# Abstract

In this project, we explored the powerful role that databases play in organizing and managing complex data systems. Focusing on a university setting, we designed and implemented a relational database capable of handling academic and administrative tasks such as managing students, instructors, courses, exams, and fees. Through this experience, we learned how structured data can be efficiently stored, accessed, and manipulated using SQL and Java. By integrating our Java application with a MySQL database via JDBC (Java

Database Connectivity), we were able to perform real-time data operations, manage

relationships between entities, and ensure data integrity through constraints and transaction control. This project not only introduced us to foundational database concepts like ER diagrams, DDL and DML statements, but also gave us hands-on experience building a fully functional system that reflects real-world university operations.

Important finding from our implementation. Databases are extremely helpful for storing data in an organized manner. Databases allow you to store, manage, and retrieve large amounts of data efficiently and consistently. Rather than using flat files or manual systems,

databases provide structure through tables, relationships, and constraints, making it easier to enforce data integrity and reduce redundancy. This organized approach is crucial for

applications that handle anything from user accounts to product inventories. Implementing SQL into Java is very useful because it greatly enhances functionality. Integrating SQL with Java using technologies like JDBC (Java Database Connectivity) allows developers to build dynamic applications that can interact with databases in real time. This makes Java

applications far more robust, as they can create, read, update, and delete records on the fly. It also separates the data layer from the application logic, which is a best practice in software design. This integration allows complex operations like filtering, sorting, and joining data to be handled efficiently by the database engine rather than manually in code. Creating a

database is easy and provides a fast way to store and sort data. Most relational database management systems (RDBMS) make it relatively simple to set up and create databases using SQL commands. Once set up, a database can store millions of records and allow for fast retrieval through indexing, sorting, and optimized queries. Tools like MySQL Workbench or SQLite make it even easier to visualize and interact with the data. This is especially helpful for projects that need to manage user input, transactions, or inventory without complex file handling. Databases are a critical part of virtually all computer systems. From websites and

mobile apps to enterprise-level systems, databases power the storage and retrieval of essential data. Whether it's a social media app fetching user posts, an e-commerce site showing products, or a healthcare system accessing patient records, databases enable seamless access to different types of information in real time. They serve as the backbone of most information systems and are crucial for data-driven decision making. JDBC simplifies the process of connecting Java applications to databases. JDBC provides a standardized API for establishing connections between Java applications and relational databases. It allows developers to

execute SQL queries, manage transactions, and ensure data integrity. One powerful feature of JDBC is its support for transaction control through methods like commit() and rollback().

With commit(), developers can ensure that a group of database operations are saved together as a single unit of work, while rollback() allows them to undo changes if an error occurs, preventing data corruption or inconsistency. This transactional control gives developers

confidence that their applications can handle real-world scenarios involving complex data operations safely and reliably.

# Introduction

The core of our project is a comprehensive relational database system designed to manage various aspects of university operations. It includes multiple interconnected entities that reflect real-world academic structures. The Student table stores essential details such as student IDs, names, semesters, courses enrolled, grades, and total credits earned. The Instructor table captures information like instructor IDs, names, departments, salaries, and courses they teach. Academic departments are represented through the Department table, which includes department names and heads. The Courses table provides data about course IDs, names, prerequisites, associated departments, and sections. Financial transactions are managed through the Fees table, which tracks tuition fees, payment amounts, and student associations. Additionally, the Exam table is used to manage assessments linked to students and the instructors responsible for them. To accurately reflect complex academic relationships, we implemented several many-to-many relationship tables. For example,

students can enroll in multiple courses, and courses can have multiple students, while

instructors may teach several courses and oversee various exams. These relationship tables help ensure that all associations between entities are clearly defined and efficiently managed. Together, these components form a robust database schema, allowing for efficient data organization, retrieval, and manipulation to support key university functions.

# The System:

Requirements we have satisfied. We drew our ER diagram for the database, wrote DDL

statements for the entities and relationships, explained our DDL statements in relation to the ER diagram. We also used sample insert statements for each table, we connected our database using JDBC, and created the database schema using JDBC. We also inserted the records into all of the tables. Our database also calculates the income tax rate of instructors and we print

the tax amount for each of the instructors. In addition, we printed the student with the highest grade for each semester. We explained the interface and showed the DML statements.

Moreover, we wrote it using java. We performed the CRUD functions, such as delete, insert, and update. Our tables were created using JDBC and it handled possible errors. Our database had the following functions: calculated the tax of the instructors and the highest grade of the student, stored data about students, instructors, and exams, courses, and departments.

# Conclusion

Throughout this project, we gained a deep understanding of how relational databases are used to organize, manage, and retrieve large amounts of structured data efficiently. We learned that databases offer powerful advantages over flat files by maintaining data integrity through tables, relationships, and constraints, which are especially important for real-world

applications like university systems. Integrating SQL into Java using JDBC proved to be a valuable skill, allowing us to build dynamic applications capable of performing real-time operations such as creating, reading, updating, and deleting data. We also explored transaction control in JDBC using commit() and rollback() to maintain consistency and reliability during complex operations. In designing our university database, we modeled

real-world academic elements by creating entities like students, instructors, departments, courses, exams, and fees. We built and implemented this structure through ER diagrams and DDL statements, and we reinforced the relationships, especially many-to-many ones, using appropriate linking tables. The practical work involved using Java and JDBC to not only

create the database schema but also insert and manipulate data, ensuring error handling throughout. Additionally, we programmed specific features such as calculating instructor tax and identifying students with the highest grade in each semester. We used DML statements to manipulate data and successfully performed all CRUD operations. By completing this project, we developed hands-on experience in full-stack database development, from design and creation to integration and functionality, highlighting the essential role of databases in modern computing.

### BIBLIOGRAPHY

I used the following article to understand how to form a JDBC connection in java.

-Author: Shreya Gupta

* Title: Establishing JDBC connection in java
* Website: geeksforgeeks.org
* URL: <https://www.geeksforgeeks.org/establishing-jdbc-connection-in-java/>

I used the following article to solidify my knowledge on DDl statements.

