in functional dependencies: HousingInfo
* Name: Divided into first name and last name in the studentInfo table.

# F. Relational Database Design

## Normalization:

From the functional dependencies mentioned above, normalization analysis done is as below:

* **1NF:** All the tables are in first normalization form as all the columns presents are made atomic such that having no repeating groups.
* **2NF**: All the non-key attributes in the tables depend on the primary key, so we can say all the tables are in 2NF second normalization form.
* **3NF**: As there is no transitive dependency between non-key attributes in the tables, all the tables satisfy 3NF.
* **4NF**: The student-course relationship should be broken into two different relationships by adding a new entity: StudentRegistrationInfo, so that 4NF is achieved. Done it below by introducing bridge entities.
* **Boyce-Codd**: As there is no transitive and partial dependency in any of the tables and all non-key attributes depend on the candidate key directly, all tables are said to be already in BCNF.

## Introducing Bridge Entities:

As the Student – course relationship has many to many relationship, breaking them into two different relationships by introducing a new entity CourseRegistrationInfo as below.

Student – CourseRegistrationInfo Relationship:

* A student can have many course registrations.
* One registration can imply to only one student.

Course – CourseRegistrationInfo Relationship:

* A course can have many registrations.
* One registration can imply to only one course.

## Foreign keys:

Listed foreign keys added to each table with respect to the relationship written above:

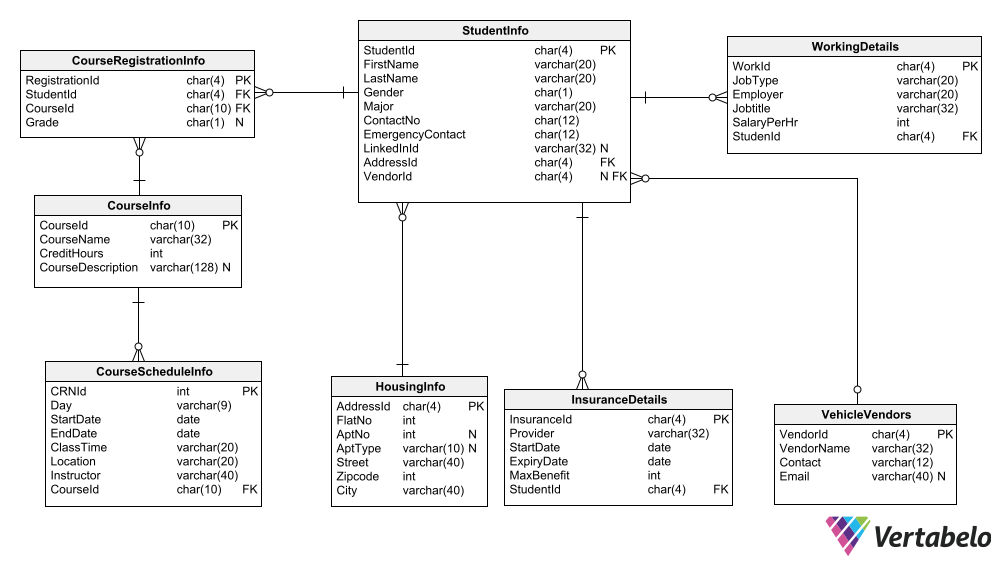
* StudentInfo table:
  + AddressId
  + VendorId
* CourseRegistrationInfo table:
  + StudentId
  + CourseId
* CourseScheduleInfo table:
  + coursed
* InsuranceDetails table:
  + StudentId
* WorkingDetails table:
  + StudentId

## Surrogate keys:

* CourseScheduleInfo table:
  + CRNId: starting from 12000.

## Database model from vertabello:

Link: <https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/Database_Model.png>



# G. Data Definition Language (DDL) Scripts

## SQL script used for creating tables:

Script name: Tables\_creation.sql

Link: <https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL_scripts/Tables_creation.sql>

## SQL script used for inserting data into tables:

Script name: Insert\_Statements.sql

Link: <https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL_scripts/Insert_Statements.sql>

## Excel spreadsheets created for all the tables after adding the foreign keys:

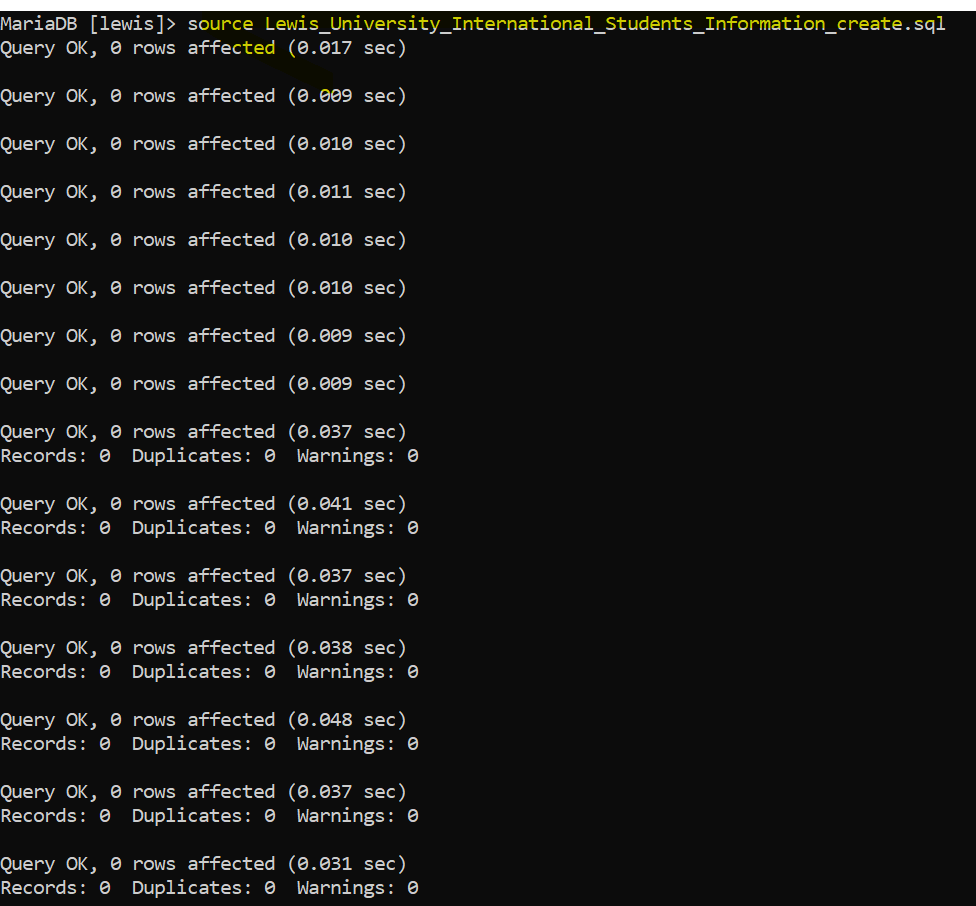
Link to file: <https://github.com/VidhyaSagarJanga/Lewis-University-International-Student-Information/blob/Sagar/Project_data.xlsx>

## Execution of Scripts by source command:

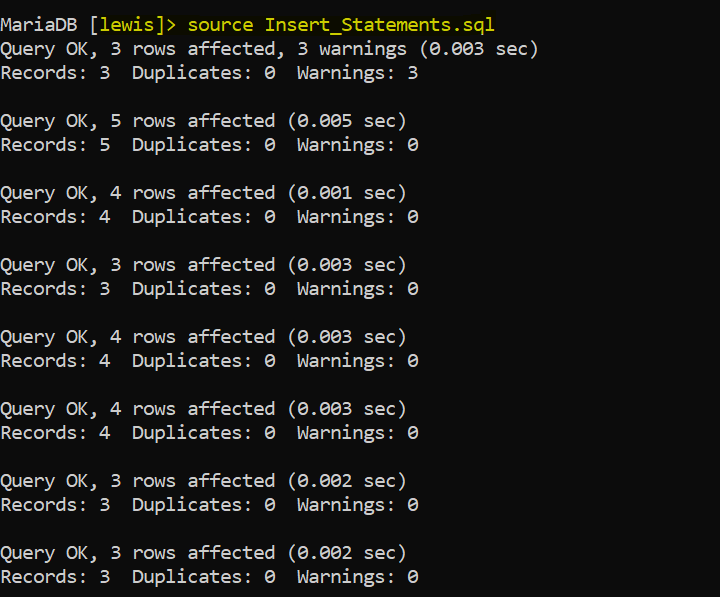
### Creating database and using it:



### Creating tables and adding foreign keys:

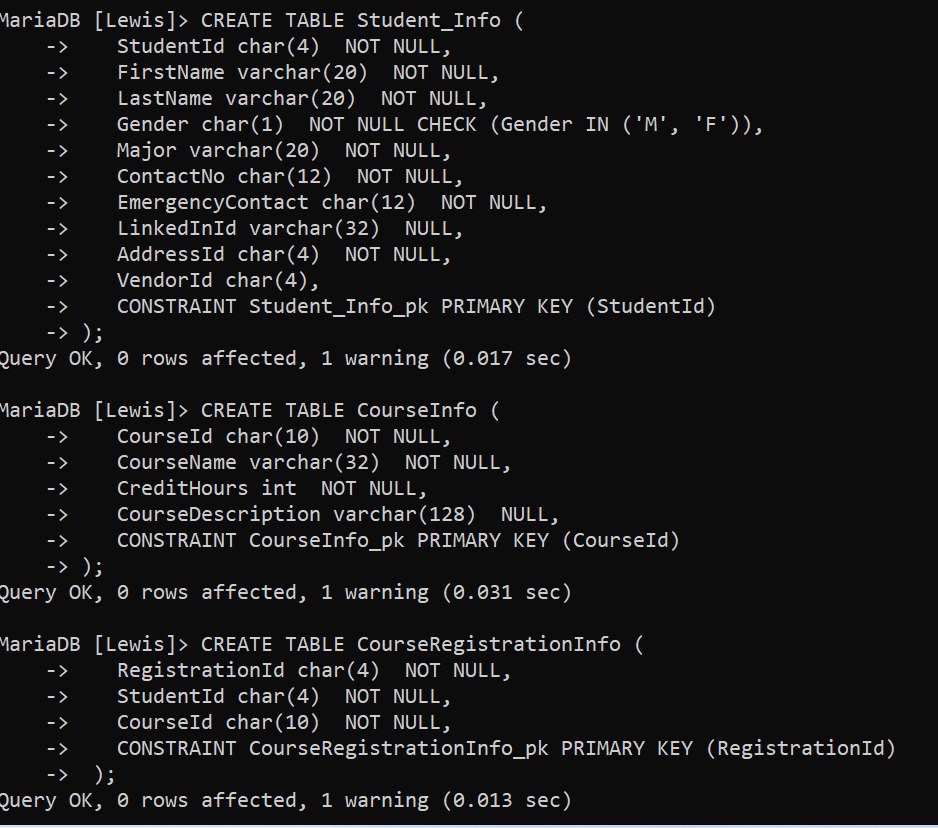


### Inserting data into the database:

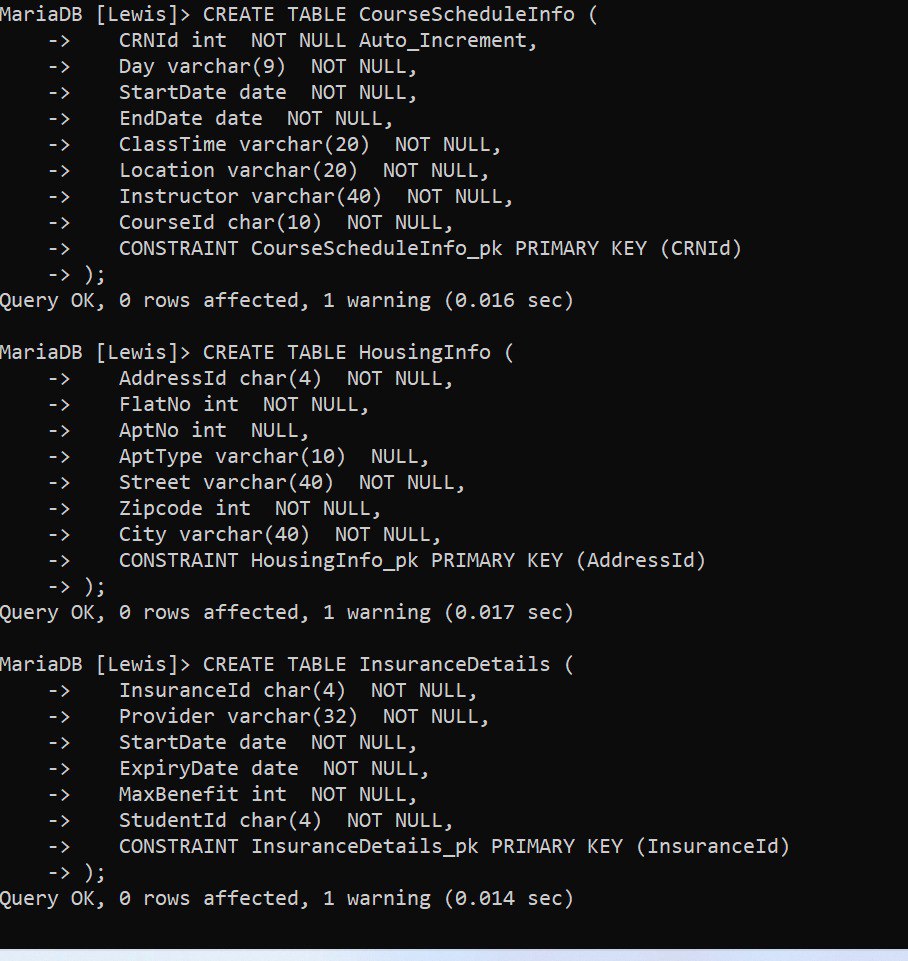


## Queries ran for creating tables in different machine:

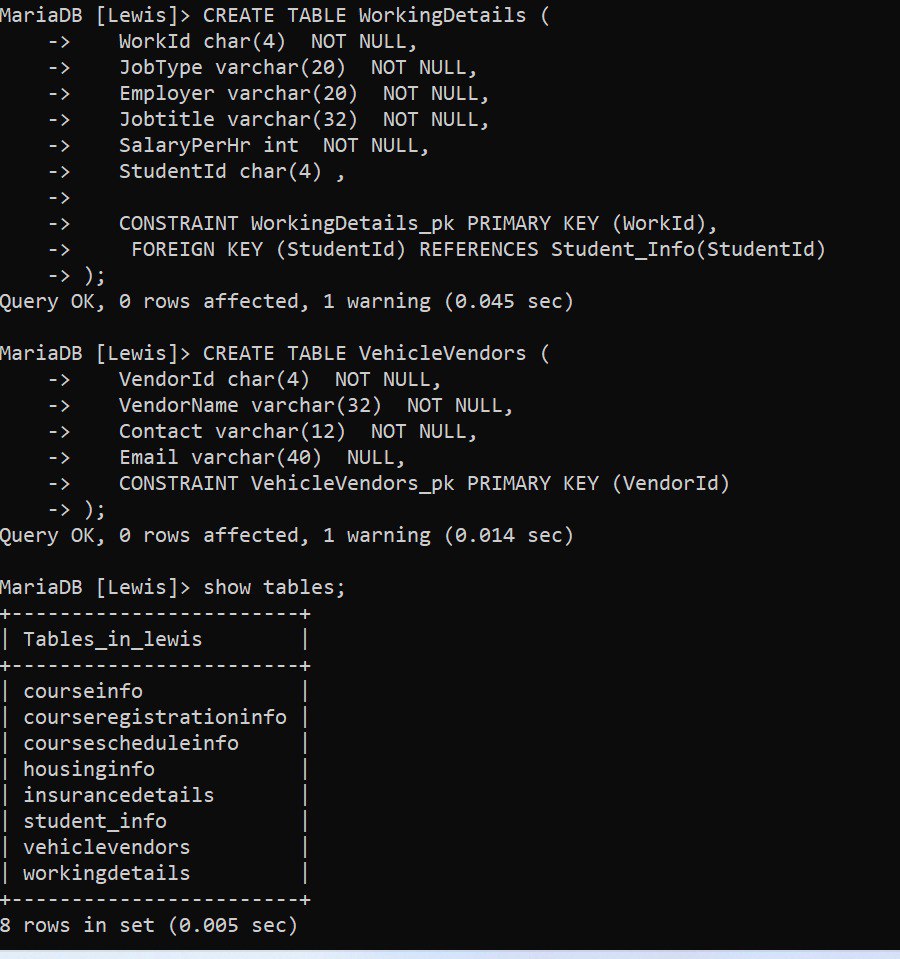
* Scripts ran for creating tables: studentinfo, courseinfo, courseregistrationinfo



* Scripts ran for creating tables: CourseScheduleInfo, HousingInfo, insurancedetails

