ENSF 607
Assignment 3

Chioma Ukaegbu
Christian Valdez
Redge Santillan

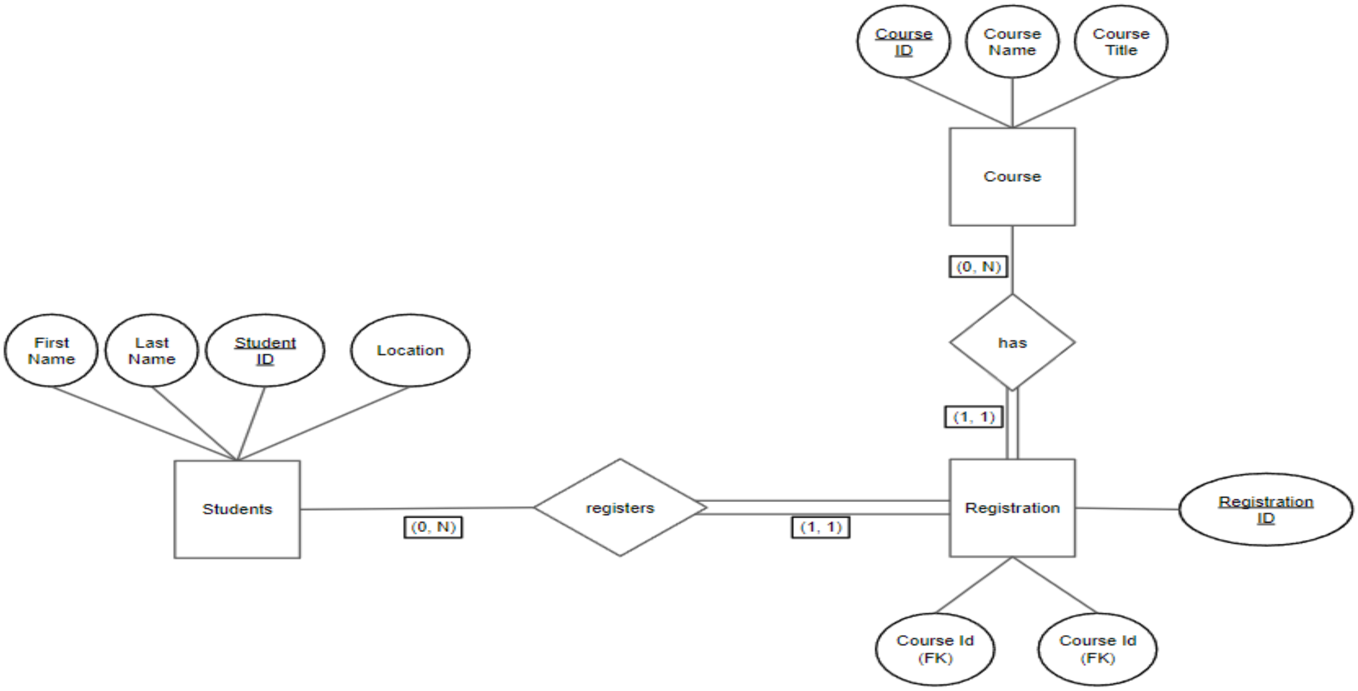
Exercise 1 - Student Database

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.

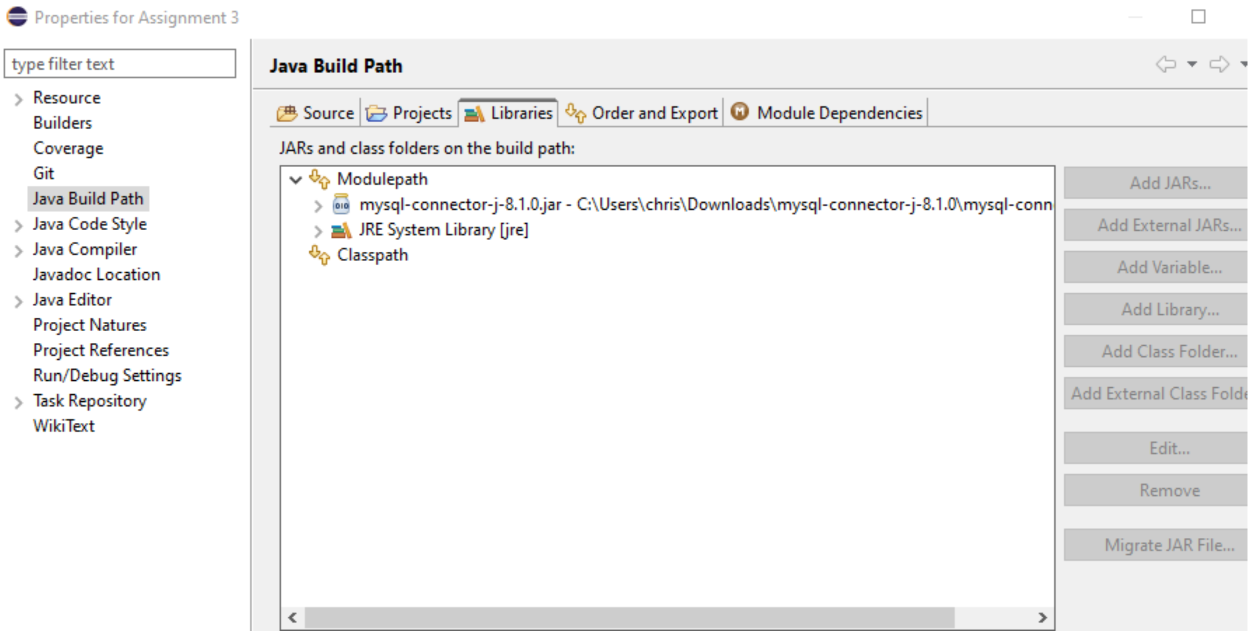
There's a defined relationship between the Student and Registration entities. Specifically, a student can register for multiple courses, with each registration uniquely corresponding to a student. This relationship has a cardinality of 0 to N for the Student (implying not every student may be registered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



