**ASSIGNMENT**

**STUDENT INFORMATION SYSTEM**

AGENDA:

* Define Classes
* Implement Constructors
* Implement Methods
* Exceptions handling and Custom Exceptions
* Collections
* Create Methods for Managing Relationships
* Database Connectivity
* Student Enrolments
* Teacher Assignment
* Payment Record
* Enrolments Report Generation

**BY**

**JASWANTH KUMAR S**

**BATCH 4**

**TASK1 : Define Classes**

class Student:

def \_\_init\_\_(self, student\_id, first\_name, last\_name, dob, email, phone):

self.\_\_student\_id = student\_id

self.\_\_first\_name = first\_name

self.\_\_last\_name = last\_name

self.\_\_dob = dob

self.\_\_email = email

self.\_\_phone = phone

self.\_\_payments = []

class Course:

def \_\_init\_\_(self, course\_id, course\_name, course\_code, instructor\_name):

self.\_\_course\_id = course\_id

self.\_\_course\_name = course\_name

self.\_\_course\_code = course\_code

self.\_\_instructor\_name = instructor\_name

self.\_\_students\_enrolled = []

self.\_\_teacher = None

class Teacher:

def \_\_init\_\_(self, teacher\_id, first\_name, last\_name, email):

self.\_\_teacher\_id = teacher\_id

self.\_\_first\_name = first\_name

self.\_\_last\_name = last\_name

self.\_\_email = email

self.\_\_assigned\_courses = []

class Enrollment:

def \_\_init\_\_(self, enrollment\_id, student, course, enrollment\_date):

self.\_\_enrollment\_id = enrollment\_id

self.\_\_student = student

self.\_\_course = course

self.\_\_enrollment\_date = enrollment\_date

class Payment:

def \_\_init\_\_(self, payment\_id, student, amount, payment\_date):

self.\_\_payment\_id = payment\_id

self.\_\_student = student

self.\_\_amount = amount

self.\_\_payment\_date = payment\_date

**TASK 2: Implement Constructors**

**STUDENT :**

def \_\_init\_\_(self, student\_id, first\_name, last\_name, dob, email, phone):

self.\_\_student\_id = student\_id

self.\_\_first\_name = first\_name

self.\_\_last\_name = last\_name

self.\_\_dob = dob

self.\_\_email = email

self.\_\_phone = phone

self.\_\_payments = []

**COURSES:**

def \_\_init\_\_(self, course\_id, course\_name, course\_code, instructor\_name):

self.\_\_course\_id = course\_id

self.\_\_course\_name = course\_name

self.\_\_course\_code = course\_code

self.\_\_instructor\_name = instructor\_name

self.\_\_students\_enrolled = [] # Collection to store enrolled students

self.\_\_teacher = None # Reference to assigned teacher

**PAYMENT:**

def \_\_init\_\_(self, payment\_id, student, amount, payment\_date):

self.\_\_payment\_id = payment\_id

self.\_\_student = student

self.\_\_amount = amount

self.\_\_payment\_date = payment\_date

# Add payment to student's payment history using the getter

student.get\_payments().append(self) # Relationship management

**TASK 3: IMPLEMENT METHODS**

**Student Methods**

# Getters

def get\_student\_id(self):

return self.\_\_student\_id

def get\_first\_name(self):

return self.\_\_first\_name

def get\_last\_name(self):

return self.\_\_last\_name

def get\_date\_of\_birth(self):

return self.\_\_dob

def get\_email(self):

return self.\_\_email

def get\_phone\_no(self):

return self.\_\_phone

def get\_payments(self):

return self.\_\_payments

# Setters

def set\_first\_name(self, first\_name):

self.\_\_first\_name = first\_name

def set\_last\_name(self, last\_name):

self.\_\_last\_name = last\_name

def set\_date\_of\_birth(self, dob):

self.\_\_dob = dob

def set\_email(self, email):

self.\_\_email = email

def set\_phone\_no(self, phone):

self.\_\_phone = phone

Course Methods

def assign\_teacher(self, teacher):

self.\_\_teacher = teacher

def update\_course\_info(self, course\_code, course\_name, instructor\_name):

