2026

OOP LAB FINAL PROJECT REPORT

GROUP MEMBERS:

Muhammad Arsal Abdullah (FA24-BDS-017)

Abdullah Bin Abdul Mannan (FA24-BDS-006)

Ameema Iman (FA24-BDS-013)

Sanan Hussain Khokhar (FA24-BDS-042)

SUBMITTED ON: December 15, 2025

Table of Contents

Oracle DataForge

Project Overview

The Oracle Database Management System is a Java desktop application built using core OOP principles. It offers a user-friendly GUI for managing, querying, and visualizing Oracle database tables. The project demonstrates encapsulation by separating backend logic from the interface, inheritance via panels extending JPanel, and polymorphism through chart generation using a common interface. Abstraction hides complex operations from the user, while modular, reusable components enhance maintainability. Event-driven programming enables seamless interaction with tables, queries, and visualizations, showcasing a practical OOP-based database management tool.

Purpose

The Oracle DB Manager is a Java-based desktop application designed to provide a graphical interface for interacting with Oracle databases. It allows users to manage, visualize, query, and analyze database tables without using command-line SQL tools. The system integrates database operations with a modern dark-themed GUI, making database management intuitive and efficient.

Problem Statement

Accessing and managing Oracle database data can be complex for users without technical expertise. This project provides a Java-based OOP desktop application that allows users to browse tables, execute queries, perform CRUD operations, and visualize data through an intuitive graphical interface.

TASK DISTRIBUTION

Member A : Ameema Iman (FA24-BDS-013)

- View Package – 4 classes
- UI Package:
  - MainFrame
  - MainFrame Panel
  - View
  - View Panel

Member B: Arsal Abdullah Khan (FA24-BDS-017)

- Charts Package – 7 classes
- AI Package – 1 class
- UI Package:
  - Visualization
  - Visualization Panel

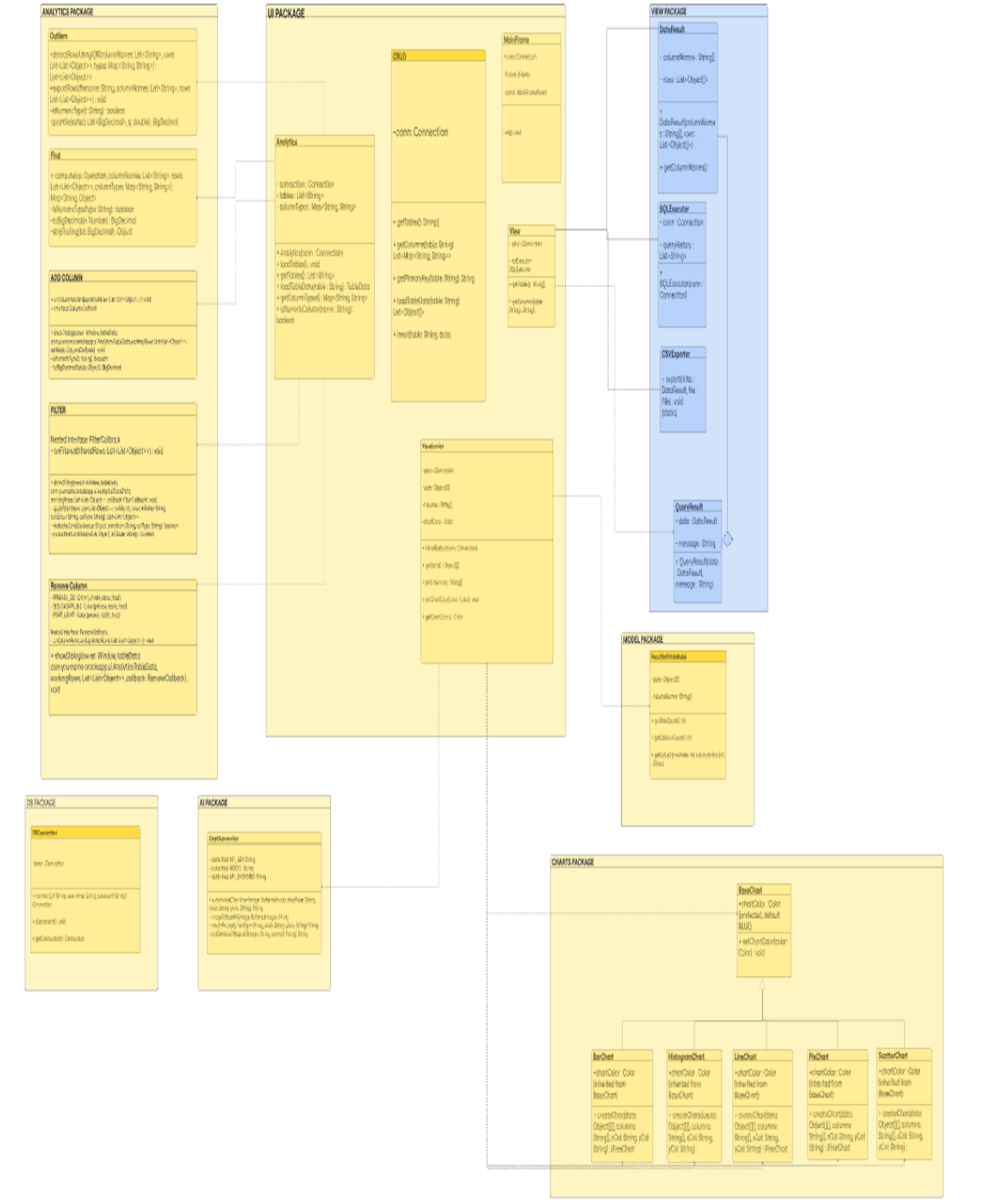
Member C: Sanan Hussain Khokhar (FA24-BDS-042)