-Author: GeeksforGeeks

* Title: DDL (Data definition language)
* Website: geeksforgeeks.org
* URL:

[https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#ddl-data-definition-langu](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#ddl-data-definition-language) [age](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#ddl-data-definition-language)

I used the following article to solidify my knowledge on DMl statements.

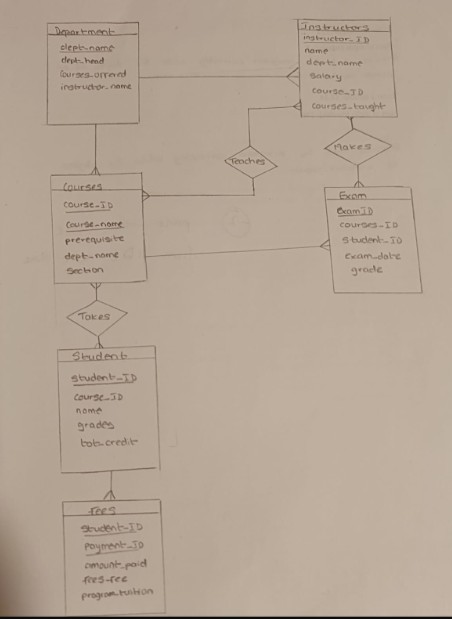
-Author: GeeksforGeeks

* Title: DML (Data manipulation language)
* Website: geeksforgeeks.org
* URL:

[https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#dmldata-manipulation-la](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#dmldata-manipulation-language) [nguage](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/#dmldata-manipulation-language)