self.\_\_course\_code = course\_code

self.\_\_course\_name = course\_name

self.\_\_instructor\_name = instructor\_name

def display\_course\_info(self):

print(f"Course: {self.\_\_course\_name} ({self.\_\_course\_code}), Instructor: {self.\_\_instructor\_name}")

def get\_enrollments(self):

return self.\_\_students\_enrolled

def get\_teacher(self):

return self.\_\_teacher

def get\_course\_id(self):

return self.\_\_course\_id

def get\_course\_name(self):

return self.\_\_course\_name

def get\_course\_code(self):

return self.\_\_course\_code

def get\_instructor\_name(self):

return self.\_\_instructor\_name

Teacher Methods

def update\_teacher\_info(self, first\_name, last\_name, email):

self.\_\_first\_name = first\_name

self.\_\_last\_name = last\_name

self.\_\_email = email

def display\_teacher\_info(self):

print(f"Teacher: {self.\_\_first\_name} {self.\_\_last\_name}, Email: {self.\_\_email}")

def get\_assigned\_courses(self):

return self.\_\_assigned\_courses

def get\_teacher\_id(self):

return self.\_\_teacher\_id

def get\_first\_name(self):

return self.\_\_first\_name

def get\_last\_name(self):

return self.\_\_last\_name

def get\_email(self):

return self.\_\_email

**TASK 4: Exceptions Handling and Custom Exceptions**

class StudentNotFoundException(Exception):

"""Raised when a student is not found in the system."""

pass

class CourseNotFoundException(Exception):

"""Raised when a course is not found in the system."""

pass

class TeacherNotFoundException(Exception):

"""Raised when a teacher is not found in the system."""

pass

class DuplicateEnrollmentException(Exception):

"""Raised when a student is already enrolled in a course."""

pass

class PaymentValidationException(Exception):

"""Raised when an invalid payment amount is provided."""

pass

class InvalidStudentDataException(Exception):

"""Raised when student data is invalid or missing."""

pass

class InvalidCourseDataException(Exception):

"""Raised when course data is invalid or missing."""

pass

class InvalidEnrollmentDataException(Exception):

"""Raised when enrollment data is invalid."""

pass

class InvalidTeacherDataException(Exception):

"""Raised when teacher data is invalid."""

pass

class InsufficientFundsException(Exception):

"""Raised when a student does not have sufficient funds for a payment."""

pass

**Collections**

self.\_\_payments = [] # In Student constructor

self.\_\_students\_enrolled = [] # In Course constructor

self.\_\_assigned\_courses = [] # In Teacher constructor def

get\_payments\_for\_student(self, student\_id):

query = "SELECT payment\_id, student\_id, amount, payment\_date FROM payments WHERE student\_id = %s"

with self.db.get\_connection() as connection:

with connection.cursor() as cursor:

cursor.execute(query, (student\_id,))

rows = cursor.fetchall()

payments = [] # Collection to store payment objects

student = self.student\_dao.get\_student\_by\_id(student\_id)

for row in rows:

payments.append(Payment(row[0], student, row[2], row[3]))

return payments

**TASK 5: Create Methods for Managing Relationships**

1.Enrollment and Student/Course Relationship

def \_\_init\_\_(self, enrollment\_id, student, course, enrollment\_date):

self.\_\_enrollment\_id = enrollment\_id

self.\_\_student = student

self.\_\_course = course

self.\_\_enrollment\_date = enrollment\_date

# Add enrollment to student and course

student.enrollments.append(self)

course.enrollments.append(self)

2.Payment and Student Relationship

def \_\_init\_\_(self, payment\_id, student, amount, payment\_date):

self.\_\_payment\_id = payment\_id

self.\_\_student = student

self.\_\_amount = amount

self.\_\_payment\_date = payment\_date

# Add payment to student's payment history

student.get\_payments().append(self)

3. Teacher and Course Relationship

def assign\_teacher(self, course\_id, teacher\_id):

query = "UPDATE courses SET teacher\_id = %s WHERE course\_id = %s"

values = (teacher\_id, course\_id)

with self.db.get\_connection() as connection:

with connection.cursor() as cursor:

cursor.execute(query, values)

connection.commit()

print(f"Teacher ID {teacher\_id} assigned to Course ID {course\_id}.")