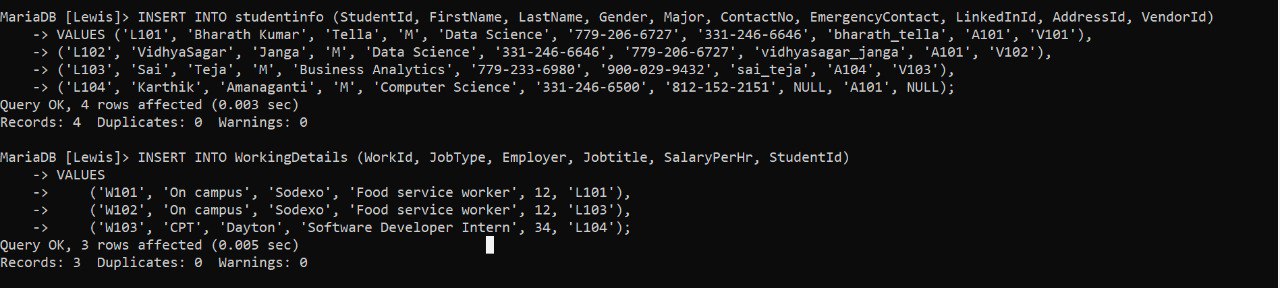


* Scripts ran for creating tables: workingdetails, vehiclevendors

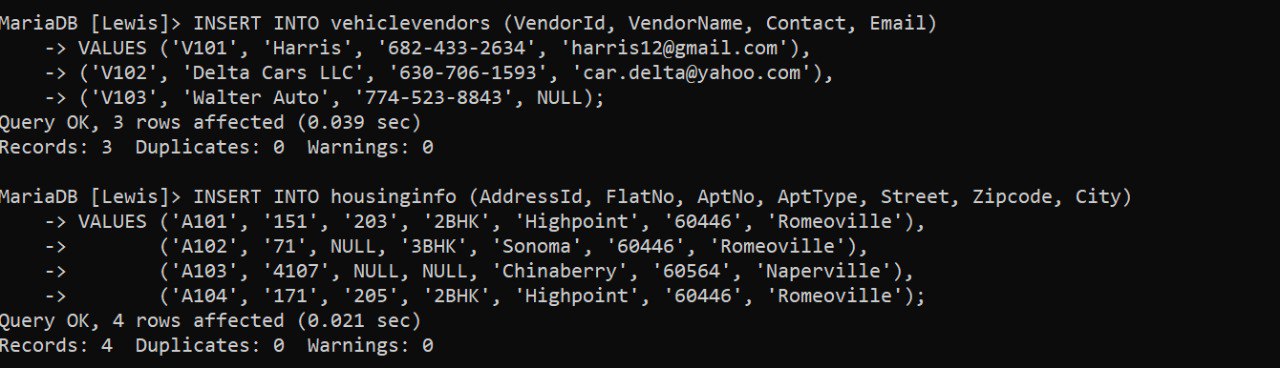


## Populating data into each table using insert queries in different machine:

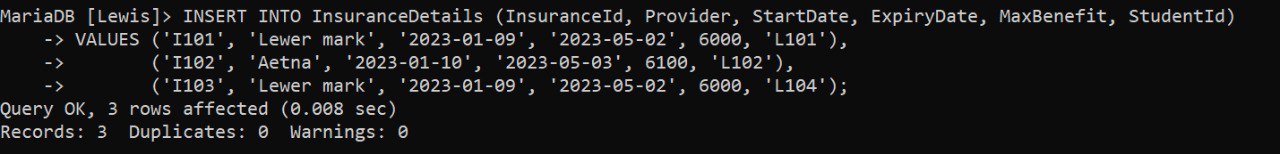
* Inserting data into tables: Studentinfo and workingdetails



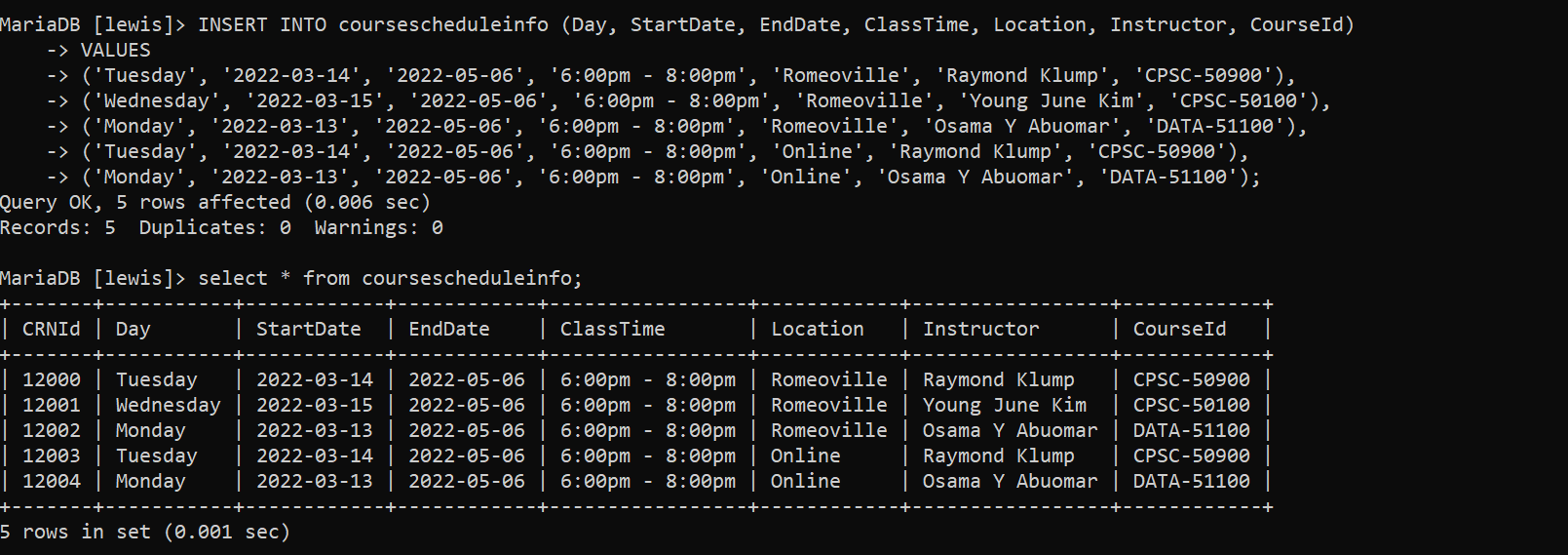
* Inserting data into tables: vehiclevendors and housinginfo



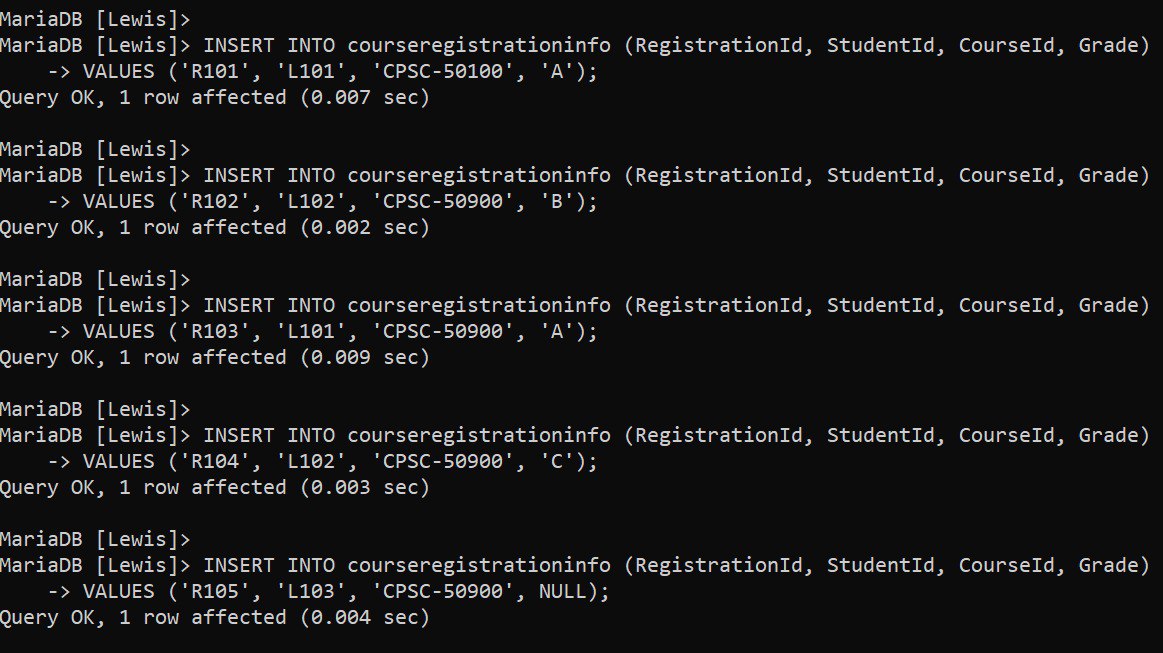
* Inserting data into table: Insurancedetails



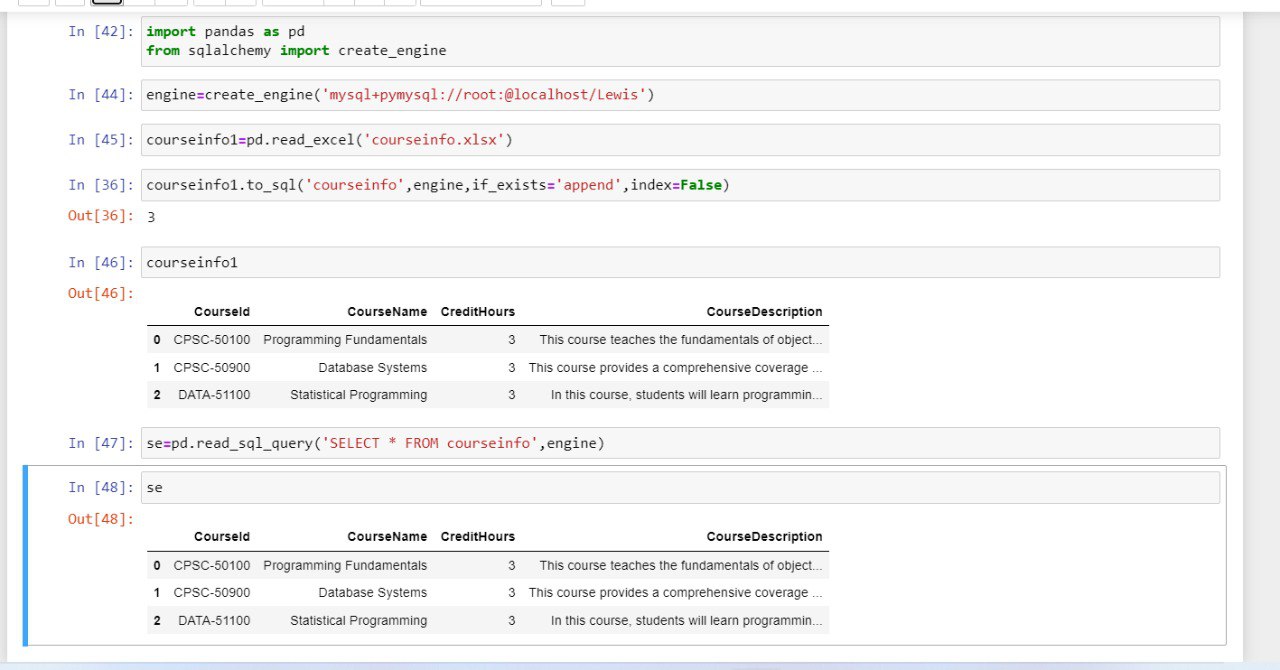
* Inserting data into table: coursescheduleinfo



* Inserting data into table: courseregistrationinfo



* Inserting data into table: Courseinfo
  + Data in to courseinfo table is done using python script as shown below.

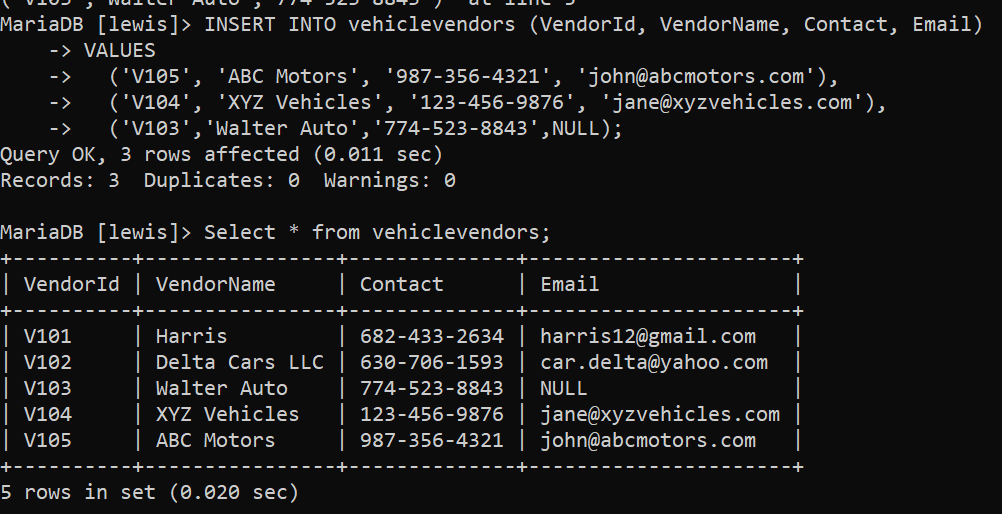


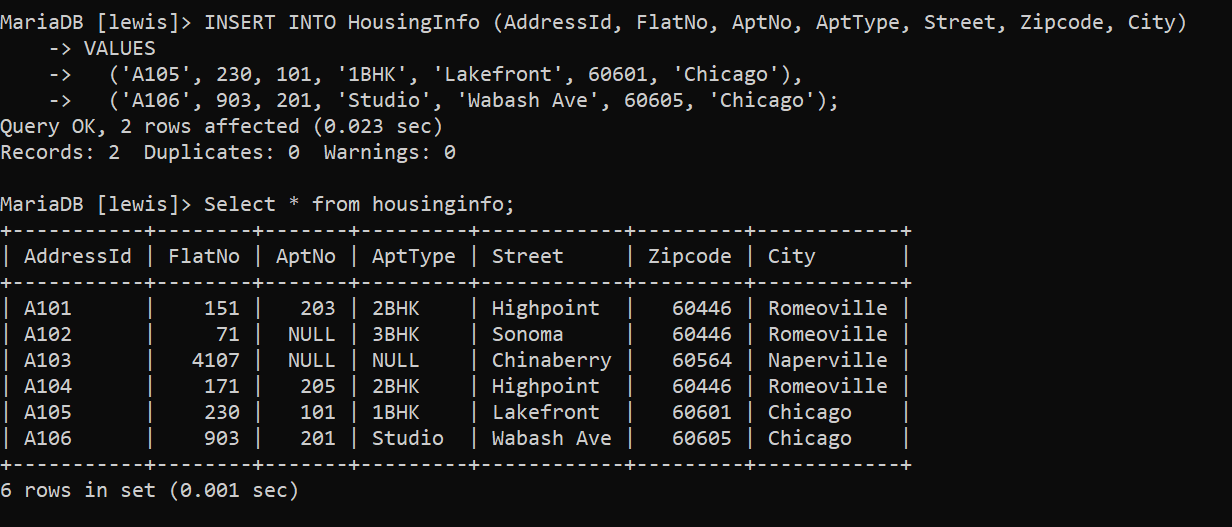
# H. Data Manipulation Language Scripts

## DML Queries link:

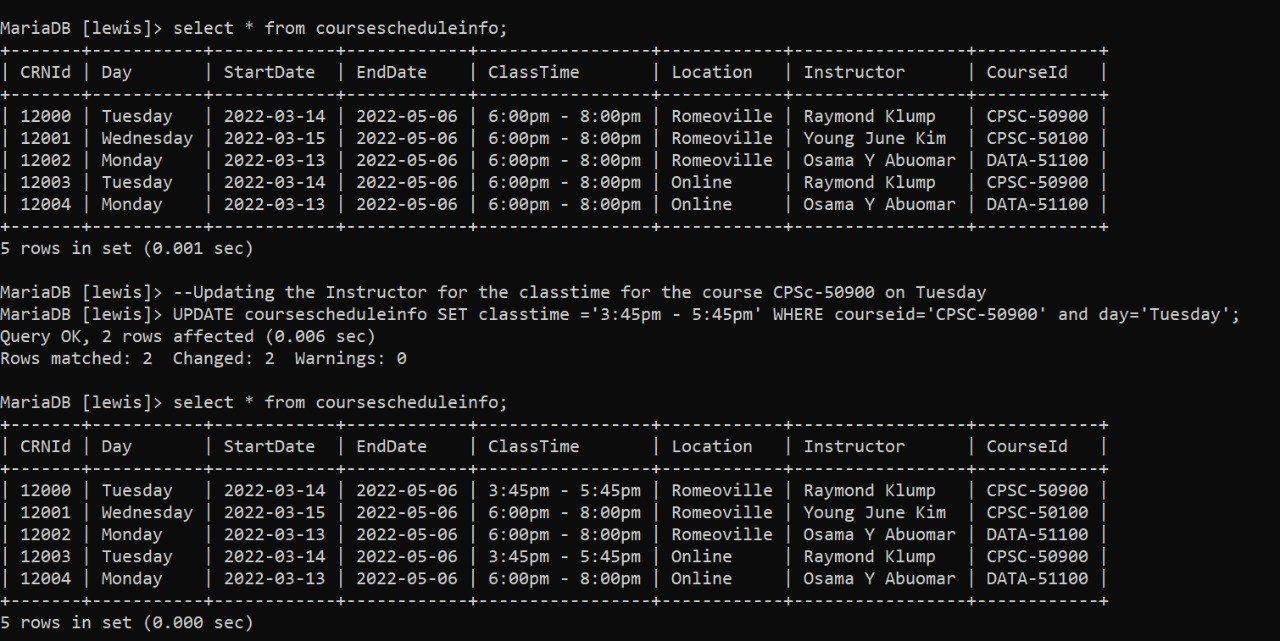
<https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL_scripts/Sample_DML.sql>