- Analytics Package – 5 classes
- UI Package:
  - Analytics
  - Analytics Panel

Member D: Abdullah bin Abdul Mannan (FA24-BDS-006)

- DB Package – 1 class
- Model Package – 1 class
- UI Package:
  - Crud
  - Crud Panel
  - DBConnection Panel

CLASS DIAGRAM



Features of Project

General Features

- **Database Connectivity:** Connects to Oracle using JDBC.
- **GUI-Based Interaction:** Swing-based modern interface with dark theme, styled buttons, tables, and fields.
- **User-Friendly & Thread-Safe:** Input highlights, hover effects; GUI runs safely on the EDT.
-

Core Modules

A. App / Entry Point

- Main method launches the application and DBConnectionPanel.
- Thread-safe GUI initialization via SwingUtilities.

B. Database Management

- DBConnection handles connect/disconnect and provides Connection objects.
- CRUD operations via Crud and CrudPanel.
- Metadata access: getTables(), getColumns(), getPrimaryKey().
- SQLExecutor handles queries and maintains history.
- DataResult & QueryResult structure query results and messages.

C. Analytics

- Add/Remove Columns, Filter, Sort, and manage outliers.
- Statistical computations: count, sum, average, min, max, median, mode, stddev.
- Type checks ensure numeric operations are safe.

D. Data Visualization / Charts

- Supports Area, Bar, Line, Pie, Histogram, Scatter charts.
- BaseChart class provides consistent chart creation; integrated with JFreeChart.
- VisualizationPanel GUI generates charts dynamically from SQL data.

E. AI Integration

- ChartSummarizer uses Google Gemini AI to generate textual summaries of charts.

F. Data Export

- CSVExporter safely exports DataResult to CSV files with proper formatting.

G. User Interface

- MainFrame & MainFramePanel manage panel navigation (Home, CRUD, Analytics, View, Visualization).
- ViewPanel: table navigation, search, SQL execution, CSV export.
- AnalyticsPanel: data exploration, filtering, column management, outlier detection.
- VisualizationPanel: query input, chart generation, axis selection, color customization.

OOP & Design

- Encapsulation, abstraction, inheritance (charts), delegation, polymorphism, and separation of concerns.
- Modular design for database, analytics, visualization, CSV export, and AI.

Advanced Features

- Dynamic table handling and metadata inspection.
- Thread-safe, responsive GUI with error handling.
- Interactive dashboards combining CRUD, analytics, visualization, and AI summaries.

CORE FUNCTIONALITIES

1. Classes and Objects

- Classes define the blueprint of components such as MainFrame, MainFramePanel, ViewPanel, VisualizationPanel, View, and Visualization.
- Objects are instantiated from these classes to perform actions, e.g., new MainFrame(conn) or new ViewPanel(conn, this::showHome).
- Encapsulates the GUI and backend logic into reusable components.

2. Inheritance

- **JPanel and JFrame inheritance:**
    - Custom panels like MainFramePanel, ViewPanel, and VisualizationPanel extend JPanel, inheriting all GUI functionality.
- This allows polymorphic usage in container layouts like CardLayout.

3. Encapsulation

- Private fields and methods to protect internal state, e.g.:
    - private Connection conn;
    - private View viewLogic;
    - Getter/setter methods like getData() or setChartColor(Color color) in Visualization class.
- Ensures controlled access to the database connection and UI components.

4. Abstraction

- Separation of UI (ViewPanel, VisualizationPanel) from backend logic (View, Visualization).
- Users interact with high-level methods like viewLogic.loadData() or visualization.generateChart() without worrying about implementation details.

5. Polymorphism

- **Method overloading**: Constructors in classes like VisualizationPanel and ViewPanel.
- **Dynamic method dispatch**:
    - For example, chart.createChart(data, columns, xCol, yCol) can call different chart types (Bar, Line, Pie) through the BaseChart abstract interface.

- Event listeners (ActionListener) also demonstrate polymorphism: the same interface handles multiple button actions.

6. Composition

- Classes contain objects of other classes to build complex behavior:
    - MainFramePanel contains ViewPanel, CrudPanel, AnalyticsPanel, VisualizationPanel.
    - VisualizationPanel contains Visualization for chart generation.
- Enables modular design and separation of responsibilities.

7. Interfaces and Lambda Expressions

- Runnable interface is used for passing onBack actions:
    - cards.add(new ViewPanel(conn, this::showHome), "VIEW");
- Event listeners like ActionListener implemented using lambda expressions for concise code.

8. Exception Handling

- Try-with-resources blocks for safe JDBC operations.
- Catching SQLException and runtime exceptions ensures robust, error-tolerant execution.