**TASK 6: Database Connectivity**

import mysql.connector

class DBConnection:

def \_\_init\_\_(self):

self.host = "localhost"

self.user = "root"

self.password = "Jash@2512"

self.database = "sisdb"

self.conn = None

self.cursor = None

def get\_connection(self):

return mysql.connector.connect(

host=self.host,

user=self.user,

password=self.password,

database=self.database

)

def disconnect(self):

if self.cursor:

self.cursor.close()

if self.conn:

self.conn.close()

def execute\_query(self, query, params=None):

self.get\_connection()

self.cursor.execute(query, params or ())

self.conn.commit()

self.disconnect()

def fetch\_results(self, query, params=None):

self.get\_connection()

self.cursor.execute(query, params or ())

results = self.cursor.fetchall()

self.disconnect()

return results

**Task 8: Student enrolment**

**John Doe's details:**

• First Name: John

• Last Name: Doe

• Date of Birth: 1995-08-15

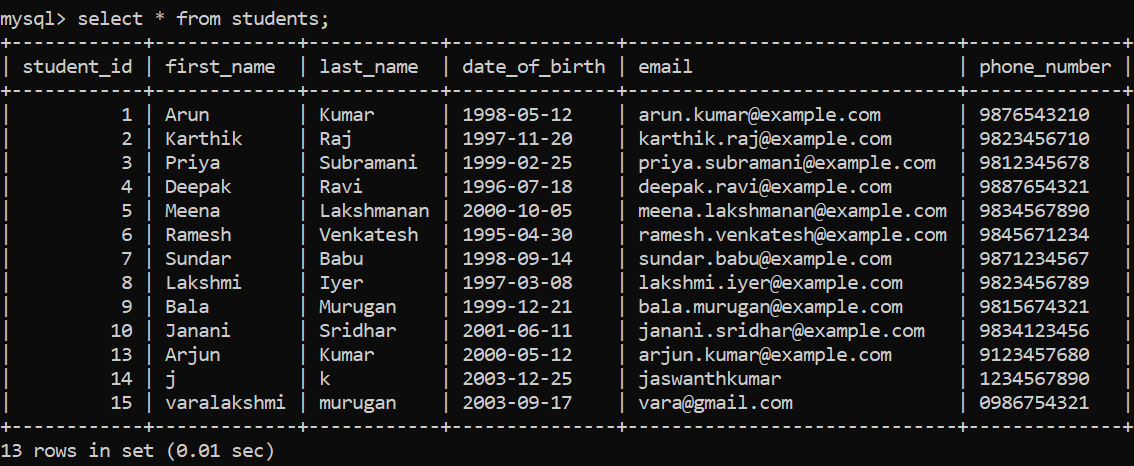
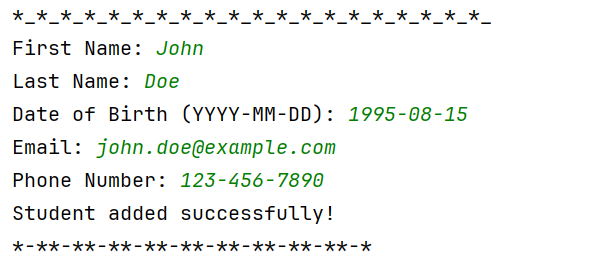
• Email: [john.doe@example.com](mailto:john.doe@example.com)

