STUDENT MANAGEMENT SYSTEM

**Final Project Report**

Subject: DATABASE SYSTEMS

Course: CPSC5000(CRN 82265)

Professor: Matthew James Clavelli

Student name: Saikumar Gunishetti

Student ID: L30065270

This database system is created to facilitate a student with all the work needed to be done in a semester. This provides digitalizing the traditional paper record to automate with computers. As a student he needs to register for a semester with his details mentioned and then choose the course work he want to do in that semester. Apart from students we have academic advisors, Professors etc. After the start of the semester a student is allotted with an academic advisor, then he can choose the courses of his choice and each course will be thought by a professor. This system keeps track of students, professors, advisors, course work, etc. the data will be coming from different parts of the intranet using FTP severs and stored on a MySQL database.

Relational Database Design:

Here we have a total of seven entities:

1. **Student**: here we store the personal details of the student like name, id, email, address, contact number etc.
2. **Advisor**: here we store the details of the advisor with name, id, phone number, address, email.
3. **Professor**: here we store the details of professor like name, id, email, address, phone number and the course thought etc.
4. **Course\_Catalog**: here we store the subject details like, name of the subject, id, the schedule like on what day it is thought.
5. **Enrolls**: here we store the courses enrolled by the student and which professor is teaching it.
6. **Teach**: here we generate a teach\_id to integrate the student with course, professor, advisor.
7. **Homework**: here we store the details of the work to be done and done by the student and the grades given for his/her work.

The aim of normalization is to reduce instances of double values. By transferring a database to one of the listed normal forms, the target schema benefits from less redundancy than the source schema. Normalization also makes database maintenance easier.

To explain the normalisation, here we have a ENTITY RELATIONSHIP DIAGRAM created using CROW’S FOOT diagram.

Diagram

Description automatically generated

This design is made using VERTABELO and the login details are mentioned below:

<https://my.vertabelo.com/login>

Username: [saikumargunishetti@lewisu.edu](mailto:saikumargunishetti@lewisu.edu)

Password: teSting@8

**Data Definition Language Scripts**

SQL file is generated through above process

**Data sources:**

Data will be coming from different sources, like student details will be coming from admission department, advisor data will be coming from administrative department, professor details will be coming from Program department and so on. Here each department enters the data manually to coma separated value files (**CSV**) and shares their files with the system over intranet.

Contents of the Entities:

1. Student: As explained earlier here we store details of student like name, address, phone, email etc. here in this table “stud\_id” is the primary key and the same will become foreign key in “Enrolls” and “Homework” entities.
2. Enrolls: In this entity, the courses that student enrolled are stored with the course id, student id and enroll id as primary key.
3. Course Catalog: In this entity all the courses that are available are saved with the course name, course day and adv\_id as the foreign key, which saves the advisor id that has added the course.
4. Home Work: The student homework subissions are stored in the entity with the details of the course, home work and student. Here the hw\_id is the primary key, teach\_id is the foregin key, hw\_name has the home work that is submitted, hw\_file stores the home work file, Hw\_grade records the grading for the homework that Is given by the professor.
5. Teach: In this entity, the course offering by the professor that is being added by the advisor details are stored. Prof\_id identifies the professor who is assigned to the course, course\_id identifies which course, adv\_id stores the details of the advisor who added the mapping of course to the professor.
6. Advisor: In this entity, the details of the advisor namely name, email , address and contact number of the advisor are stored.
7. Professor: In this entity, the details of the professor are stored namely, professor name, professor designation, address, contactno and email.

**To Create Table:**

Here the load data infile command takes the comma separated files of data which are in the same order of columns in the SQL table and inserts it into the SQL Table.

The separator is also mentioned by the command terminated by ‘,’. To distinguish each value in the table.

**Commands for each table data insertion:**

**Query 1:** load data infile ‘student.csv’ into table student fields terminated by ‘,’;

This command takes the data from student.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the student Table.

**Query 2:** load data infile ‘professor.csv’ into table professor fields terminated by ‘,’;

This command takes the data from professor.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the professor Table.

**Query 3:** load data infile ‘advisor.csv’ into table advisor fields terminated by ‘,’;

This command takes the data from advisor.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the advisor Table.

**Query 4:** load data infile ‘course\_catalog.csv’ into table course\_catalog fields terminated by ‘,’;

This command takes the data from course\_catalog.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the course\_catalog Table.

**Query 5:** load data infile ‘enroll.csv’ into table enrolls fields terminated by ‘,’;

This command takes the data from enroll.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the enrolls Table.

**Query 6:** load data infile ‘teach.csv’ into table enrolls fields terminated by ‘,’;

This command takes the data from teach.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the teach Table.

**Query 7:** load data infile ‘homework.csv’ into table enrolls fields terminated by ‘,’;

This command takes the data from homework.csv and breaks the data at ‘,’ for each column by using the specified terminated by ‘,’. To insert data into the homework Table.