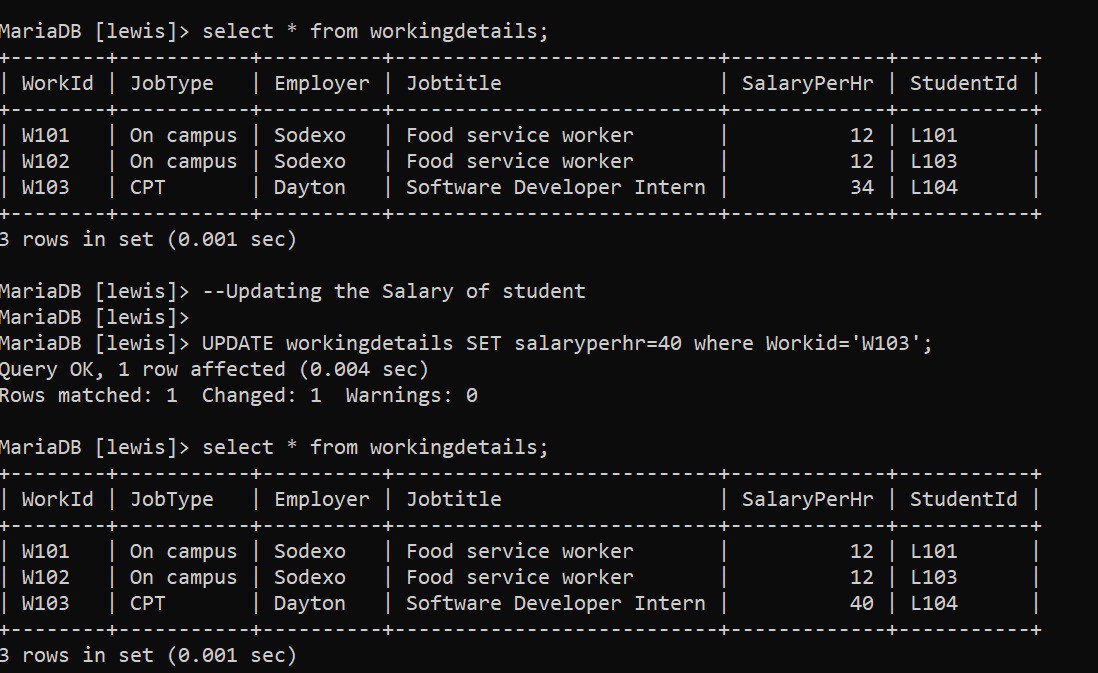
## Insert Queries:



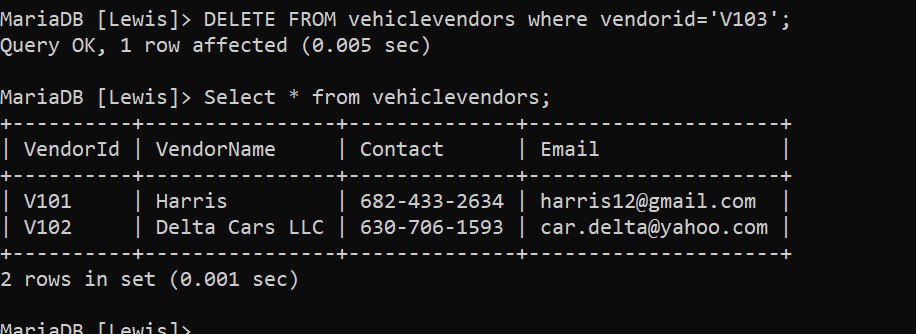


## Update Queries:

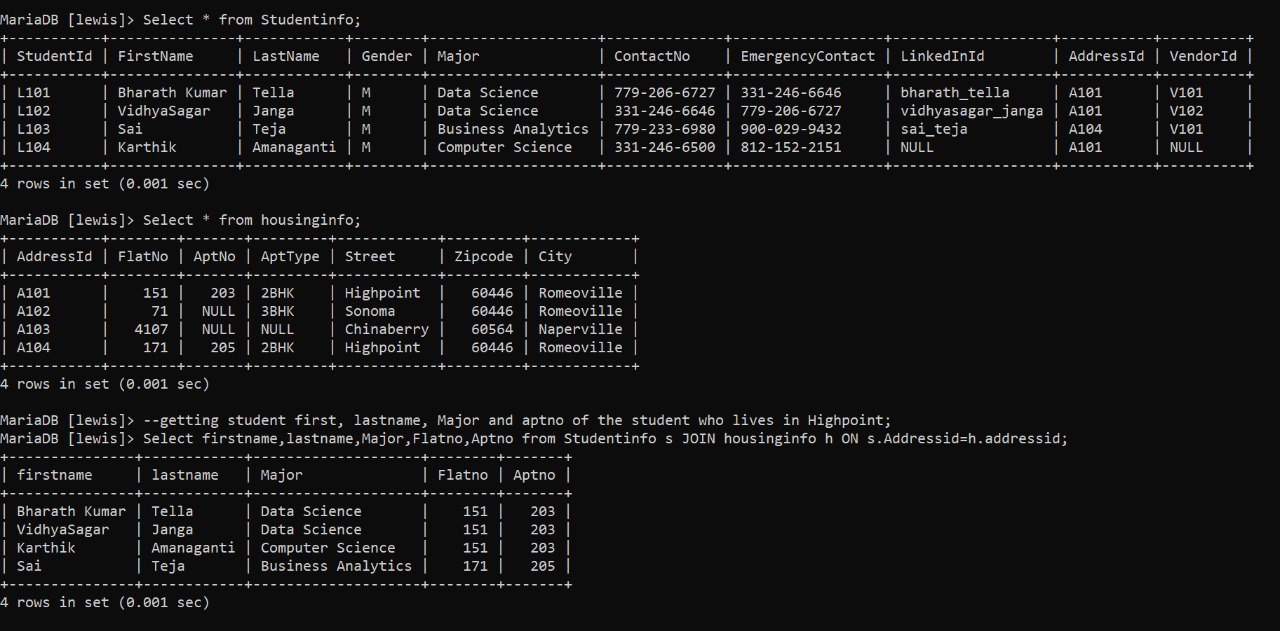




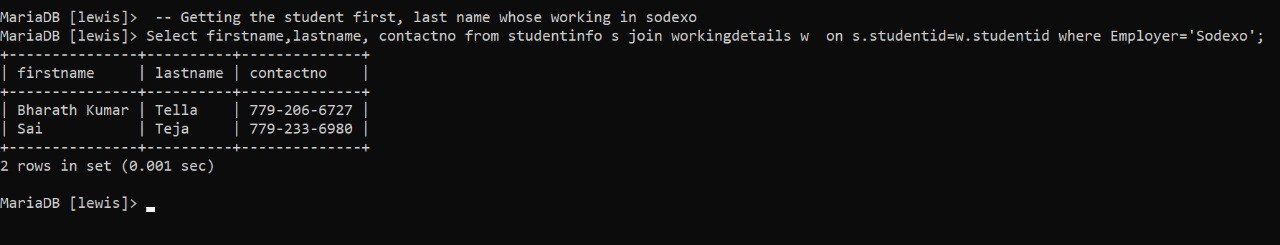
## Delete Query:

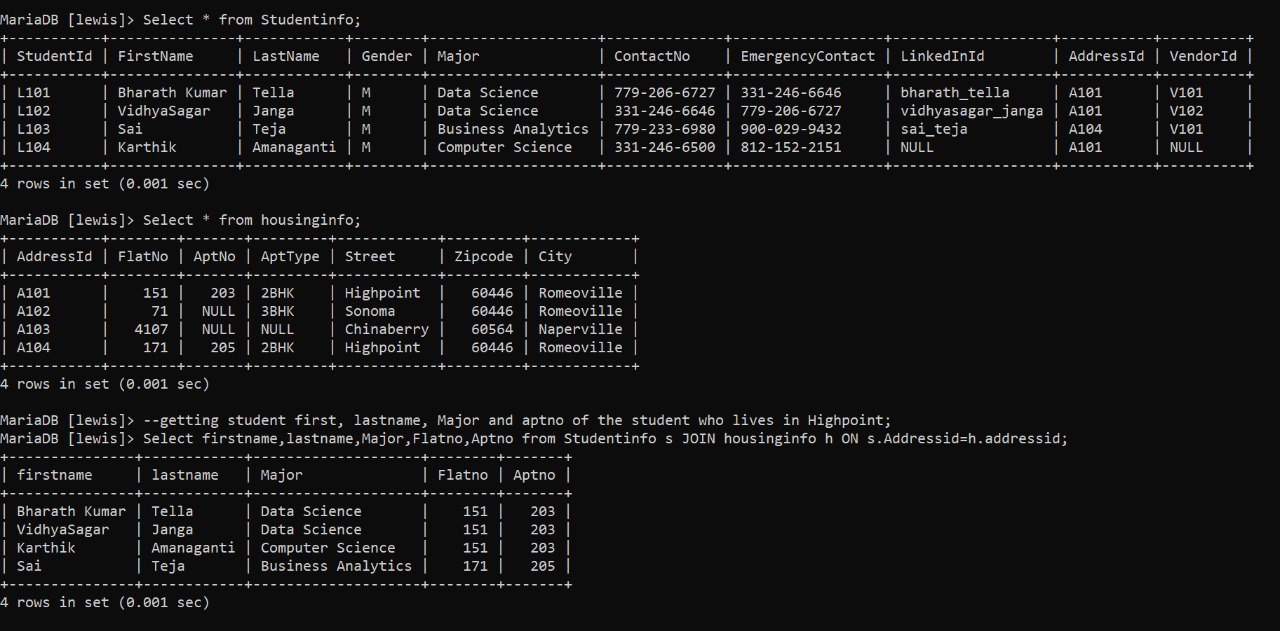


## Select Query:

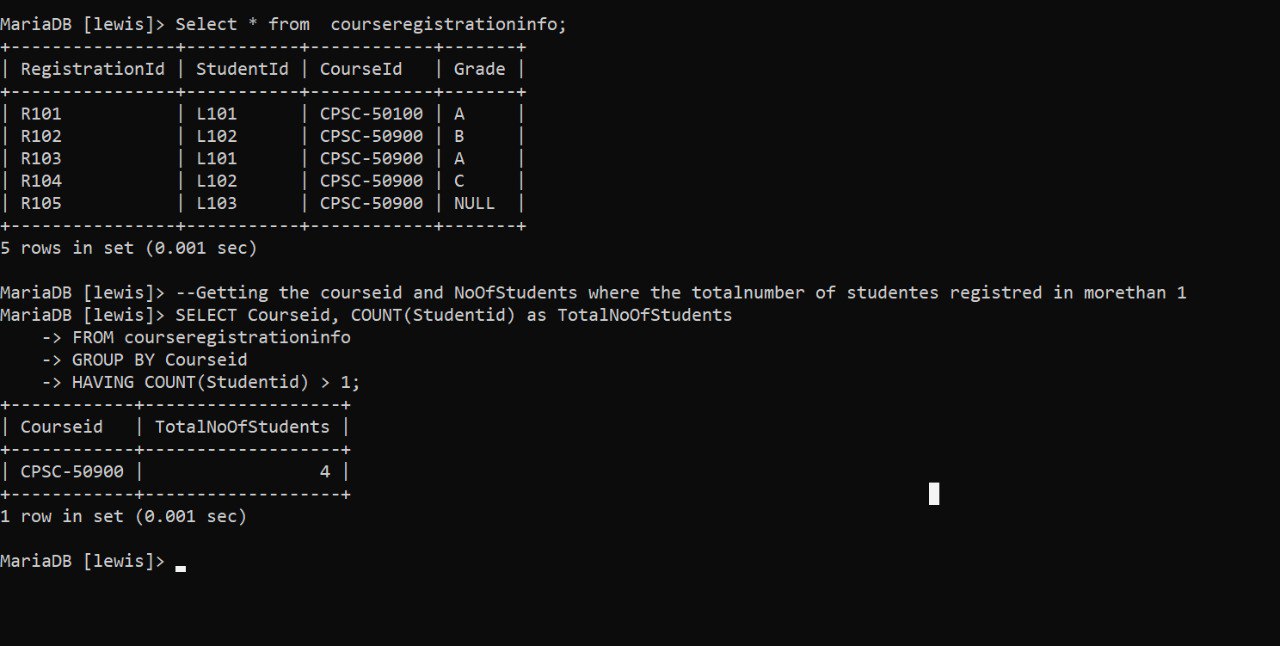


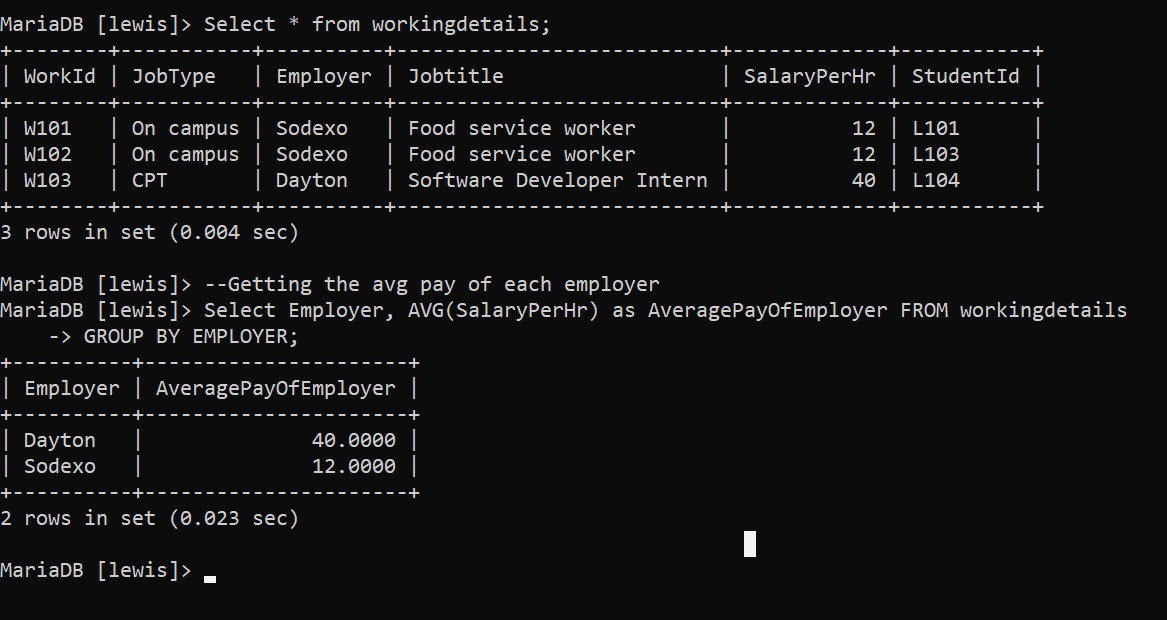
## Join Queries:



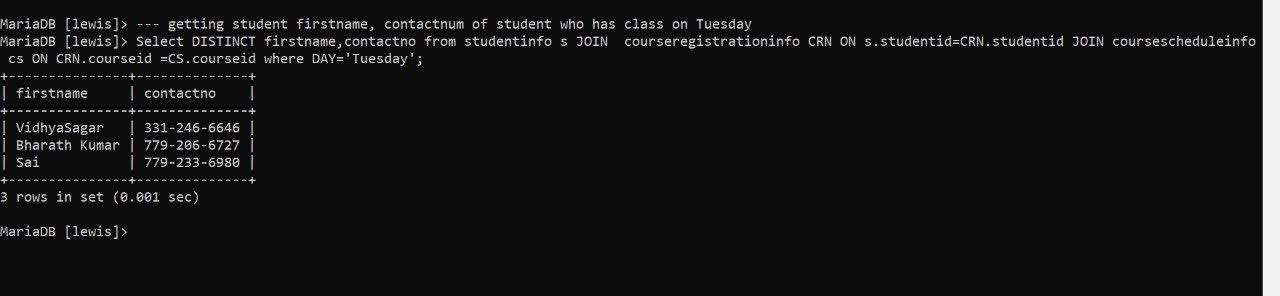


## Summary Queries:

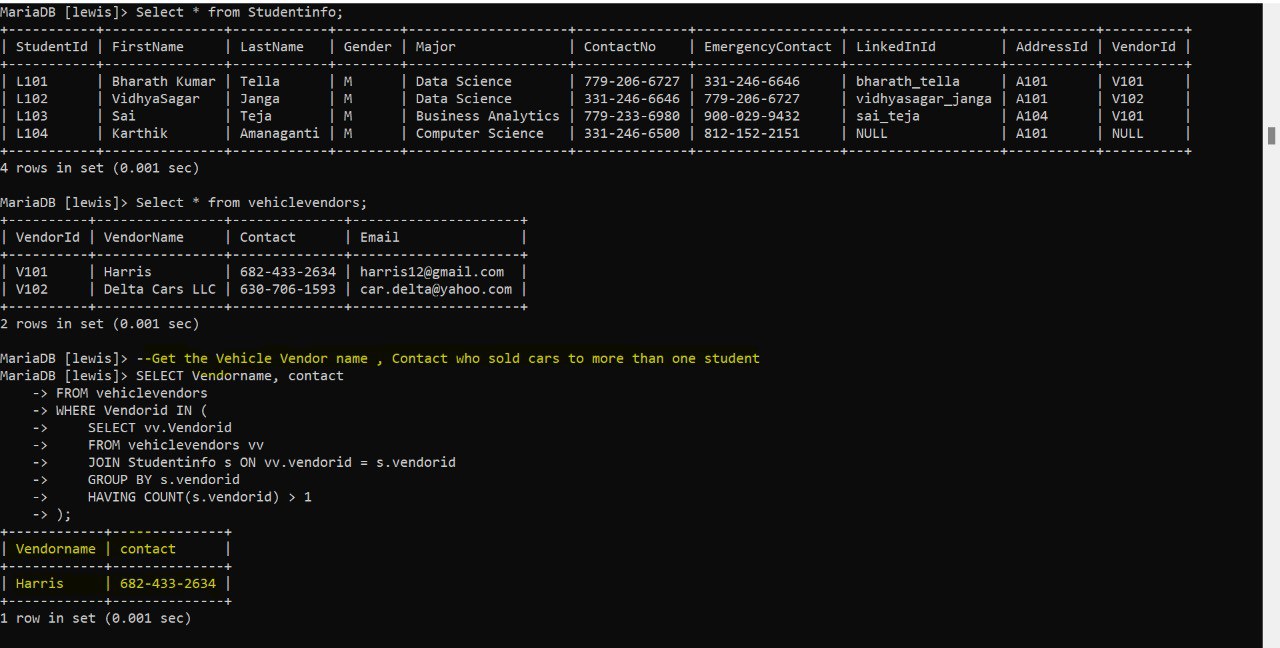


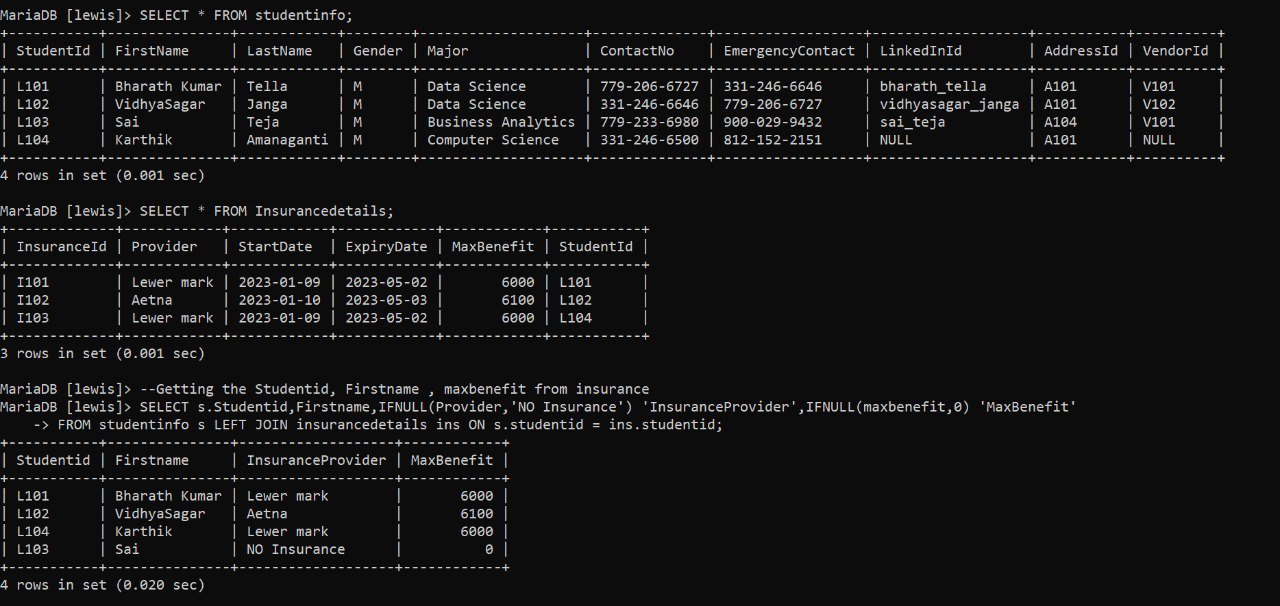


## Multi-table query:



## Query of choice:





# I. Indexes

## Index 1:

### Importance:

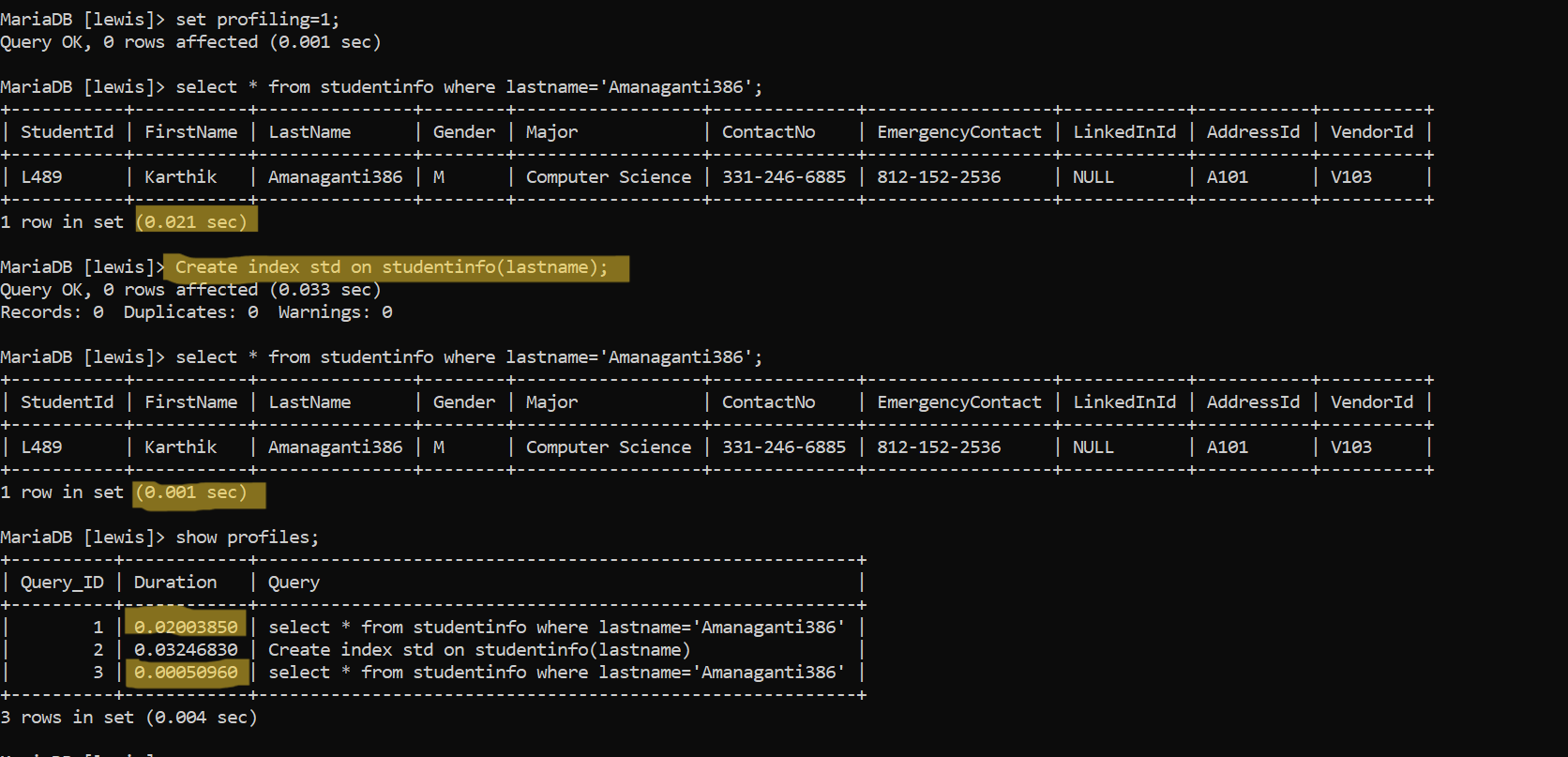
The index created on the "lastname" attribute in the "studentinfo" table can improve the performance of queries that involve searching or sorting by last name. This can be particularly useful in situations where there are many records in the table, as it can help to reduce the time required to retrieve data of students based on their last name.

### SQL generated:

create index std on studentinfo(lastname)

### Performance improvement:

The performance improvement is shown below in the screenshot by reducing the data retrieval time while running the query on the table studentinfo based on the column lastname.   
  
Time taken before index created: **0.02003 sec**Time taken after index created: **0.00051 sec**



## Index 2:

### Importance:

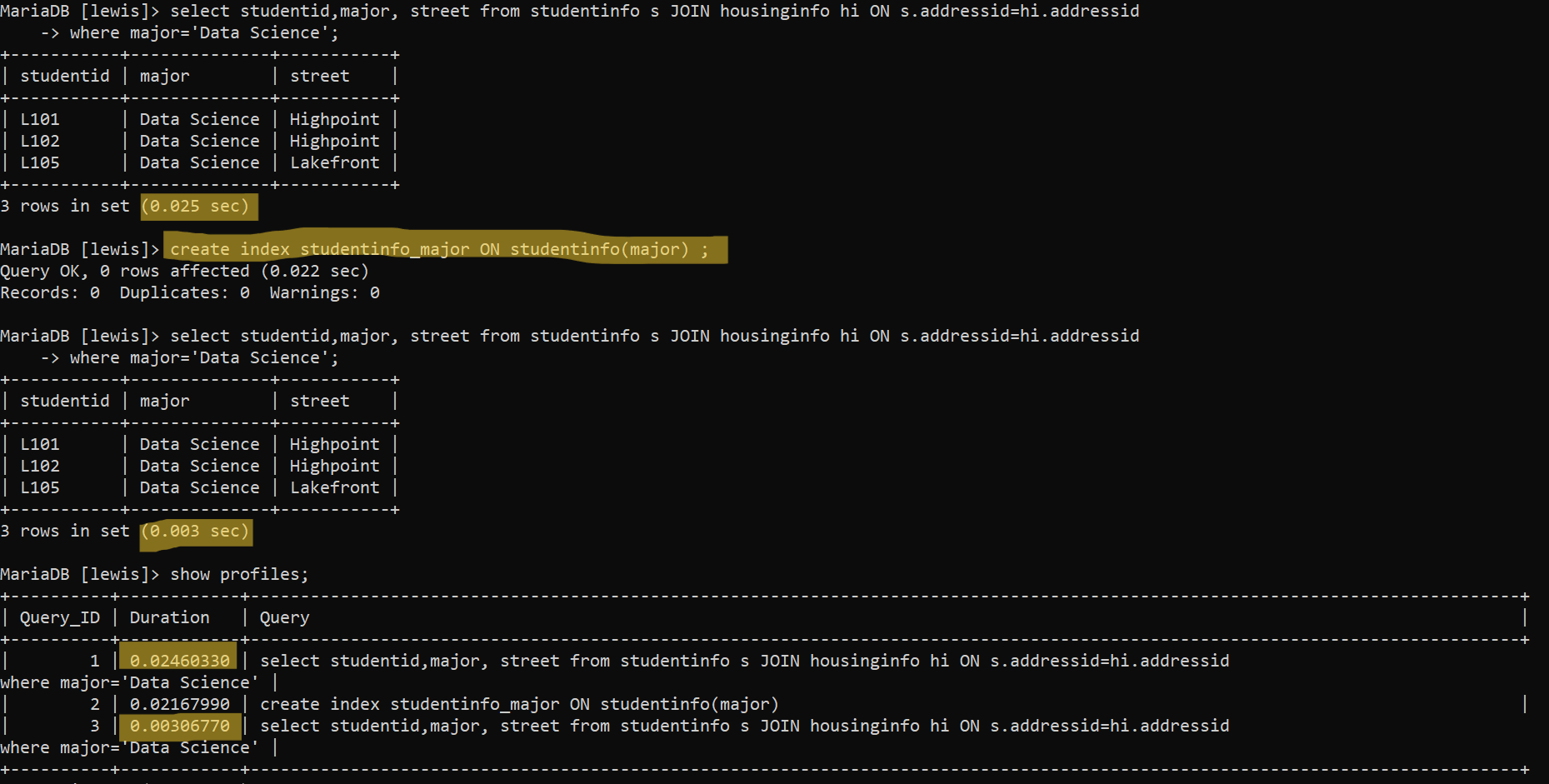
The index created on the "major" attribute in the "studentinfo" table can help to optimize queries that involve filtering or sorting by major. This can be specifically useful in cases when there are many records in the table and the find the students on particular major.

### SQL generated:

create index studentinfo\_major on studentinfo(major)

### Performance improvement:

The performance improvement is shown below in the screenshot by reducing the data retrieval time while running the query on the table studentinfo based on the column major.   
  
Time taken before index created: **0.02460 sec**Time taken after index created: **0.00306 sec**



## Index 3:

### Importance:

The index created on the "vendorname" attribute in the "vehiclevendors" table can improve the performance of queries that involve searching or sorting by vendor name. This can be particularly useful if there are a large number of vendors in the table and get the contact email details of a particular user based on vendor name.

### SQL generated:

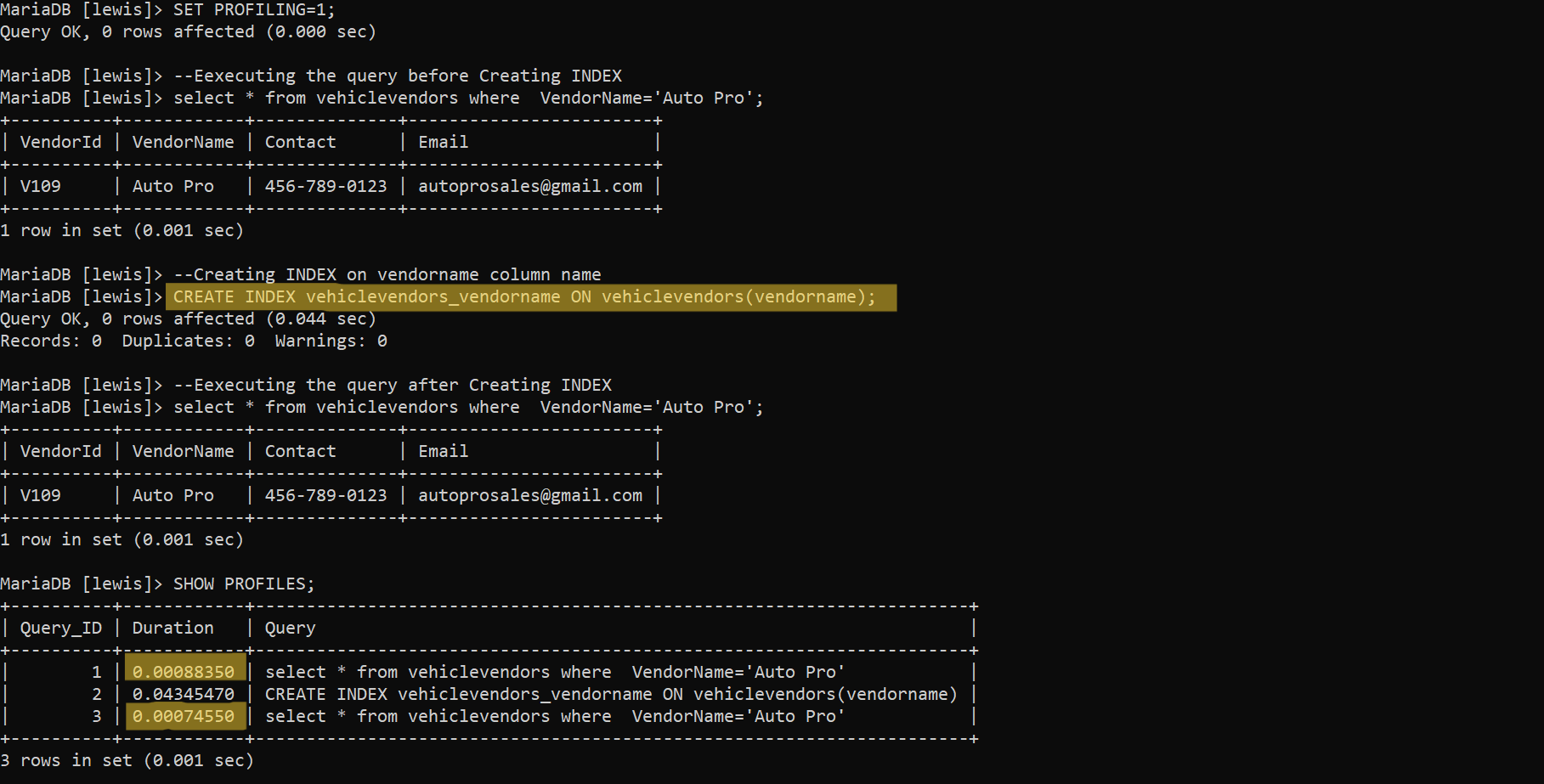
create index vehiclevendors\_vendorname on vehiclevendors(vendorname)

### Performance improvement:

The performance improvement is shown below in the screenshot by reducing the data retrieval time while running the query on the table vehiclevendors based on the column vendorname.