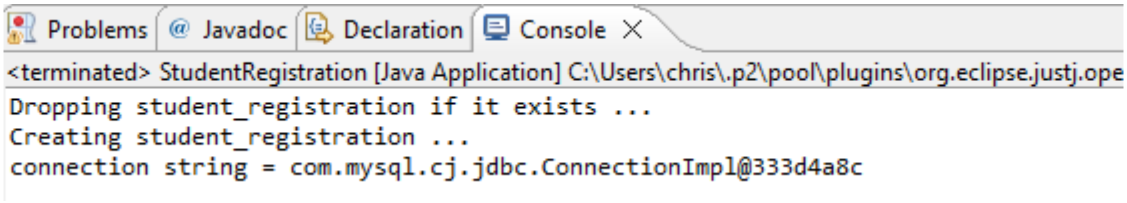
JARs and class folders on the build path:

- Modulepath
  - mysql-connector-j-8.1.0.jar - C:\Users\chris\Downloads\mysql-connector-j-8.1.0\mysql-conn
  - JRE System Library [jre]
- Classpath

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();
        }
    }
```

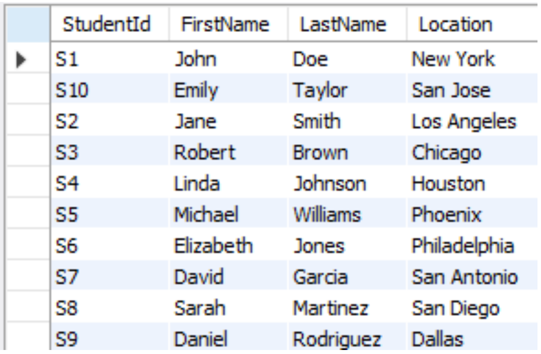
A sample output looks like the following:



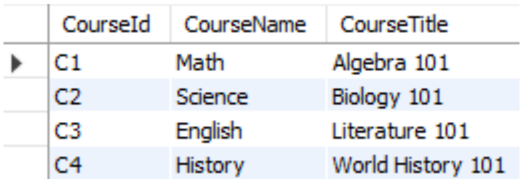
```
<terminated> StudentRegistration [Java Application] C:\Users\chris\.p2\pool\plugins\org.eclipse.justj.ope
Dropping student_registration if it exists ...
Creating student_registration ...
connection string = com.mysql.cj.jdbc.ConnectionImpl@333d4a8c
```

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.

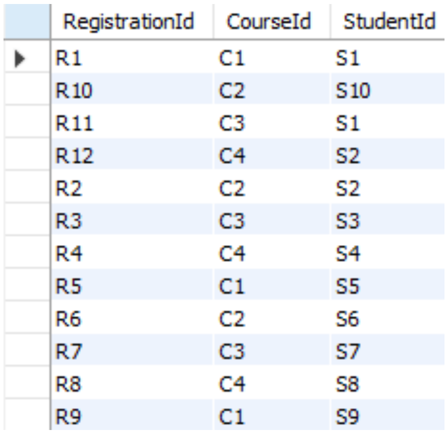
This is how the database looks from MySQL Workbench - shown below are the Student, Course, and Registration tables.



StudentId	FirstName	LastName	Location
S1	John	Doe	New York
S10	Emily	Taylor	San Jose
S2	Jane	Smith	Los Angeles
S3	Robert	Brown	Chicago
S4	Linda	Johnson	Houston
S5	Michael	Williams	Phoenix
S6	Elizabeth	Jones	Philadelphia
S7	David	Garcia	San Antonio
S8	Sarah	Martinez	San Diego
S9	Daniel	Rodriguez	Dallas

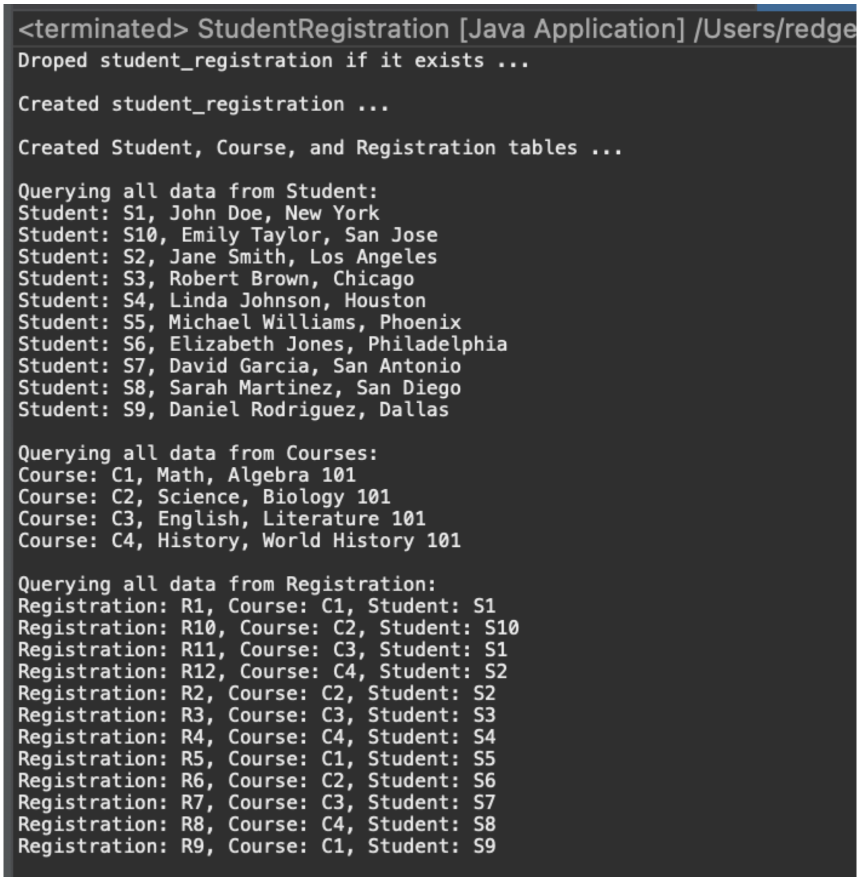


CourseId	CourseName	CourseTitle
C1	Math	Algebra 101
C2	Science	Biology 101
C3	English	Literature 101
C4	History	World History 101



RegistrationId	CourseId	StudentId
R1	C1	S1
R10	C2	S10
R11	C3	S1
R12	C4	S2
R2	C2	S2
R3	C3	S3
R4	C4	S4
R5	C1	S5
R6	C2	S6
R7	C3	S7
R8	C4	S8
R9	C1	S9

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


<terminated> StudentRegistration [Java Application] /Users/redge