Screenshot: Text

Description automatically generated

**Data Manipulation Language Scripts**

**Insert, Update, Delete and select:**

**Query 1:** Insert into student (stud\_id, stud\_name, stud\_email, stud\_address, stud\_contactno) values (4, ‘david smith’, ‘davidsm@gmail.com’, ‘120 melissa, texas’, 4545454545);

**Query 2:** Insert into professor(prof\_id, prof\_name, prof\_designation, prof\_contactno, prof\_address, prof\_email) values (6, ‘deogun’, ‘preofessor’, 3434343434, ‘121 new York, new jersey’, ‘deokgun@gmail.com’);

**Query 3:** Update professor set prof\_name=’deokgun park’ where prof\_id=6;

**Query 4:** Update student set stud\_name=’deokgun park’ where prof\_id=4;

**Query 5:** Delete from student where stud\_id = 4;

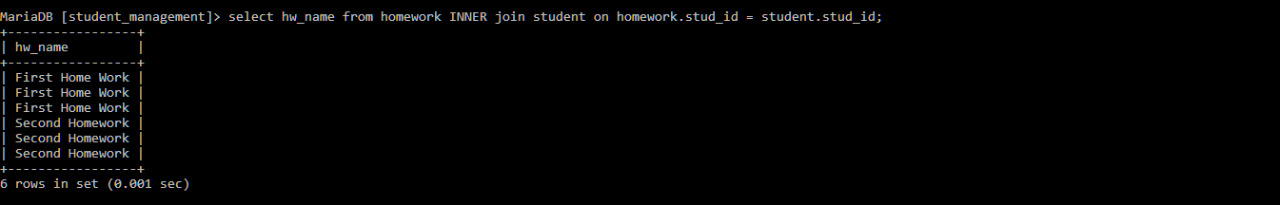
**Query 6:** Select \* from student;

Text

Description automatically generated

**Query 7:** select hw\_name from homework inner join student on homework.stud\_id = student.stud\_id;

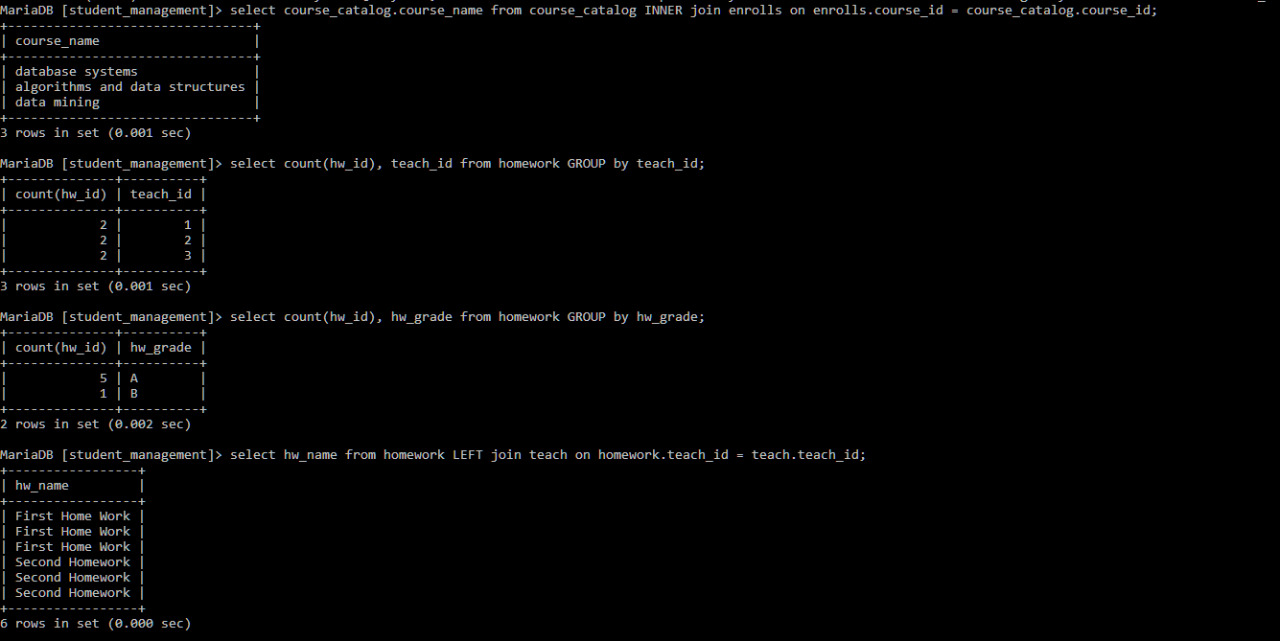
**Query 8:** select course\_catalog.course\_name from course\_catalog inner join enrolls on enrolls.course\_id = course\_catalog.course\_id



