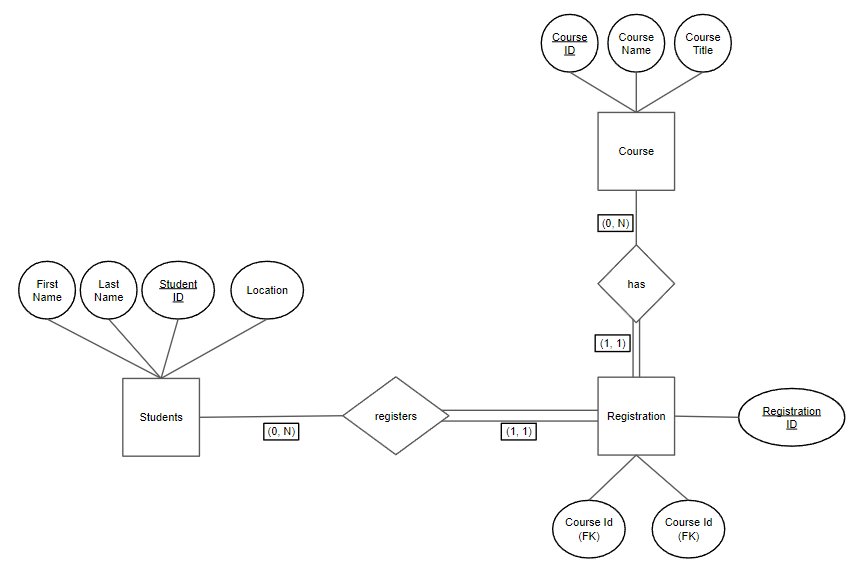
**Exercise 1 - Student Database**

The Entity Relationship Diagram (ERD) describes the relationship between three entities: Student, Course, and Registration.The **Student** entity has the following attributes: StudentId (which serves as its primary key), FirstName, LastName, and Location. The **Course** entity encompasses the attributes: CourseId (the primary key), CourseName, and CourseTitle. Meanwhile, the **Registration** entity is characterized by its attributes: RegistrationId (primary key), CourseId, and StudentId, both of which are foreign keys.

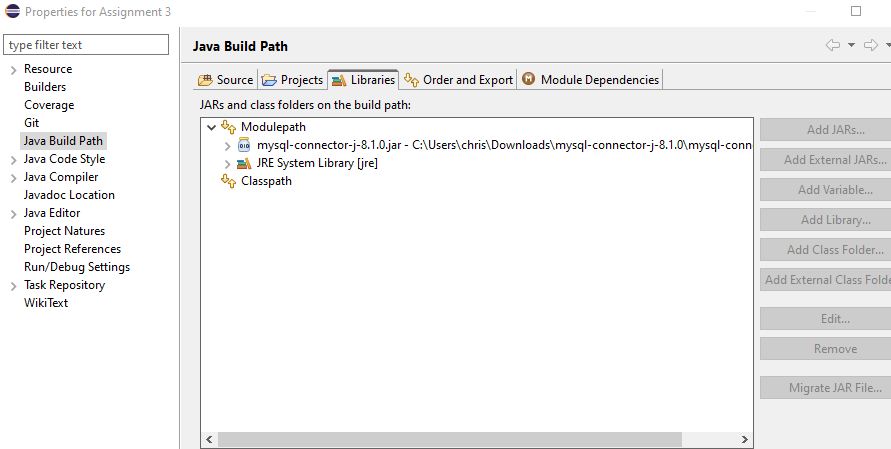
There's a defined relationship between the Student and Registration entities. Specifically, a student can register for multiple courses, with each registration uniquely corresponding to a student. This relationship has a cardinality of 0 to N for the Student (implying not every student may be registered for a course) and a strict 1 to 1 for the Registration (indicating each registration is uniquely linked to a student).

Similarly, the Course and Registration entities are interrelated. A course might have multiple students enrolled, and each registration distinctly correlates to a specific course. The cardinality for this relationship is 0 to N for the Course (meaning not every course might have students registered) and 1 to 1 for the Registration (ensuring each registration is tied to a single course).