9. Static Members

- Static constants for colors, dimensions, and button sizes for consistency across the UI.
- Example: private static final Color PRIMARY_BG = new Color(28, 48, 74);

Libraries and Packages Used:

1. Java Standard Libraries

- **java.util, java.math.BigDecimal, java.io** – Collections, high-precision computations, and file handling.
- **javax.swing & java.awt** – GUI components, layouts, colors, fonts, and event handling.
- **javax.imageio, java.net** – Image processing and API/URL handling.

2. Custom Project Packages

- **ui** – GUI panels and visualization (MainFramePanel, ViewPanel, VisualizationPanel).
- **analytics** – Data analysis, filtering, outlier detection, column operations.
- **ai** – AI integration, image handling, API requests.
- **model** – Database-to-GUI mapping (ResultSetTableModel).
- **charts** – Chart generation (BarChart, PieChart, LineChart).
- **db** – Database connectivity and management.

3. External Libraries (JARs)

- **jcommon & jfreechart** – Charting utilities and visualization.
- **json** – JSON parsing for API/AI integration.
- **ojdbc17** – Oracle JDBC driver for database connectivity.

APP

Purpose:

- Serves as the entry point of the application.
- Launches the graphical user interface for database interaction.

Key Functionality:

- main method initializes the application.
- Instantiates DBConnectionPanel to allow users to connect to the database.

Implementation Details:

- Uses SwingUtilities.invokeLater to ensure GUI components run on the Event Dispatch Thread (EDT).
- Guarantees thread safety for all Swing operations.

Summary:

- App.java reliably starts the application and presents a user-friendly interface for database connection.

PACKAGES

AI

Class: ChartSummarizer
Package: com.yourname.oracleapp.ai
Type: Concrete Class
Purpose:
The ChartSummarizer class is responsible for analyzing chart images and generating textual summaries using Google's Gemini AI. It handles image conversion, prompt creation, API interaction, and response parsing to produce insights from chart visuals.

Key Attributes and Methods:

- **private static final String API_KEY** – Stores the API key for accessing the Gemini AI service.
- **private static final String MODEL** – Specifies the Gemini model used (gemini-2.5-flash).
- **summarizeChart(BufferedImage chartImage, String chartType, String xAxis, String yAxis)** – Main method that converts the chart to Base64, creates a prompt, and returns AI-generated insights.
- **private String imageToBase64(BufferedImage image)** – Converts a chart image into a Base64-encoded string suitable for API submission.
- **private String createPrompt(String chartType, String xAxis, String yAxis)** – Generates a structured textual prompt including chart type, axis labels, and instructions for summarization.
- **private String callGeminiAPI(String base64Image, String prompt)** – Sends the HTTP request to the Gemini API, handles errors, and extracts the generated summary text from the response.

ANALYTICS

Class: AddColumn
Purpose:
Provides a GUI dialog to add a new column to an existing dataset by selecting two columns, choosing an operation (arithmetic or concatenation), and specifying a name. Updates both column metadata and dataset values.

Key Features:

- Color constants for consistent UI theme.
- Callback interface to return updated rows.
- Dialog handles input, validation, and computation of new column values.
- Numeric type checks and safe conversion to BigDecimal.

Class: Filter
Purpose: Provides a GUI dialog to filter dataset rows based on user-defined conditions (e.g., comparisons, IN, LIKE, BETWEEN, IS NULL).

Key Features:

- Color-themed UI for consistency.
- Callback interface to return filtered rows.
- Supports numeric and text comparisons safely.
- Handles a wide range of filter operators.

Class: Find
Purpose:
Performs common statistical and aggregate computations on dataset columns, such as count, sum, average/mean, min, max, median, mode, standard deviation, and null counts.

Key Features:

- **Operation enum** – Defines supported computations (COUNT, SUM, AVERAGE, MIN, MAX, NULLS, MEAN, MEDIAN, MODE, STDDEV).
- **compute(...)** – Main method that applies the selected operation across one or more columns. Handles numeric and non-numeric types appropriately.
- **isNumericType(String type)** – Checks if a column is numeric for arithmetic operations.

- **toBigDecimal(Number n) / stripTrailing(BigDecimal bd)** – Safely converts numbers to BigDecimal and formats results without unnecessary trailing zeros.

Class: Outliers
Purpose:
Detects and manages outlier rows in numeric datasets using the Interquartile Range (IQR) method and allows exporting flagged rows to a file.

Key Features:

- **detectRowsUsingIQR(...)** – Identifies rows containing outliers in numeric columns based on 1.5×IQR thresholds.
- **exportRows(String filename, List<String> columnNames, List<List<Object>> rows)** – Exports selected rows to a tab-delimited text file.
- **isNumericType(String t)** – Checks if a column is numeric.
- **quantile(List<BigDecimal> sorted, double q)** – Calculates quartiles for IQR computation.

Class: RemoveColumn
Purpose:
Provides a GUI dialog to remove an existing column from a dataset in the analytics module. Ensures safe removal with user validation and updates both column metadata and dataset values.

Key Features:

- Color-themed UI for consistency with the application.
- **RemoveCallback interface** – Returns updated rows after column removal.
- **showDialog(...)** – Displays a modal dialog, handles selection and validation, removes the column, and triggers the callback.
- Prevents removing the last remaining column to maintain dataset integrity.