**Query 9:** select count(hw\_id), teach\_id from homework group by teach\_id;

**Query 10:** select count(hw\_id), teach\_id from homework group by teach\_id;

**Query 11:** select hw\_name from homework left join teach on homework.teach\_id = teach.teach\_id;

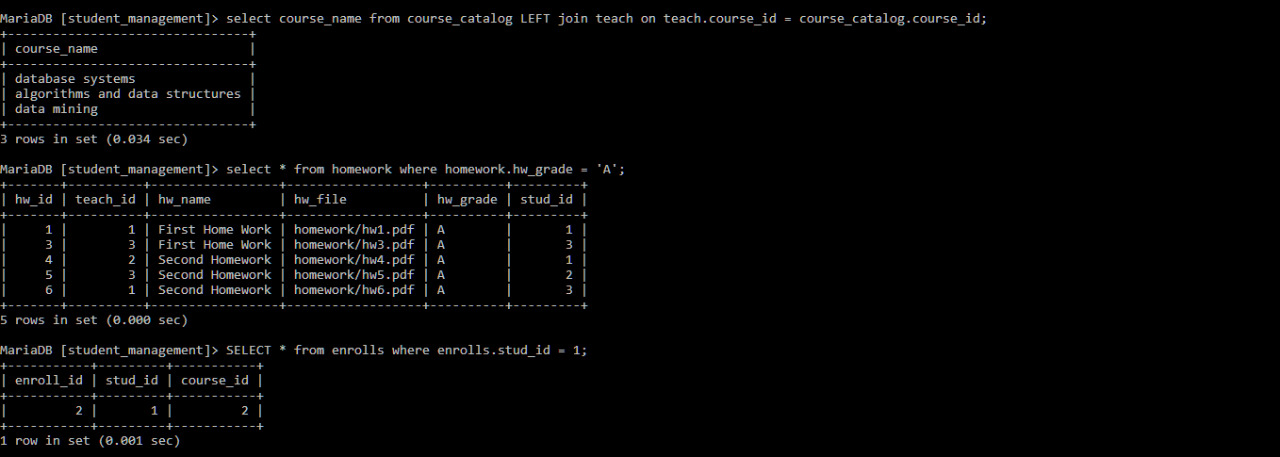


**Query 12:**

Select course\_name from course\_catalog left join teach on teach.course\_id = course\_catalog.course\_id;

**Query 13:** select \* from homework where homework.hw\_grade = ‘A’;

**Query 14:** select \* from enrolls where enrolls.stud\_id = 1;



**Indexes:**

**Student Name Index:** The index of student name helps in making the search easier to find the queries especially for string matching queries. As every name in the table must be compared and there can be many names. In case of millions of records in the table this index is helpful to reduce the query runtime



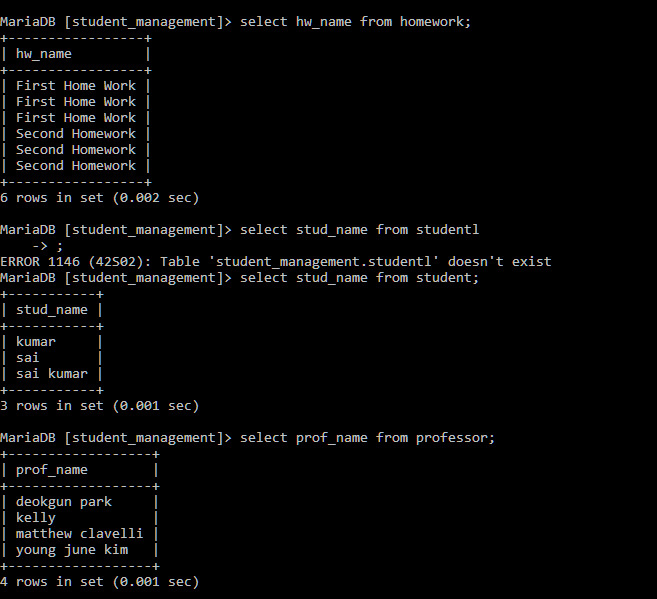
**Professor Name Index:** The index on professor name is helpful when there are a lot of records in the table. As mentioned above the queries that are related to the string matching will be more effective in reducing the query runtime.



**Homework name Index:** As mentioned above for the same reason the home work name index is helpful in reducing the query runtime in case of large number of records in the homework table.



**Note:** As there are only few records in the database the query runtime remains the same.

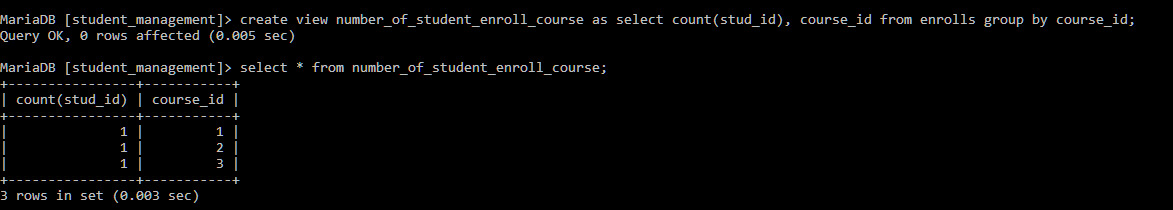


**Views:**

**Query 1:** create view number\_of\_student\_enroll\_course as select count(stud\_id), course\_id from enrolls group by course\_id;

This view is useful to get the count statistic of number of students enrolled in particular course.