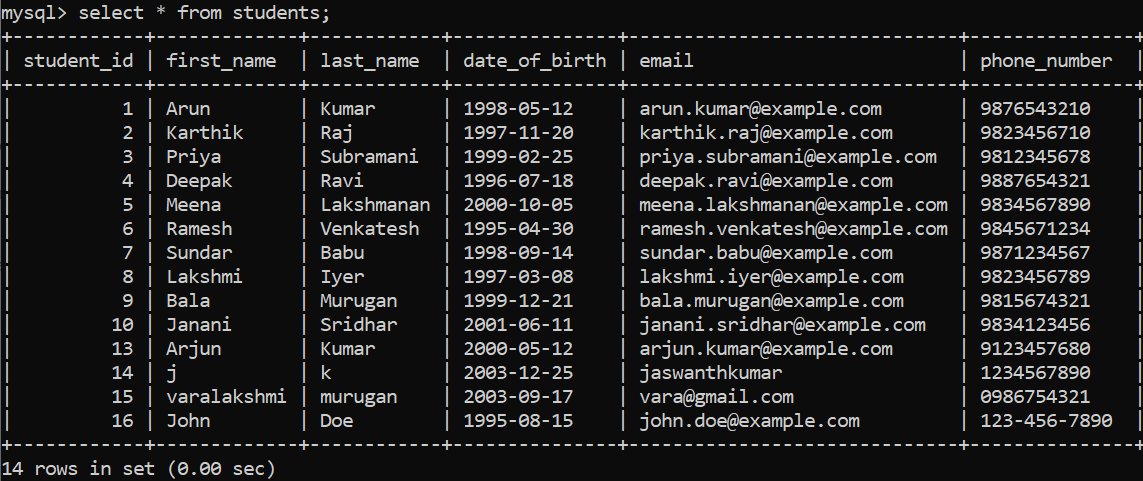
• Phone Number: 123-456-7890

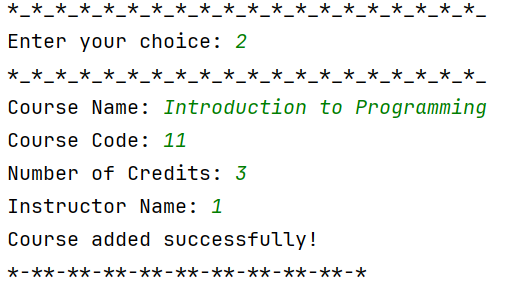
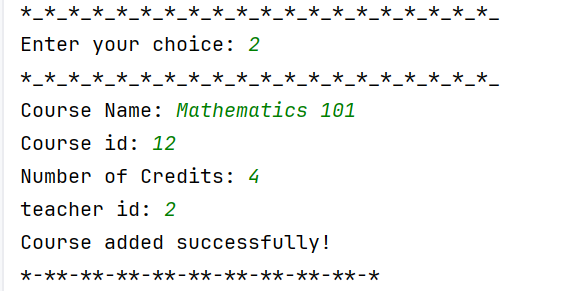
**John is enrolling in the following courses:**

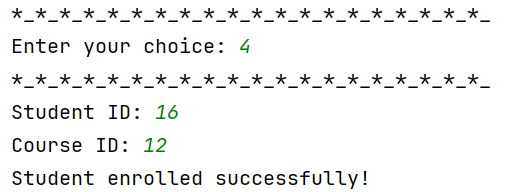
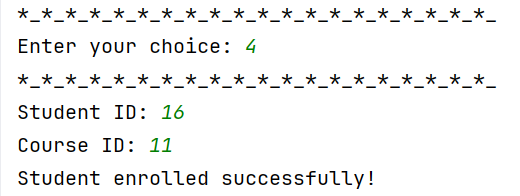
• Course 1: Introduction to Programming