In essence, the system is designed such that students can register for various courses, with the Registration entity acting as the intermediary that links students to their chosen courses.

****

Connecting a MySQL local server to a Java application was achieved through the Java Database Connectivity (JDBC) framework. For This assignment, we used **version 8.1.0**. To simplify the code, we simply added the jar file to the project build path:



To establish a connection to a MySQL database through Java using JDBC, you'd typically utilize the DriverManager.getConnection() method. Here's a pseudo code that demonstrates this:

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/[yourDatabase]";

String username = [yourUsername];

String password = [yourPassword];

try {

Connection connection = DriverManager.getConnection(url, username, password);

System.out.println("connection string = " + connection);

connection.close();

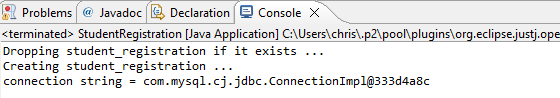
} catch (SQLException e) {

e.printStackTrace();

}

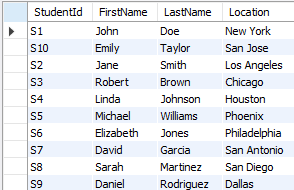
}

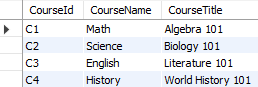
A sample output looks like the following:

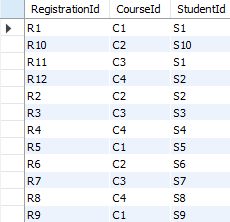


For the first exercise, it was not stated how the table was populated. As a group, we decided to create a static table given the requirements given in the ERD. Every time the code is run, the tables will be populated with the same data.

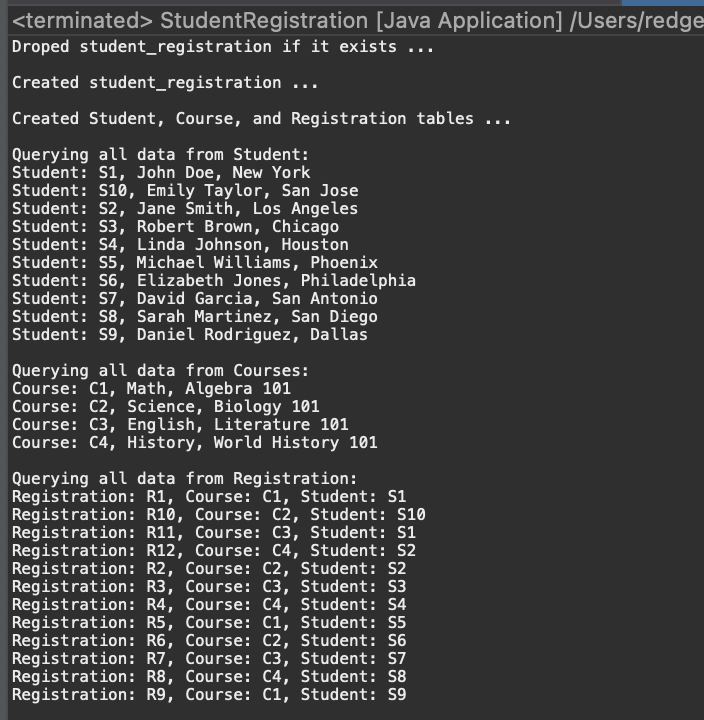
This is how the database looks from MySQL Workbench - shown below are the Student, Course, and Registration tables.