# Appendix

## ER Diagram:

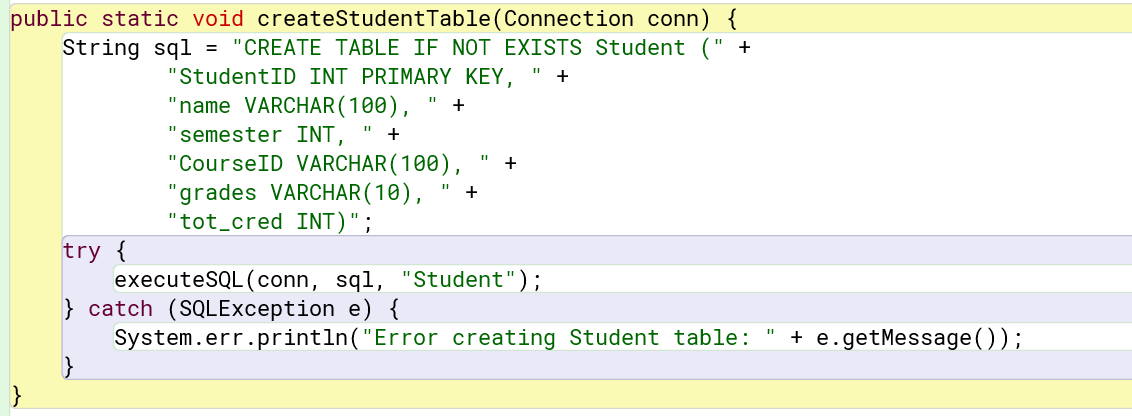
****

**DDL statements**

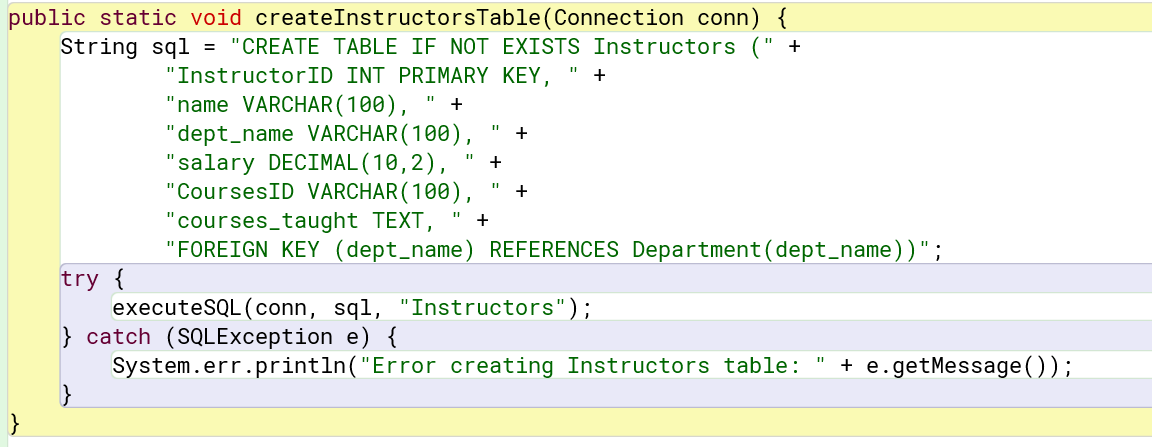
1. Department table

## A screenshot of a computer code AI-generated content may be incorrect.9

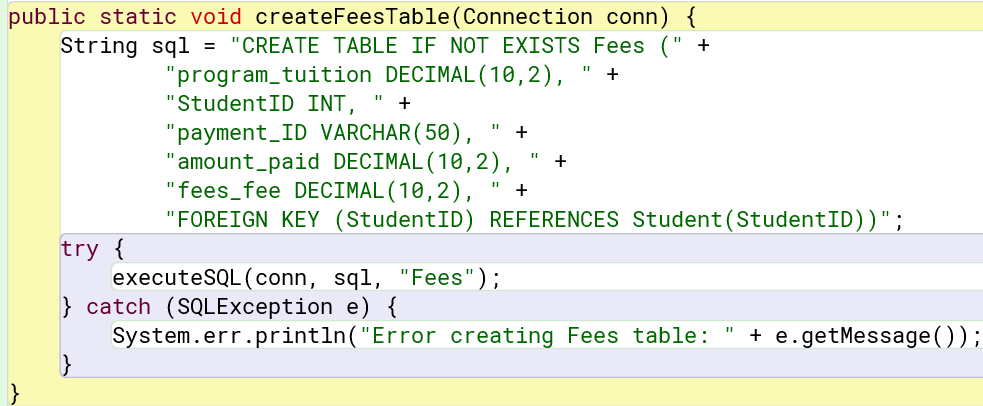
1. Student table



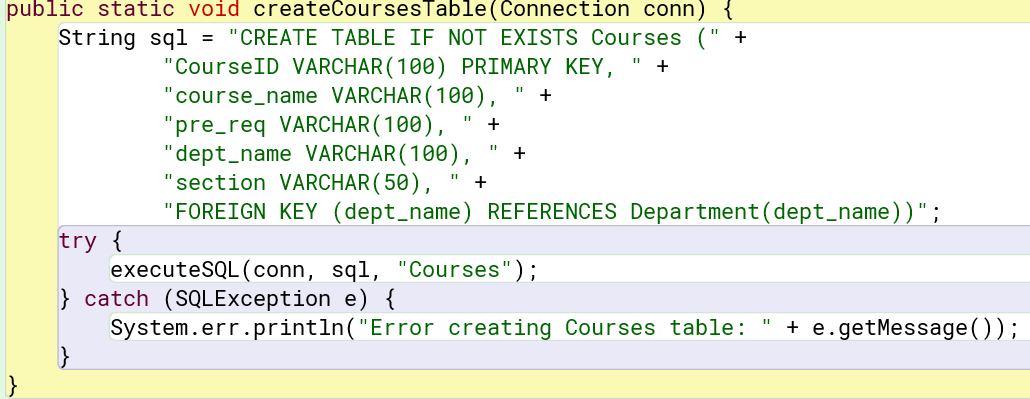
1. Instructor table



1. Feese Table

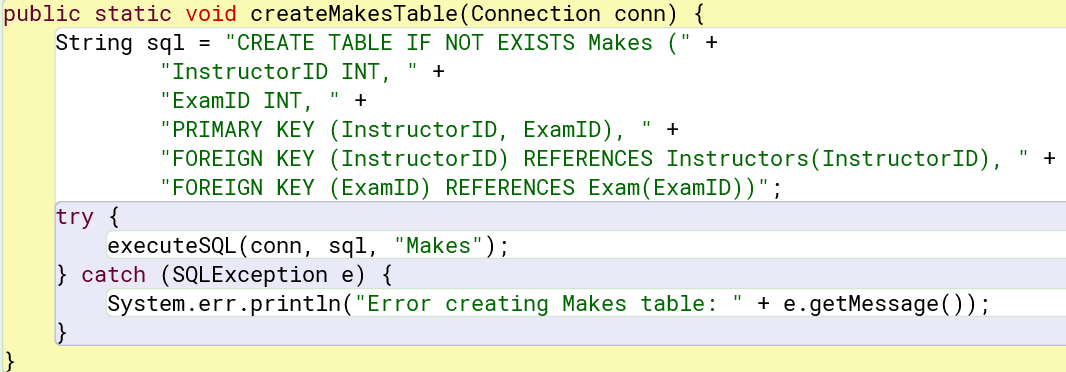


1. Courses Table

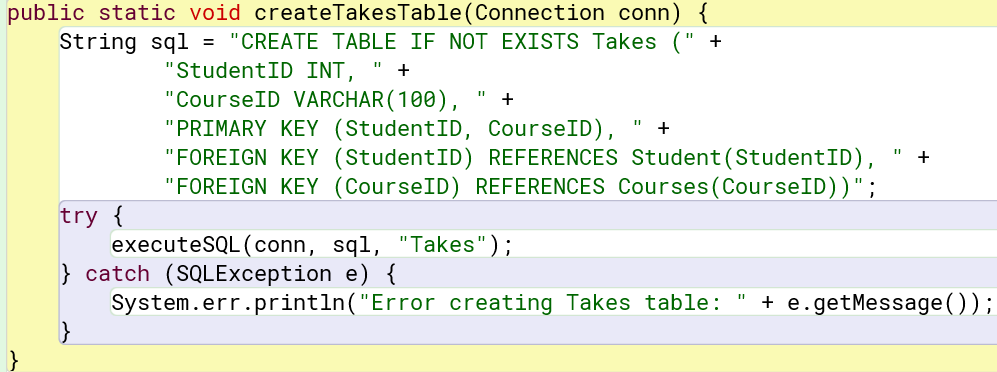


## Relationship Tables

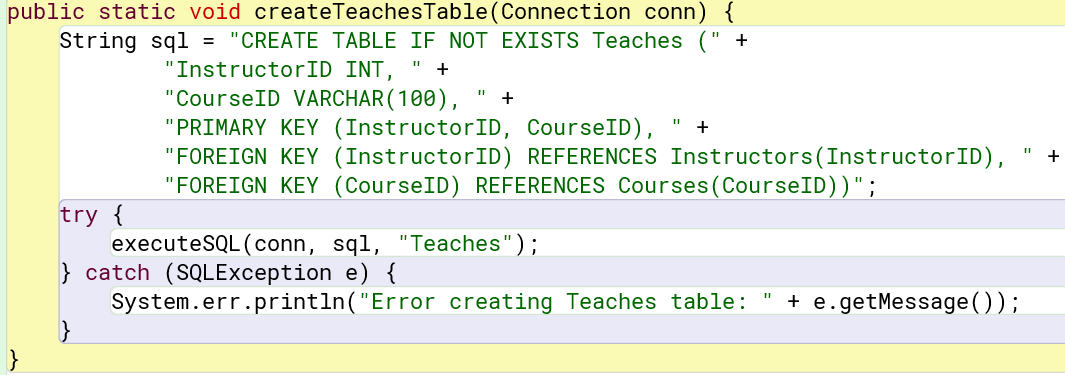
1. Makes table



1. Takes Table



1. Teaches table

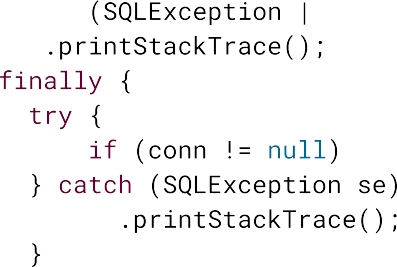
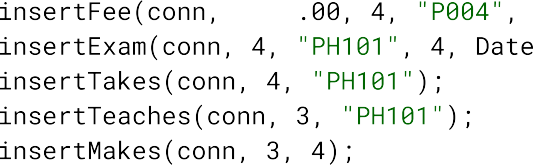
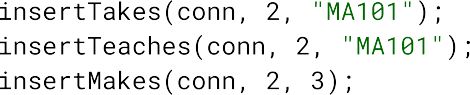


