ABSTRACT:

The Fee Management System is a robust solution designed to streamline the complex process of managing student fees within educational institutions. With a focus on efficiency and transparency, this system offers administrators and staff a comprehensive set of features to handle fee-related tasks seamlessly. One of its key functionalities includes the ability to add, edit, or update fees for students enrolled in specific courses, ensuring accuracy and flexibility in fee management. Administrators can effortlessly navigate through the system interface to input fee details, assign courses, and track payment statuses.

Moreover, the system facilitates course management by allowing administrators to add, edit, or update course details as needed. This feature enables educational institutions to adapt to changing academic requirements and offerings while maintaining an up-to-date course catalog. Additionally, the system provides a convenient way to generate fee receipts for students upon successful payment, enhancing the overall user experience and providing a tangible record of transactions.

Furthermore, the Fee Management System offers robust reporting capabilities, empowering administrators to gain insights into fee collections and financial performance. Through customizable reporting features, administrators can generate detailed reports on fee collection trends, outstanding payments, and revenue projections. These reports serve as valuable tools for decision-making and strategic planning within educational institutions, facilitating informed financial management and resource allocation.

In summary, the Fee Management System is a comprehensive solution designed to streamline fee-related processes, enhance administrative efficiency, and promote transparency within educational institutions. By offering a range of features such as fee management, course administration, receipt generation, and reporting, this system empowers administrators to effectively manage fee-related tasks and optimize financial operations for the benefit of students and staff alike.

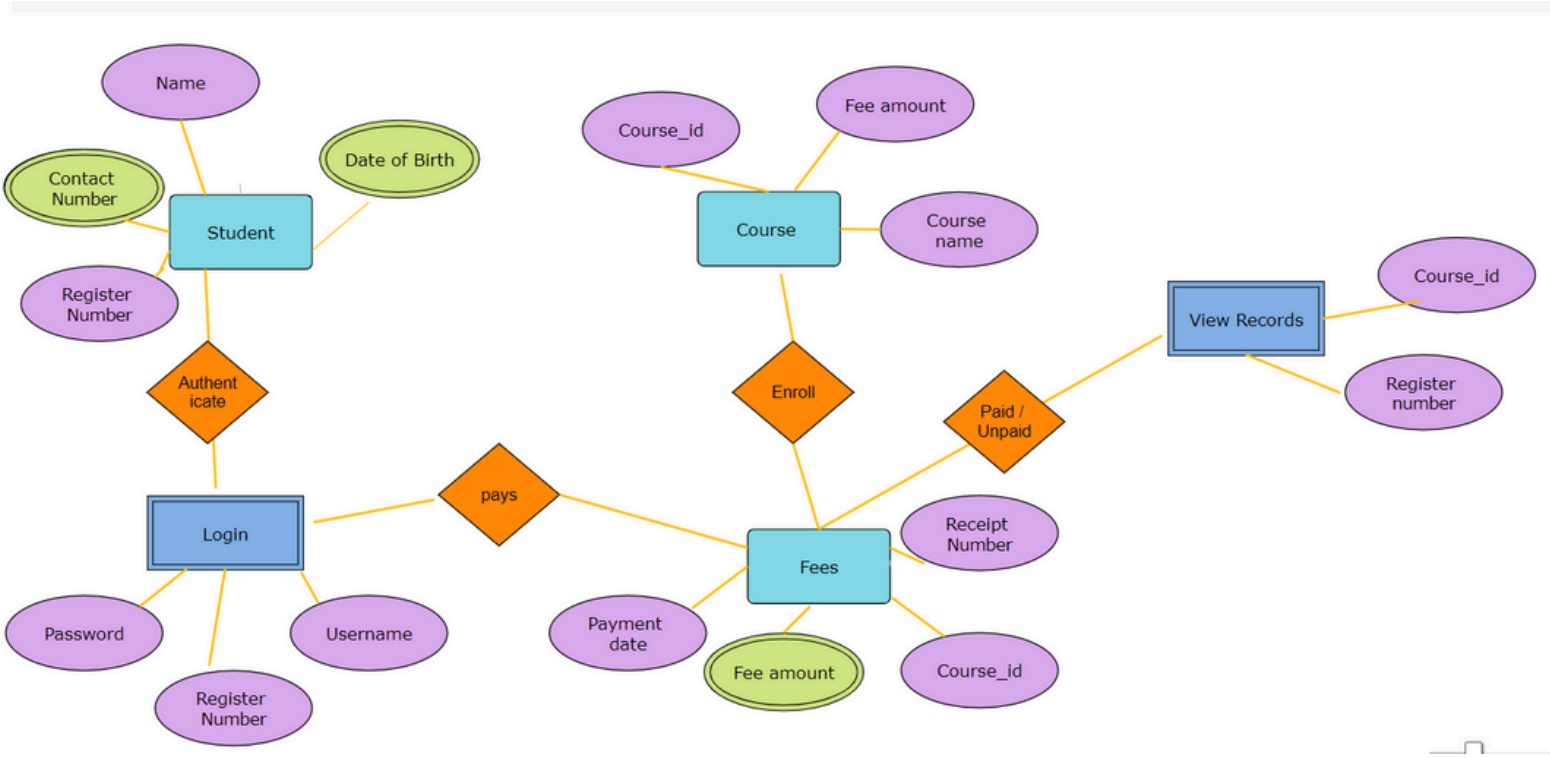
PROBLEM UNDERSTANDING :

The Fee Management System simplifies fee administration within educational institutions. It enables seamless management of student fees for specific courses, with options to add, edit, or update fee details. Course management functionalities allow easy modification of course information to adapt to changing academic needs. The system automates fee receipt generation upon payment, ensuring a smooth transaction process. Additionally, robust reporting features provide administrators with valuable insights into fee collections and financial performance, facilitating informed decision-making. In summary, the system streamlines fee-related tasks, enhances administrative efficiency, and promotes transparency in financial operations within educational institutions.

ENTITY AND RELATIONSHIP DIAGRAM (ER DIAGRAM) :

ENTITIES AND ATTRIBUTES :

Student - Name , Date of Birth , Register Number , Contact Number
Course - Course _Id , Fee amount , Course Name
View Records (Weak Entity) - Course_Id , Register Number
Fees - Receipt Number , Course_Id , Fee amount , Payment Date
Login (Weak Entity) - Username , Password , Register Number



Databases Used :

Course - ID (Primary Key) , Cname, Cost .
Fees_Details - Receipt_No (Primary Key) , Student_Name ,Roll_No, Payment_Mode, Cheque_No , Bank_Name, DD_No, Course_Name , GSTIN , Total_Amount , Date , Amount , CGST, SGST , Total_in_words , Remark , Year1 , Year2.
Signup - Id(Primary Key) , Firstname , Lastname , Username, Password , DOB , Contact_No.