DB

Class: DBConnection
Purpose:
Manages connections to an Oracle database, providing methods to connect, disconnect, and retrieve the active Connection object.

Key Features:

- **connect(String url, String username, String password)** – Establishes a JDBC connection, disables auto-commit for transaction control, and handles connection errors.
- **disconnect()** – Safely closes the active connection if open.
- **getConnection()** – Returns the current Connection object for use in database operations.

MODEL

Class: ResultSetTableModel

Purpose:

ResultSetTableModel converts a JDBC ResultSet into a table model compatible with Swing's JTable. This allows database query results to be displayed in a graphical table format within the application. It also supports initialization from static data arrays.

Key Features and Methods:

Constructors:

- From ResultSet: Loads data and column names from a live database query.
- From 2D array + column names: Initializes with custom or pre-extracted data.

Data Loading:

- loadResultSet(ResultSet rs): Reads all rows and columns into a 2D array and stores column names.

Table Model Methods (AbstractTableModel overrides):

- getRowCount() / getColumnCount(): Return number of rows/columns.
- getValueAt(row, col): Get value of a specific cell.
- getColumnName(col): Get column name.

CHARTS

Class: BaseChart
Type: Abstract Class

Purpose:

BaseChart serves as a base class for all chart types in the application. It provides common functionality such as defining a chart color and enforcing a consistent method for chart creation.

Key Attributes and Methods:

- **protected Color chartColor** – Stores the color used for chart elements (default is blue).
- **setChartColor(Color color)** – Allows changing the chart color dynamically.
- **abstract createChart(Object[][] data, String[] columns, String xCol, String yCol)** –

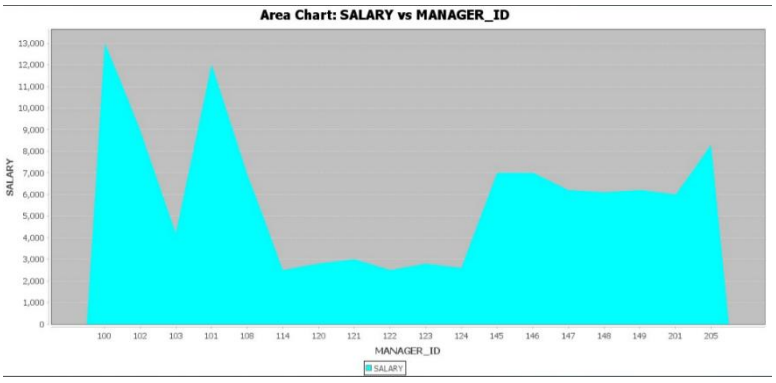
Class: AreaChart
Extends: BaseChart

Purpose:

AreaChart generates area charts to visualize trends and comparisons in data. It uses the **JFreeChart** library to display the magnitude of values over a continuous interval.

Key Method:

- **createChart(Object[][] data, String[] columns, String xCol, String yCol)**



Class: BarChart
Extends: BaseChart

Purpose:

BarChart is used to create bar charts that visually compare values across categories. It uses **JFreeChart** to generate charts with vertical bars representing data magnitude.

Key Method:

- createChart(Object[][] data, String[] columns, String xCol, String yCol)

Class: HistogramChart
Extends: BaseChart

Purpose:

HistogramChart generates histograms to visualize the distribution of numerical data. Each bar represents the frequency of values within a range, helping to identify patterns such as clustering or spread.

Key Method:

- createChart(Object[][] data, String[] columns, String xCol, String yCol)

Class: LineChart
Extends: BaseChart

Purpose:

LineChart creates line charts to visualize trends and changes over a sequence or time. It connects data points with lines, making it easy to identify patterns or progressions.

Key Method:

- createChart(Object[][] data, String[] columns, String xCol, String yCol)

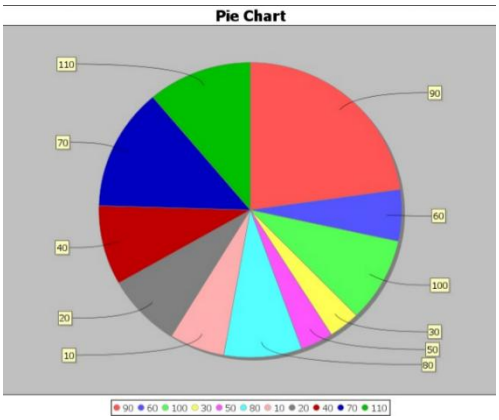
Class: PieChart
Extends: BaseChart

Purpose:

PieChart generates pie charts to visualize data as proportions of a whole. Each slice represents the contribution of a category to the total, making it ideal for comparing relative sizes.

Key Method:

- createChart(Object[][] data, String[] columns, String xCol, String yCol**)**



VIEW

Class: CSVExporter
Purpose:
• **Export Data:** Converts a DataResult object into a CSV file.
• **Column Handling:** Writes column names as the first row, separated by commas.
• **Row Handling:** Writes each row, escapes quotes, and encloses values containing commas or quotes in double quotes.
• **File Writing:** Uses PrintWriter with try-with-resources to safely write to the specified file.
• **Integration:** Can be used anywhere DataResult needs to be saved as a CSV for external use.