### Java Source Code

<https://github.com/MedhaMadhub/Database-Project>



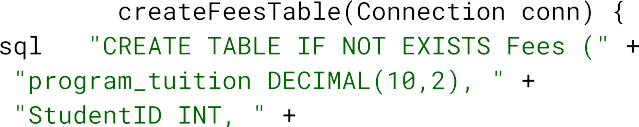
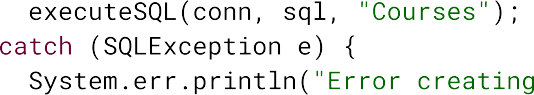
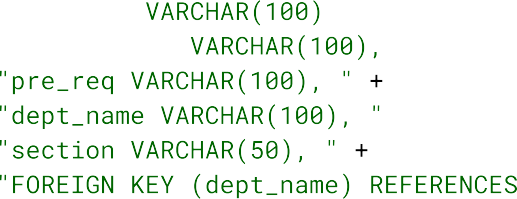
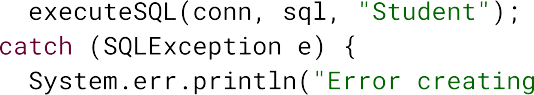
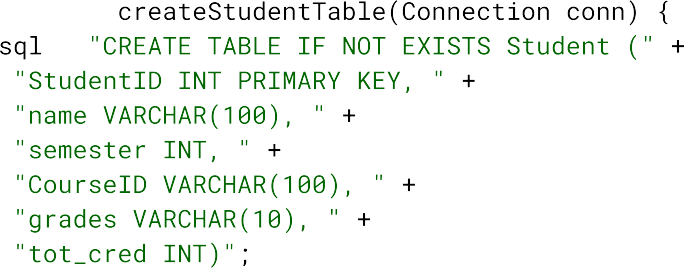
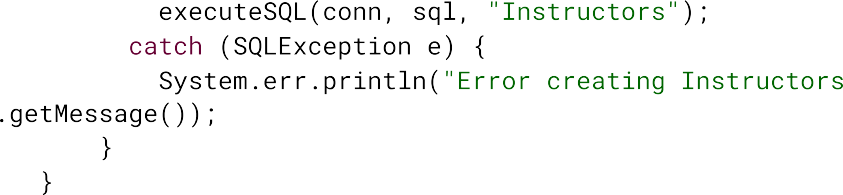
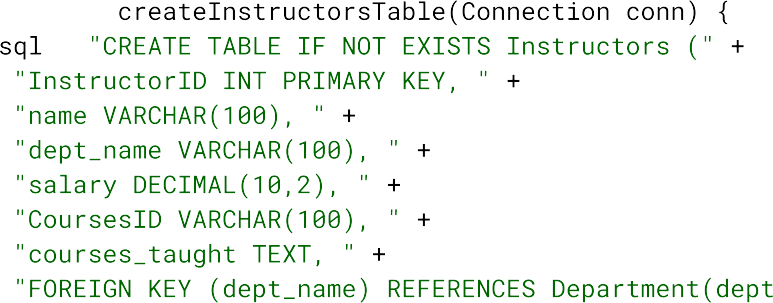




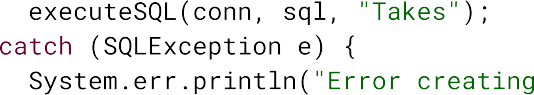
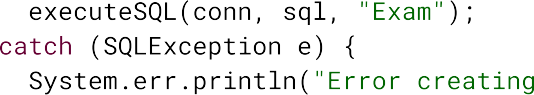


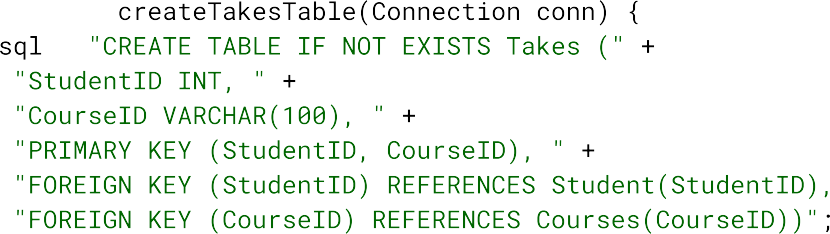
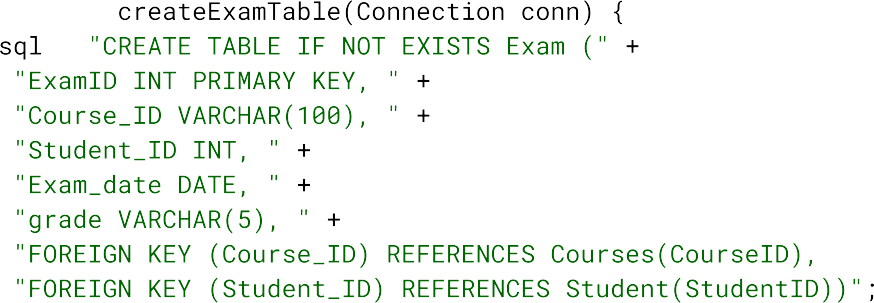
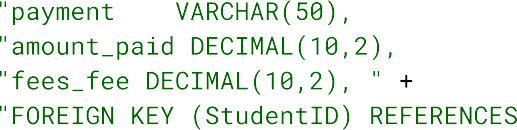