RELATIONAL SCHEMA AND CREATION OF DATABASE :

Creation of Database Course :

```
CREATE TABLE Course (
Course_ID INT PRIMARY KEY,
Cname VARCHAR(255),
Cost DECIMAL(10, 2));
```

Creation of Database Fees_Details :

```
CREATE TABLE Fees_Details (
Receipt_No INT PRIMARY KEY,
Student_Name VARCHAR(255)
Roll_No VARCHAR(20),
Payment_Mode VARCHAR(20),
Cheque_No VARCHAR(20),
Bank_Name VARCHAR(50),
DD_No VARCHAR(20),
Course_Name VARCHAR(255),
GSTIN VARCHAR(20),
Total_Amount DECIMAL(10, 2),
Date DATE,
Amount DECIMAL(10, 2),
CGST DECIMAL(10, 2),
SGST DECIMAL(10, 2),
Total_in_words VARCHAR(255),
Remark TEXT,
Year1 INT,
Year2 INT,
FOREIGN KEY (Course_Name) REFERENCES
Course(Cname));
```

Creation of Database Signup :

```
CREATE TABLE Signup (
Id INT PRIMARY KEY,
Firstname VARCHAR(50),
Lastname VARCHAR(50),
Username VARCHAR(50),
Password VARCHAR(50),
DOB DATE,
Contact_No VARCHAR(15));
```

CONSTRAINTS , SETS AND JOINS

Constraints:

Constraints are rules that enforce data integrity and define restrictions on the data stored in database tables. They ensure that data values meet certain conditions or requirements, helping to maintain consistency and accuracy in the database. Common types of constraints include:

- **Primary Key Constraint:** Ensures that each row in a table is uniquely identified by a primary key column or combination of columns.
- **Foreign Key Constraint:** Enforces referential integrity by requiring that values in a column (or columns) of one table match the values in a primary key column (or columns) of another table.
- **Unique Constraint:** Ensures that values in a column (or columns) are unique across all rows in the table.
- **Check Constraint:** Specifies a condition that must be satisfied for each row in the table.
- **Not Null Constraint:** Prevents null values from being inserted into a column.

Sets:

Sets in SQL refer to the result sets returned by SQL queries. They represent collections of rows that satisfy the specified conditions or criteria. SQL provides several set operations to manipulate and combine result sets, including:

- **Union:** Combines the results of two or more SELECT queries into a single result set, eliminating duplicate rows.
- **Intersect:** Returns only the rows that are common to two or more SELECT queries.
- **Except (or Minus):** Returns the rows that are present in the first SELECT query but not in the second SELECT query.

Set operations are useful for performing complex data analysis, combining data from multiple tables, and identifying common or unique records between result sets.

Joins:

Joins in SQL are used to retrieve data from multiple tables based on a related column between them. There are several types of joins:

- **Inner Join:** Returns only the rows that have matching values in both tables.
- **Left Join (or Left Outer Join):** Returns all rows from the left table and the matched rows from the right table. If there is no match, NULL values are returned for the columns from the right table.
- **Right Join (or Right Outer Join):** Returns all rows from the right table and the matched rows from the left table. If there is no match, NULL values are returned for the columns from the left table.
- **Full Join (or Full Outer Join):** Returns all rows from both tables, matching rows from both tables where available and NULL values where there is no match.

Joins allow you to retrieve related data from multiple tables in a single query, enabling complex data analysis and reporting.

APPLYING NORMALISATION

If we apply normalization techniques to a project, particularly a database project, several things will happen, each with its own implications and benefits:

1. Improved Data Structure: Normalization involves organizing data into multiple related tables to reduce redundancy and dependency. This leads to a more structured and organized database schema, which improves data integrity and reduces the risk of anomalies such as update anomalies, insertion anomalies, and deletion anomalies.

2. Data Integrity: Normalization helps maintain data integrity by reducing or eliminating data redundancy. By storing data in separate tables and linking them using relationships, you ensure that each piece of data is stored in only one place. This minimizes the chances of inconsistent or contradictory information being stored in the database.

3. Efficient Storage: Normalization can lead to more efficient storage of data. By removing redundant data and optimizing the database schema, you can reduce the amount of disk space required to store the data. This can result in cost savings, particularly for large databases.

4. Improved Query Performance: In some cases, normalization can improve query performance by reducing the amount of data that needs to be accessed and processed. By breaking down data into smaller, more manageable tables, you can often write more efficient queries that return results more quickly.

5. Flexibility and Scalability: A normalized database schema is typically more flexible and scalable than a denormalized schema. As your application evolves and your data requirements change, it's easier to modify and extend a normalized schema without introducing inconsistencies or performance issues.

6. Maintenance and Updates: Normalization can make it easier to maintain and update your database schema over time. Because each piece of data is stored in only one place, you only need to update it in one location. This reduces the risk of errors and makes it easier to enforce data consistency.

7. Normalization Overhead: While normalization offers many benefits, it can also introduce some overhead. Maintaining relationships between tables and joining them in queries can add complexity to your database design and queries. In some cases, denormalization may be necessary to improve query performance or simplify certain types of queries.

In summary, applying normalization techniques to a project can lead to a more structured, efficient, and maintainable database schema, with improved data integrity and query performance. However, it's important to strike the right balance between normalization and denormalization based on your specific requirements and performance considerations.

CONCURRENY CONTROL AND RECOVERY MECHANISM

Concurrency control and recovery mechanisms are essential components of database management systems (DBMS) to ensure data consistency, integrity, and reliability, especially in multi-user environments.

1. Concurrency Control:

Concurrency control refers to the management of simultaneous access to shared data by multiple transactions in a database system. It ensures that the execution of transactions does not result in data inconsistencies or conflicts. There are several concurrency control techniques:

- **Locking:** Transactions acquire locks on data items to prevent concurrent access by other transactions. Locks can be exclusive (write lock) or shared (read lock). Different locking protocols, such as two-phase locking (2PL) and deadlock detection, are used to manage locks effectively.

- **Timestamp-based Protocols:** Transactions are assigned timestamps based on their start times. Techniques like timestamp ordering and optimistic concurrency control use these timestamps to determine the order of transaction execution and detect conflicts.

-**Multiversion Concurrency Control (MVCC):** Maintains multiple versions of data items to allow concurrent transactions to read consistent snapshots of data. It reduces contention by allowing reads and writes to proceed concurrently.