• Course 2: Mathematics 101





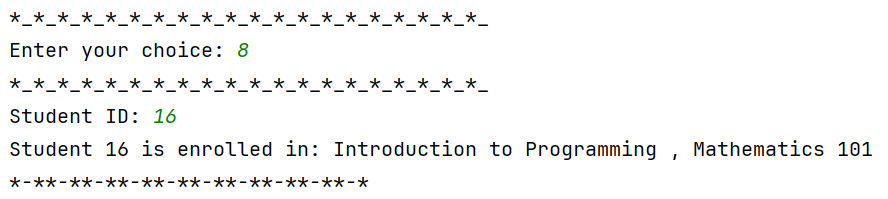


**Introduction to Programming -john**

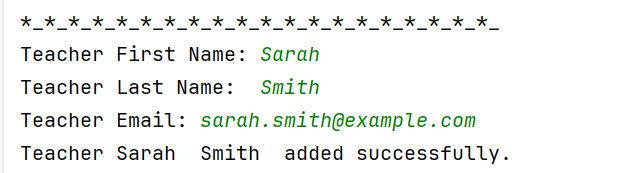
**Mathematics 101 -john**

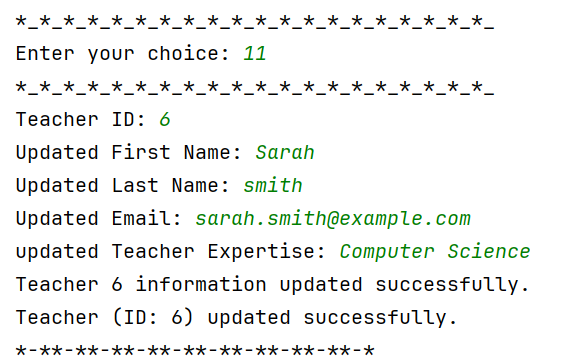
**John is enrolling in the following courses:**