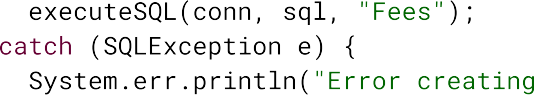






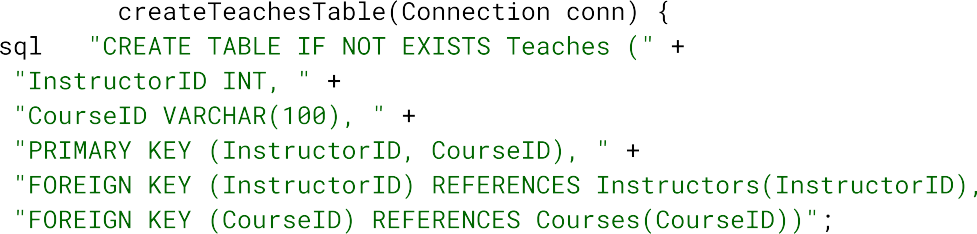


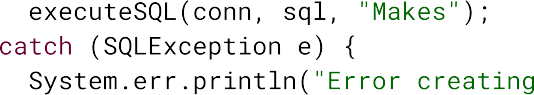


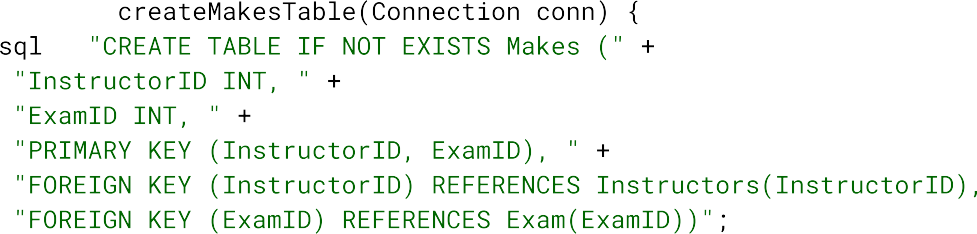
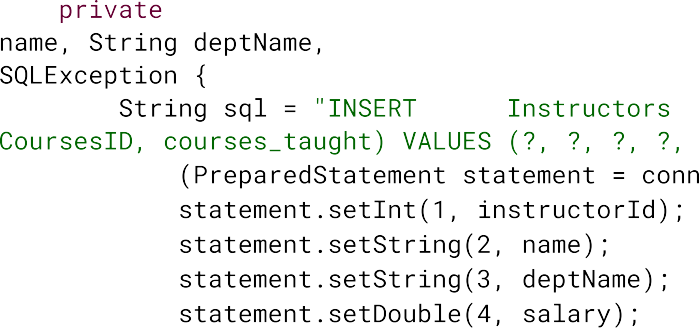
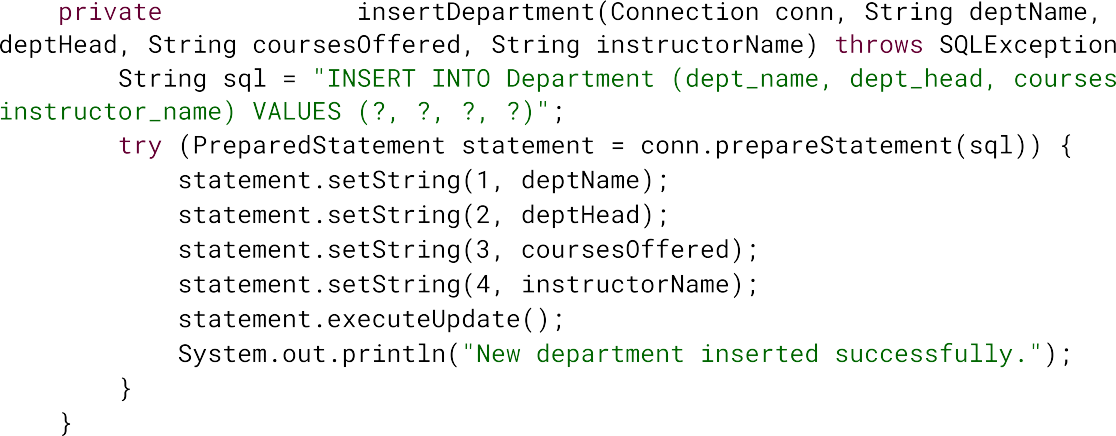
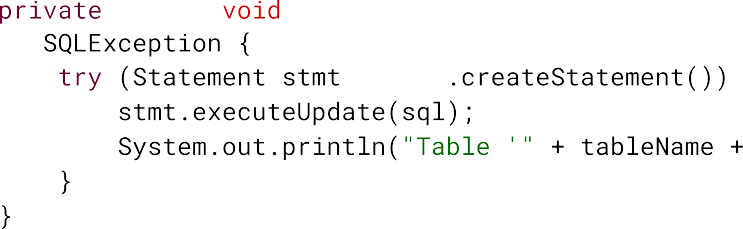