Class: DataResult
Purpose:
• **Data Storage:** Holds tabular data with column names and rows.
• **Columns:** Stores column headers in a String[] array.
• **Rows:** Stores each row of data as an Object[] in a List.
• **Accessors:** Provides getColumnNames() and getRows() to retrieve column names and row data.
• **Integration:** Used as a standard container for passing query results or table data between components.

Class: QueryResult
Purpose:
• **Encapsulate Results:** Holds the outcome of a database query.
• **Data Storage:** Contains a DataResult object for tabular query data.
• **Message Storage:** Stores a status or informational message about the query.
• **Accessors:** Provides getData() and getMessage() to retrieve the results and message.
• **Integration:** Serves as a standard way to return both query data and messages from database operations.

Class: SQLExecutor
Purpose:
• **Execute SQL Queries:** Handles execution of both SELECT and update/DDL statements on a database connection.
• **Query History:** Maintains a list of all executed SQL queries (queryHistory).
• **Run SQL:** runSQL(String sql) executes the query and returns a QueryResult.

- For SELECT queries: Reads column names and rows into a DataResult.

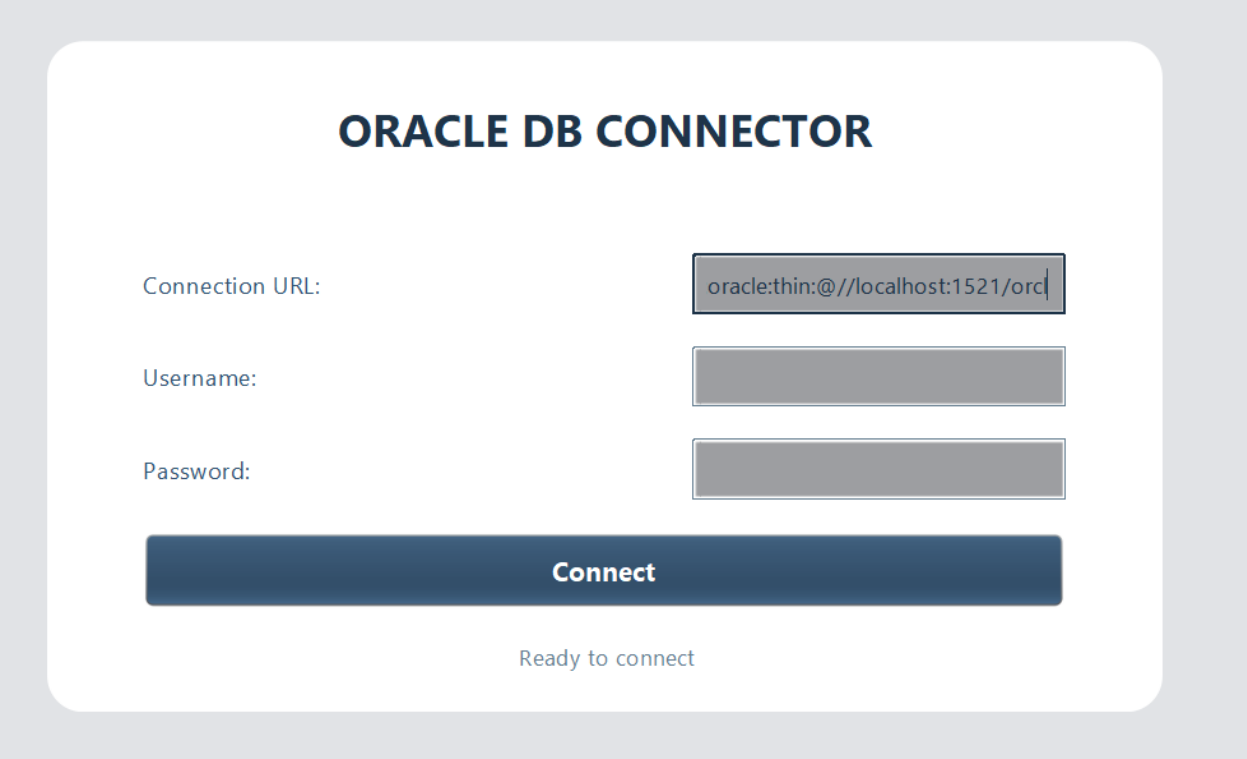
- For UPDATE/INSERT/DELETE: Returns affected row count.
- For DDL: Returns a success message.
  • **Error Handling:** Rolls back on SQL exceptions and returns error messages in QueryResult.
  • **Integration:** Provides a unified interface for executing SQL and retrieving structured results and messages.

UI

Class: DBConnectionPanel

Purpose:

- **Connection Inputs:** Text fields and a password field with placeholders and focus highlights for better usability.
- **Connect Button:** Initiates connection via the DBConnection class, opens MainFrame on success, and displays real-time status messages.
- **UI Styling:** Modern, rounded panel design with a consistent color palette, styled buttons/labels/fields, and visual enhancements like shadows and rounded corners.
- **Integration:** Passes the active Connection object to MainFrame for backend operations.



Class: MainFrame

Purpose:

The MainFrame class serves as the main window of the application. It acts as a container for the core GUI (MainFramePanel) and provides a centralized frame for interacting with all features once a database connection is established.