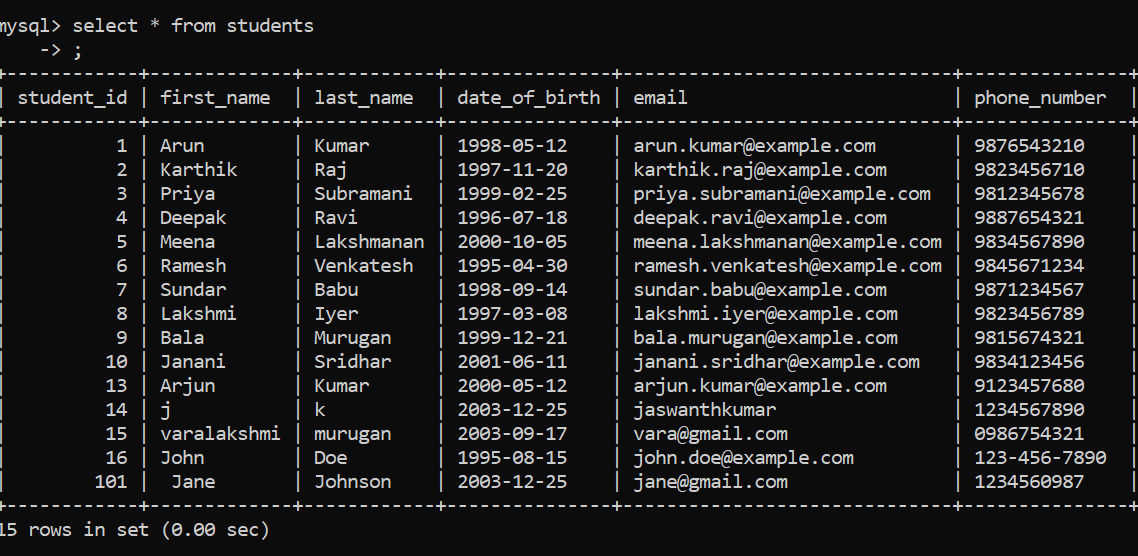
• Course 1: Introduction to Programming

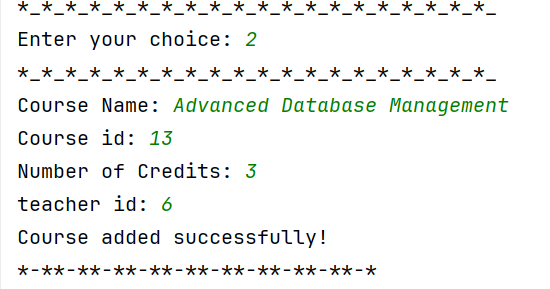
**** • Course 2: Mathematics 101

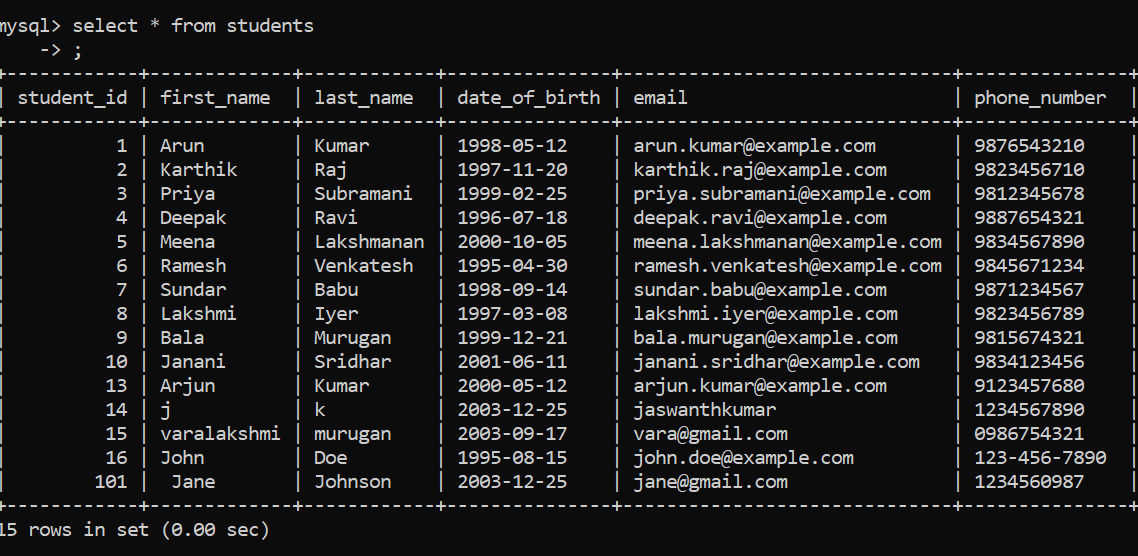
**Task 9: Teacher Assignment**

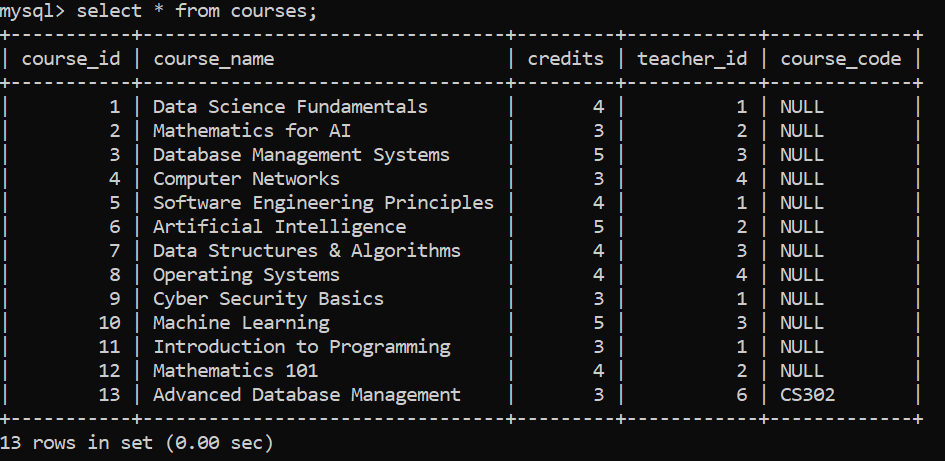


\*\*\* I don’t have a column called expertise before in my existing database (not mentioned before in schema creation). I’ve altered the database and added expertise here\*\*\*

