Group 1

Members
1.Adrian Osogo
2.Terrence Katua
3.Emmanuel Kanyiri
4.Innocent Ndeti
5.James Njao

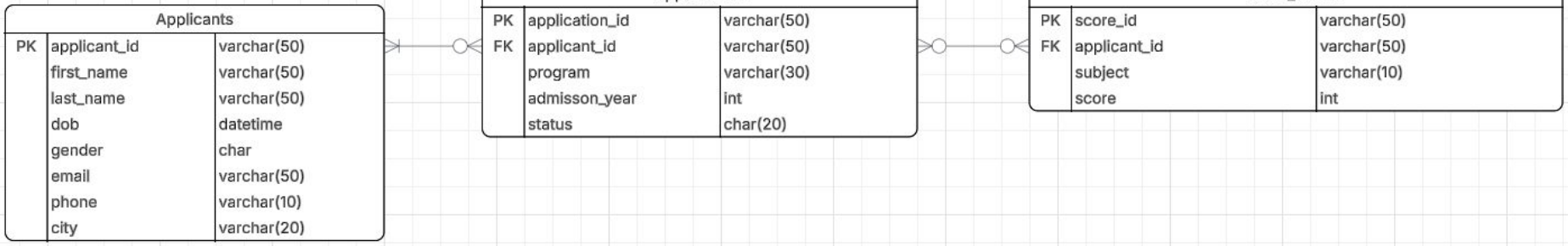
Project Goal:

- Develop a java desktop application (Swing) that imports CSV admissions data into a normalized relational DB, performs analytics and displays results with visualizations.

- The university application analyzes university admission data from 3 csv files:
- applicants.csv
- applications.csv
- exam_scores.csv

Table of Contents

Normalised Data Structure



Code for Creation of tables and columns

1. Applicants table.

- This table keeps all the details about each applicant i.e name, birth date, gender, email, phone, and city.
- Every applicant gets a unique applicant ID so the system can identify them easily when linking to other tables like applications or exam scores.

```
CREATE TABLE applicants (
  applicant_id varchar(50) NOT NULL,
  first_name varchar(50) DEFAULT NULL,
  last_name varchar(50) DEFAULT NULL,
  dob datetime DEFAULT NULL,
  gender char(1) DEFAULT NULL,
  email varchar(50) DEFAULT NULL,
  phone varchar(10) DEFAULT NULL,
  city varchar(20) DEFAULT NULL,
  PRIMARY KEY (applicant_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

2. Applications Table

- This table stores every program an applicant has applied for.
- It links each application to the applicant's ID, the program name, the year of admission, and whether they were accepted or not.
- The foreign key connection keeps all records properly linked back to the main applicants table.

```
CREATE TABLE applications (
  application_id VARCHAR(50) NOT NULL,
  applicant_id VARCHAR(50) DEFAULT NULL,
  program VARCHAR(30) DEFAULT NULL,
  admisson_year INT DEFAULT NULL,
  status CHAR(20) DEFAULT NULL,
  PRIMARY KEY (application_id),
  KEY applicant_id (applicant_id),
  CONSTRAINT applications_ibfk_1
      FOREIGN KEY (applicant_id) REFERENCES applicants (applicant_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

3. Exam Scores table

- The `exam_scores` table records applicants' performance in various subjects.
- Each entry is uniquely identified by a **`score_id`** primary key, with a **foreign key (`applicant_id`)** linking each score to its corresponding record in the `applicants` table.
- The table captures subject name and the numeric score, facilitating computation of average scores and comparative analysis across programs.

```
CREATE TABLE exam_scores (
  score_id varchar(50) NOT NULL,
  applicant_id varchar(50) DEFAULT NULL,
  subject varchar(10) DEFAULT NULL,
  score int DEFAULT NULL,
  PRIMARY KEY (score_id),
  KEY applicant_id (applicant_id),
  CONSTRAINT exam_scores_ibfk_1 FOREIGN KEY (applicant_id) REFERENCES applicants
(applicant_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

B. JDBC Code

- The University's Admission system uses **JDBC (Java Database Connectivity)** to establish a secure and dynamic link between the Java application and the database.
- The system loads the appropriate database driver using `Class.forName()` and establishes a connection through `DriverManager.getConnection()`.
- If something goes wrong — like a missing driver or wrong password — the system shows a clear error message instead of crashing through exception handling mechanisms such as `catch (ClassNotFoundException cnfe)`