In our script, we printed out the contents of the database tables to confirm that we were querying the database correctly (shown below):

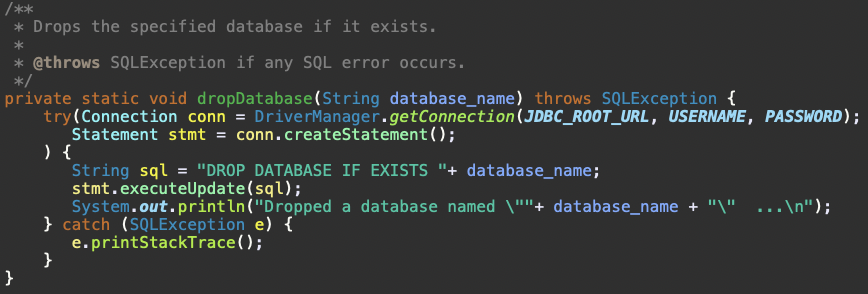


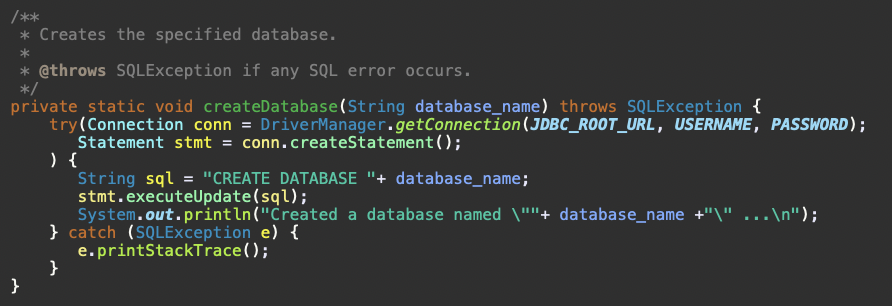
**Exercise 2 - Ticket Generator and Power BI dashboards**

Our SQL statements used to create the database and its tables were formed in TicketGenerator.java using String concatenation.

**Creating the database**

The two methods shown below utilizes the JDBC driver to drop the database if it exists, and then create the database. The SQL statements are wrapped in a try-with-resources block with their respective methods.