- **Serializability:** Ensures that the execution of concurrent transactions produces the same results as if they were executed serially. Techniques like strict two-phase locking and Serializable Snapshot Isolation (SSI) guarantee serializability.

2. Recovery Mechanism:

Recovery mechanisms ensure the durability of data and the system's ability to recover from failures, such as system crashes or disk failures. The primary goal is to bring the database back to a consistent state after a failure. Key components of recovery mechanisms include:

- **Write-Ahead Logging (WAL)**: Log records are written to a persistent log before modifying corresponding data in the database. This ensures that changes are durable and can be replayed in case of a crash.

- **Checkpointing:** Periodic checkpoints are taken to create consistent snapshots of the database. Checkpoint records indicate the state of transactions at the time of the checkpoint, allowing for efficient recovery.

- **Transaction Undo and Redo:** During recovery, transactions are undone or redone based on the contents of the log. The undo phase reverses changes made by incomplete transactions, while the redo phase re-applies committed transactions' changes to restore the database to a consistent state.

Concurrency control and recovery mechanisms work together to maintain data consistency and system reliability in database management systems, ensuring that transactions are executed correctly and the database remains resilient to failures.

MODULE DESCRIPTION AND FUNCTIONALITIES

1. LOGIN+SIGNUP:

- This module facilitates user authentication and registration.
- **Login:** Existing users can log in using their username and password.
- **Signup:** New users can create an account by providing their details such as first name, last name, username, password, date of birth, and contact number.
- Upon successful login, users gain access to other modules of the system.

2. ADD FEES:

- This module enables administrators to add fees for students.
- **Add Fees Form**: Administrators can fill in details such as student name, roll number, payment mode, course name, total amount, payment date, GST details, and any remarks.
- Upon submission, the fee details are stored in the database, and a receipt is generated for the transaction.

3. SEARCH RECORD:

- This module allows users to search for specific fee records based on various criteria.
- **Search Form:** Users can enter search parameters such as student name, roll number, course name, payment date range, etc.
- The system retrieves matching fee records from the database and displays them to the user.

4. VIEW RECORD:

- This module enables users to view all fee records stored in the system.
- **Record Listing:** All fee records are displayed in a tabular format, showing details such as student name, roll number, course name, payment amount, payment date, etc.
- Users can scroll through the records or apply filters to narrow down the list based on specific criteria.

5. EDIT COURSE:

- This module allows administrators to edit course details such as course name and cost.
- **Edit Course Form:** Administrators can select a course from the list and modify its attributes.
- Upon submission, the changes are updated in the database, ensuring that the course details are up to date.

6. VIEW REPORT:

- This module generates reports on fee collection and financial performance.
- **Report Generation:** Users can generate reports for a specific time period, course, or any other relevant parameter.
- The system calculates total fee collection, outstanding payments, taxes collected, etc., and presents the information in a structured format.
- Reports provide valuable insights for decision-making and financial analysis within the institution.

These modules collectively form a comprehensive fee management system, providing functionalities for user authentication, fee management, record search and viewing, course editing, and report generation, thereby facilitating efficient administration of fee-related processes within educational institutions.

DATABASE CONNECTIVITY

The Java method `insertDetails()` is responsible for inserting new user details into a database table named "signup". Here's an explanation of the content and functionality of this method:

1. SimpleDateFormat: This class is used to format the date of birth (`dob`) into a string in the format "yyyy-MM-dd". This formatted date (`myDob`) is then used to insert into the database.

2. Class.forName(): This statement loads the JDBC driver class for the Derby database, which is used to establish a connection to the database.

3. Connection: The `getConnection()` method establishes a connection to the Derby database named "feesmanagement" running on the localhost with the port number 1527. The username and password for the database are provided as "root".

4. PreparedStatement: A prepared statement is created with an SQL query to insert values into the "signup" table. Placeholders (?) are used for parameterized values to prevent SQL injection attacks.

5. Setting Parameters: The values for each parameterized placeholder in the prepared statement are set using methods like `setInt()` and `setString()`.

6. executeUpdate(): The `executeUpdate()` method is called on the prepared statement to execute the SQL query and perform the insertion operation. It returns the number of rows affected by the operation.

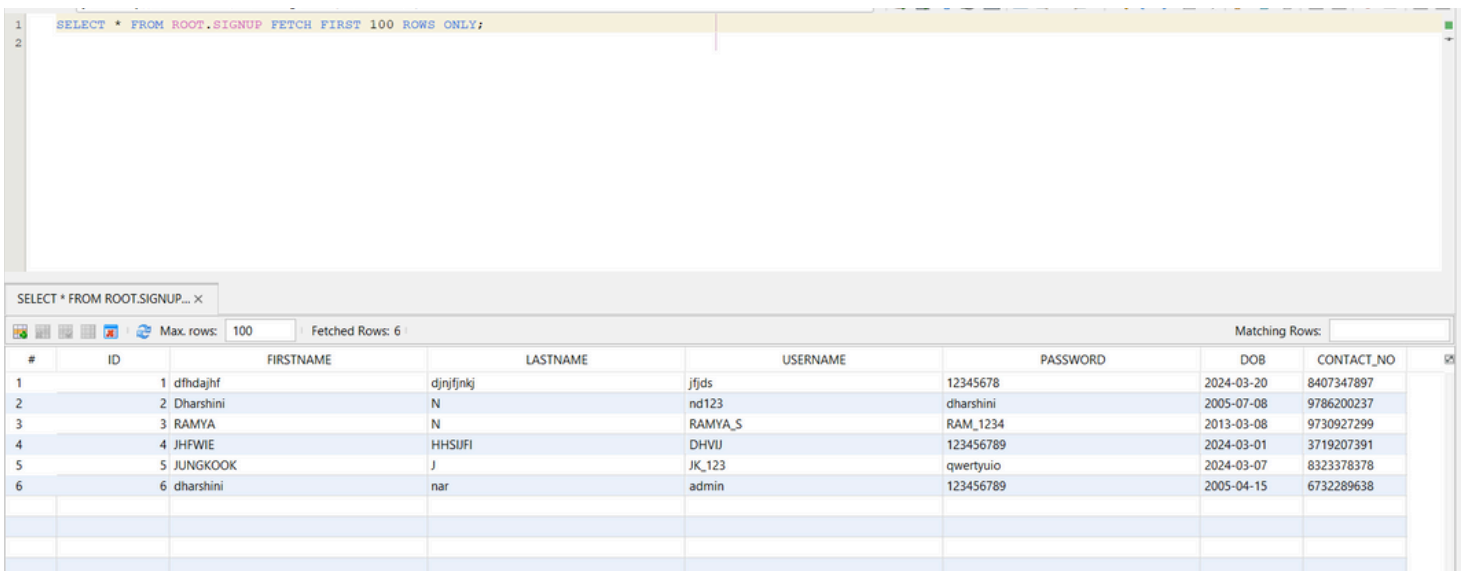
7. Message Dialog: Depending on the result of the insertion operation (`i`), a message dialog is displayed indicating whether the record was successfully inserted or not.