**output in database:**

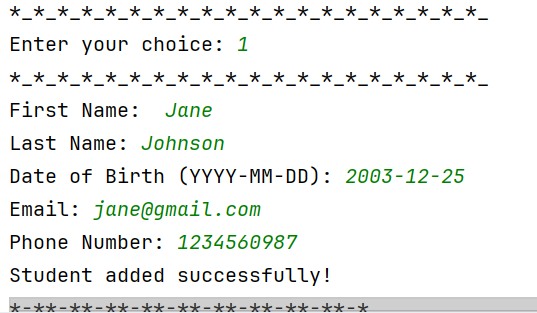
**output in database:**

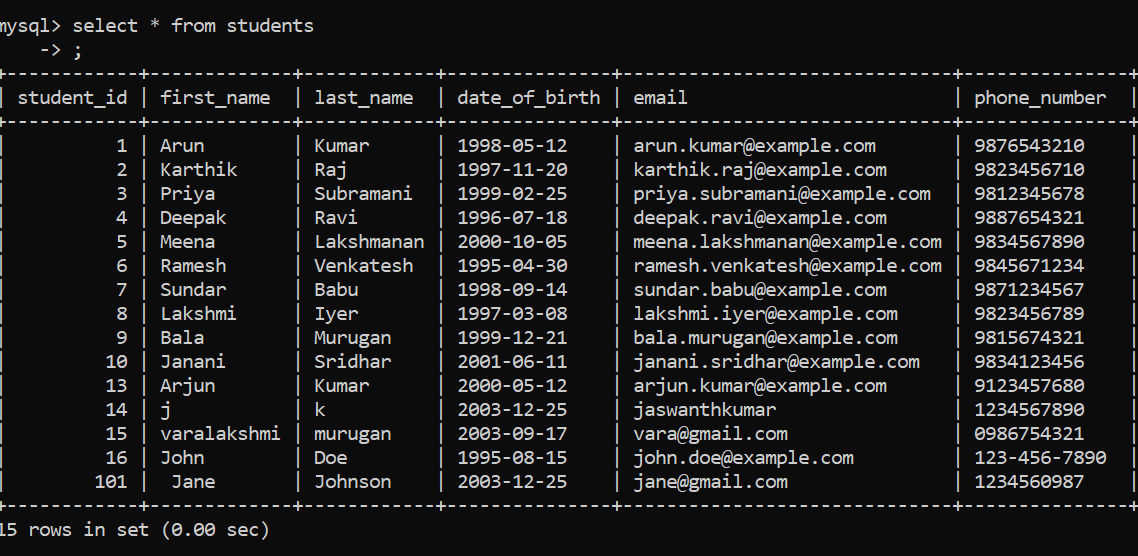
****



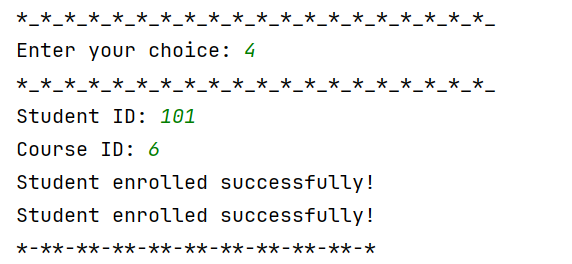
**Task 10: Payment Record**

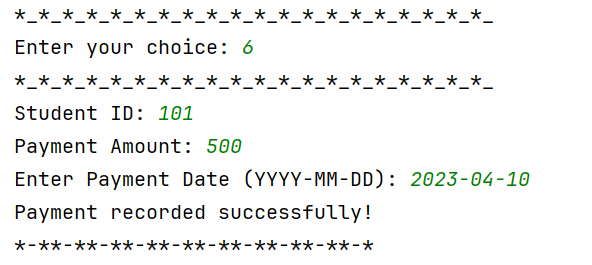
**New student registration**

****

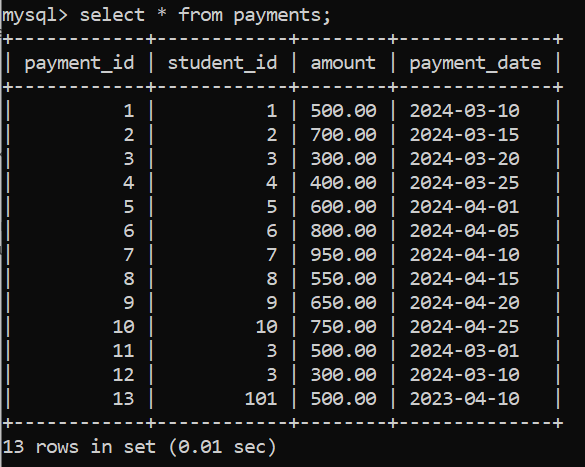
**output in database:**

**course enrolment**



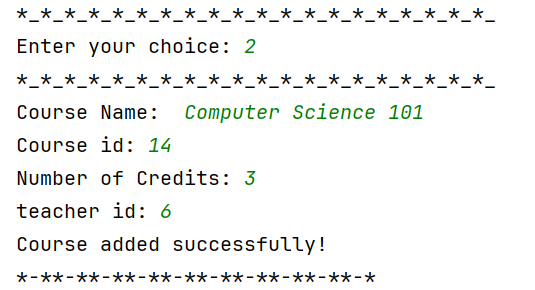
**payment record :**

**output in database:**

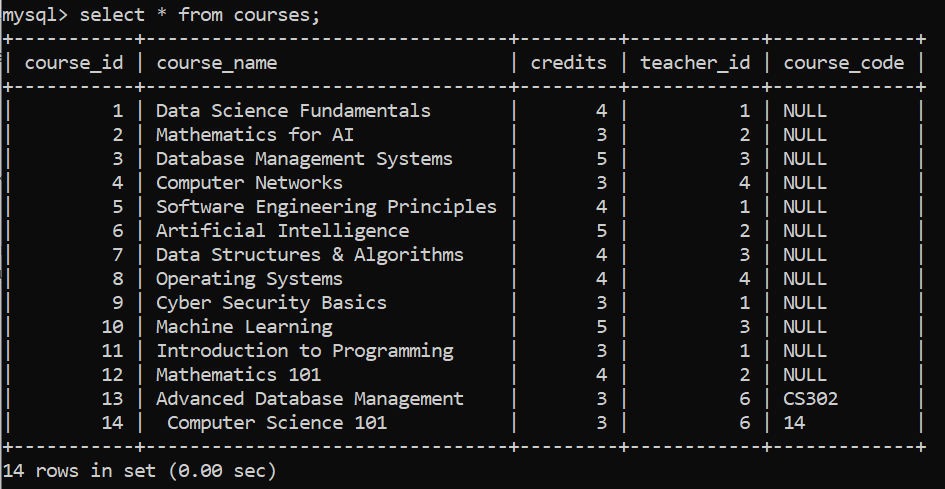