```
try {
            Class.forName(driverClass);
            conn = DriverManager.getConnection(url, user, password);
        } catch (ClassNotFoundException cnfe) {
            JOptionPane.showMessageDialog(null, "JDBC Driver not found: " +
driverClass + "\nPlease add the JDBC driver JAR to the classpath.", "Driver Error",
JOptionPane.ERROR_MESSAGE);
            return;
        } catch (SQLException sqle) {
            JOptionPane.showMessageDialog(null, "Failed to connect: " +
sqle.getMessage(), "Connection Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
```

Code for running queries

1. Acceptance rates per program

- The code below finds out what percentage of applicants were accepted into each program. It runs a database query and returns results so they can be displayed in a bar chart.

```
String query = "SELECT b.program, " +
                "COUNT(CASE WHEN b.status='Accepted' THEN 1
END)*100.0/COUNT(b.application_id) AS acceptanceRate " +
                "FROM applicants AS a LEFT JOIN applications AS b
ON a.applicant_id=b.applicant_id " +
                "GROUP BY b.program";
        ResultSet rs = stmt.executeQuery(query);
```

Program	Accepted	Total	AcceptanceRate(%)
CS	8	19	42.11
Engineering	6	13	46.15
Medicine	14	22	63.64
Business	6	16	37.50

2. Average exam score per program

- The following query computes the mean examination score for each program by joining the exam_score and applications tables. The result set enables comparative performance analysis across programs.

```
ResultSet rs = stmt.executeQuery(
                "SELECT b.program, AVG(score) as avg_score " +
                    "FROM exam_scores e " +
                    "JOIN applications b ON e.applicant_id =
b.applicant_id " +
                    "GROUP BY b.program"
            );
```

Program	AverageScore
CS	66.00
Engineering	70.63
Medicine	70.42
Business	72.26

3. Distribution of applicants by city and gender

- The query groups applicant records by both city and gender to illustrate demographic distribution. Executed through JDBC, the results populate a Swing table model to visualize regional and gender-based applicant trends.

```
    try (Statement stmt = connection.createStatement()) {
            ResultSet rs = stmt.executeQuery(
                    "SELECT gender, COUNT(*) as count FROM
applicants GROUP BY gender"
            );
            while (rs.next()) {
                dataset.setValue(rs.getString("gender"),
rs.getInt("count"));
            }
        }
```

City	Gender	Count
Eldoret	F	5
Eldoret	M	2
Kisumu	F	11
Kisumu	M	7
Mombasa	F	5
Mombasa	M	6
Nairobi	F	7
Nairobi	M	7

4. Top 10 applicants by average exam score

- This query ranks applicants based on their mean examination scores and returns the top ten performers.

```
try (Statement stmt = connection.createStatement()) {
    ResultSet rs = stmt.executeQuery(
            "SELECT b.program, AVG(score) as avg_score " +
                "FROM exam_scores e " +
                "JOIN applications b ON e.applicant_id =
b.applicant_id " +
                "GROUP BY b.program"
        );
```

Top 10 Applicants by Average Exam Score:

John Okafor: 100.00
Kim Ndegwa: 90.00
Brian Khan: 89.00
Paul Ruiz: 89.00
James Karanja: 88.00
Sam Otieno: 83.50
Kim Mwangi: 82.50
Paul Doe: 81.25
Mark Okafor: 78.00
Jane Doe: 77.67

Code for Charts and Outputs

1. BAR CHART (Exam Scores)

```
private JPanel createExamScoresHistogram() throws SQLException {
JPanel panel = new JPanel(new BorderLayout());
List<Double> scores = new ArrayList<>();