Time taken before index created: **0.02460 sec**

Time taken after index created: **0.00306 sec**



# J. Views

## View 1:

### Git link:

<https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL_scripts/view1.sql>  
Code:

CREATE VIEW StudentCourseSchedule AS

SELECT DISTINCT

s.firstname,

s.contactno,

hs.street,

cs.Day,

cs.courseID

FROM

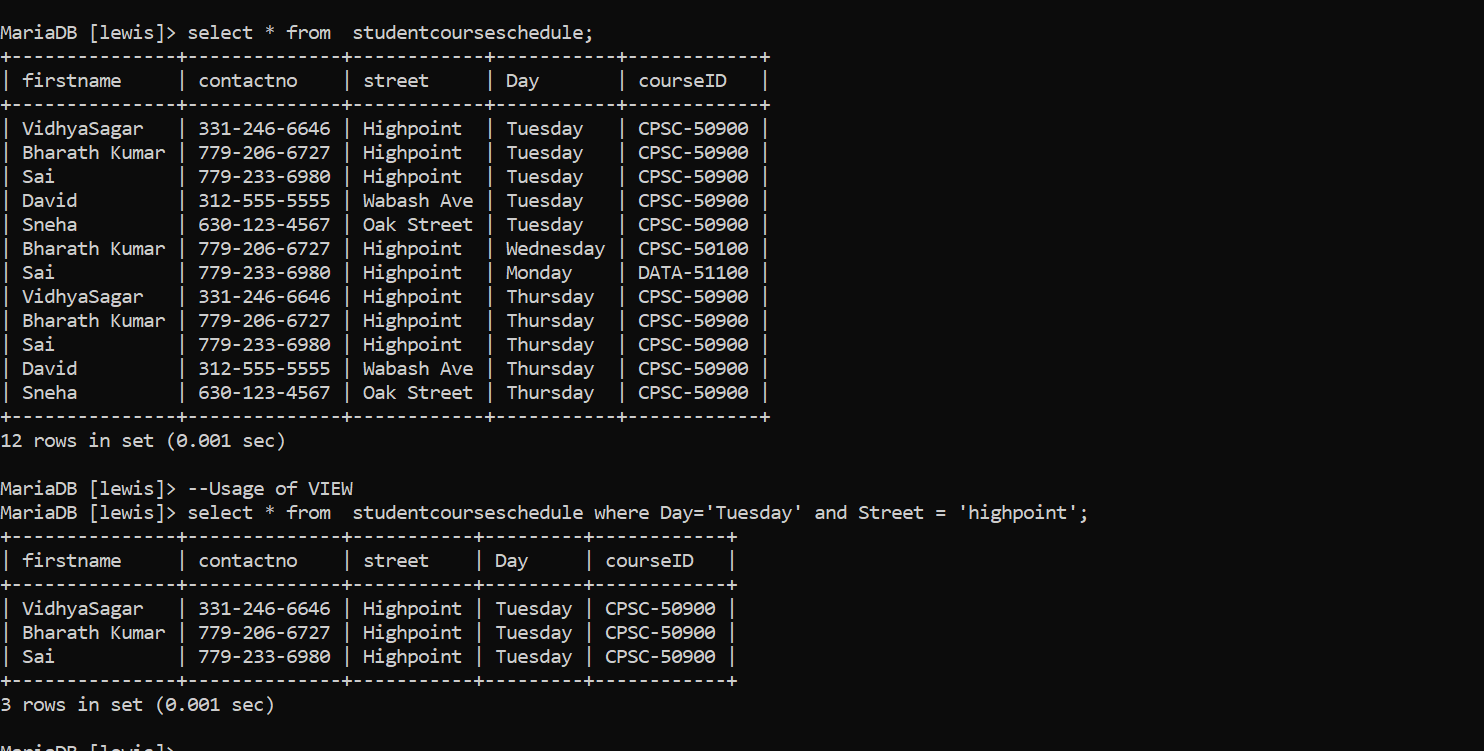
studentinfo s

JOIN courseregistrationinfo CRN ON s.studentid = CRN.studentid

JOIN housinginfo hs ON hs.addressid = s.addressid

JOIN coursescheduleinfo cs ON cs.courseID = CRN.courseId;

### Data contained:



### Importance for database:

* Students can quickly and easily access the contact details of other students who live in the same area and attend classes on the same day.
* Can get the set of students taking offline classes on a administrative level.

## View 2:

### SQL created:

https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL\_scripts/view2.sql

### Code:

CREATE VIEW student\_housing\_insurance\_info AS

SELECT -- selecting all the required attributes required to display

si.StudentId,

hi.FlatNo,

hi.AptNo,

hi.AptType,

hi.Street,

hi.Zipcode,

hi.City,

id.Provider,

id.StartDate AS InsuranceStartDate,

id.ExpiryDate AS InsuranceExpiryDate,

id.MaxBenefit,

CASE -- case to define the conditions

WHEN id.InsuranceId LIKE 'I%' THEN 'HealthInsurance'

WHEN id.InsuranceId LIKE 'D%' THEN 'VehicleInsurance'

ELSE NULL

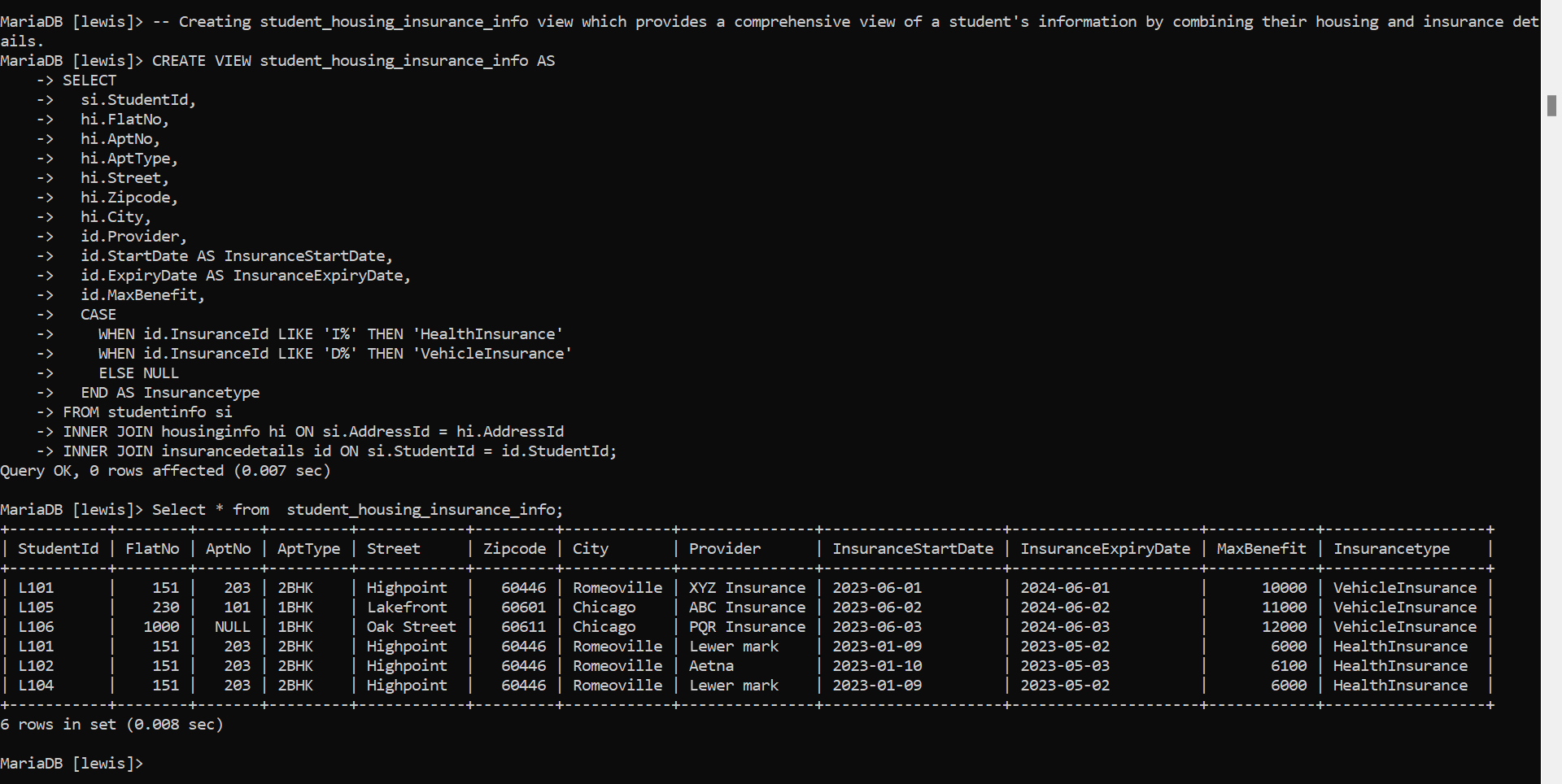
END AS Insurancetype

FROM studentinfo si

INNER JOIN housinginfo hi ON si.AddressId = hi.AddressId

INNER JOIN insurancedetails id ON si.StudentId = id.StudentId;

### Data contained:



### Importance for database:

The student\_housing\_insurance\_info view is important because

* It gives all the information about a student's housing and insurance in one place.
* Useful for administrative tasks, tracking student records, and managing student services.
* It saves time and reduces errors by not having to search multiple tables.

# K. Triggers

## SQL for creating trigger:

### Git link:

<https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL_scripts/trigger.sql>

### Code:

Begin;

delimiter //

CREATE TRIGGER trg\_studentinfo\_insert

BEFORE INSERT ON studentinfo

FOR EACH ROW

BEGIN

DECLARE last\_id VARCHAR(4);

SELECT MAX(studentid) INTO last\_id FROM studentinfo;

IF last\_id IS NULL THEN

SET NEW.studentid = 'L101';

ELSE

SET NEW.studentid = CONCAT('L', LPAD(SUBSTRING(last\_id, 2) + 1, 3, '0'));

END IF;

END//

delimiter ;

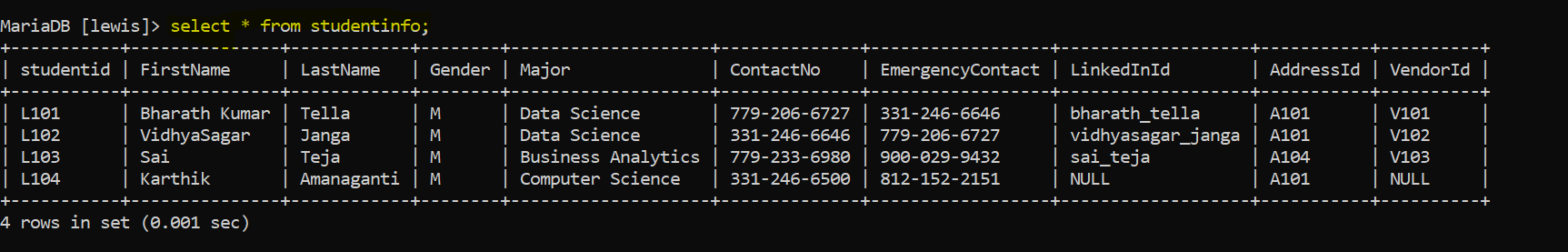
commit;

## Purpose:

* To automatically generate a unique student ID for each new record inserted into the "studentinfo" table.
* If there are no existing records in the table, it will set the student ID to 'L101'.
* The trigger is executed before each new record is inserted into the table, ensuring that the student ID is always generated and assigned automatically.
* This helps to maintain data integrity and consistency in the database.

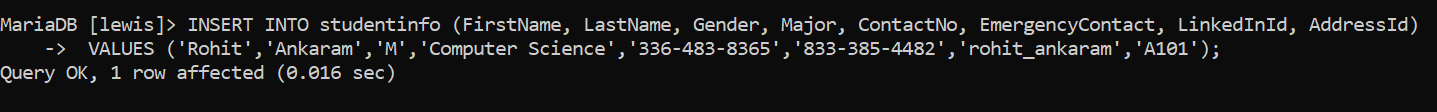
## Screenshots of output:

### Table contents before creating trigger:

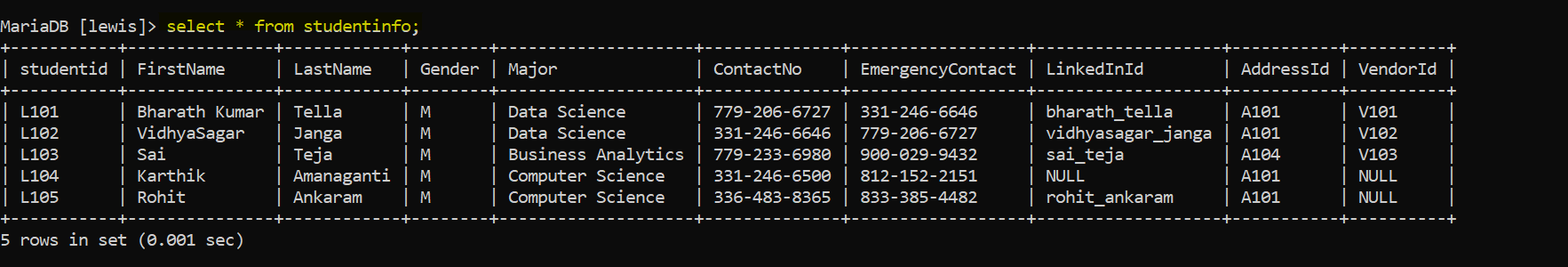


### Trigger created on database:

### Inserting into table without PK:



### Table contents after creating trigger and inserting values:



# L. Transactions

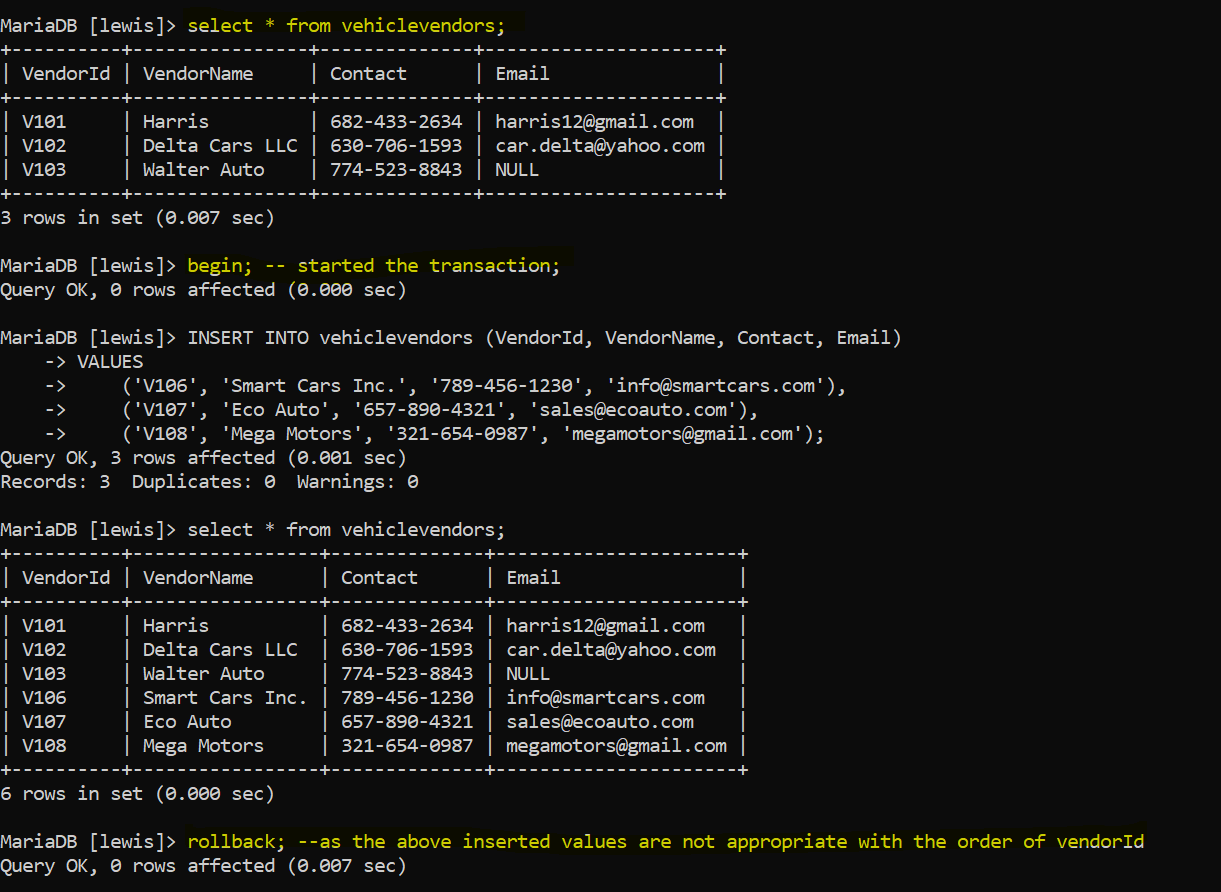
ACID: Atomicity, Consistency, Isolation, Durability