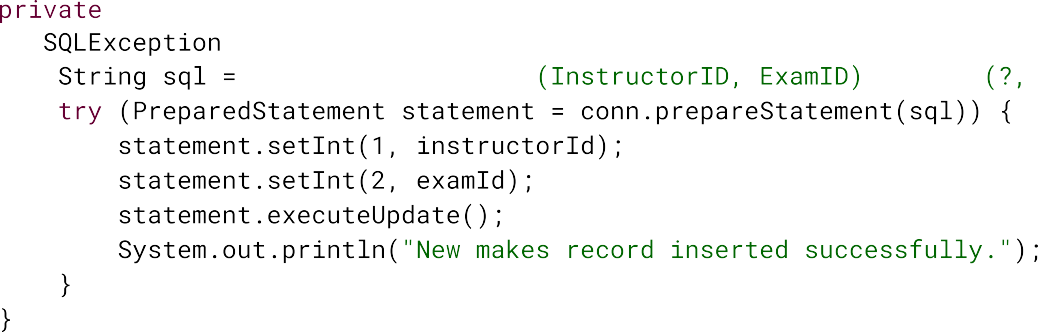
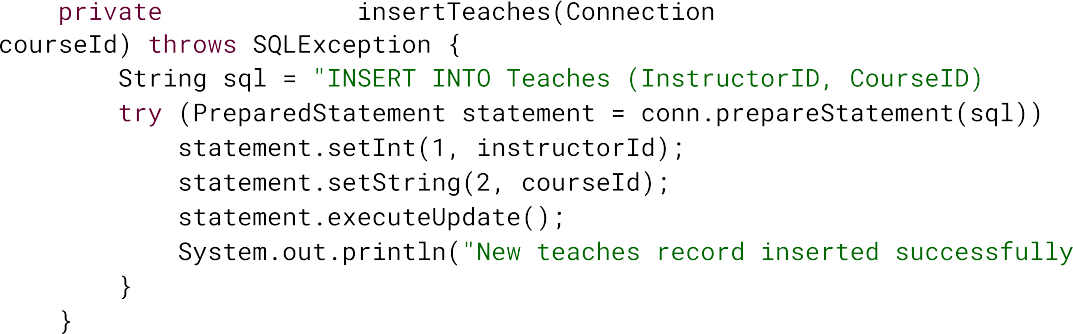
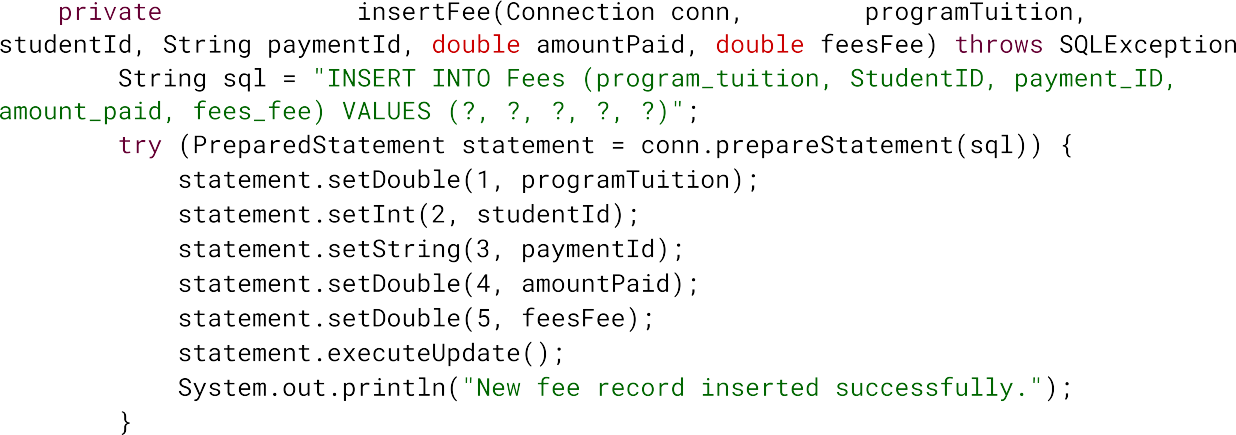
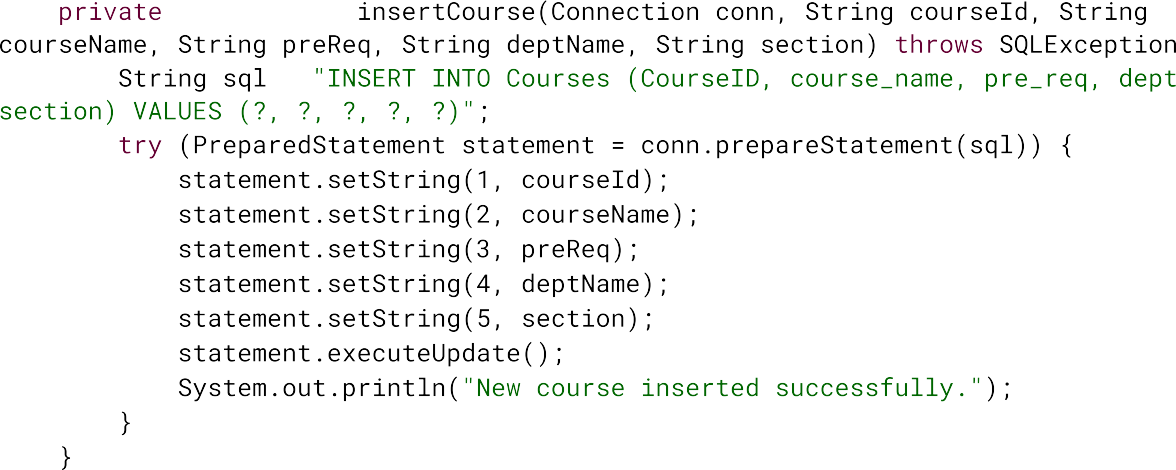
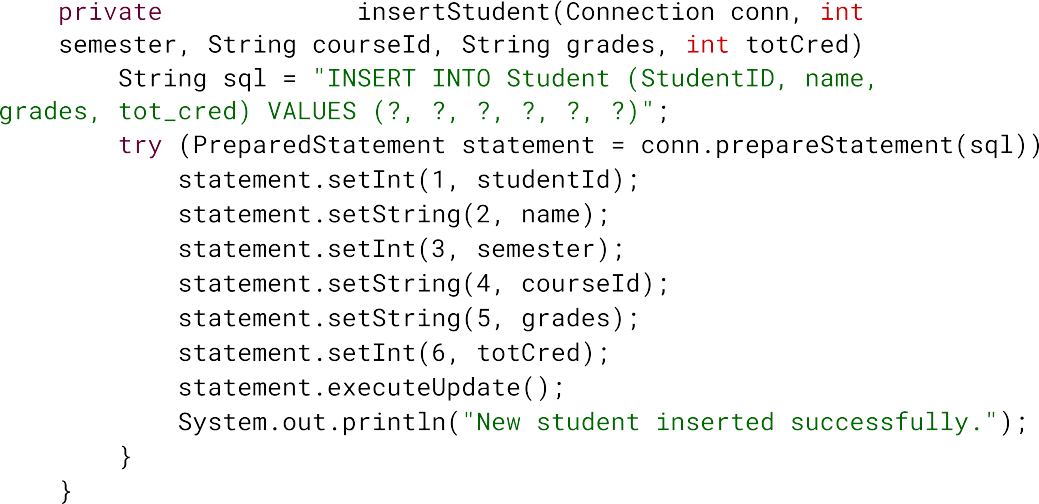