try (Statement stmt = connection.createStatement()) {
ResultSet rs = stmt.executeQuery("SELECT score FROM exam_scores");
while (rs.next()) {
scores.add(rs.getDouble("score"));
}
}
```

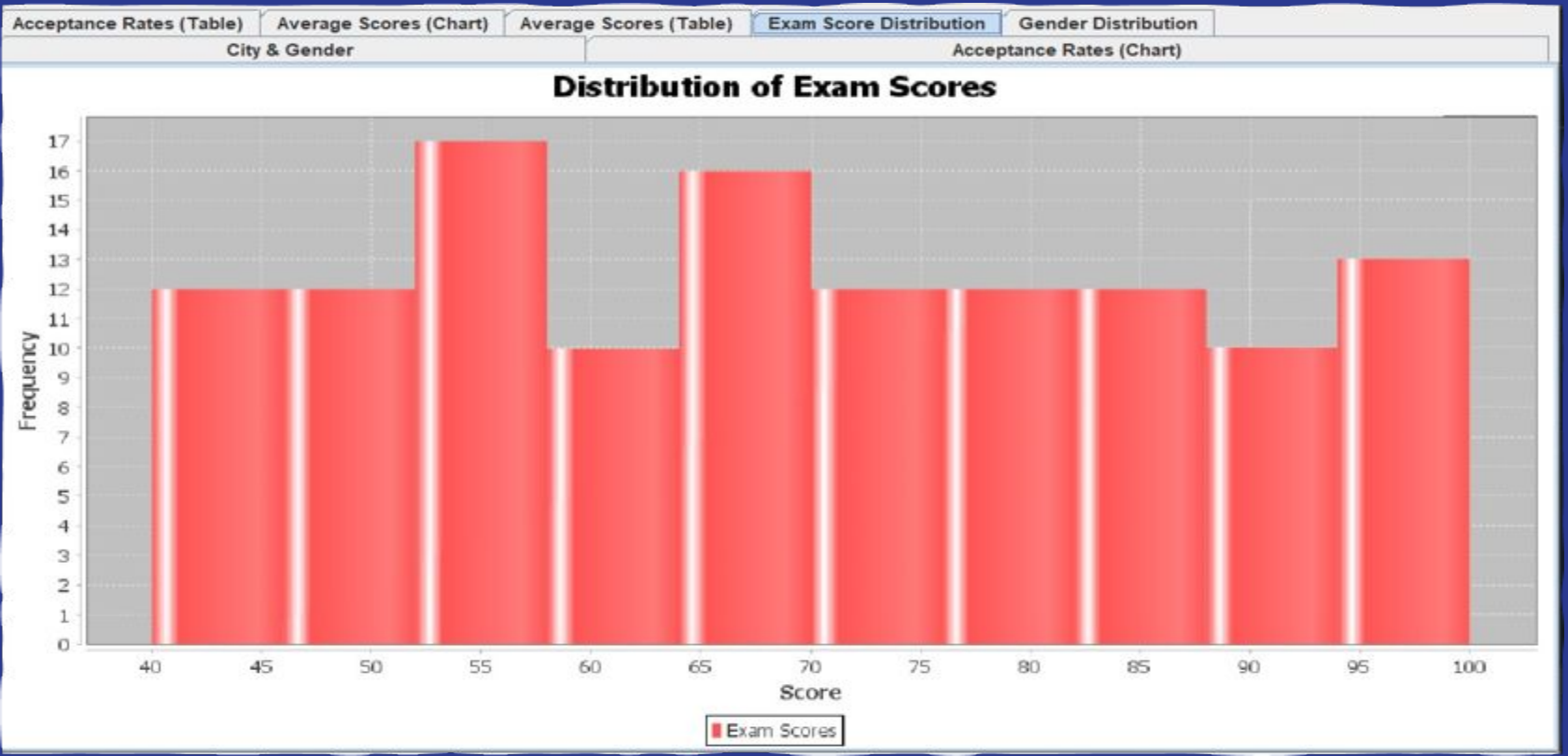
```
double[] scoreArray = scores.stream().mapToDouble(d -> d).toArray();
HistogramDataset dataset = new HistogramDataset();
dataset.addSeries("Scores", scoreArray, 20);

JFreeChart chart = ChartFactory.createHistogram(
"Distribution of Exam Scores",
"Score",
"Frequency",
dataset,
PlotOrientation.VERTICAL,
true, true, false
);

ChartPanel chartPanel = new ChartPanel(chart);
panel.add(chartPanel);
return panel;
}
```

Bar Chart

Output



2. Histogram (Acceptance rate)

Code:

```
private JPanel createAcceptanceRatesChart() throws SQLException {
JPanel panel = new JPanel(new BorderLayout());
DefaultCategoryDataset dataset = new DefaultCategoryDataset();