## Importance of transactions ensuring ACID behavior:

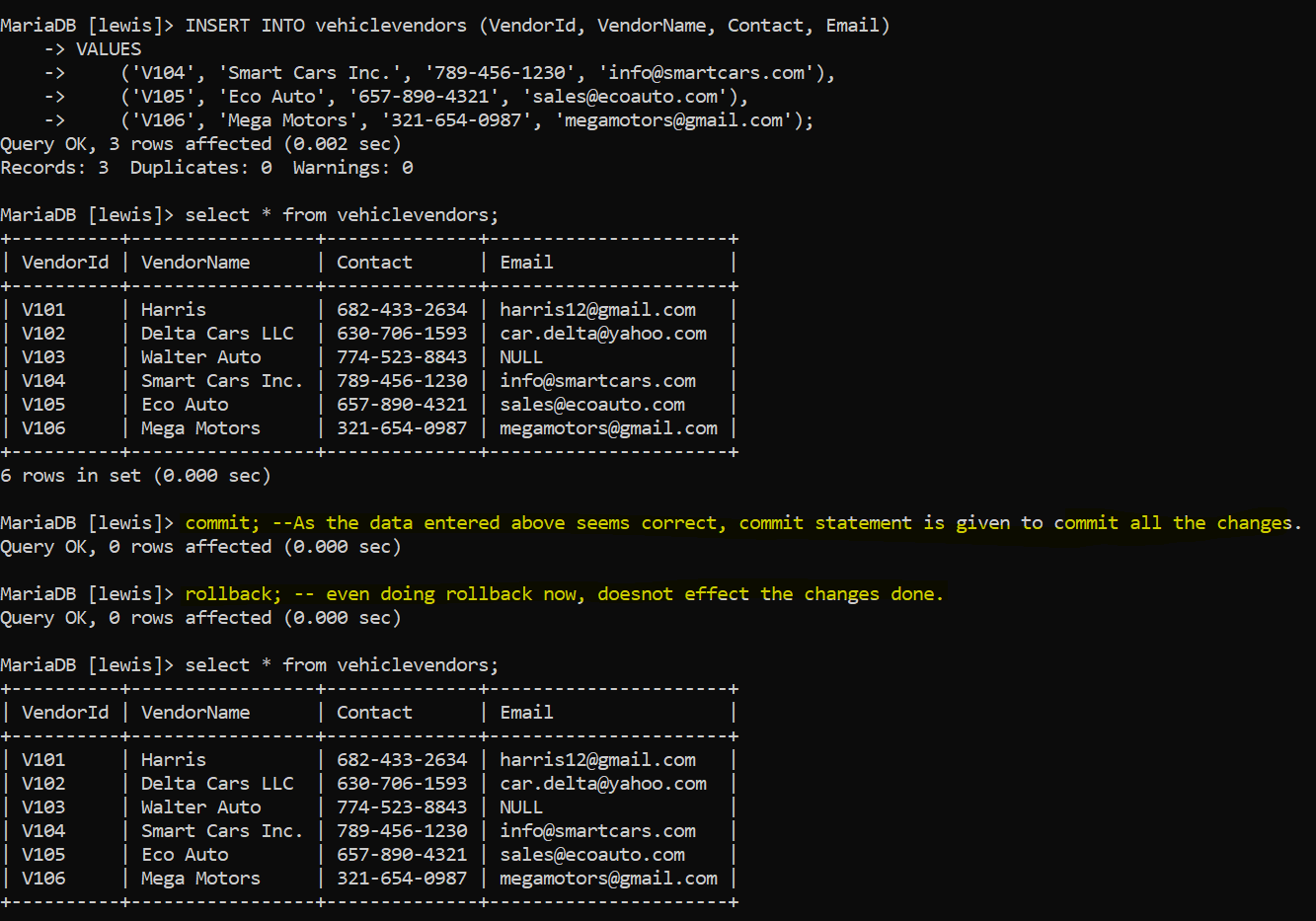
They are important for ensuring ACID behavior because they allow a set of related database operations to be executed as a single unit of work that is either all committed, or all rolled back. This means that the database remains in a consistent state, even in the event of errors or system failures, and ensures that the database satisfies the ACID properties.

* **Atomicity** ensures that either all the operations in the transaction are completed successfully or none of them are, so the database remains in a consistent state.
* **Consistency** ensures that the database remains in a valid state throughout the transaction.
* **Isolation** ensures that concurrent transactions do not interfere with each other,
* **Durability** ensures that the effects of a committed transaction are permanent even in the event of system failures.

## Screenshots performing transactions on database:



In the above screenshot, as the data entered using the insert statements is not correct, it is reverted using rollback statement after checking the database.



In the above screenshot once the rollback of the illegal transaction is done, corrected data is inserted, and changes are committed. So, even though the rollback command is executed, no change happened.

## Operations used while doing transactions:

### Begin:

Starts the transaction.

### Commit:

Commits all the statements executed after the begin statement and ends the transaction.  
Commits only if no errors occurred while executing all the statements above.

### Rollback:

Roll backs all the changes done earlier.

# M. Database Security

## Various users on the database:

The following are the various kinds of users that will use the databases and the privileges that each should have:

**Administrators**: These are the users who will manage the database and have full control over it. They should have privileges to create, modify, and delete all database objects, as well as grant and revoke permissions to other users.

**Students**: These are the users who will access their personal information, such as course registration, housing details, insurance details, and working details. They should have SELECT, INSERT, and UPDATE privileges on their respective tables to view, add, and modify their information.

**Professors**: These are the users who will access course-related information, such as course schedules, course registration details, and course information. They should have SELECT, INSERT, and UPDATE privileges on the respective tables to view, add, and modify course information.

**VehicleVendors**: These are the users who will access information related to vehicles, such as the list of vehicle vendors and their details. They should have SELECT, INSERT, and UPDATE privileges on the vehiclevendors table to view, add, and modify vendor information. They should also have SELECT privileges on student and housing information to provide their services.

**Employers**: These are the users who will access the working details of students, such as their work schedule, employer information, and payment details. They should have SELECT, INSERT, and UPDATE privileges on the workingdetails table to view, add, and modify working details. They should also have SELECT privileges on student and housing information to verify the student's employment eligibility.

**Insurance Providers**: These are the users who will access insurance-related information, such as insurance details of students and their housing details. They should have SELECT, INSERT, and UPDATE privileges on the insurancedetails table to view, add, and modify insurance details. They should also have SELECT privileges on student and housing information to verify the student's insurance eligibility.

## GRANT statements that assign privileges to these different kinds of users

--SQL CODE for creating adminstrators

CREATE USER 'Sagar' IDENTIFIED BY '8106064932a@A';

CREATE USER 'bharath' IDENTIFIED BY '765997998a@A';

--SQL code for granting permissions

GRANT SELECT, UPDATE, INSERT ON \*.\* TO 'Sagar','bharath';

--SQL code for creating student users

CREATE USER 'karthik' IDENTIFIED BY 'karthik123';

CREATE USER 'rohit' IDENTIFIED BY 'rohit123';

--SQL CODE for Granting permissions to students

GRANT SELECT,INSERT, UPDATE ON studentinfo TO 'karthik','rohit';

GRANT SELECT,INSERT, UPDATE ON housinginfo TO 'karthik','rohit';

GRANT SELECT,INSERT, UPDATE ON courseregistrationinfo TO 'karthik','rohit';

GRANT SELECT ON courseinfo TO 'karthik','rohit';

GRANT SELECT ON workingdetails TO 'karthik','rohit';

GRANT SELECT ON insurancedetails TO 'karthik','rohit';

GRANT SELECT ON coursescheduleinfo TO 'karthik','rohit';

GRANT SELECT ON vehiclevendors TO 'karthik','rohit';

--SQL CODE FOR CREATING USERS 'PROFESSORS'

CREATE USER 'rklump' IDENTIFIED BY 'klump123';

CREATE USER 'choward' IDENTIFIED BY 'howard123';

--SQL CODE for Granting permissions to 'PROFESSORS'

GRANT SELECT,INSERT, UPDATE ON courseregistrationinfo TO 'rklump','choward';

GRANT SELECT,INSERT, UPDATE ON coursescheduleinfo TO 'rklump','choward';

GRANT SELECT,INSERT, UPDATE ON courseinfo TO 'rklump','choward';

GRANT SELECT ON housinginfo TO 'rklump','choward';

GRANT SELECT ON workingdetails TO 'rklump','choward';

GRANT SELECT ON studentinfo TO 'rklump','choward';

GRANT SELECT ON vehiclevendors TO 'rklump','choward';

--SQL code for creating VehicleVendors

CREATE USER 'harrisvv' IDENTIFIED BY 'harris123';

CREATE USER 'haribabu' IDENTIFIED BY 'haribabu123';

--SQL CODE for Granting permissions to 'VehicleVendors'

GRANT SELECT, INSERT, UPDATE ON vehiclevendors TO 'harrisvv','haribabu';

GRANT SELECT ON studentinfo TO 'harrisvv','haribabu';

GRANT SELECT ON housinginfo TO 'harrisvv','haribabu';

GRANT SELECT ON insurancedetails TO 'harrisvv','haribabu';

--SQL code for creating Employers

CREATE USER 'sodexo\_manager' IDENTIFIED BY 'sodexo123';

CREATE USER 'dayton\_manager' IDENTIFIED BY 'dayton123';

--SQL CODE for Granting permissions to 'VehicleVendors'

GRANT SELECT, INSERT, UPDATE ON workingdetails TO 'sodexo\_manager','dayton\_manager';

GRANT SELECT ON studentinfo TO 'sodexo\_manager','dayton\_manager';

GRANT SELECT ON housinginfo TO 'sodexo\_manager','dayton\_manager';

GRANT SELECT ON courseregistrationinfo TO 'sodexo\_manager','dayton\_manager';

--SQL code for creating Insurance Providers

CREATE USER 'Lewer\_mark' IDENTIFIED BY 'lewer123';

CREATE USER 'aetna' IDENTIFIED BY 'aetna123';

--SQL CODE for Granting permissions to Insurance Providers

GRANT SELECT, INSERT, UPDATE ON insurancedetails TO 'Lewer\_mark','aetna';

GRANT SELECT ON studentinfo TO 'Lewer\_mark','aetna';

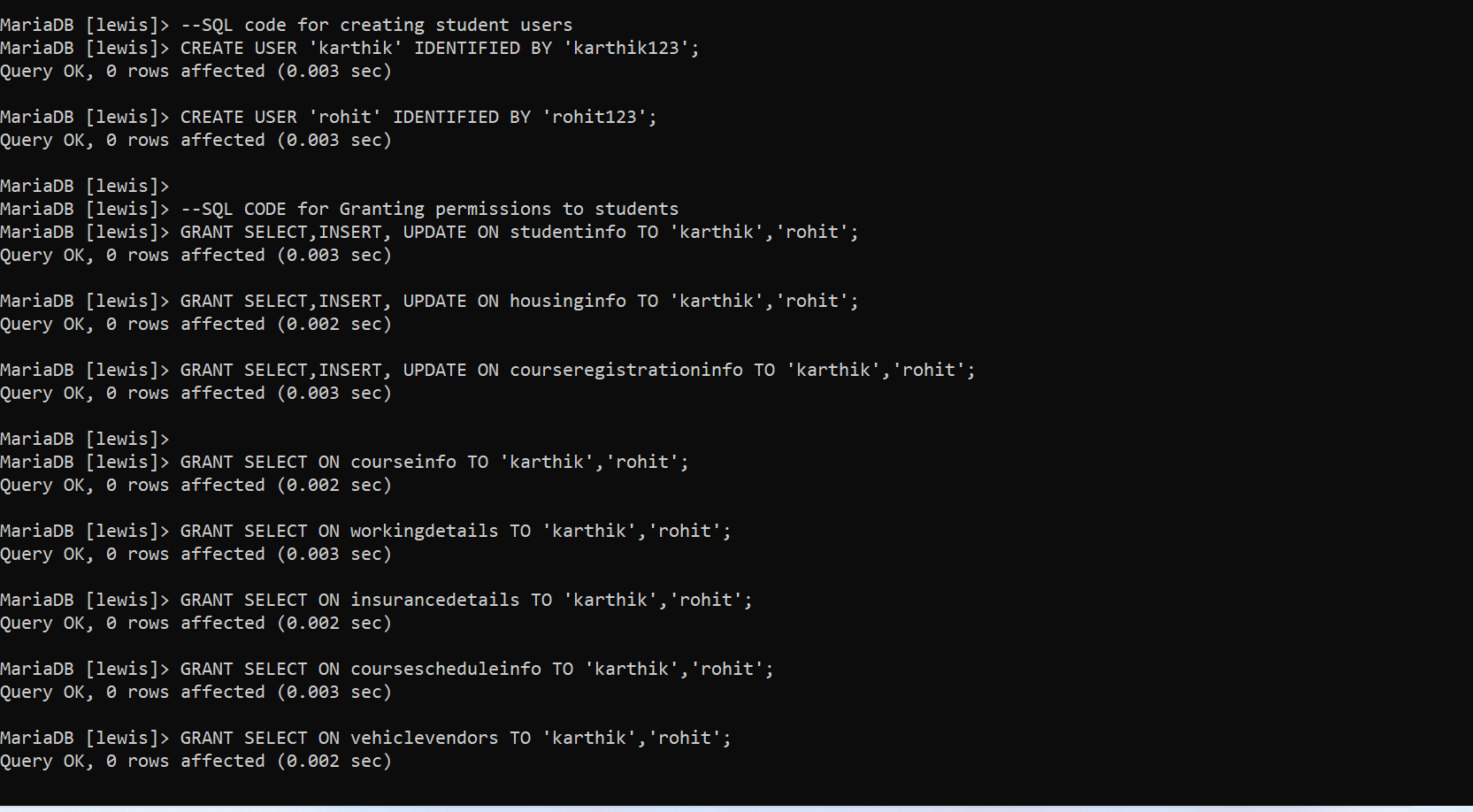
GRANT SELECT ON housinginfo TO 'Lewer\_mark','aetna';

## Sql executed upload to git:

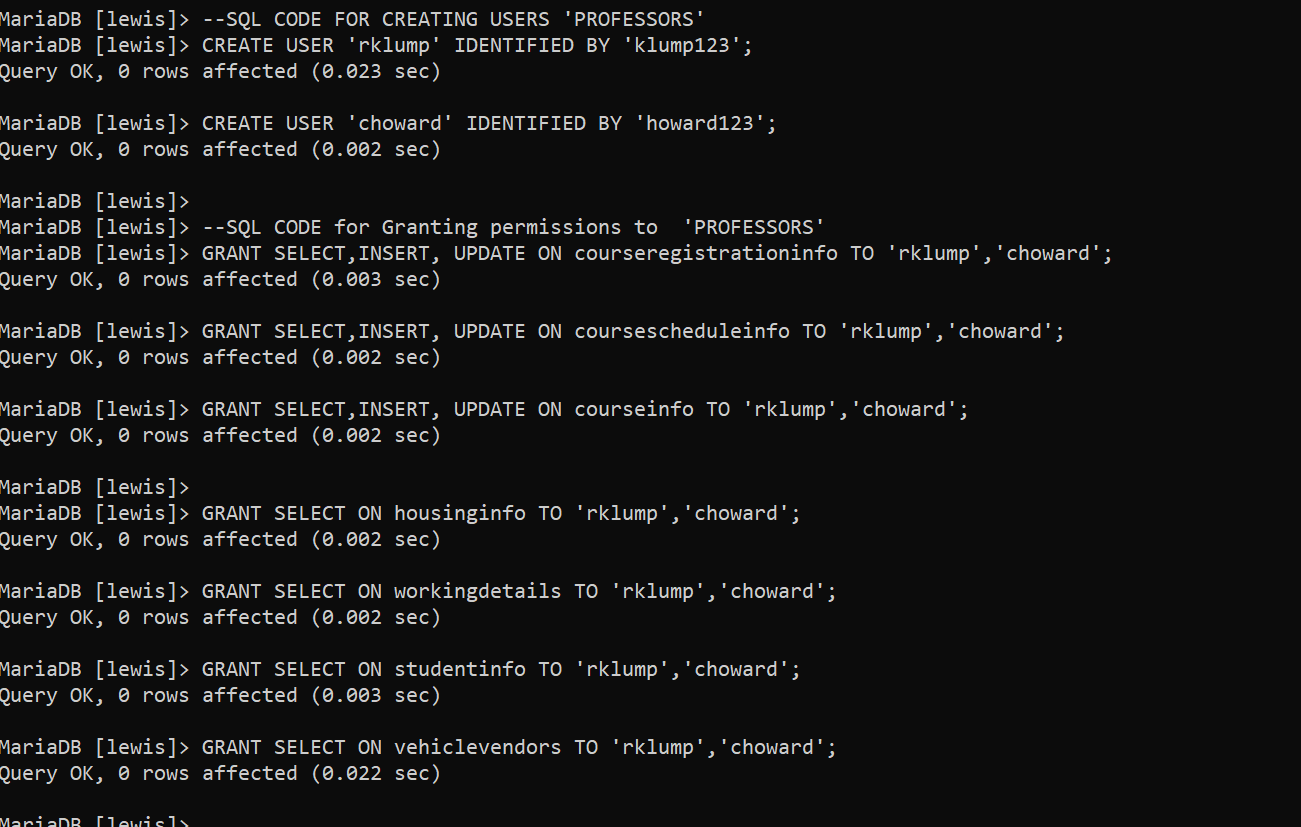
<https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/SQL_scripts/Grant_statements.sql>