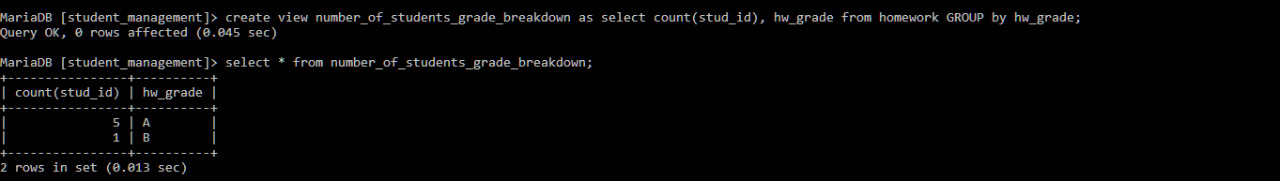
This view helps the advisor to know the number of students for each course and for example decide to increase or decrease allocation of students for each course.



**Query 2:** create view number\_of\_students\_grade\_breakdown as select count(stud\_id), hw\_grade from homework group by hw\_grade;

This view is useful in getting the number of students count for the homework based on grade secured by the student.

This view is helpful for the advisor and professor to know the count of students who got a particular grade.



**Triggers:**

**Trigger 1:**

CREATE TRIGGER `student\_bckp` AFTER INSERT ON `student` FOR EACH ROW insert into student\_bckp(stud\_address, stud\_contactno, stud\_email, stud\_id, stud\_name) VALUES(NEW.stud\_address, NEW.stud\_contactno, NEW.stud\_email, NEW.stud\_id, NEW.stud\_name);

This trigger helps in backing up the data to a different table called student\_bckp, After every insertion into the student table this trigger is triggered and the same record will the stored in the student\_bckp table.

**Trigger 2:**

CREATE TRIGGER `professor\_bckp` AFTER INSERT ON `professor` FOR EACH ROW insert into professor\_bckp(prof\_address, prof\_contactno, prof\_designation, prof\_email, prof\_id, prof\_name) values(NEW.prof\_address, NEW.prof\_contactno, NEW.prof\_designation, NEW.prof\_email, NEW.prof\_id, NEW.prof\_name);

This Trigger helps in backing up the data in the professor table to the professor\_bckp table after every insertion into the professor table. This trigger is triggered after every insertion of record into the professor table.