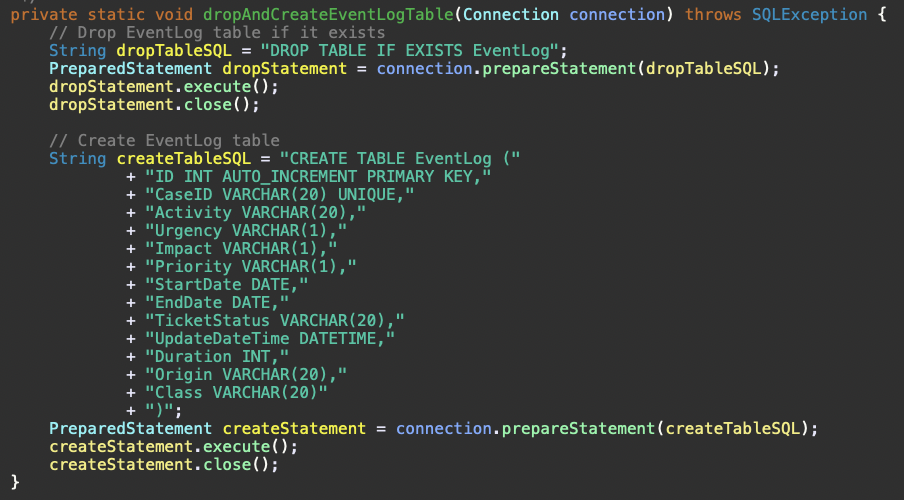




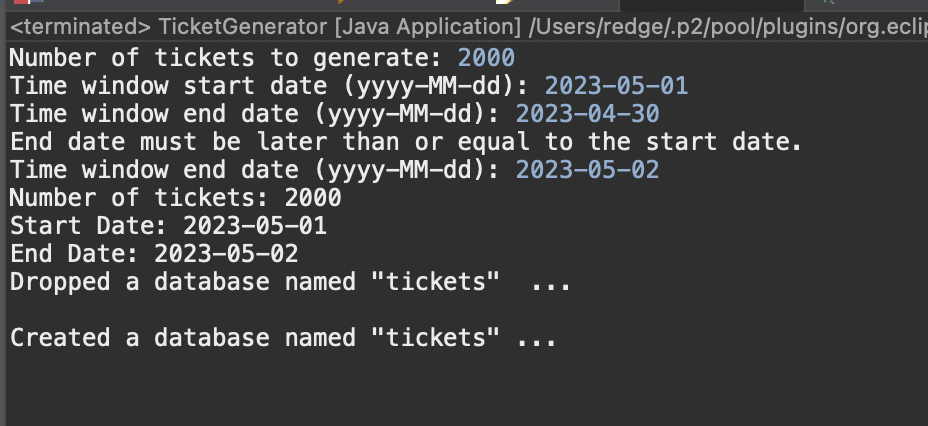
**Creating and populating the tables**

The two methods shown below create our SQL statements. Each activity requiring SQL statements were broken down into separate Strings. The general structure is as follows: one String to check if the table exists, another String to create the table, and the last String to insert the records into the table.





Sample console output for the user input section.



**Dashboard Description**

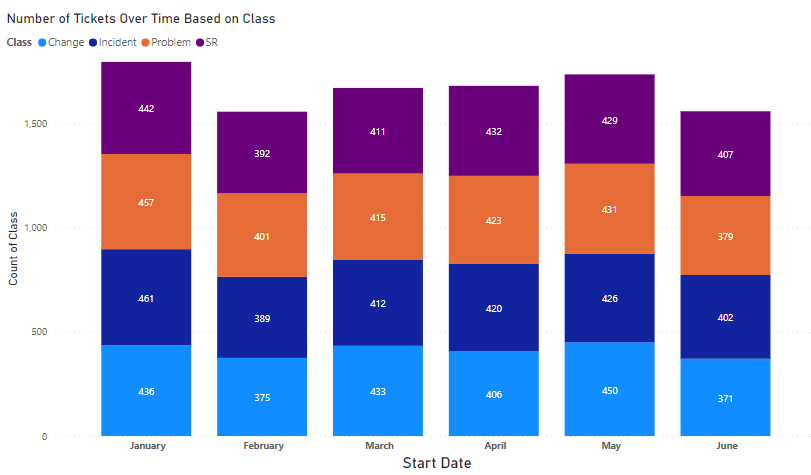
After creating the database, we used Power BI to visualize the results.

The graph below displays ticket counts for four classes (Change, Incident, Problem, SR) from January to June.

Some of the trends seen in the graph:

* Change Tickets: Peak in January (461), lowest in June (371).
* SR Tickets: Highest in April (432), moderate decrease by June (407).
* Incident & Problem Tickets: Relatively stable counts, minor fluctuations.

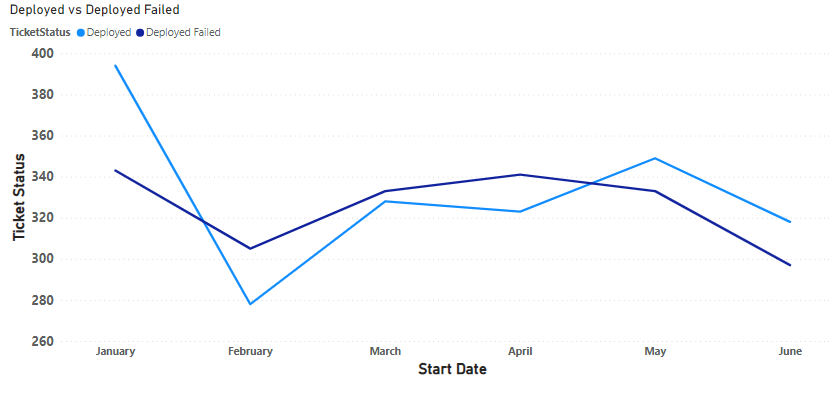
Overall, the ticket distribution is balanced across classes. Change tickets show a declining trend, while SRs peaked in April. Incidents and Problems remain consistent.



The graph below compares ticket counts for two statuses, "Deployed" and "Deployed Failed," from January to June. Some observations from the following in the graph:

* Deployed Tickets (Blue Line): Start high in January, dip in February, peak in March, and show a steady decline thereafter.
* Deployed Failed Tickets (Dark Line): Begin moderately in January, peak in February, decrease in March, and then maintain a more gradual decline through June.