## Screenshots creating differrent users:

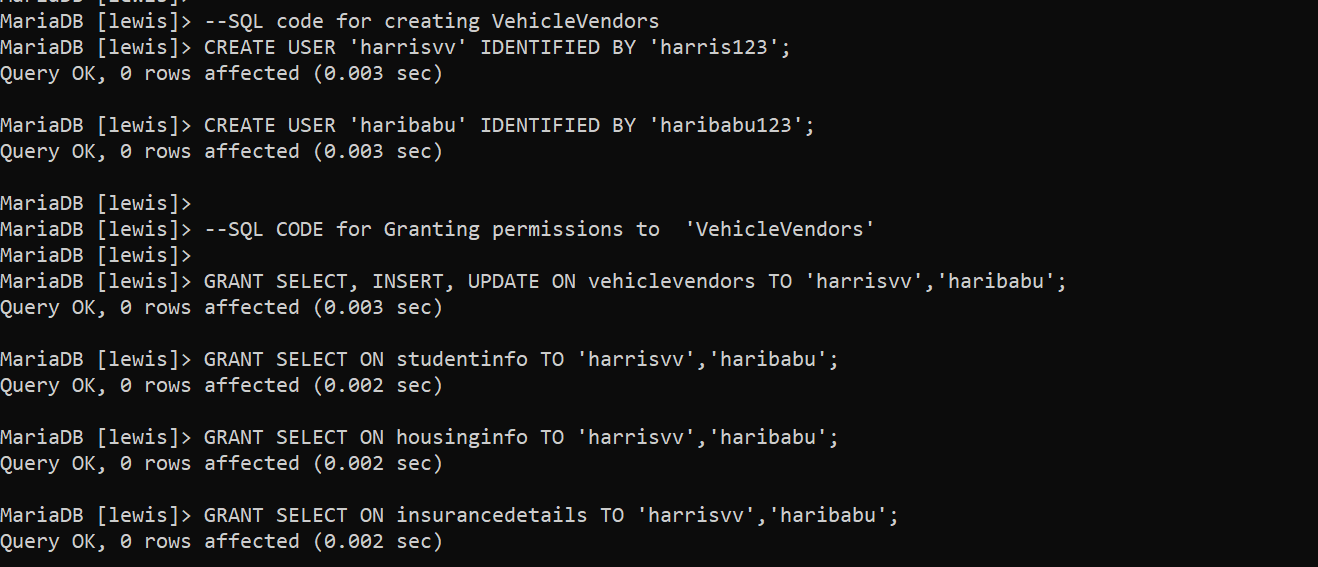
### Creating student user

****

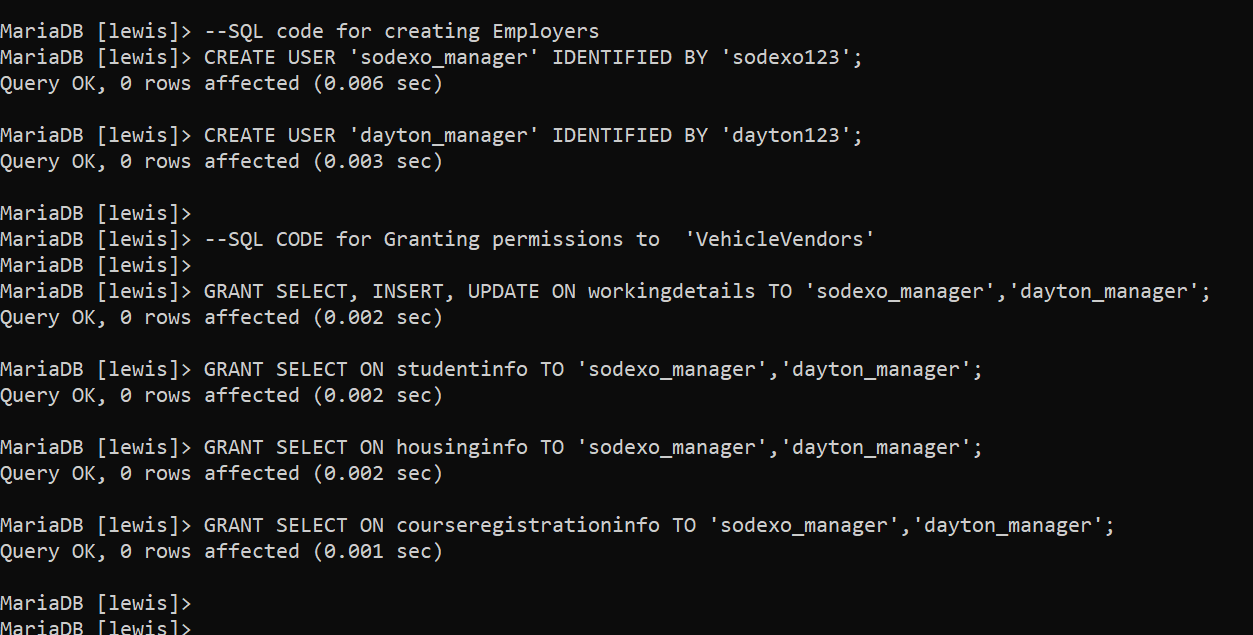
### CREATING USERS: PROFESSORS:

****

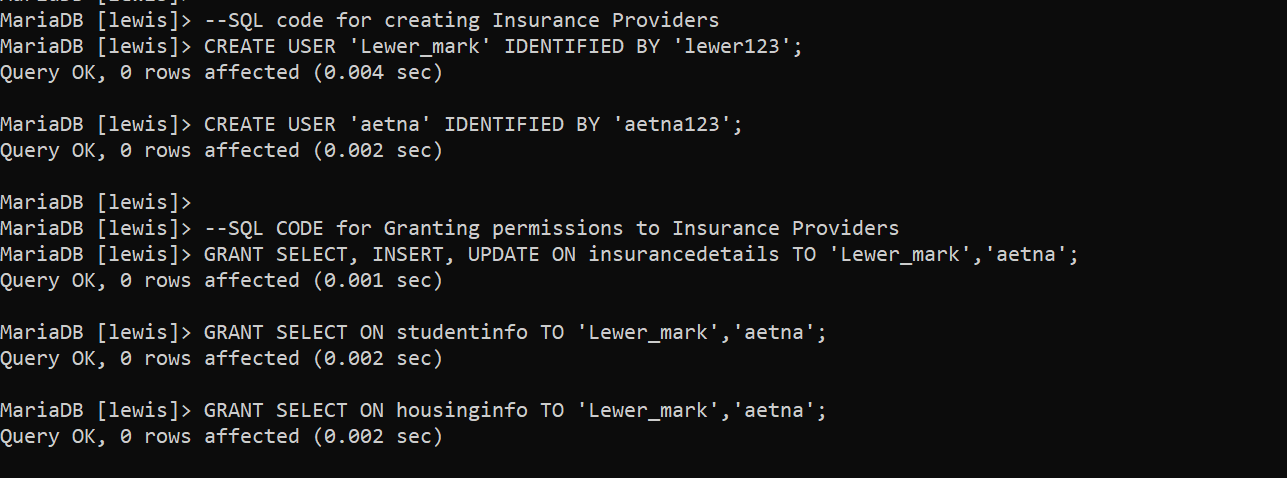
### VEHICLEVENDORS:

****

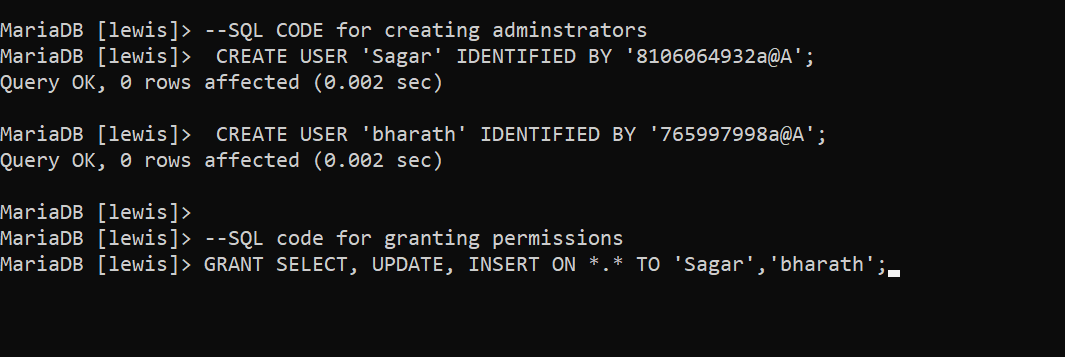
### Employers:

****

### INSURANCE PROVIDERS:

****

### ADMINSTRATORS:



# N. Locking and Concurrent Access

## Purpose of locking tables:

**Data consistency**: Locking of tables ensures that the data in a table is not modified by multiple users simultaneously, which can lead to inconsistent data. When a table is locked, only one user can modify the data at a time, ensuring that the data remains consistent.

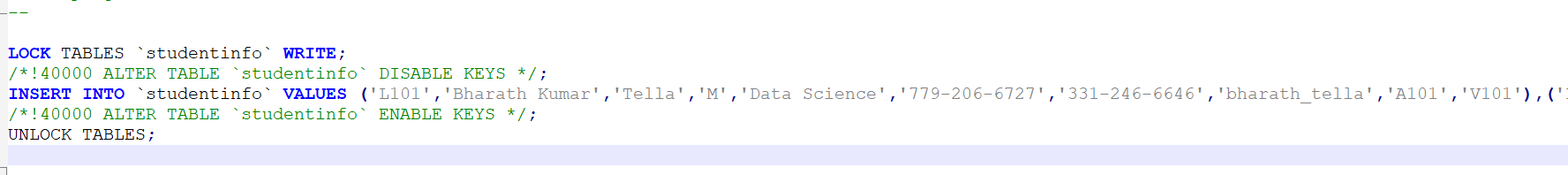
**Preventing conflicts**: Locking of tables can help prevent conflicts between transactions that are accessing the same data. By locking a table, a transaction can prevent other transactions from modifying the same data, which can help prevent conflicts that could lead to data inconsistencies.

**Maintaining data integrity**: Locking of tables can help maintain data integrity by ensuring that transactions are completed in a consistent and controlled manner. By locking a table, a transaction can ensure that it has exclusive access to the data it needs to complete its task, and that no other transactions are modifying the data at the same time.

**Ensuring atomicity**: Locking of tables is essential for ensuring the atomicity of transactions, which means that a transaction is either completed in its entirety or not at all. By locking a table, a transaction can ensure that all the changes it makes to the data are completed before the lock is released, ensuring that the transaction is atomic.

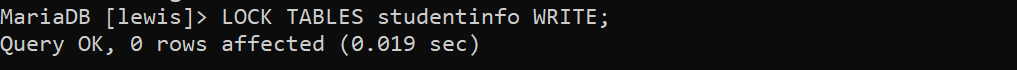
## Screenshots performing locking using backup file:

The below example from the backup file ensures the locking the ‘write’ operation on the ‘studentinfo‘ table so that no inserting to the table is done while backing up the table.



## Performing locking on our database:

Locking the write operation on the database

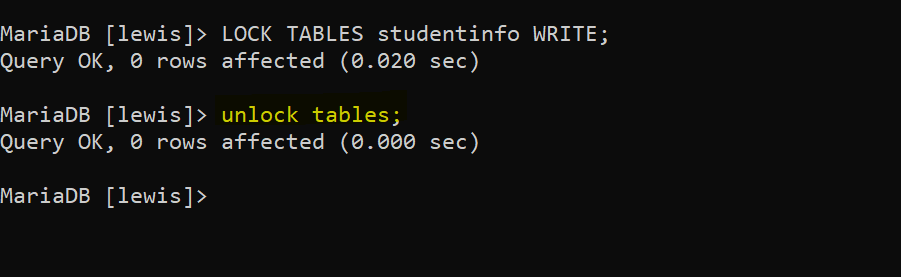


Executing the update operation on another cmd after locking:

Performing this operation waits until the unlocking happens



Unlocking the tables



Releases the update operation and updates the table:



# O. Backing Up Your Database

## Steps taken to back up database:

### Frequency of backup:

As the database is not changed frequently, the frequency that needs to be maintained is twice per week.

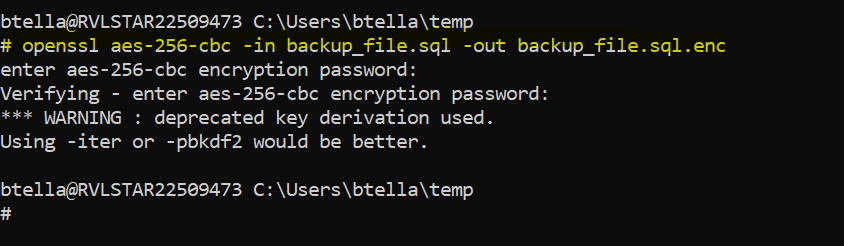
### Storage location:

The backup file of the database is saved as a file backup\_file.sql where it is saved in the temp location created in the personal directory.

### Security of backup file:

The file created is encrypted with a password using the below command.

openssl aes-256-cbc -in backup\_file.sql -out backup\_file.sql.enc

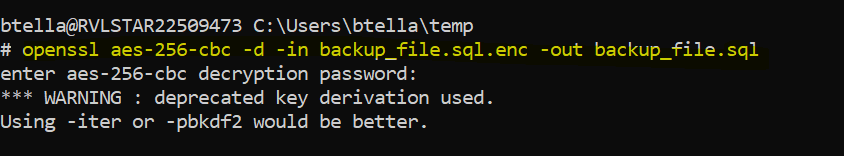


The encrypted file is then uploaded to git and link is as below:

<https://github.com/VidhyaSagarJanga/Lewis-University-International-Student-Information/blob/Sagar/backup_file.sql.enc>

Command used for decrypting:

openssl aes-256-cbc -d -in backup\_file.sql.enc -out backup\_file.sql

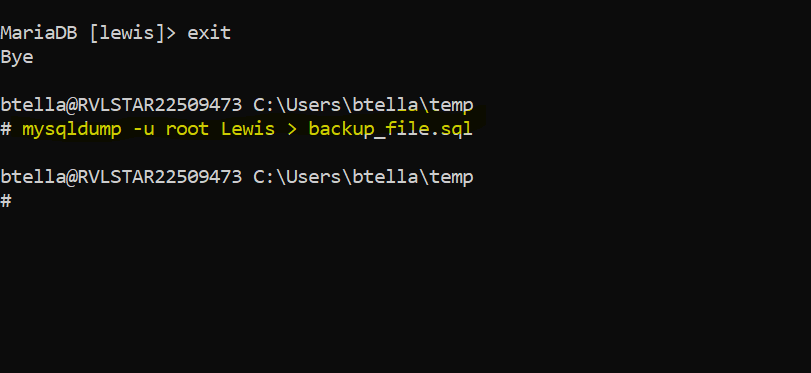


### Automation of backup:

Automation of backup is done using windows task scheduler.