8. Exception Handling: Any exceptions that occur during the execution of the method are caught and printed to the console using `e.printStackTrace()`. This helps in debugging and identifying any issues that may arise during database operations.

In summary, this method encapsulates the functionality to insert new user details into the "signup" table of a Derby database, ensuring data integrity and providing feedback to the user regarding the success or failure of the operation.

CODE FOR DATABASE CONNECTIVITY OF TABLE SIGNUP

```
void insertDetails()
    {  SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
       String myDob=format.format(dob);
    try
      {
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
Connection_con=
DriverManager.getConnection("jdbc:derby://localhost:1527/feesmanagement","root","root");
        String sql="insert into signup values(?,?,?,?,?,?,?)";
        PreparedStatement stmt= con.prepareStatement(sql);
        stmt.setInt(1,getId());
        stmt.setString(2, fname);
        stmt.setString(3,lname);
        stmt.setString(4,uname);
        stmt.setString(5,password);
        stmt.setString(6,myDob);
        stmt.setString(7,contact_no);
        int i=  stmt.executeUpdate();
        if(i>0)
        {
          JOptionPane.showMessageDialog(this,"Record Inserted") ;
        }
        else{
           JOptionPane.showMessageDialog(this,"Record Not Inserted") ;
        }

      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
```



#	ID	FIRSTNAME	LASTNAME	USERNAME	PASSWORD	DOB	CONTACT_NO
1	1	dfhdajhf	djnjfjnkj	jfjds	12345678	2024-03-20	8407347897
2	2	Dharshini	N	nd123	dharshini	2005-07-08	9786200237
3	3	RAMYA	N	RAMYA_S	RAM_1234	2013-03-08	9730927299
4	4	JHFWIE	HHSIJFI	DHVIJ	123456789	2024-03-01	3719207391
5	5	JUNGKOOK	J	JK_123	qwertyuio	2024-03-07	8323378378
6	6	dharshini	nar	admin	123456789	2005-04-15	6732289638

EMBEDDED SQL OPERATIONS

Embedded SQL refers to SQL statements that are embedded within a programming language, typically to interact with a database management system (DBMS) from within an application. Here's some content explaining embedded SQL operations:

Embedded SQL Operations:

Embedded SQL allows developers to seamlessly integrate database operations into their application code, enabling efficient and dynamic interaction with the underlying database. This integration enhances the functionality of the application by providing real-time access to database resources and facilitating data manipulation based on user input or application logic.

Key Features:

1. Integration: Embedded SQL allows SQL statements to be directly embedded within the source code of programming languages such as Java, C/C++, COBOL, and others. This integration enables developers to leverage the full power of SQL while programming application logic.

2. Parameterization: Embedded SQL supports parameterized queries, allowing developers to pass parameters dynamically to SQL statements. This feature enhances security by preventing SQL injection attacks and promotes code reusability by facilitating the execution of similar queries with different parameter values.

3. Prepared Statements: Prepared statements are a key feature of embedded SQL, enabling developers to precompile SQL queries for improved performance and efficiency. Prepared statements can be reused multiple times with different parameter values, reducing overhead associated with query compilation and optimization.

4. Transaction Management: Embedded SQL provides support for transaction management, allowing developers to define transaction boundaries within their application code. This enables atomicity, consistency, isolation, and durability (ACID properties) for database transactions, ensuring data integrity and reliability.

5. Error Handling: Embedded SQL frameworks offer robust error handling mechanisms, allowing developers to handle database errors gracefully within their application code. Error codes and messages returned by the database engine are captured and processed, enabling developers to take appropriate actions based on the nature of the error.

Benefits:

- Efficiency: Embedded SQL operations are executed directly within the application code, eliminating the need for separate database client programs or middleware layers. This results in faster data access and reduced latency, improving overall application performance.

- **Simplicity:** Embedded SQL simplifies database interactions by providing a familiar SQL syntax within the context of the programming language used for application development. Developers can seamlessly integrate database operations into their code without having to learn additional query languages or frameworks.

PL/SQL PROCEDURES ON DERBY DATABASE

PL/SQL (Procedural Language/Structured Query Language) procedures play a crucial role in database operations by providing a powerful mechanism for encapsulating business logic and executing SQL statements within the database environment. These procedures are stored and executed on the database server, offering several advantages:

1. Data Manipulation: PL/SQL procedures enable the manipulation of data stored in database tables. They can perform various operations such as inserting new records, updating existing records, deleting records, and retrieving data based on specified criteria. By encapsulating these operations within procedures, developers can ensure consistent and reliable data management across different parts of the application.

2. Transaction Management: Procedures facilitate transaction management in the database by allowing developers to define transaction boundaries and control the execution of multiple SQL statements as a single logical unit of work. Transactions can be started, committed, or rolled back within procedures, ensuring data consistency and integrity.

3. Error Handling: PL/SQL procedures support robust error handling mechanisms to deal with exceptions that may occur during database operations. Developers can implement error handling logic within procedures to capture and handle exceptions gracefully, ensuring that the application remains stable and responsive in the event of errors.