```
Droped student_registration if it exists ...

Created student_registration ...

Created Student, Course, and Registration tables ...

Querying all data from Student:
Student: S1, John Doe, New York
Student: S10, Emily Taylor, San Jose
Student: S2, Jane Smith, Los Angeles
Student: S3, Robert Brown, Chicago
Student: S4, Linda Johnson, Houston
Student: S5, Michael Williams, Phoenix
Student: S6, Elizabeth Jones, Philadelphia
Student: S7, David Garcia, San Antonio
Student: S8, Sarah Martinez, San Diego
Student: S9, Daniel Rodriguez, Dallas

Querying all data from Courses:
Course: C1, Math, Algebra 101
Course: C2, Science, Biology 101
Course: C3, English, Literature 101
Course: C4, History, World History 101

Querying all data from Registration:
Registration: R1, Course: C1, Student: S1
Registration: R10, Course: C2, Student: S10
Registration: R11, Course: C3, Student: S1
Registration: R12, Course: C4, Student: S2
Registration: R2, Course: C2, Student: S2
Registration: R3, Course: C3, Student: S3
Registration: R4, Course: C4, Student: S4
Registration: R5, Course: C1, Student: S5
Registration: R6, Course: C2, Student: S6
Registration: R7, Course: C3, Student: S7
Registration: R8, Course: C4, Student: S8
Registration: R9, Course: C1, Student: S9
```

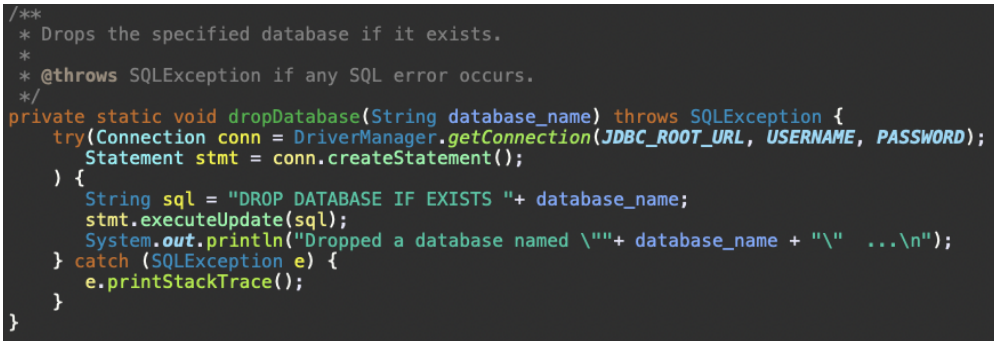
Chioma Ukaegbu
Christian Valdez
Redge Santillan

Exercise 2 - Ticket Generator and Power BI dashboards

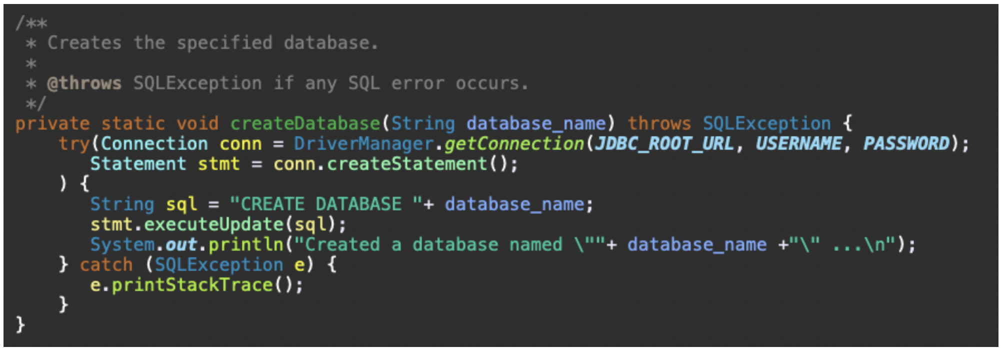
Our SQL statements used to create the database and its tables were formed in `TicketGenerator.java` using String concatenation.

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.



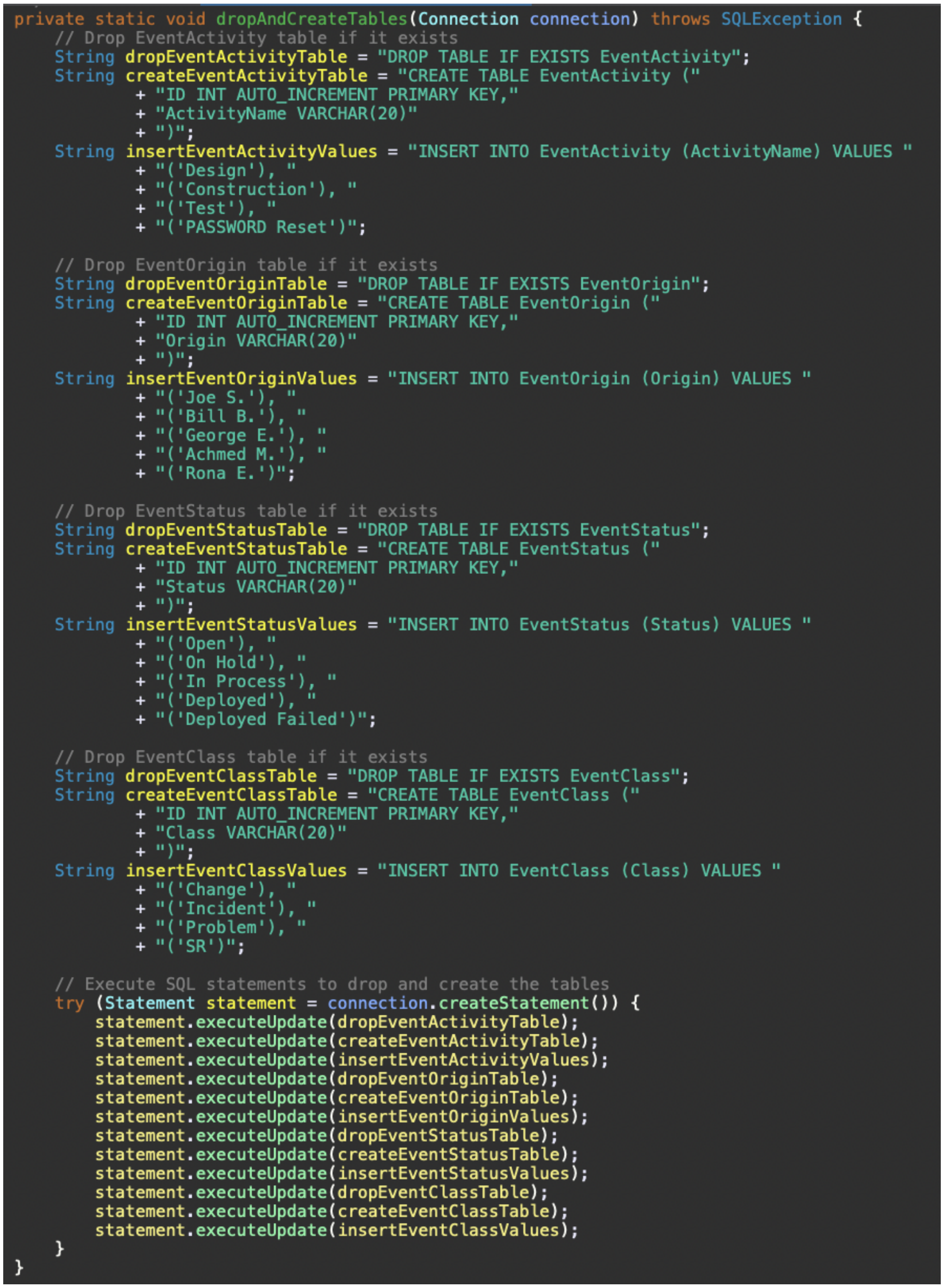
```java
/**
 * Drops the specified database if it exists.
 *
 * @throws SQLException if any SQL error occurs.
 */
private static void dropDatabase(String database_name) throws SQLException {
    try(Connection conn = DriverManager.getConnection(JDBC_ROOT_URL, USERNAME, PASSWORD);
        Statement stmt = conn.createStatement();
    ) {
        String sql = "DROP DATABASE IF EXISTS "+ database_name;
        stmt.executeUpdate(sql);
        System.out.println("Dropped a database named \""+ database_name + "\"  ...\n");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```