**Trigger 3:**

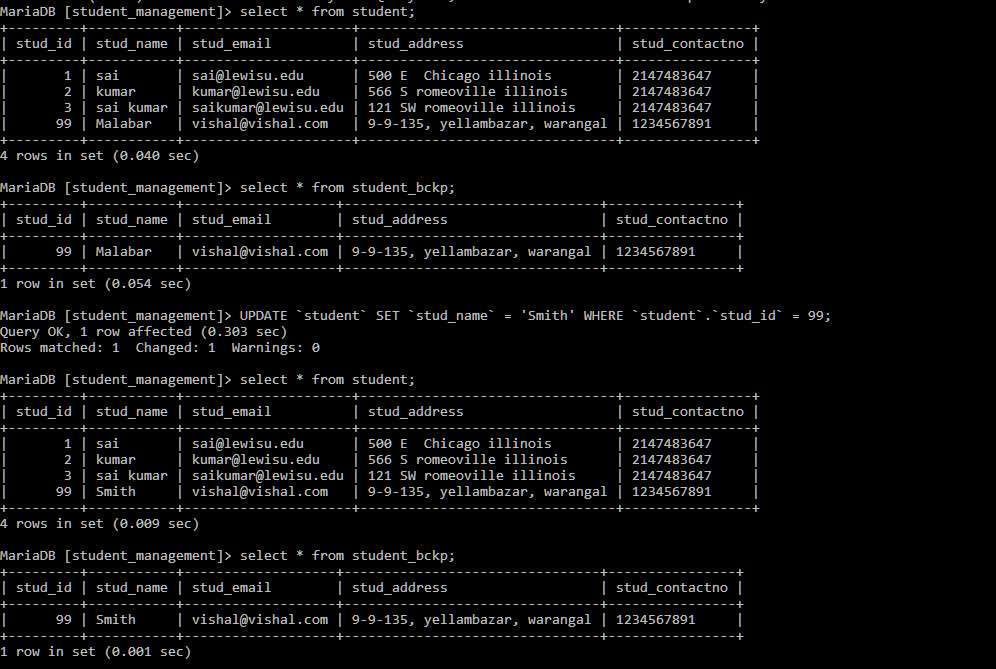
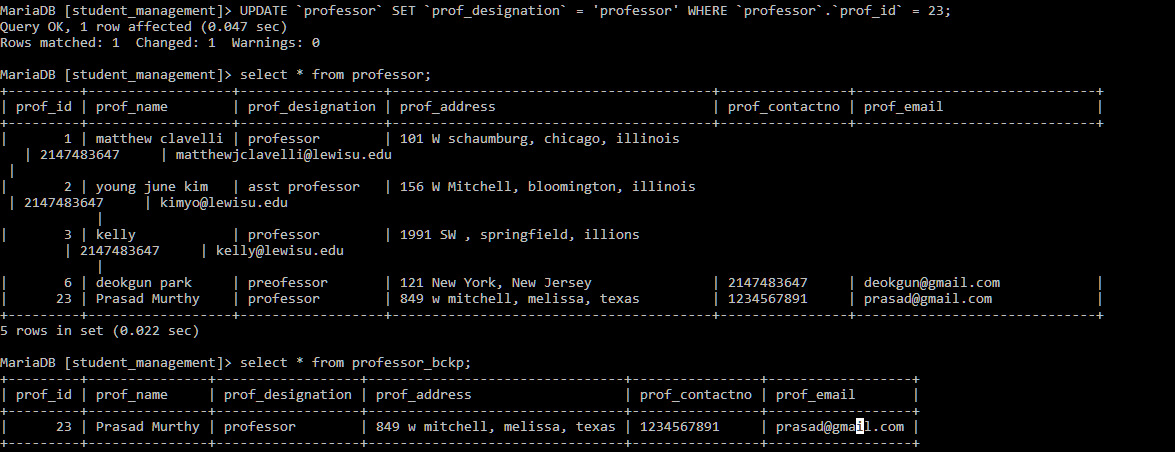
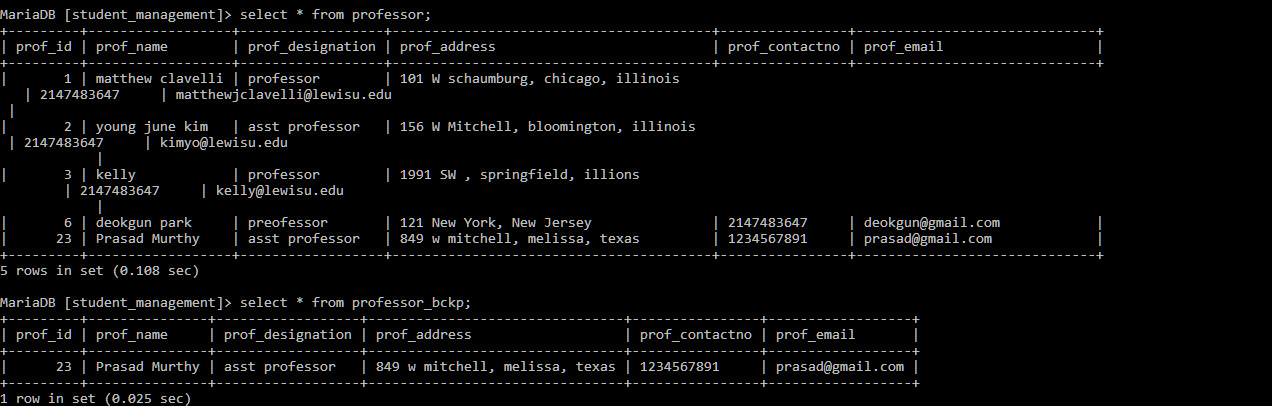
CREATE TRIGGER `student\_name\_update` AFTER UPDATE ON `student`

FOR EACH ROW update student\_bckp set stud\_name = NEW.stud\_name where stud\_id = NEW.stud\_id;

This Trigger is useful in updating the student\_bckp table for the values changed in the records in the student table that are present in the student\_bckp table. This trigger is triggered after updating the student name for a particular record in the student table for updating the same In the student\_bckp table.

**Trigger 4:**

CREATE TRIGGER `professor\_desig\_update` AFTER UPDATE ON `professor` FOR EACH ROW update professor\_bckp set prof\_designation = NEW.prof\_designation where professor\_bckp.prof\_id = NEW.prof\_id;

This trigger is used to update the professor\_bckp table for the designation change in the professor table. After updating the designation in the professor table this trigger is triggered to update the same record in the professor\_bckp table.

**Transactions:**

Transaction processing is a type of computing that supports interactive applications and is often conducted by big server computers. Work is separated into individual, indivisible activities called transactions.

It coordinates the processing of multiple transactions at the same time. Enables the sharing of data and manages the prioritization.

* **ACID properties of transactions**  
  In the context of transaction processing, the acronym *ACID* refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.
* **Commit and rollback**  
  To assure the ACID properties of a transaction, any changes made to data in the course of a transaction must be *committed* or *rolled back*.
* **Recovery and restart**  
  CICS continually records information about the state of the region and about the state of each unit of work in the region. This information is preserved and used when a region is restarted, thus enabling CICS to restart with no loss of data integrity.

**Atomicity**

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.

For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

**Consistency**

Data is in a consistent state when a transaction starts and when it ends.

For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

**Isolation**

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized.

For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

**Durability**

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

**Text

Description automatically generated**

**Security:**

In this system, we have three users-professor, student and advisor.

Student - Here the student is given the permission to only upload a homework and nothing else.

Advisor – Here the Advisor is given the permission to add, delete, update and select the professor and course mapping in the teach table and nothing else.

Professor – Here the professor is given the permission to add, select, delete and update the homework table.