While "Deployed" tickets peaked in March and then decreased, "Deployed Failed" tickets had their highest count in February, with a consistent drop-off towards June.

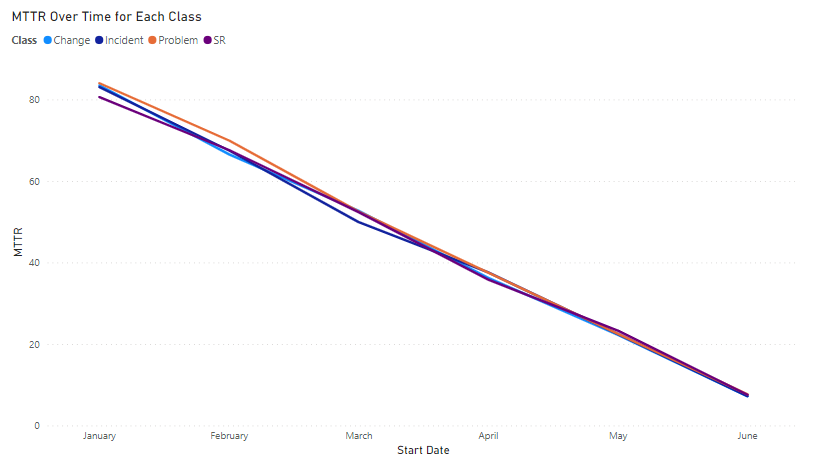


The graph below depicts the "Mean Time to Resolve (MTTR)" for the four ticket classes from January to June.

All ticket classes (Change, Incident, Problem, SR) have shown a consistent decline in MTTR from January to June.

* The MTTR values start close to 80 in January and reduce to near 20 by June.
* The lines for all classes overlap closely, indicating similar resolution times for each class.

Across all ticket classes, the time taken to resolve issues has improved significantly over the six months, indicating more efficient resolution processes or fewer complexities encountered in recent months.

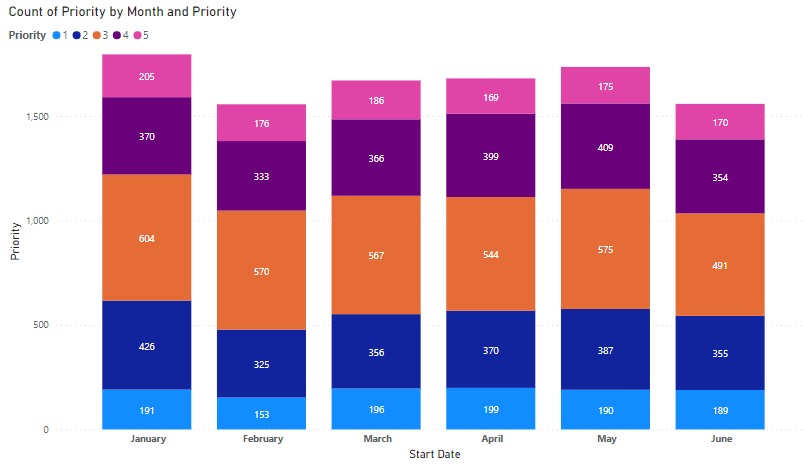


We decided to explore the count of priority by month from January to June.

Some observations about the plot:

* Priority 1 (Blue) tickets range from around 190 in January to 189 in June, remaining fairly stable.
* Priority 2 (Orange) tickets see an increase from 426 in January to a peak of 575 in May, then a slight drop in June.
* Priority 3 (Purple) tickets are relatively steady, hovering around 550-600 tickets.
* Priority 4 (Pink) tickets fluctuate, peaking at 409 in May.
* Priority 5 (Teal) gradually decreases from 205 in January to 170 in June.

Overall, the priority 2 tickets have shown a notable rise, peaking in May. Priority 5 tickets saw a consistent decline. The rest maintained a more or less consistent trend across the six months.



Another exploratory trend we explored was the distribution of ticket holders among the five individuals.

The ticket distribution among the holders is fairly balanced, with each individual managing roughly one-fifth of the total tickets. George E. has the highest count, but the difference is minimal.