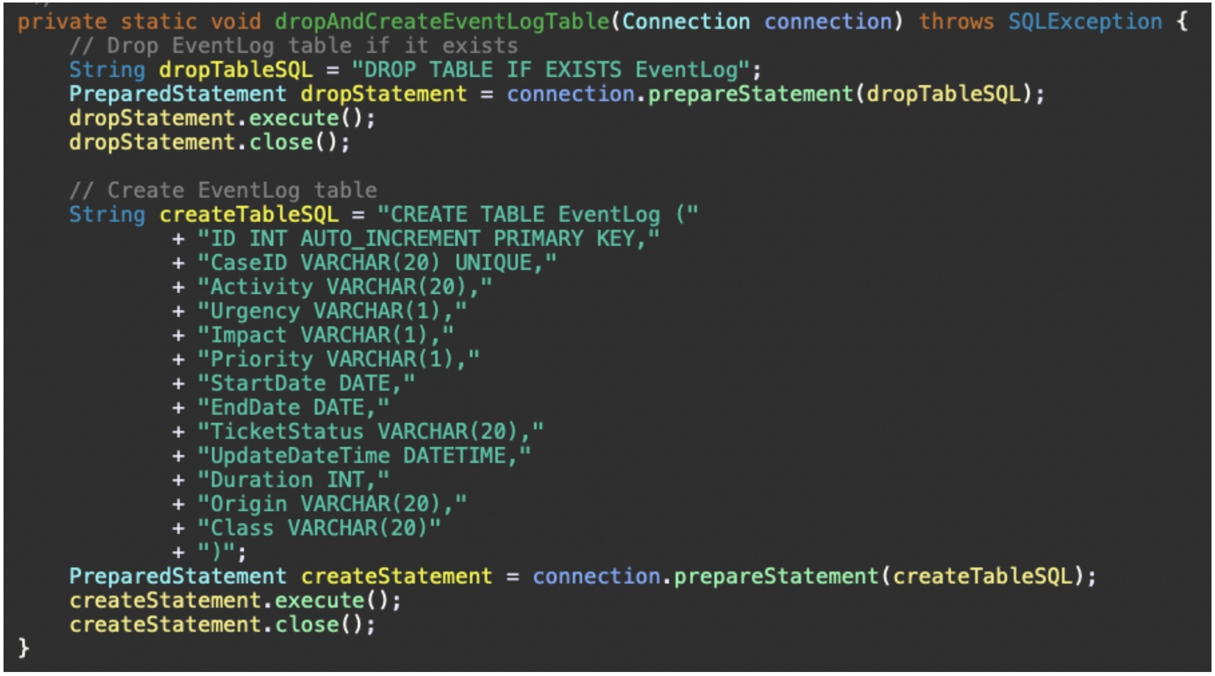
```java
/**
 * Creates the specified database.
 *
 * @throws SQLException if any SQL error occurs.
 */
private static void createDatabase(String database_name) throws SQLException {
    try(Connection conn = DriverManager.getConnection(JDBC_ROOT_URL, USERNAME, PASSWORD);
        Statement stmt = conn.createStatement();
    ) {
        String sql = "CREATE DATABASE "+ database_name;
        stmt.executeUpdate(sql);
        System.out.println("Created a database named \""+ database_name +"\" ...\n");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

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.

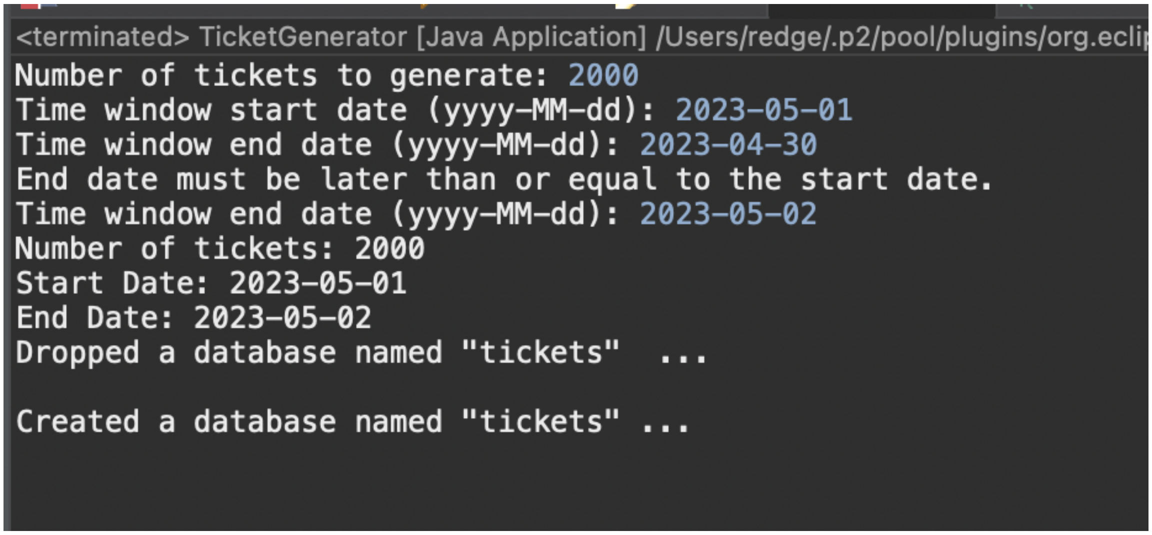