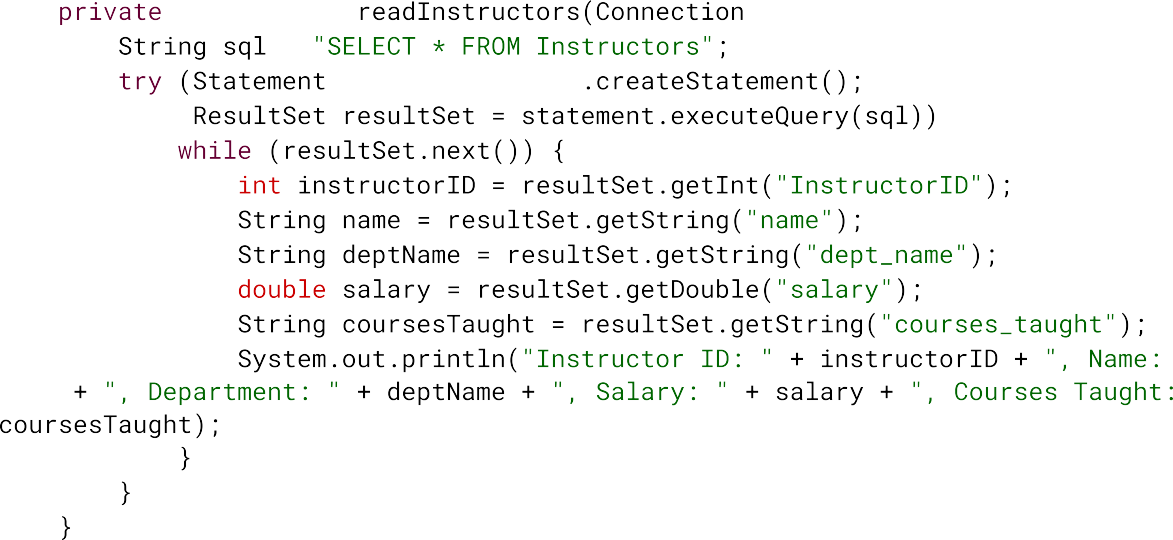
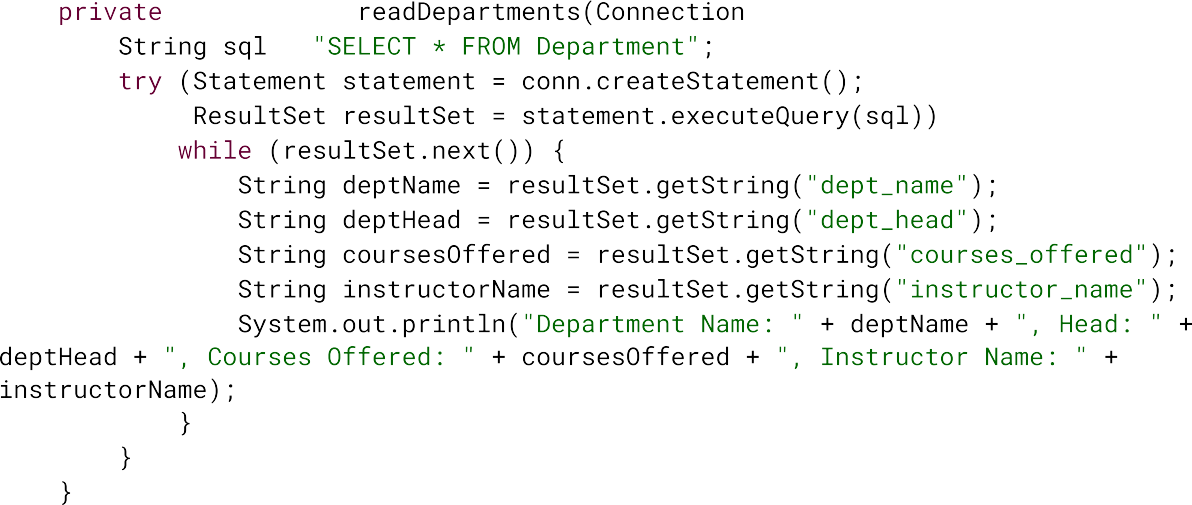
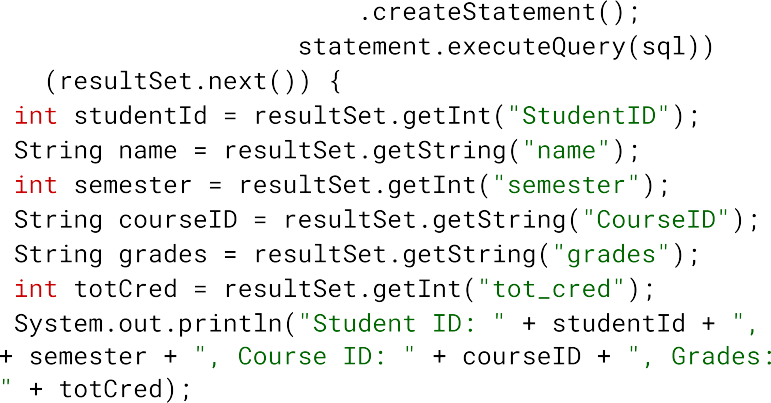
  