Key Features:

Database Connection: Receives a Connection object and passes it to MainFramePanel for database operations.
• **JFrame Setup:** Creates a window titled “Oracle Swing Application”, sets size to 900×600, centers it, and closes the app on exit.
• **Content Management:** Loads MainFramePanel as the main content pane to manage UI components.
• **Visibility:** Displays the frame after initialization.

Class: MainFramePanel

Purpose:

The **MainFramePanel** class serves as the central GUI of the Oracle Swing application, providing a home/dashboard interface and managing multiple functional panels—CRUD, Data View, Analytics, and Visualization—using a **CardLayout** for smooth panel switching. It features a sidebar with icon buttons for easy navigation, hover effects for better user experience, and a modern, clean theme with consistent colors, fonts, and spacing. The class dynamically displays panels when sidebar buttons are clicked and allows returning to the home panel, while also passing the active **Connection** object to all panels to enable database operations.

Class: Crud

Purpose:

The Crud class provides core Create, Read, Update, and Delete operations for database tables. It acts as a backend utility for managing table data in the application.

Key Features:

• **Table & Column Metadata:**

getTables(), getColumns(String table), getPrimaryKey(String table)

• **Data Manipulation:**

loadTableData(String table), insert(...), update(...), delete(...)

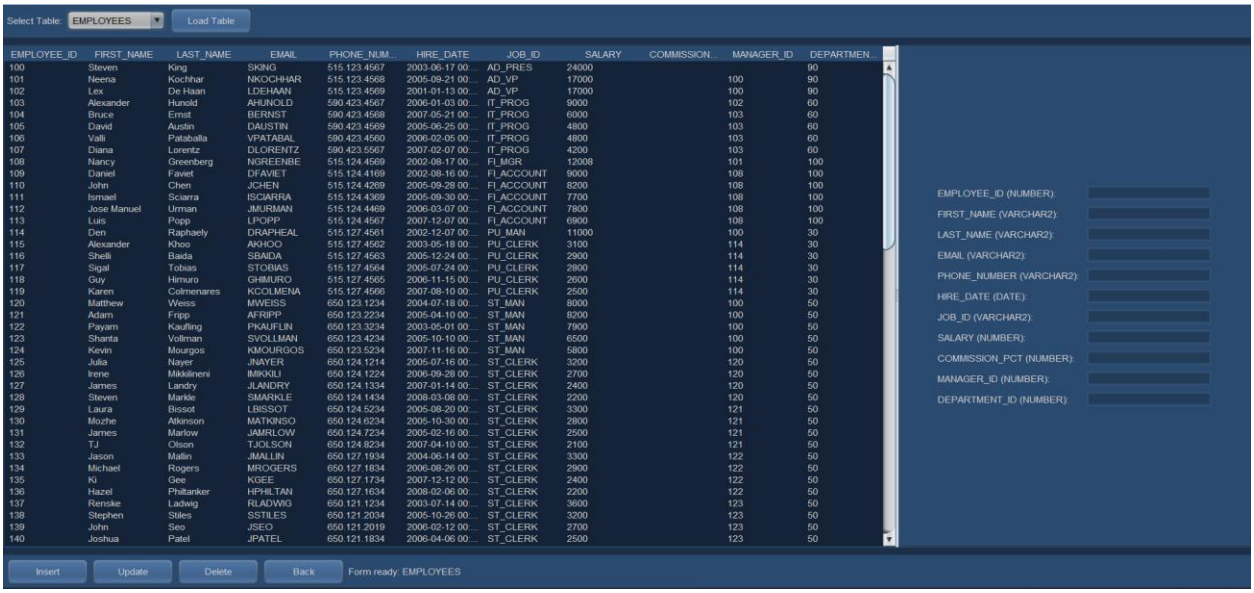
• **Database Integration:**

PreparedStatement, auto-commit mode

Class: CrudPanel

Purpose:

The CrudPanel allows users to select tables, view data dynamically, and perform insert, update, and delete operations with real-time feedback. Input fields are generated from table metadata, and row selection populates forms, ensuring an interactive and user-friendly interface. It integrates with the Crud class and uses a Connection for database operations.



Class: View

Purpose:

• **Database Interaction:** Acts as a bridge between the UI and the database using an active Connection.