```
private static void dropAndCreateTables(Connection connection) throws SQLException {
    // Drop EventActivity table if it exists
    String dropEventActivityTable = "DROP TABLE IF EXISTS EventActivity";
    String createEventActivityTable = "CREATE TABLE EventActivity ("
            + "ID INT AUTO_INCREMENT PRIMARY KEY,"
            + "ActivityName VARCHAR(20)"
            + ")";
    String insertEventActivityValues = "INSERT INTO EventActivity (ActivityName) VALUES "
            + "('Design'), "
            + "('Construction'), "
            + "('Test'), "
            + "('PASSWORD Reset')";

    // Drop EventOrigin table if it exists
    String dropEventOriginTable = "DROP TABLE IF EXISTS EventOrigin";
    String createEventOriginTable = "CREATE TABLE EventOrigin ("
            + "ID INT AUTO_INCREMENT PRIMARY KEY,"
            + "Origin VARCHAR(20)"
            + ")";
    String insertEventOriginValues = "INSERT INTO EventOrigin (Origin) VALUES "
            + "('Joe S.'), "
            + "('Bill B.'), "
            + "('George E.'), "
            + "('Achmed M.'), "
            + "('Rona E.')";

    // Drop EventStatus table if it exists
    String dropEventStatusTable = "DROP TABLE IF EXISTS EventStatus";
    String createEventStatusTable = "CREATE TABLE EventStatus ("
            + "ID INT AUTO_INCREMENT PRIMARY KEY,"
            + "Status VARCHAR(20)"
            + ")";
    String insertEventStatusValues = "INSERT INTO EventStatus (Status) VALUES "
            + "('Open'), "
            + "('On Hold'), "
            + "('In Process'), "
            + "('Deployed'), "
            + "('Deployed Failed')";

    // Drop EventClass table if it exists
    String dropEventClassTable = "DROP TABLE IF EXISTS EventClass";
    String createEventClassTable = "CREATE TABLE EventClass ("
            + "ID INT AUTO_INCREMENT PRIMARY KEY,"
            + "Class VARCHAR(20)"
            + ")";
    String insertEventClassValues = "INSERT INTO EventClass (Class) VALUES "
            + "('Change'), "
            + "('Incident'), "
            + "('Problem'), "
            + "('SR')";