**User** **creations**:

Advisor: create user ‘advisor’@’localhost’ identified by ‘password’

Professor: create user ‘professor’@’localhost’ identified by ‘password’

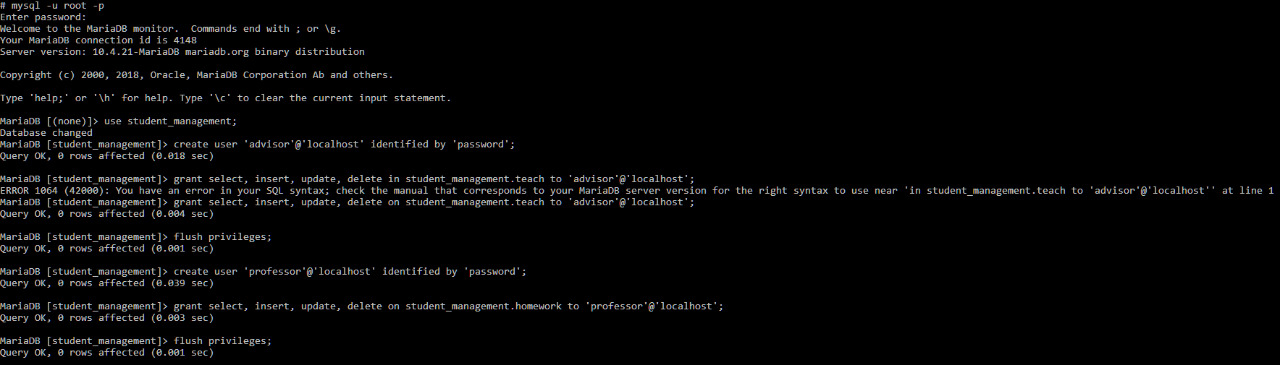
Student: create user ‘student’@’localhost’ identified by ‘password’

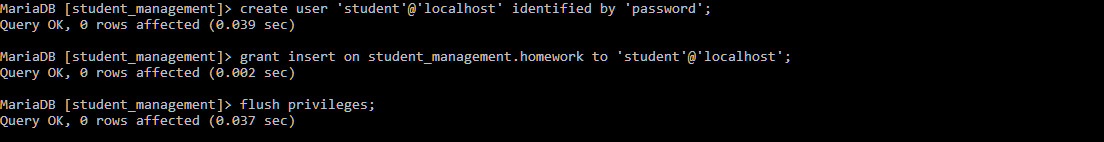
Grant Permissions:

Advisor: grant select, insert, update, delete on student\_management.teach to ‘advisor’@’localhost’;

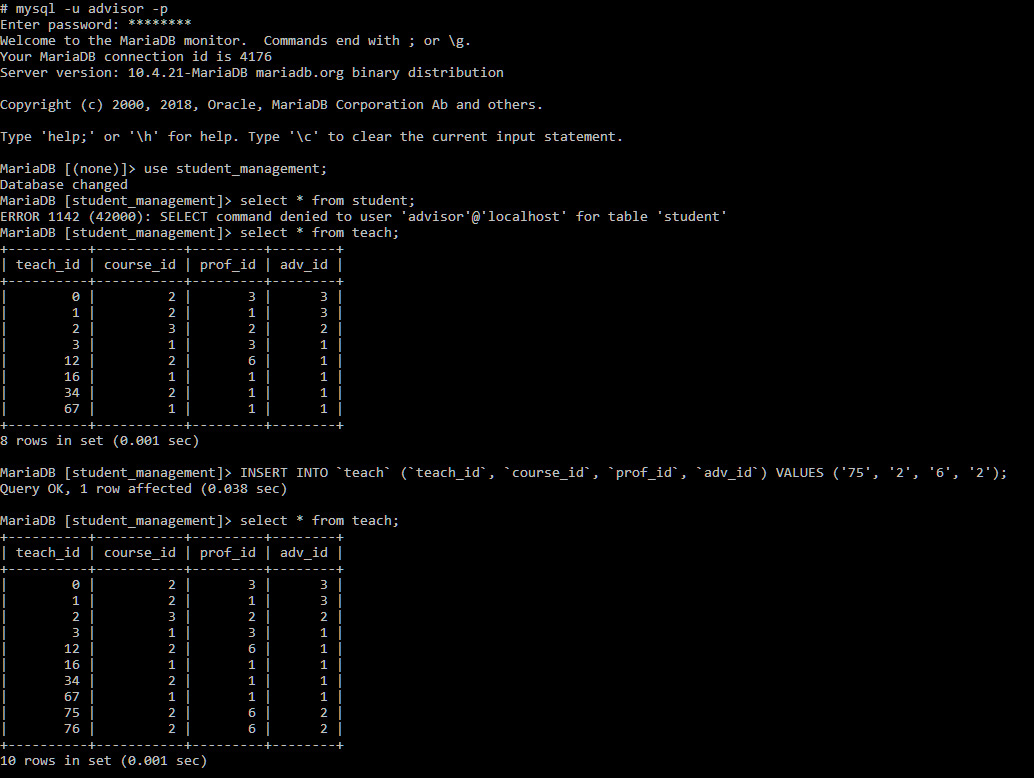
Professor: grant select, insert, update, delete on student\_management.teach to ‘professor’@’localhost’;