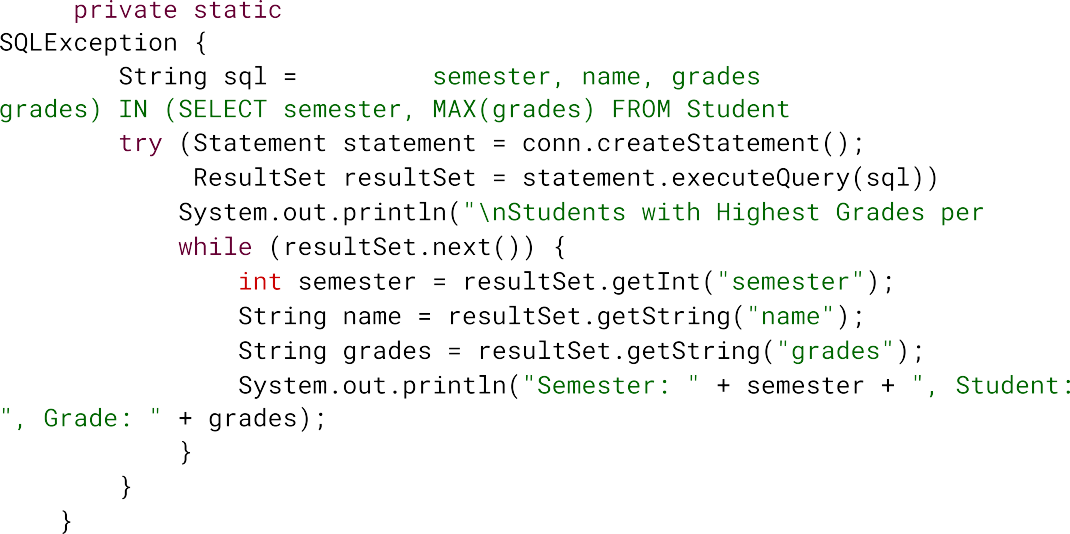
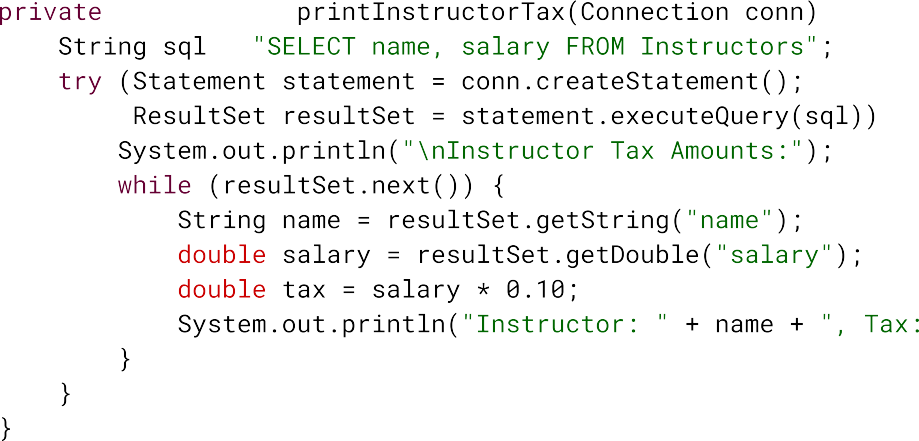
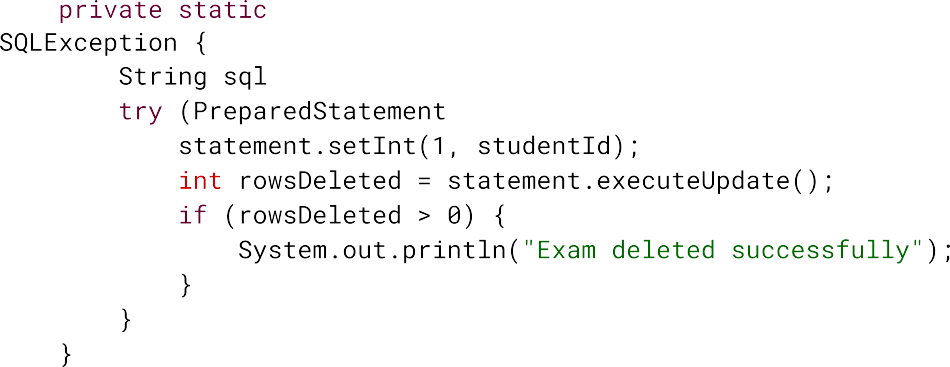
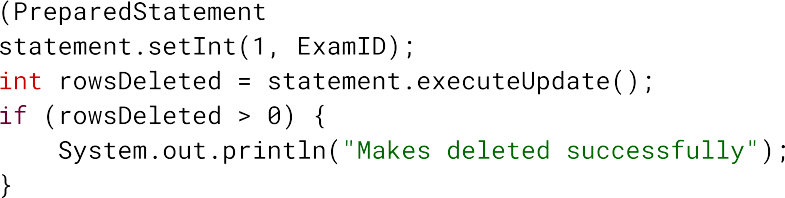












**USER’S GUIDE**

The interface allows the user to insert new values into tables such as Department, Student, Instructor, Fees, and Courses or delete values from these tables. It also allows the user to change data within these tables. The interface starts with providing the user 3 options asking what type of change the user wants to make to the database: Insert, Delete or Change or exit the interface. If the user selects Insert, it then asks the user which table it wants to insert data into: The Department, Student, Instructor, Fees, and Courses tables. Once the user selects which table they want to insert values into, the interface provides a sample format the user has to enter the new values in, then allows the user to enter the new values for the table. Similarly, if the user selects delete then the interface asks which table they want to remove data from: The Department, Student, Instructor, Fees, and Courses tables, when the user selects the table, they want to remove the values from, the interface asks the primary key of the record they want to remove (e.g., DepartmentID, StudentID). Once the user enters the primary key they want to remove, the corresponding values are removed. If the user selects Change, the interface allows for modification of fields within the tables; the interface asks for the relevant record's primary key and the new data value.