    // Execute SQL statements to drop and create the tables
    try (Statement statement = connection.createStatement()) {
        statement.executeUpdate(dropEventActivityTable);
        statement.executeUpdate(createEventActivityTable);
        statement.executeUpdate(insertEventActivityValues);
        statement.executeUpdate(dropEventOriginTable);
        statement.executeUpdate(createEventOriginTable);
        statement.executeUpdate(insertEventOriginValues);
        statement.executeUpdate(dropEventStatusTable);
        statement.executeUpdate(createEventStatusTable);
        statement.executeUpdate(insertEventStatusValues);
        statement.executeUpdate(dropEventClassTable);
        statement.executeUpdate(createEventClassTable);
        statement.executeUpdate(insertEventClassValues);
    }
}
```



```
private static void dropAndCreateEventLogTable(Connection connection) throws SQLException {
    // Drop EventLog table if it exists
    String dropTableSQL = "DROP TABLE IF EXISTS EventLog";
    PreparedStatement dropStatement = connection.prepareStatement(dropTableSQL);
    dropStatement.execute();
    dropStatement.close();

    // Create EventLog table
    String createTableSQL = "CREATE TABLE EventLog ("
            + "ID INT AUTO_INCREMENT PRIMARY KEY,"
            + "CaseID VARCHAR(20) UNIQUE,"
            + "Activity VARCHAR(20),"
            + "Urgency VARCHAR(1),"
            + "Impact VARCHAR(1),"
            + "Priority VARCHAR(1),"
            + "StartDate DATE,"
            + "EndDate DATE,"
            + "TicketStatus VARCHAR(20),"
            + "UpdateDateTime DATETIME,"
            + "Duration INT,"
            + "Origin VARCHAR(20),"
            + "Class VARCHAR(20)"
            + ")";
    PreparedStatement createStatement = connection.prepareStatement(createTableSQL);
    createStatement.execute();
    createStatement.close();
}
```

Sample console output for the user input section.



```
<terminated> TicketGenerator [Java Application] /Users/redge/.p2/pool/plugins/org.ecli
Number of tickets to generate: 2000
Time window start date (yyyy-MM-dd): 2023-05-01
Time window end date (yyyy-MM-dd): 2023-04-30
End date must be later than or equal to the start date.
Time window end date (yyyy-MM-dd): 2023-05-02
Number of tickets: 2000
Start Date: 2023-05-01
End Date: 2023-05-02
Dropped a database named "tickets"  ...