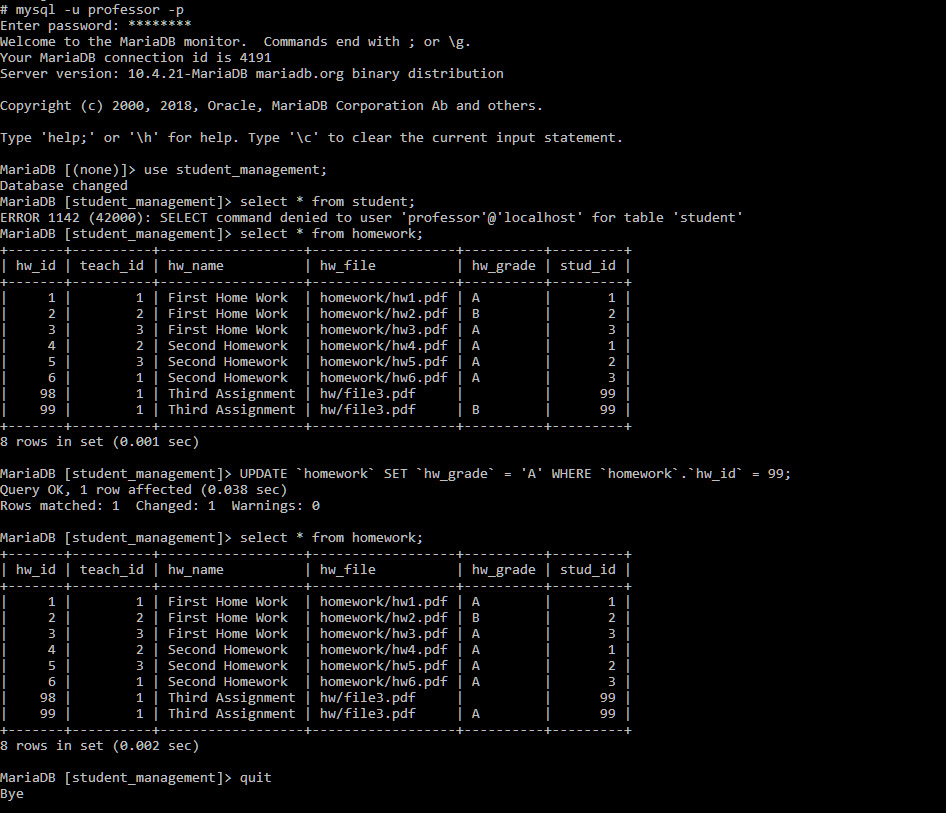
Student: grant insert on student\_management.homework to ‘student’@’localhost’;





Demonstration of grant permissions

: 

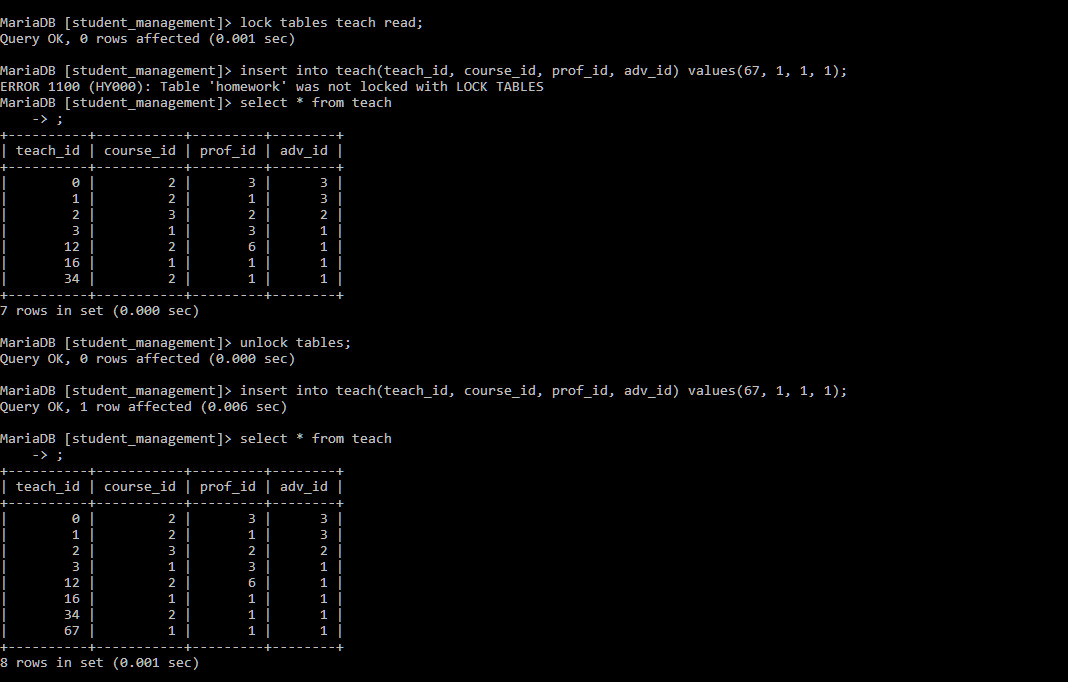


Text

Description automatically generated

**Locking:**

The teach table has the course to professor mapping which can only be managed by advisor. As the mapping is crucial for professor to teach the course. To avoid any updating or deleting of these mapping it is good practice to lock the tables.