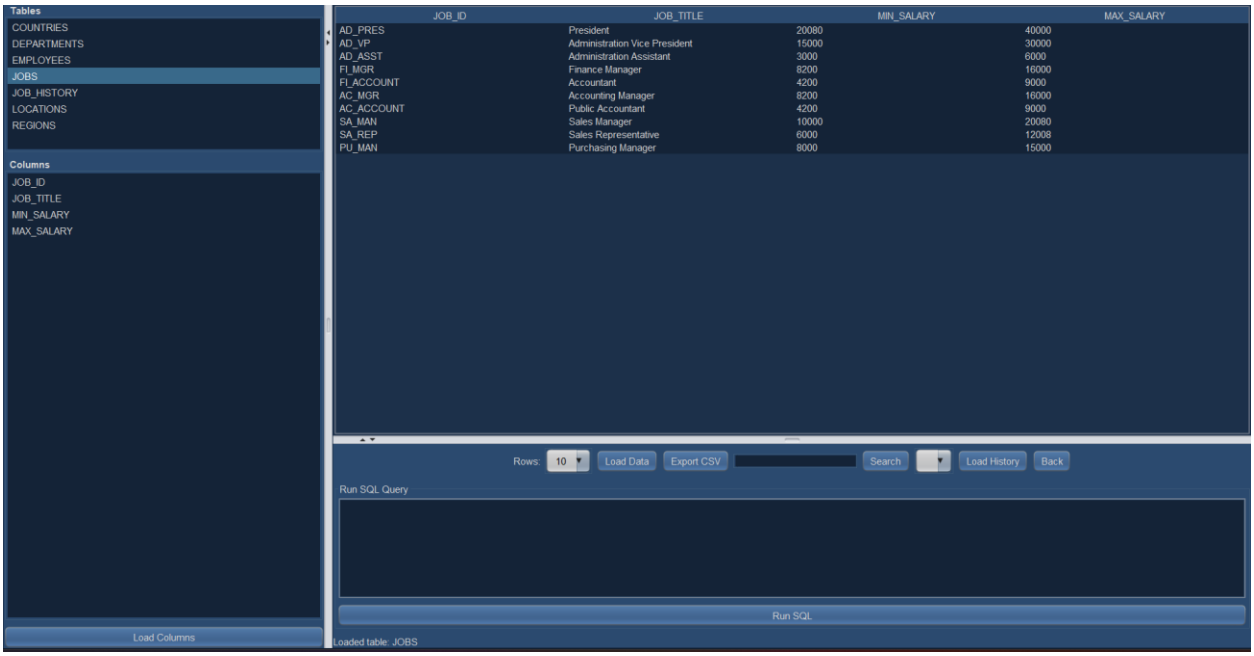
• **SQL Execution:** Uses SQLExecutor to run queries (runSQL) and retrieve query history.

• **Metadata Retrieval:**

- getTables() → Returns all table names in the connected database.

- getColumns(table) → Returns all column names for a given table.
  • **Data Loading:**
- loadData(table, cols, limit) → Loads table data with optional column selection and row limit.
- searchInData(data, keyword) → Filters a DataResult by a search keyword (case-insensitive).
  • **Data Export:** exportToCSV(data, file) → Saves a DataResult to a CSV file via CSVExporter.
  • **Integration:** Provides a unified API for retrieving, filtering, exporting, and executing SQL data from the database.

Class: ViewPanel
Purpose:
The **ViewPanel** connects to the **View** backend to load, search, execute SQL queries, and export table data, integrating seamlessly into the MainFramePanel's card layout. It features table and column navigation, dynamic data display using a table model, controls for filtering, row limits, query execution, CSV export, and a consistent dark-themed UI. All operations update dynamically and work directly with the **View** class for database interaction.



Class: Analytics
Purpose:
Facilitates database table inspection and data retrieval for analytics purposes. Provides methods to list database tables, load table data, and manage column metadata.

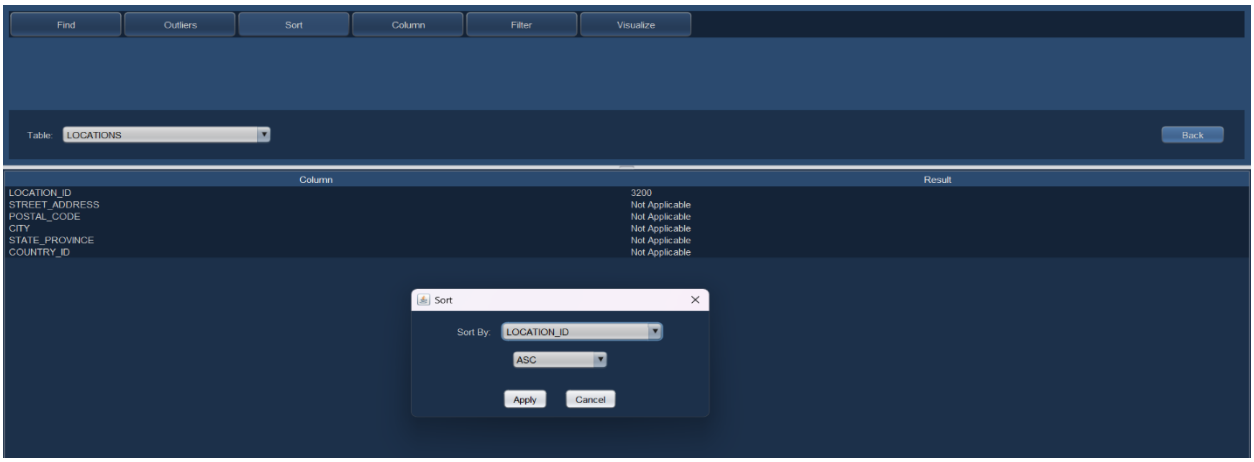
Key Features:

- **loadTables()** – Retrieves all table names from the connected database schema.
- **getTables()** – Returns the list of loaded table names.
- **loadTableData(String table)** – Fetches all rows and column metadata for a specified table, returning a TableData object.
- **getColumnTypes()** – Returns the data types of the loaded table's columns.
- **isNumericColumn(String name)** – Checks if a column type is numeric for analytics operations.
- **TableData inner class** – Encapsulates table information including column names, rows, and column types.