Created a database named "tickets" ...
```

Dashboard Description

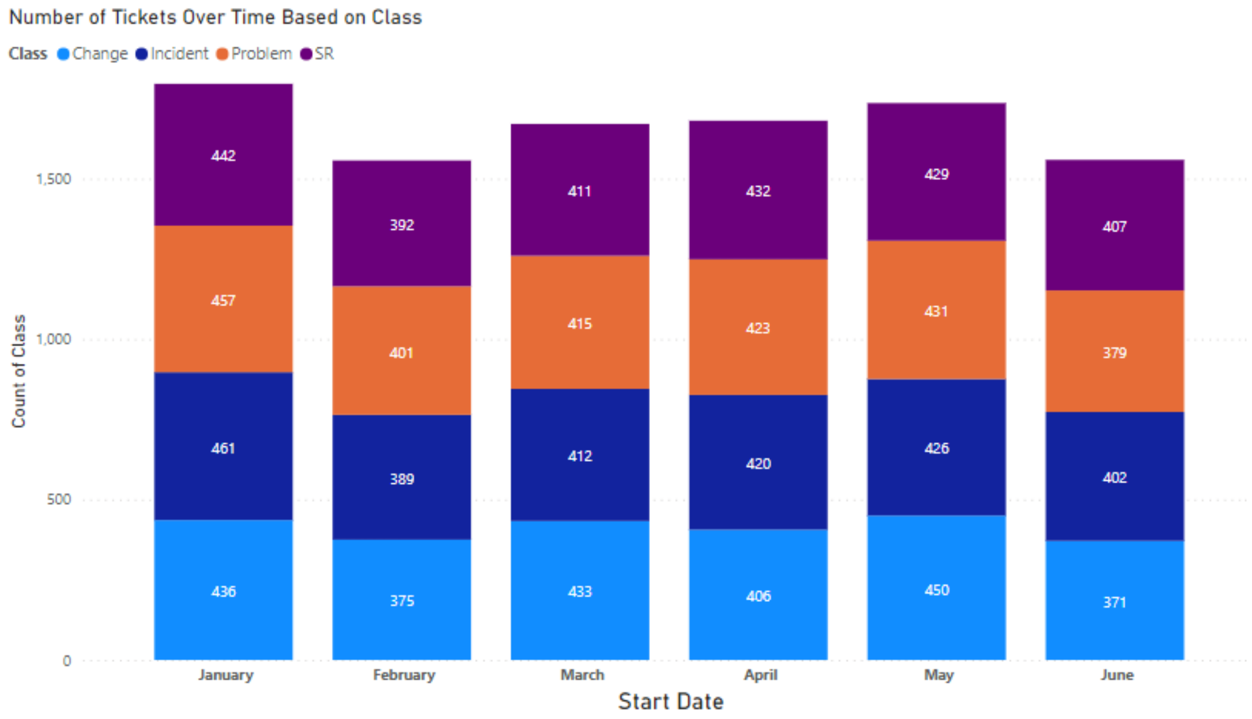
After creating the database, we used Power BI to visualize the results.

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.

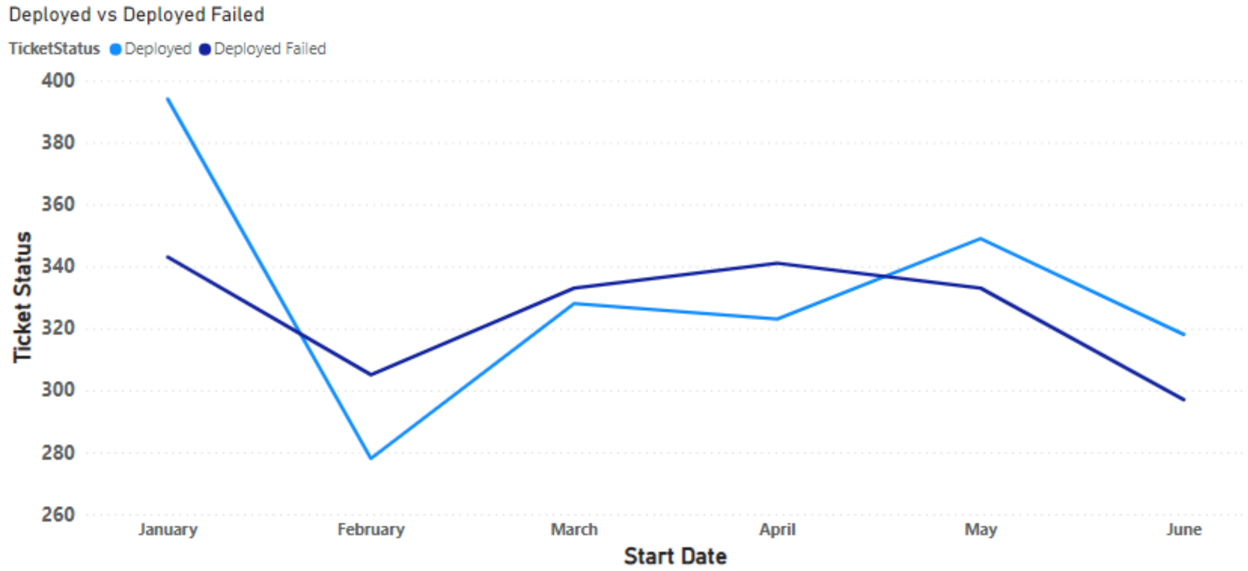
Overall, the ticket distribution is balanced across classes. Change tickets show a declining trend, while SRs peaked in April. Incidents and Problems remain consistent.



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.

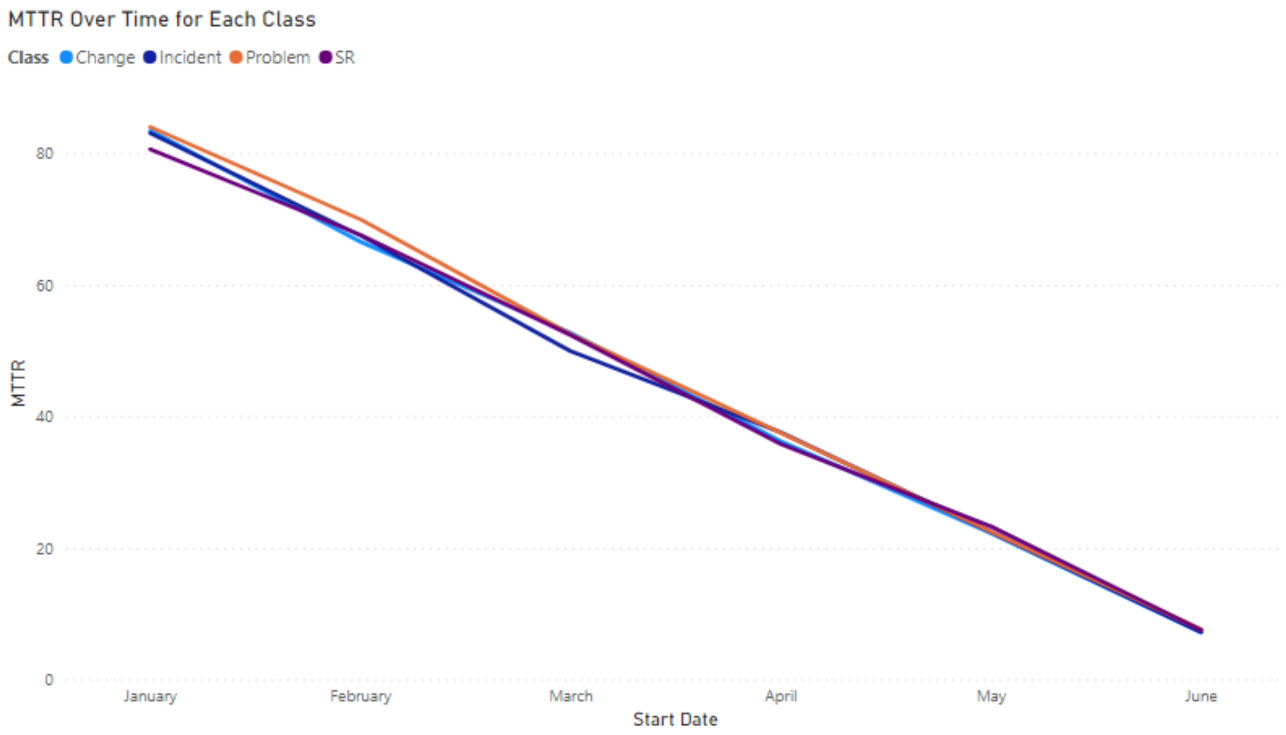


The graph below depicts the "Mean Time to Resolve (MTTR)" for the four ticket classes from January to June.

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.