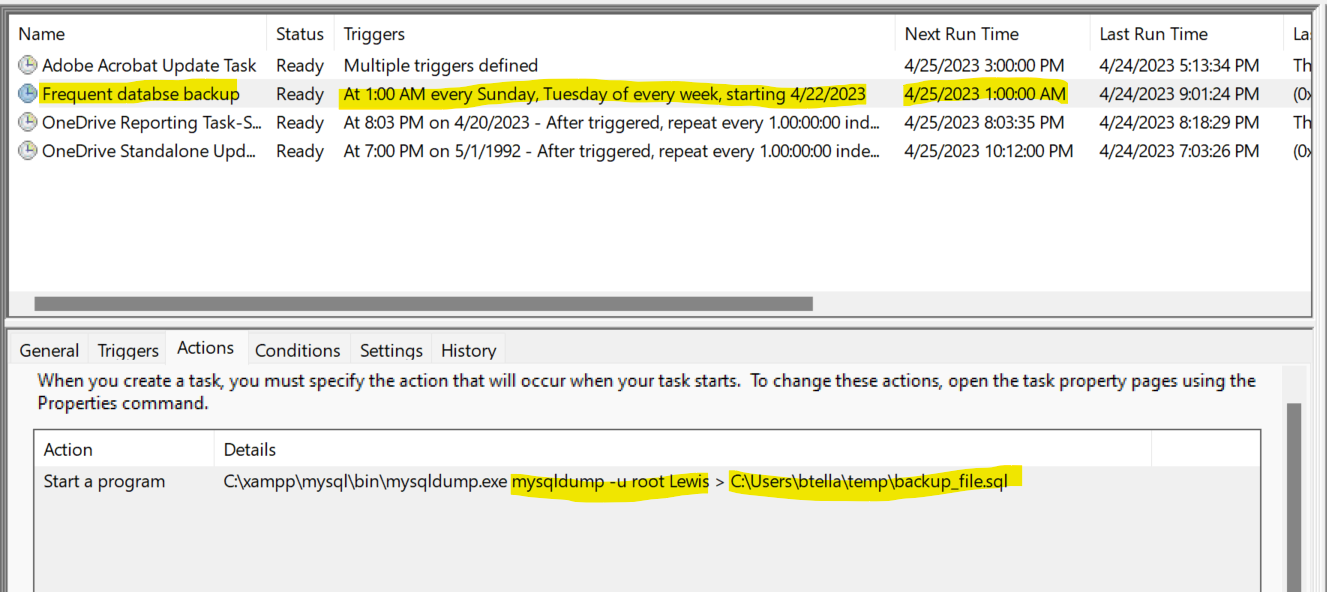
## Commands performed and backup file:

### Backup command performed on console:

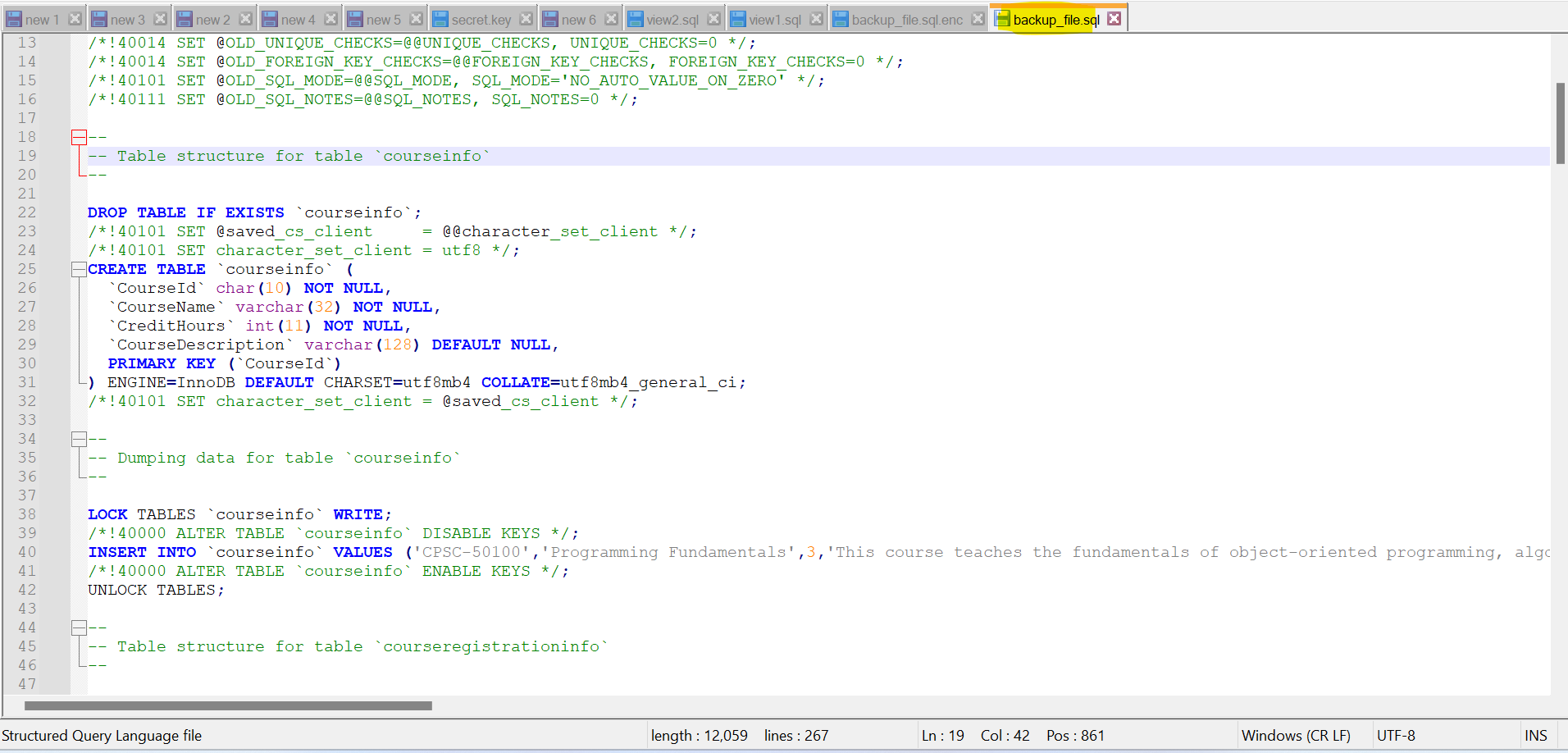


### Automation and frequency of backup:

The frequency of backup is twice a week on Tuesday and Sunday as the database are not done frequently.



### Screenshot of backup file generated:



# P. Programming

## Link for python script:

<https://github.com/bharathtella/Lewis-University-International-Student-Information/blob/main/Sample_Python_script.py>

## Python script:

#importing the padas and sqlalchemy packages

import pandas as pd

from sqlalchemy import create\_engine

from sqlalchemy.exc import SQLAlchemyError

# Creating a connection to the database engine

try:

engine = create\_engine('mysql+pymysql://root:@localhost/Lewis')

print("Connection to MySQL database successful!")

except SQLAlchemyError as e:

print(f"An error occurred while connecting to the database: {e}")

# Running an SQL query to select all rows from the courseinfo table in the database

course\_info\_data=pd.read\_sql\_query('SELECT \* FROM courseinfo',engine)

# Displaying the results of the SQL query

print('')

print('Data in the courseinfo table :')

print('')

print(course\_info\_data)

print('')

# Saving the Data in courseinfo table to an Excel file at the given path, without including the index column

file=r"C:\Users\vjanga\Desktop\course\_info\_data.xlsx"

try:

course\_info\_data.to\_excel(file, index=False)

print("Excel file saved successfully.")

except Exception as e:

print(f"An error occurred while saving the Excel file: {e}")

# Running an SQL query to show all tables in the database

tables=pd.read\_sql('show tables',engine)

print('')

print('tables in the LEWIS database :')

print('')

print(tables)

print('')

# Saving the tables in the database to a txt file at the given path, including the index column

tables\_file=r"C:\Users\vjanga\Desktop\tablesinfo.txt"

try:

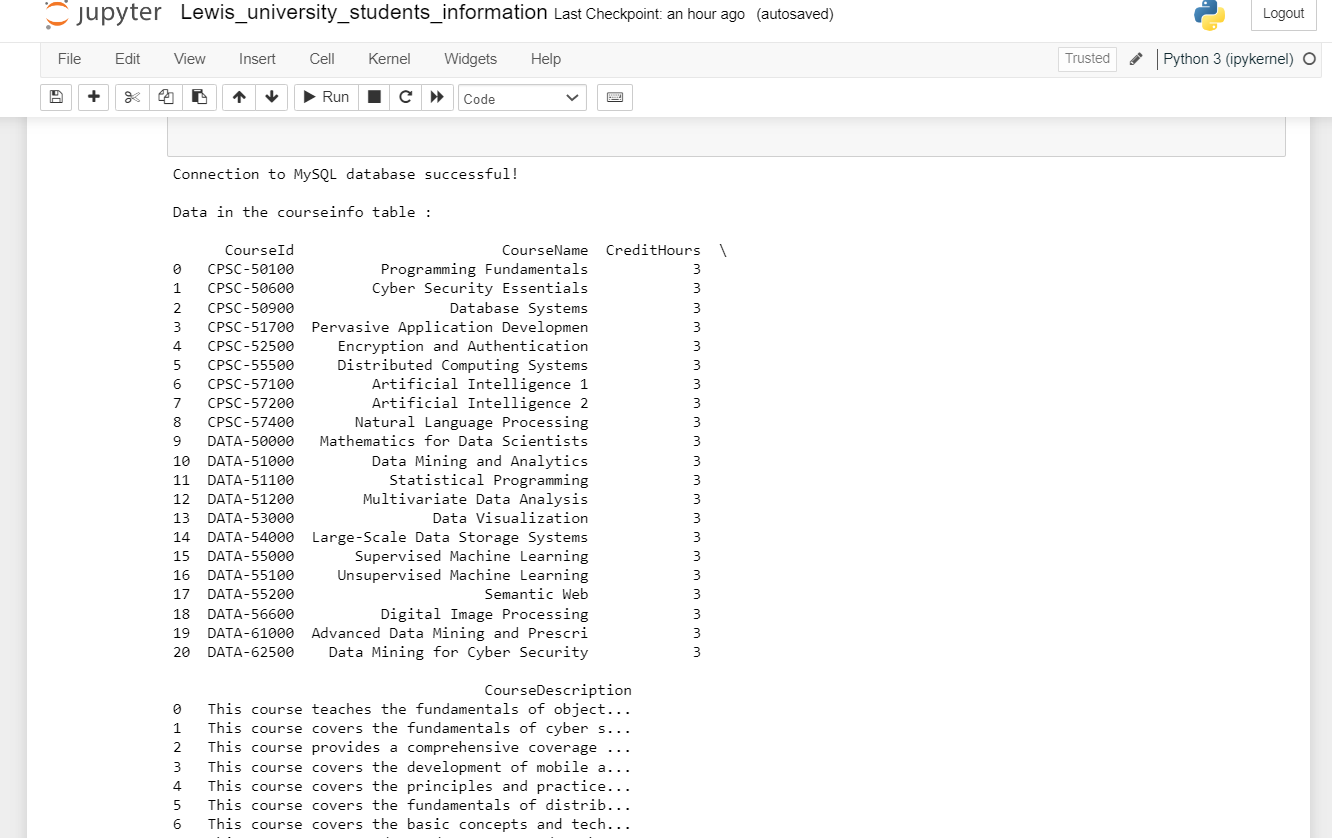
tables.to\_csv(tables\_file)

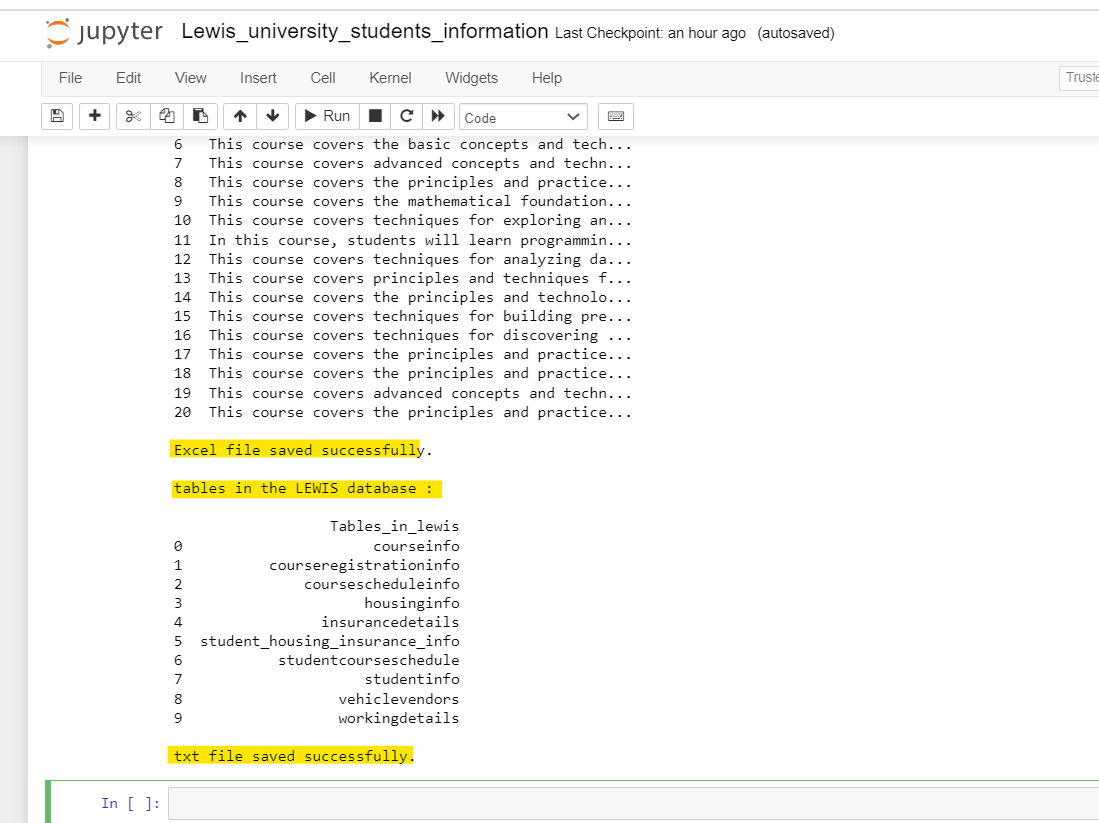
print("txt file saved successfully.")

except Exception as e:

print(f"An error occurred while saving the Excel file: {e}")

## OUTPUT:

****

****

## Git Links for data exported from MySQL database:

### course\_data exported as excel file:

<https://github.com/VidhyaSagarJanga/Lewis-University-International-Student-Information/blob/Sagar/course_data_from_mysql.xlsx>

### **tablesinfo exported as text file :**

<https://github.com/VidhyaSagarJanga/Lewis-University-International-Student-Information/blob/Sagar/tablesinfo_from_mysql.txt>

# R. Suggested Future Work

## Limitations of our database:

* Data type of attribute ‘Day’ used for coursescheduleinfo table uses a varchar (9) instead of more appropriate data type such as an enumeration.
* The provided schema only includes a limited number of tables and does not cover all possible relationships between entities.
* The grading of the student can be added to a new table to avoid data redundancy and update anomalies.
* As the database is created in localhost, accessing it from other system would be difficult.
* The vendor details can contain the address data of the vendors which can help the students access the database.

## Overcoming these shortcomings:

1. **Data type of attribute ‘Day’**: We can update the data type of the 'Day' attribute in the coursescheduleinfo table to an enumeration or a more appropriate data type. This will ensure that only valid values are entered and improve the consistency of the data.
2. **Limited number of tables**: We can identify any missing relationships or entities and add them to the database schema. This will help to ensure that all necessary data is being captured and that the database is more comprehensive.
3. **Student grading**: We can create a new table for storing student grades, rather than including them in the studentinfo table. This will help to avoid data redundancy and update anomalies, as you mentioned.
4. **Accessing the database**: We can consider hosting the database on a server or in the cloud to make it accessible from other systems. Alternatively, you could set up a VPN (Virtual Private Network) to securely access the database from remote locations.
5. **Address data of vendors**: Including the address in the housing info table and adding a column in the vehicle vendors table.

## Migration steps of database to cloud AWS:

1. Choose the appropriate service from AWS that fits the requirement of database
2. Create an AWS environment that can host the database, such as an Amazon EC2 instance or an Amazon RDS database instance.
3. Set up security groups and networking so that the database can communicate with other services and be accessed securely.
4. Use the chosen AWS migration tool to migrate the data from the current database to the AWS environment. This can be done using a one-time migration.
5. Test the migrated database to ensure that it is functioning properly.

## Advantages migrating to cloud:

* **Cost-friendly**: Can reduce hardware, software, and maintenance costs as payment done for only what we use.
* **Easy to manage**: Set up and managing the database can be done easy and faster comparatively as per the business needs.
* **Increased Security**: Compared to physical database systems, security is highly increased as they have robust security measures.
* **High availability and reliability**: Cloud databases are highly available and offer reliable access to data. Most cloud service providers have multiple data centers that ensure data availability in case of any disaster or downtime.
* **Accessibility**: Can be easily accessed from anywhere in the world which makes easier for employees to collaborate and work from different locations.

## Advantages of using document-based *NoSQL format:*

* **Optimized Performance**: Can handle the large number of read and write operations at one instance compared to SQL.
* **Reduced Development Time**: As NoSQL doesn’t require any data normalization and has flexible schema, it can significantly reduce development time.
* **Cost effective**: As it requires less hardware and infrastructure to support, they are cost-friendly.
* **Faster Data Access**: Compared to SQL, Document-based NoSQL databases are optimized for fast data access and retrieval.

## Disadvantages of using document-based *NoSQL format:*

* **Limited Query Capabilities**: While you can perform basic queries, more complex queries may require additional coding and processing.
* **Security Considerations**: Document-based NoSQL databases can be more vulnerable to security threats than SQL databases, particularly if they are not properly configured or secured.
* **Difficult while migration**: When compared to SQL, it is difficult in NoSQL to migrate data to different database.
* **Difficult to Learn**: Document-based NoSQL databases can be more difficult to learn and use than SQL databases, especially for developers who are used to working with SQL databases. This can result in higher training costs and a longer learning curve.