try (Statement stmt = connection.createStatement()) {
String query = "SELECT b.program, " +
"COUNT(CASE WHEN b.status='Accepted' THEN 1 END)*100.0/COUNT(b.application_id) AS acceptanceRate " +
"FROM applicants AS a LEFT JOIN applications AS b ON a.applicant_id=b.applicant_id " +
"GROUP BY b.program";
ResultSet rs = stmt.executeQuery(query);
```

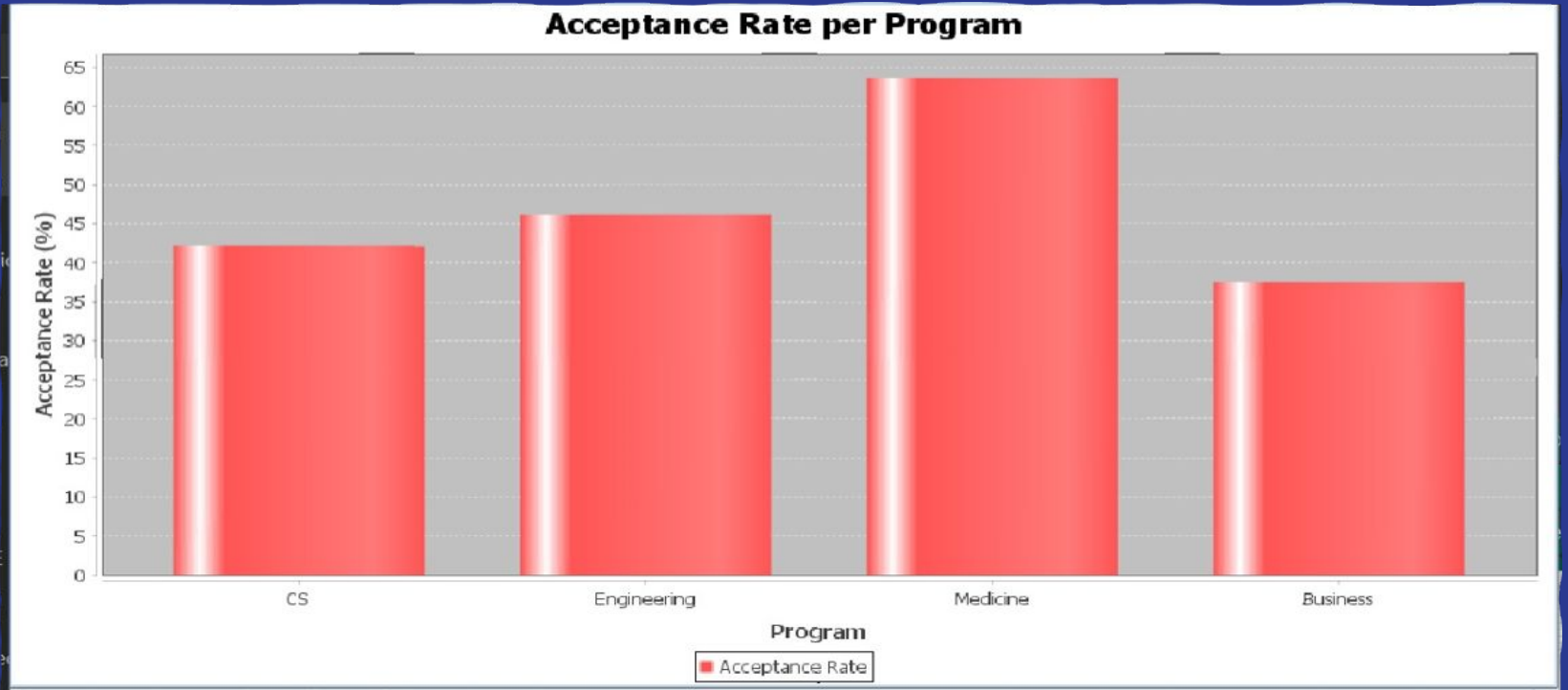
```
while (rs.next()) {
dataset.addValue(rs.getDouble("acceptanceRate"),
"Acceptance Rate",
rs.getString("program"));
}
}

JFreeChart chart = ChartFactory.createBarChart(
"Acceptance Rate per Program",
"Program",
"Acceptance Rate (%)",
dataset,
PlotOrientation.VERTICAL,
true, true, false
);