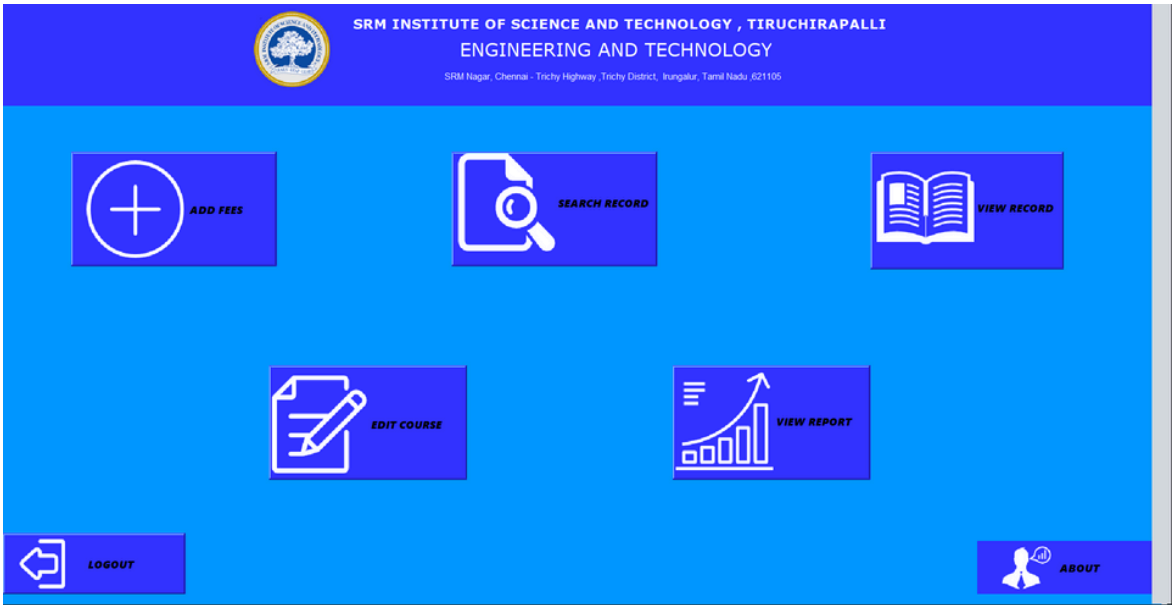
4. Parameterization: Procedures support parameterization, allowing developers to define input and output parameters that can be passed to and from the procedure. Parameterized procedures enhance code reusability and flexibility by enabling the execution of similar operations with different parameter values.

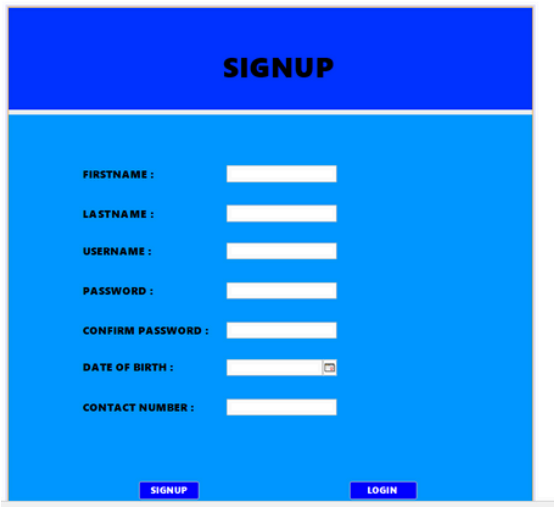
5. Security: Procedures contribute to database security by encapsulating sensitive operations and controlling access to database resources. Developers can grant appropriate permissions to procedures based on user roles and privileges, ensuring that only authorized users can execute them.

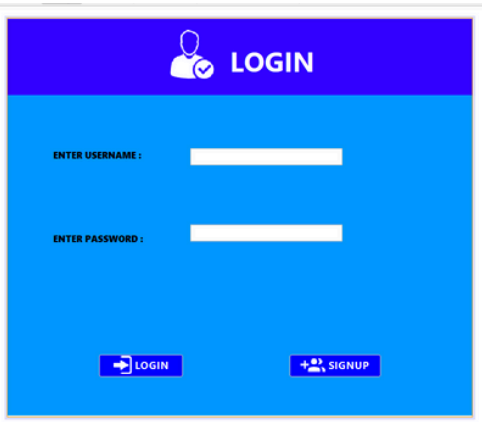
6. Performance Optimization: PL/SQL procedures can improve database performance by reducing network overhead and minimizing the number of round-trips between the application and the database server. By executing SQL statements directly on the server, procedures eliminate the need to transfer large volumes of data between client and server, resulting in faster query execution and reduced latency.

In summary, PL/SQL procedures are a fundamental component of database development, offering a rich set of features and capabilities for implementing business logic, managing transactions, handling errors, and optimizing performance. By leveraging procedures effectively, developers can build robust and efficient database applications that meet the evolving needs of users and business requirements.

FRONT END DESIGNS :



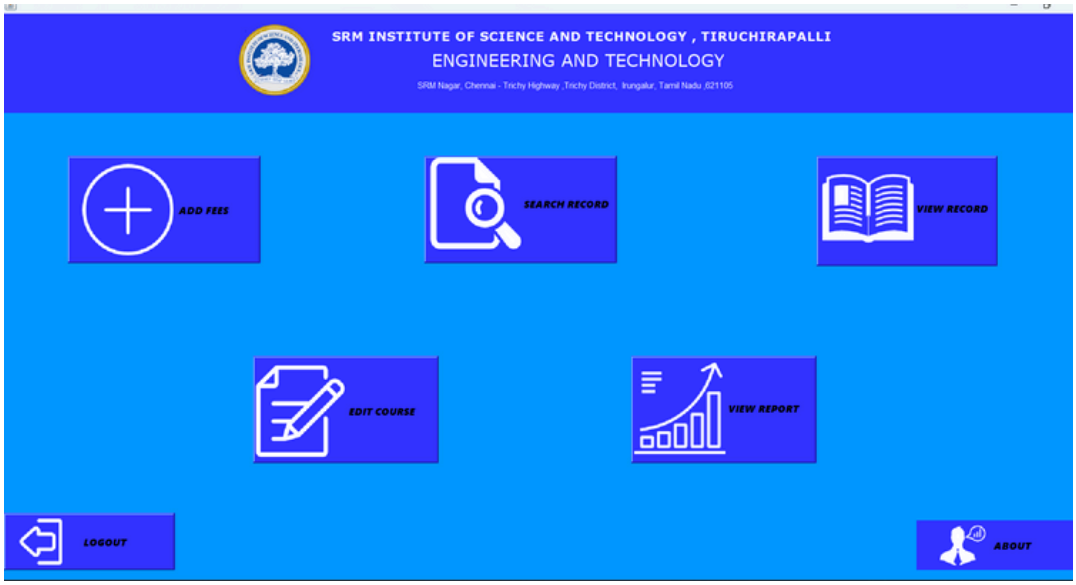




LOGIN MODULE :