****

**Task 11: enrolment Report Generation:**

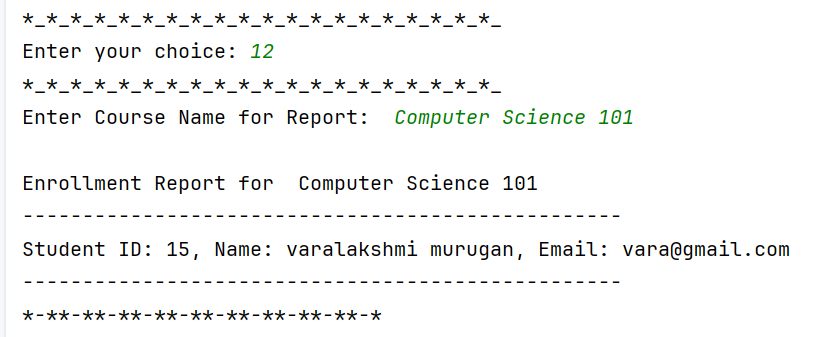
**New course creation :**

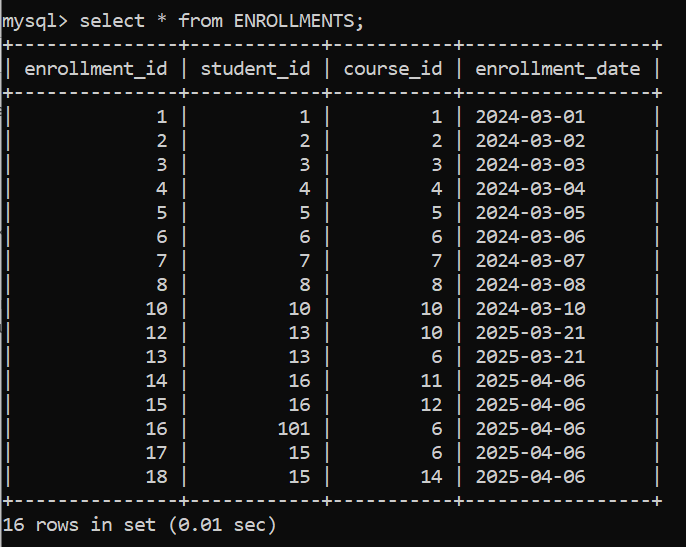
****

**output in database:**



**ENROLLMENT REPORT:**

as per the course computer science 101, it returned as the student named Varalakshmi has enrolled in computer science 101. even though she’s enrolled in another course called artificial intelligence.



**END OF THE DOCUMENT**