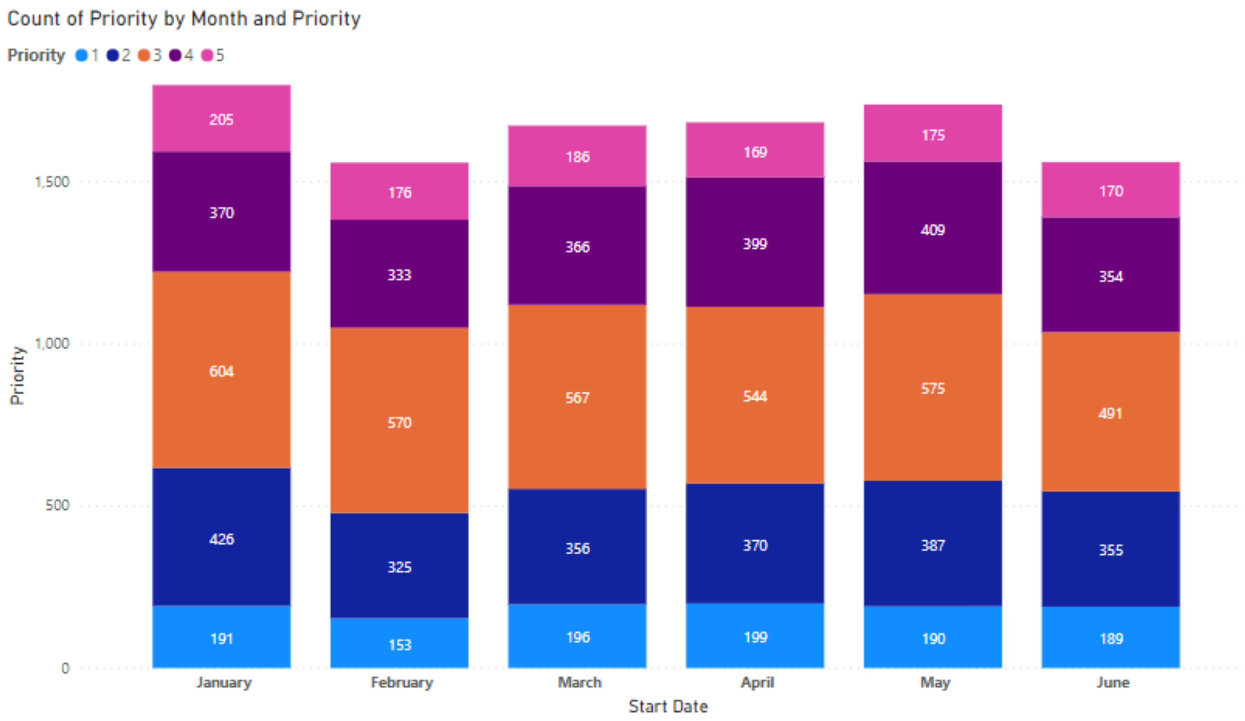


We decided to explore the count of priority by month from January to June.

Some observations about the plot:

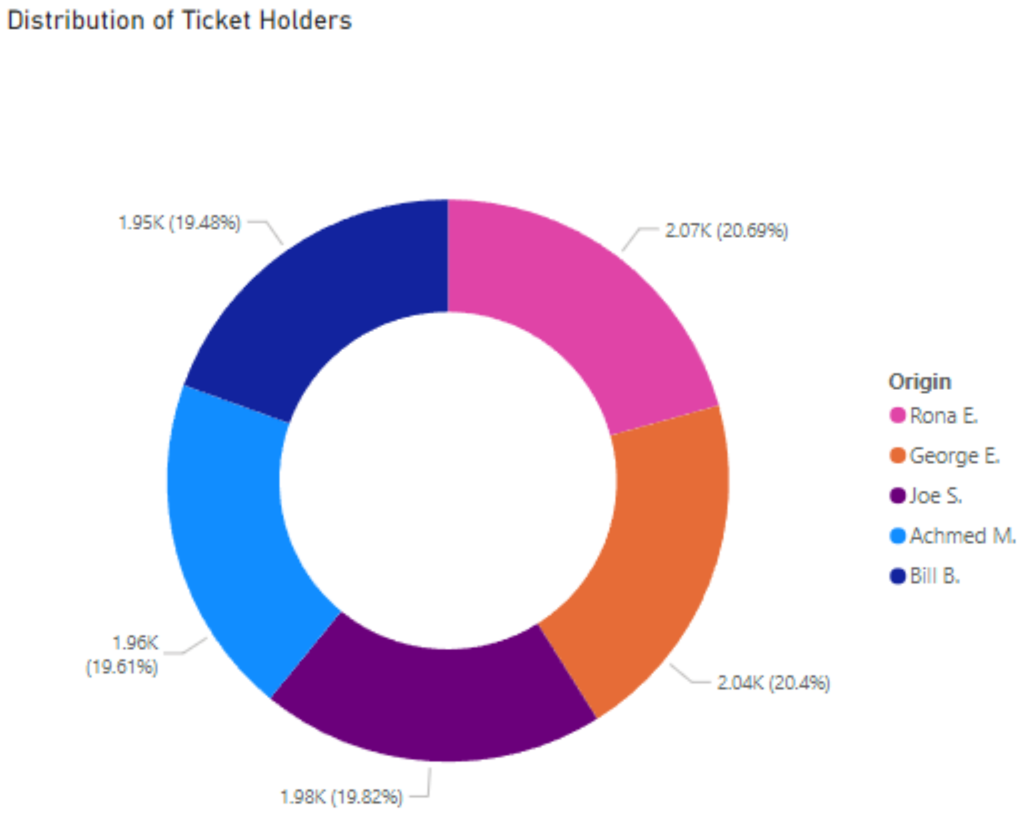
- Priority 1 (Blue) tickets range from around 190 in January to 189 in June, remaining fairly stable.
- Priority 2 (Orange) tickets see an increase from 426 in January to a peak of 575 in May, then a slight drop in June.
- Priority 3 (Purple) tickets are relatively steady, hovering around 550-600 tickets.
- 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.


Distribution of Ticket Holders
1.95K (19.48%)
2.07K (20.69%)
Origin
● Rona E.
● George E.
● Joe S.
● Achmed M.
● Bill B.
1.96K (19.61%)
2.04K (20.4%)
1.98K (19.82%)