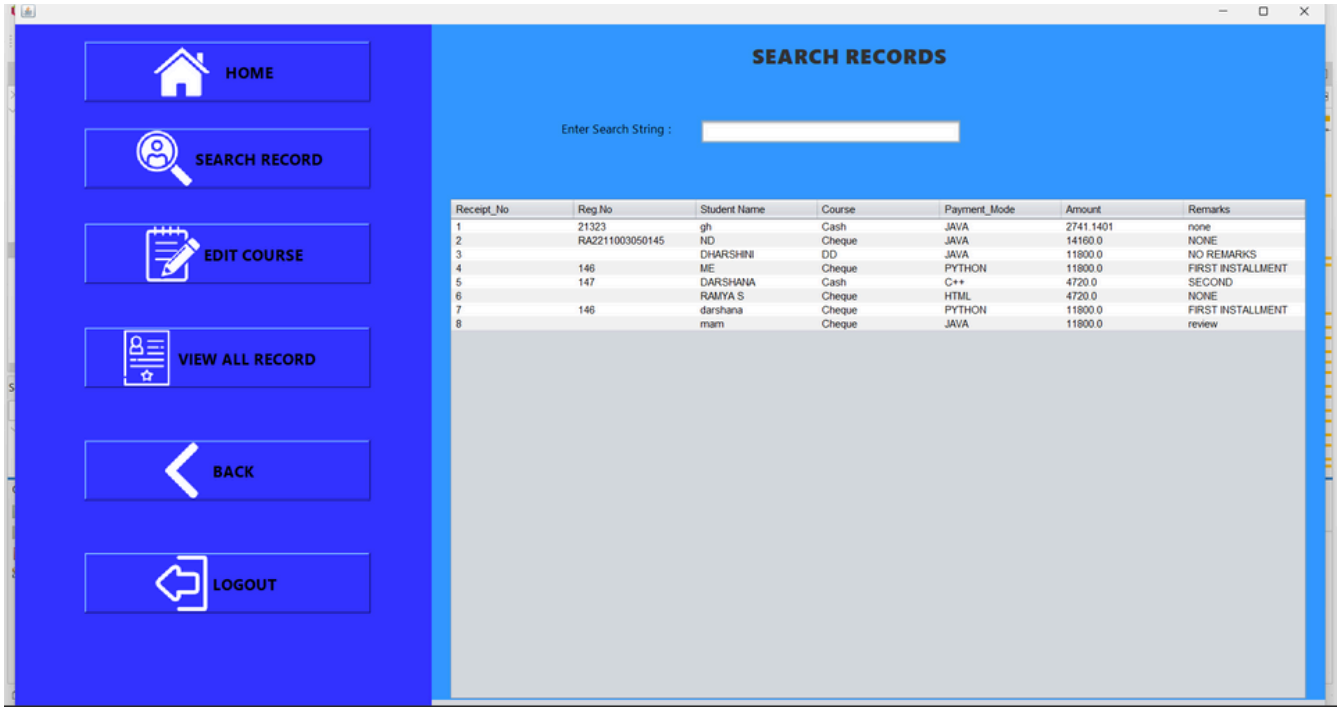

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JOptionPane;


public class Login extends javax.swing.JFrame {

  /**
   * Creates new form Login
   */
  public Login() {
    initComponents();
  }

  void userVerification(String username , String password)
  {
    try
    {
      Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
      Connection con = DriverManager.getConnection("jdbc:derby://localhost:1527/feesmanagement","root","root");
      String sql = "select * from signup where username = ? and password =?";
      PreparedStatement pst = con.prepareStatement(sql);
      pst.setString(1,username);
      pst.setString(2,password);
      ResultSet rs = pst.executeQuery();
      if(rs.next())
      {
        JOptionPane.showMessageDialog(this, "Login Successful!");
        Home home = new Home();
        home.show();
        this.dispose();


      }
      else
      {
        JOptionPane.showMessageDialog(this, "Wrong Username or Password");
      }
    }
    catch(Exception e)
    {

    }
  }
```

HOME PAGE MODULE :




```
import jav
public class Home extends javax.swing.JFrame {
    /**
     * Creates new form Home
     */
    public Home() {
        initComponents();
    }

    private void initComponents() {
        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
        jLabel12 = new javax.swing.JLabel();
        jPanel3 = new javax.swing.JPanel();
        jLabel4 = new javax.swing.JLabel();
        jPanel7 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jPanel8 = new javax.swing.JPanel();
        jLabel5 = new javax.swing.JLabel();
        jPanel10 = new javax.swing.JPanel();
        jLabel7 = new javax.swing.JLabel();
        jPanel11 = new javax.swing.JPanel();
        jLabel9 = new javax.swing.JLabel();
        jPanel14 = new javax.swing.JPanel();
        jLabel13 = new javax.swing.JLabel();
        jPanel30 = new javax.swing.JPanel();
        jLabel8 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
```

SEARCH RECORD MODULE :

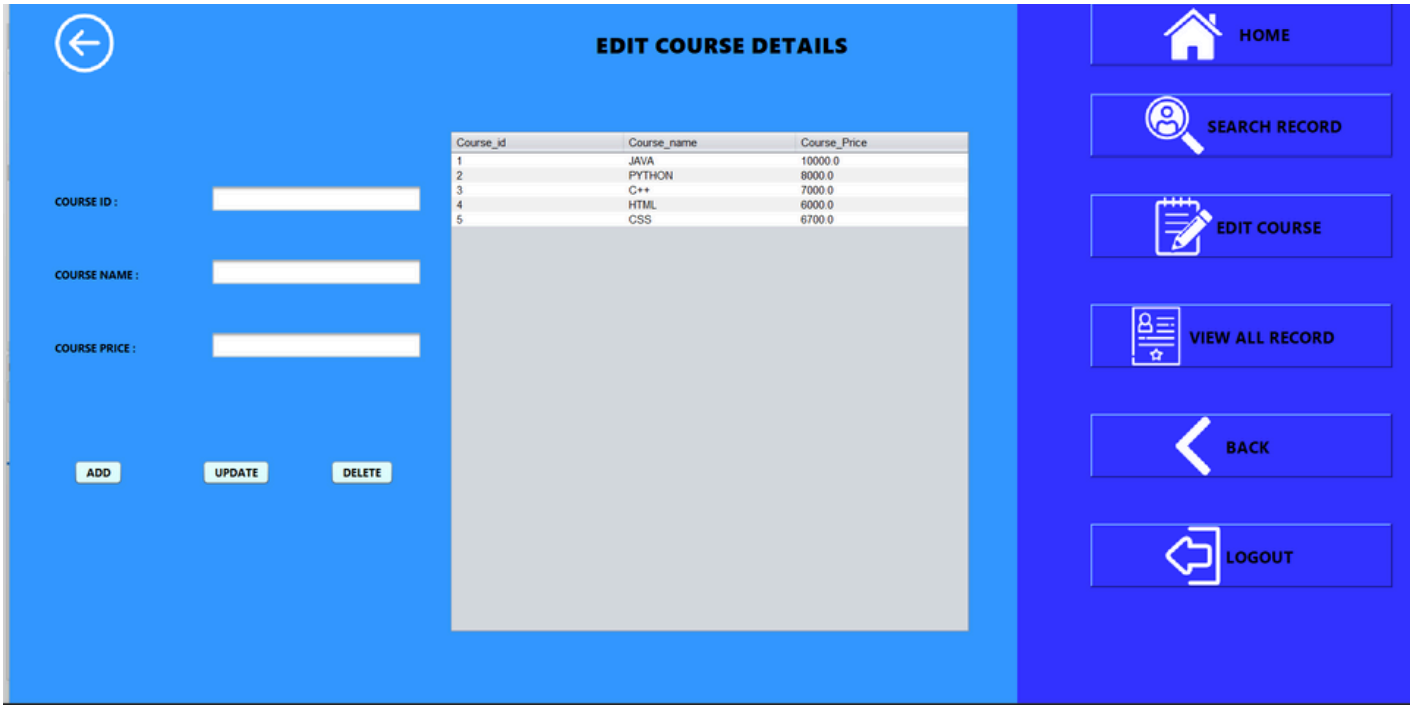