ChartPanel chartPanel = new ChartPanel(chart);
panel.add(chartPanel);
return panel;
}
```

Histogram

Output



CS	8	19	42.11
Engineering	6	13	46.15
Medicine	14	22	63.64

3. Pie chart (Gender Distribution)

Code:

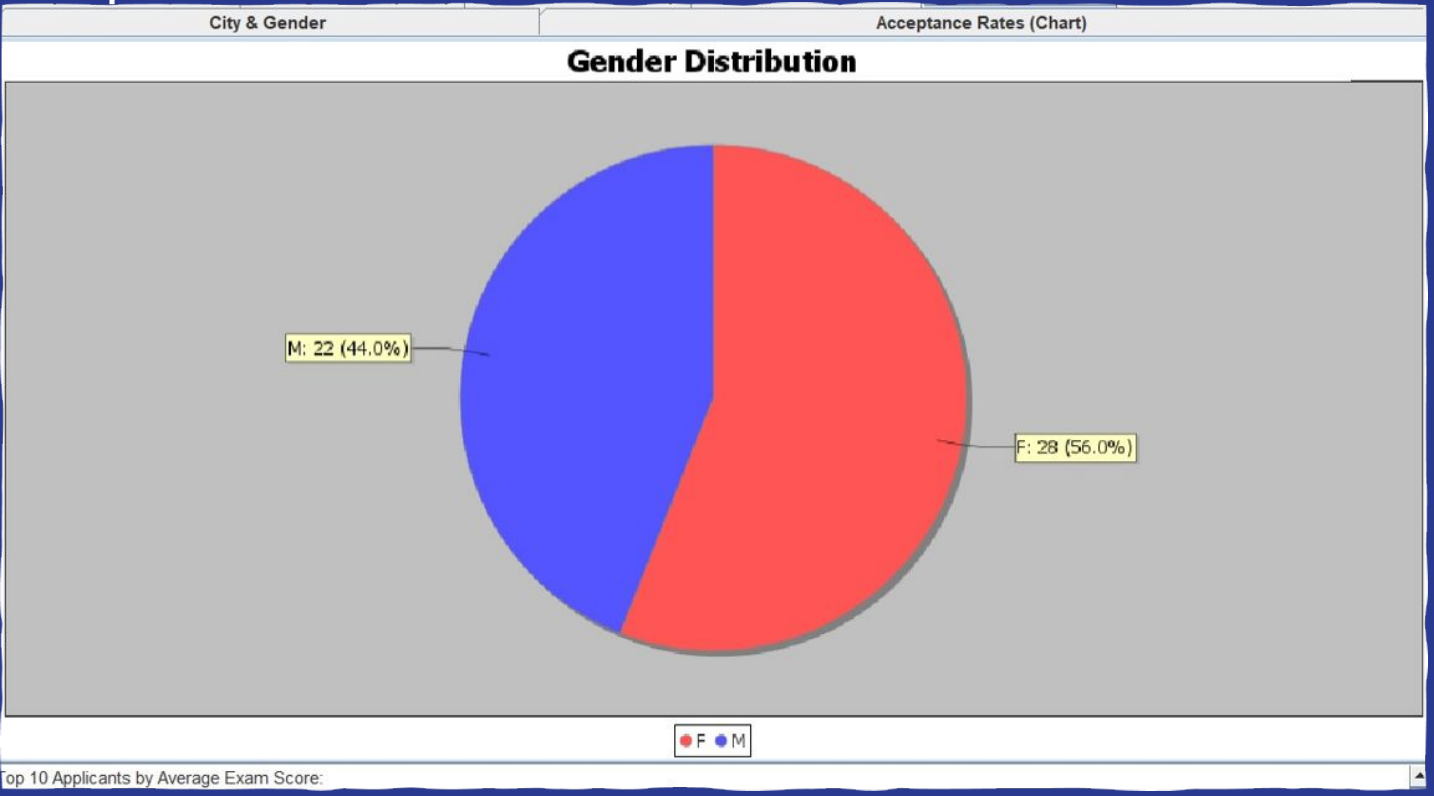
```
private JPanel createGenderDistributionChart() throws SQLException {
JPanel panel = new JPanel(new BorderLayout());
DefaultPieDataset dataset = new DefaultPieDataset();

try (Statement stmt = connection.createStatement()) {
ResultSet rs = stmt.executeQuery(
"SELECT gender, COUNT(*) as count FROM applicants GROUP BY
gender"
);
while (rs.next()) {
dataset.setValue(rs.getString("gender"), rs.getInt("count"));
}
```

```
}
JFreeChart chart = ChartFactory.createPieChart(
"Gender Distribution",
dataset,
true, true, false
);
PiePlot plot = (PiePlot) chart.getPlot();
plot.setLabelGenerator(new StandardPieSectionLabelGenerator(
"{0}: {1} ({2})",
new DecimalFormat("0"),
new DecimalFormat("0.0%")
));
ChartPanel chartPanel = new ChartPanel(chart);
panel.add(chartPanel);
return panel;
}
```

Pie chart

Output



Summary

a. Findings
b. Insights

Challenges faced