**Backup:**

In my scenario, I have manually taken backup of the system by giving the command below. The location of the backup was in D drive of my machine, with the file name “ stud\_mgmt.sql ”

**Frequency:** For the backup frequency, scheduling a backup every day at 00:00 HRS helps in maintaining the large amount of backup data.

**Location:** The backup file (.sql) file can be stored in a folder on my machine.

**Automation:** For automating the database backup operation, we can use the task scheduler in Windows 10, for automating the Microsoft Windows Batch File. That has the backup command that is to be issued to get the backup.

**Security:** The backed up .sql files that are in the folder is zipped with password protection.

**Database Backup command:**

mysqldump -u root -p student\_management > “D:\stud\_mgmt.sql”



**Python programming:**

Code:

#!/usr/bin/env python

# coding: utf-8

# In[22]:

import pymysql

db = pymysql.connect(*host*='localhost',*user*='root',*password*='', *database*= "student\_management") # creating the connection

cursor = db.cursor() # creating the cursor

query\_stud = "select \* from student"# select query

result = cursor.execute(query\_stud)# course execute runs the query on database

tuples = cursor.fetchall()# gets all the data

print(tuples)# prints the tuples

resultt = [str(i) + "\n" for i in tuples]# making the lines for the data to be stored in the file

fo = open('student.txt', "w")# opening the file

fo.writelines(resultt)# writing the file with the tuples

fo.close()# closing the opened file

print('-'\*100)

# In[23]:

query\_stud = "select \* from professor"# select query

result = cursor.execute(query\_stud)# course execute runs the query on database

tuples = cursor.fetchall()# gets all the data

print(tuples)# prints the tuples

resultt = [str(i) + "\n" for i in tuples]# making the lines for the data to be stored in the file

fo = open('professor.txt', "w")# opening the file

fo.writelines(resultt)# writing the file with the tuples

fo.close()# closing the opened file

print('-'\*100)

# In[25]:

query\_stud = "select \* from enrolls"# select query

result = cursor.execute(query\_stud)# course execute runs the query on database

tuples = cursor.fetchall()# gets all the data

print(tuples)# prints the tuples

resultt = [str(i) + "\n" for i in tuples]# making the lines for the data to be stored in the file

fo = open('enroll.txt', "w")# opening the file

fo.writelines(resultt)# writing the file with the tuples

fo.close()# closing the opened file

print('-'\*100)

# In[26]:

query\_stud = "select \* from homework"# select query

result = cursor.execute(query\_stud)# course execute runs the query on database

tuples = cursor.fetchall()# gets all the data

print(tuples)# prints the tuples

resultt = [str(i) + "\n" for i in tuples]# making the lines for the data to be stored in the file

fo = open('homework.txt', "w")# opening the file

fo.writelines(resultt)# writing the file with the tuples

fo.close()# closing the opened file

print('-'\*100)

# In[ ]:

query\_stud = "select \* from course\_catalog"# select query

result = cursor.execute(query\_stud)# course execute runs the query on database

tuples = cursor.fetchall()# gets all the data

print(tuples)# prints the tuples

resultt = [str(i) + "\n" for i in tuples]# making the lines for the data to be stored in the file

fo = open('course\_catalog.txt', "w")# opening the file

fo.writelines(resultt)# writing the file with the tuples

fo.close()# closing the opened file

print('-'\*100)

# In[27]:

query\_stud = "select \* from advisor"# select query

result = cursor.execute(query\_stud)# course execute runs the query on database

tuples = cursor.fetchall()# gets all the data

print(tuples)# prints the tuples

resultt = [str(i) + "\n" for i in tuples]# making the lines for the data to be stored in the file

fo = open('advisor.txt', "w")# opening the file

fo.writelines(resultt)# writing the file with the tuples

fo.close()# closing the opened file

print('-'\*100)

# In[28]:

query\_stud = "select \* from teach" # select query

result = cursor.execute(query\_stud) # course execute runs the query on database

tuples = cursor.fetchall() # gets all the data

print(tuples) # prints the tuples

resultt = [str(i) + "\n" for i in tuples] # making the lines for the data to be stored in the file

fo = open('teach.txt', "w") # opening the file

fo.writelines(resultt) # writing the file with the tuples

fo.close() # closing the opened file

print('-'\*100)

# In[ ]:

Output Of the database: A picture containing text, monitor, screenshot, black

Description automatically generated

**All the documents, file sets, design png, commands used, backup file, sql script, python script and necessary files are placed in git repository and the link for the repo is** [**https://github.com/saiguni/student\_management**](https://github.com/saiguni/student_management)

**<<<<<<<<<<<<<<<<<<<--------------------------------------Thank you---------------------------------->>>>>>>>>>>>>>>>>>>>>>>>>>>>**