```
import java.awt.Color;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.RowFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableRowSorter;


public class SearchRecord extends javax.swing.JFrame {


  DefaultTableModel model;

  public SearchRecord() {
    initComponents();
    setRecordsToTable();
  }

  public void setRecordsToTable()
  {
    try
    {
      Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
      Connection con =DriverManager.getConnection("jdbc:derby://localhost:1527/feesmanagement","root","root");
      PreparedStatement pst = con.prepareStatement("select * from fees_details ");
      ResultSet rs=pst.executeQuery();

      while(rs.next()){
        String receiptNo= rs.getString("reciept_no");
        String rollNo= rs.getString("roll_no");
        String studentName= rs.getString("student_name");
        String paymentMode= rs.getString("payment_mode");
        String courseName= rs.getString("course_name");
```

VIEW RECORD MODULE :

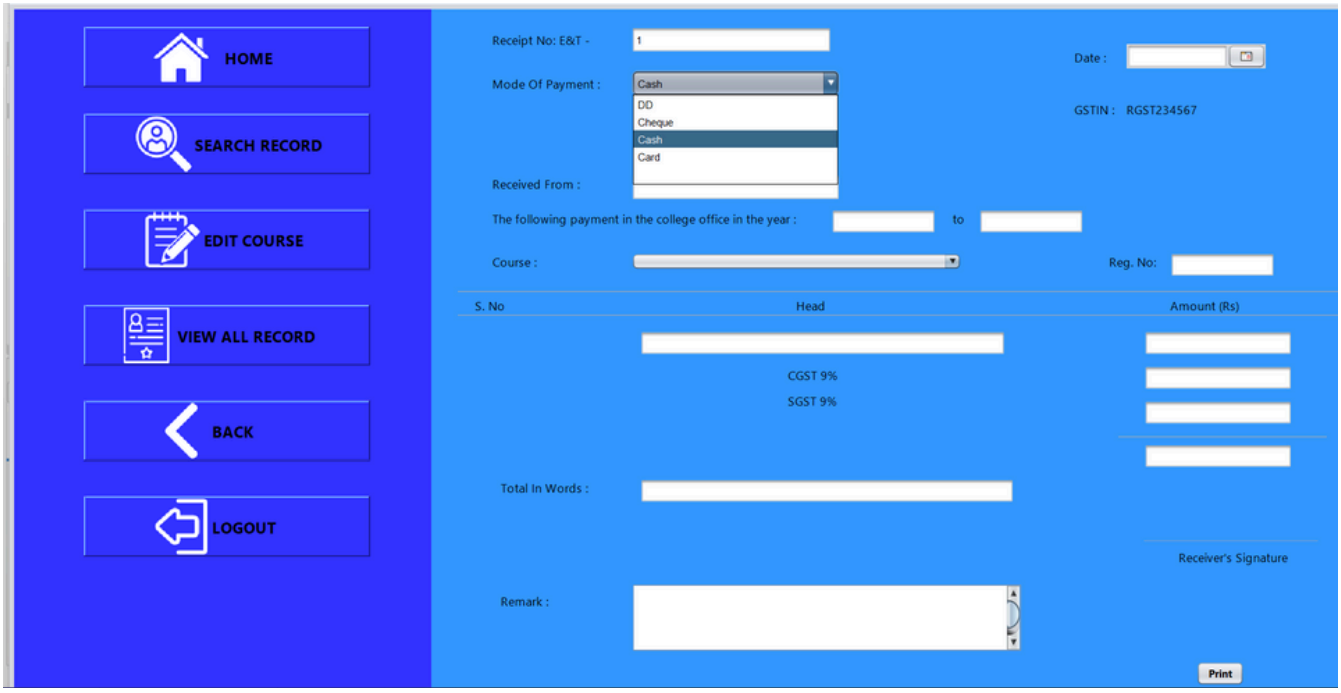

```
import java.awt.Color;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.text.SimpleDateFormat;
import javax.swing.table.DefaultTableModel;
public class ViewAllRecords extends javax.swing.JFrame {
DefaultTableModel model;
 public ViewAllRecords() {
     initComponents();
     viewRecords();
  }

  public void viewRecords()
  {
     try
     {
        Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
        Connection con =DriverManager.getConnection("jdbc:derby://localhost:1527/feesmanagement","root","root");
        PreparedStatement pst = con.prepareStatement("select * from fees_details ");
        ResultSet rs=pst.executeQuery();

        while(rs.next()){
           String receiptNo= rs.getString("reciept_no");
           String rollNo= rs.getString("roll_no");
           String studentName= rs.getString("student_name");
           String paymentMode= rs.getString("payment_mode");
           String courseName= rs.getString("course_name");
           float amount= rs.getFloat("total_amount");