Class: AnalyticsPanel

Purpose:

The **AnalyticsPanel** provides a user-friendly interface for exploring, manipulating, and visualizing database tables. It supports operations like Find, Outliers, Sort, Column Management, Filter, and Visualization, with options to export results to CSV/TXT. The panel features a dark-themed, responsive UI and tracks original, working, and filtered data to maintain state during operations.



Class: Visualization

Purpose:

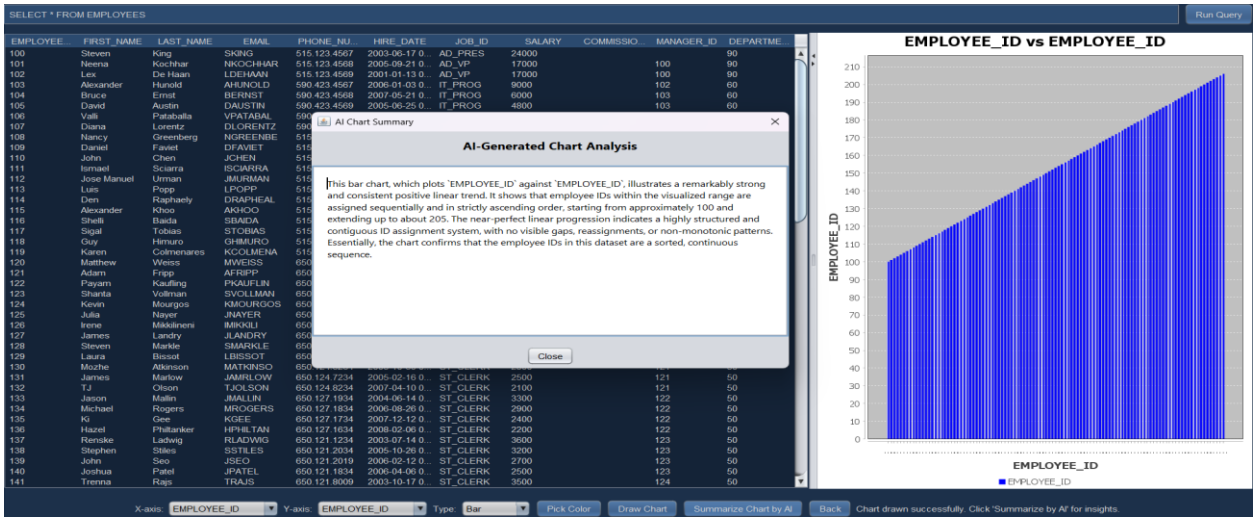
The Visualization class retrieves data from the database and generates charts using JFreeChart. It

separates charting logic from database queries, allowing safe and flexible visualization of SQL data.

Key Features:

- **Database Connection:** Uses conn (JDBC connection) to execute queries.
- **Data Storage:** Stores query results in a 2D array (data) and column names in columns for safe, in-memory processing.
- **Chart Customization:** chartColor allows setting and retrieving the color of generated charts.
- **Query Execution:** runQuery(String query) executes SELECT queries, stores results in memory, and closes ResultSet and Statement immediately.
- **Chart Generation:** generateChart(String type, String xCol, String yCol) supports Line, Bar, Pie, Scatter, Area, and Histogram charts using stored data; returns a ready-to-render JFreeChart object.
- **Color Methods:** setChartColor(Color color) and getChartColor() customize chart appearance.

Class: VisualizationPanel
Purpose:
The **VisualizationPanel** provides a GUI for executing SQL queries and generating charts from the results. It features a dark-themed interface with a scrollable result table, multiple chart types (Bar, Line, Pie, Scatter, Area, Histogram), axis and chart controls, and customization options. Integrated with the **Visualization** backend, it displays data and charts side by side within the MainFramePanel's card layout.



Future Improvements

Security

- Implement role-based access control
- Secure credential handling and session management

Database & Architecture

- Introduce DAO pattern for better separation of concerns
- Use connection pooling for improved performance
- Support dynamic schema switching

Analytics & Visualization

- Add advanced analytics (grouping, trends, calculated fields)
- Enhance charts with interactivity and multi-series support
- Enable chart and data export (PDF/PNG/Excel)

Query & Reporting

- Provide SQL syntax highlighting and auto-completion
- Include execution plan analysis for query optimization
- Support automated and scheduled reports

User Experience & Maintainability

- Improve UI customization and keyboard shortcuts
- Add comprehensive testing and centralized error handling
- Modernize architecture using REST services and scalable frontends

CONCLUSION

This project successfully delivers a comprehensive Oracle-based Swing application that integrates database connectivity, CRUD operations, analytics, and data visualization within a unified and user-friendly interface. The system follows a modular and well-structured design, ensuring clear separation between the user interface, business logic, and database operations.

By supporting dynamic querying, data analysis, and chart-based visualization, the application enhances data accessibility and decision-making capabilities. Its extensible architecture allows for future enhancements in security, scalability, analytics, and usability, making it suitable not only for academic purposes but also as a foundation for enterprise-level database management and business intelligence solutions.

Overall, the project demonstrates effective application of Java, JDBC, Swing, and database concepts, fulfilling its objectives and providing a strong platform for further development and modernization.