           String remark= rs.getString("remark");
```

EDIT COURSE MODULE :



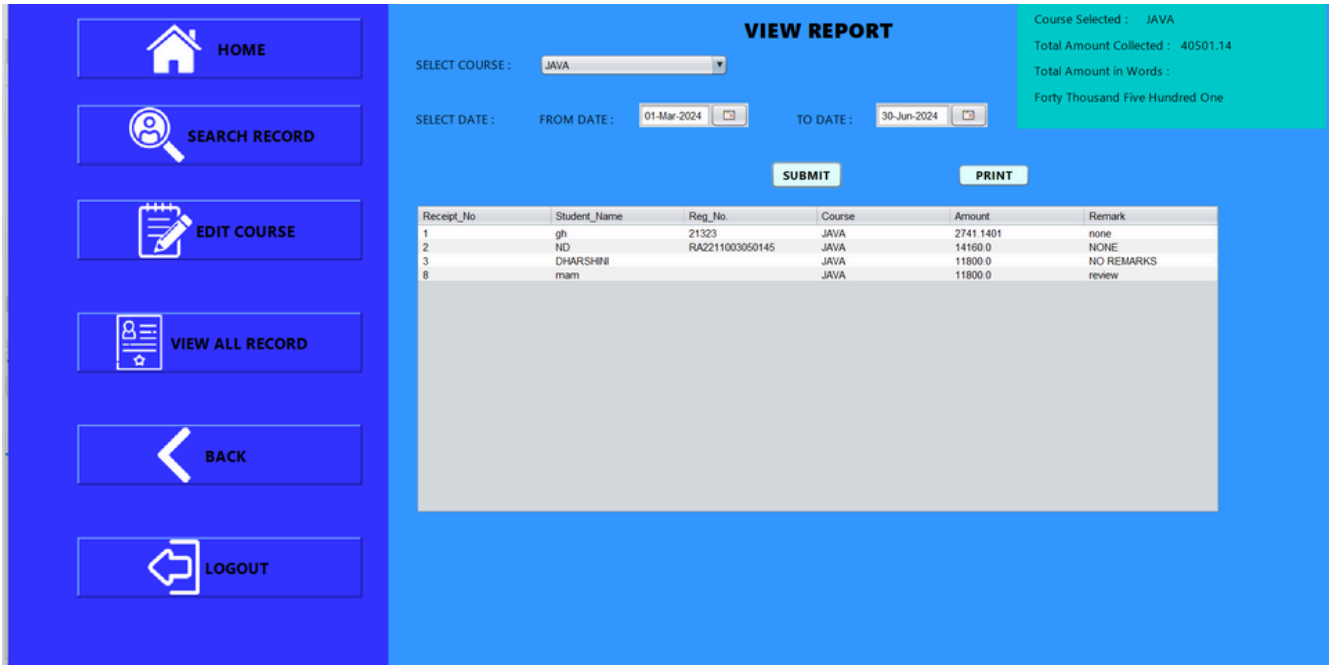

```
import java.awt.Color;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

public class EditCourse extends javax.swing.JFrame {
   DefaultTableModel model;
   public EditCourse() {
      initComponents();
      setRecordsToTable();
   }
   public void setRecordsToTable()
   {
      try
      {
         Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
         Connection con
=DriverManager.getConnection("jdbc:derby://localhost:1527/feesmanagement","root","root");
         PreparedStatement pst = con.prepareStatement("select * from course");
         ResultSet rs=pst.executeQuery();

         while(rs.next()){
            String course_id= rs.getString("id");
            String course_name= rs.getString("cname");
            String course_price= rs.getString("cost");


            Object[] obj ={course_id,course_name,course_price};
            model = (DefaultTableModel) tbl_course_data.getModel();
            model.addRow(obj);
```

ADD FEES MODULE :




```
import java.awt.Color;
import javax.swing.JOptionPane;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JOptionPane;
public class AddFees extends javax.swing.JFrame {
 public AddFees() {
      initComponents();
      displayCashFirst();
      fillComboBox();
     int receiptNo = getReceiptNo();
      txt_receiptno.setText(Integer.toString(receiptNo));

   }
   public void displayCashFirst()
   {
      lbl_ddno.setVisible(false);
      lbl_chequeno.setVisible(false);
      lbl_bankname.setVisible(false);
      txt_ddno.setVisible(false);
      txt_chequeno.setVisible(false);
      txt_bankname.setVisible(false);
   }
   public boolean validation()
   {
      if(txt_receivedfrom.getText().equals(""))
      {
         JOptionPane.showMessageDialog(this,"Please Enter
Username");
         return false;
```

VIEW REPORT MODULE :




```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.awt.print.PrinterJob;
import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.text.MessageFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.JFileChooser;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

public class ViewReport extends javax.swing.JFrame {
    DefaultTableModel model;
   public ViewReport() {
      initComponents();
      fillComboBox();
      clearTable();
   }
    public void fillComboBox()
   {
      try
      {
         Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
  Connection con
=DriverManager.getConnection("jdbc:derby://localhost:1527/feesmanagement","root","root");
         PreparedStatement pst = con.prepareStatement("select cname from course");
         ResultSet rs = pst.executeQuery();
         while(rs.next())
         {
          combo_course.addItem(rs.getString("cname"));
```

REPORT GENERATION USING CRYSTAL REPORTS/REPORT GENERATION TOOL

Report generation using Crystal Reports or any other report generation tool involves the creation of formatted reports based on data retrieved from a database or other data sources. Here's an overview of the process:

1. Designing the Report:

- The first step is to design the layout and structure of the report. This includes defining the report's header, footer, sections, and formatting elements such as fonts, colors, and styles.
- Report designers typically use drag-and-drop interfaces to add data fields, text objects, images, charts, and other elements to the report canvas.

2. Connecting to Data Sources:

- Once the report layout is defined, the next step is to connect the report to one or more data sources. This can include databases (here derby), spreadsheets, XML files, web services, or other data repositories.
- Report generation tools provide connectivity options and wizards to establish connections to various data sources and retrieve data for the report.

3. Defining Data Queries:

- After connecting to the data source, the report designer defines queries or commands to retrieve the required data for the report. This may involve writing SQL queries, stored procedures, or using query builders provided by the reporting tool.
- The designer specifies criteria, filters, sorting, and grouping options to retrieve the data in the desired format.

4. Mapping Data to Report Elements:

- Once the data is retrieved, the designer maps the data fields to the corresponding report elements. This involves linking database fields to text boxes, labels, tables, charts, and other components on the report layout.
- The report generation tool provides options for data binding and parameterization to dynamically populate report elements with data from the dataset.

5. Formatting and Styling:

- After mapping the data, the designer applies formatting and styling to the report elements to enhance readability and visual appeal. This includes adjusting fonts, colors, borders, alignment, and spacing.
- Conditional formatting may be applied to highlight specific data based on criteria or thresholds.

6. Testing and Previewing:

- Once the report design is complete, the designer tests and previews the report to ensure that it displays the data accurately and meets the requirements.
- Preview options allow the designer to view the report output within the report generation tool before finalizing and distributing it.

7. Exporting and Distribution:

- After the report is validated, it can be exported to various formats such as PDF, Excel, Word, HTML, or printed directly from the report generation tool.
- Reports can be distributed via email, shared on a web portal, or integrated into other applications for consumption by end users.

In summary, report generation using Crystal Reports or other similar tools involves designing, connecting to data sources, defining queries, mapping data to report elements, formatting and styling, testing, exporting, and distributing reports to end users, with a focus on accuracy, usability, and visual presentation.

ONLINE COURSE COMPLETION CERTIFICATE