### **Modified Business Requirements Document**

#### **Main Objective**

To build an **offline application** (no internet required) that allows users to connect to any offline database. The application will:

* Enable users to ask natural language questions.
* Provide accurate responses from the database.
* Offer a high-quality user experience (UI/UX) with dynamic features.

#### **Functional Requirements**

**1. Login Page:**

* Admin users will input the following:
  + Database Type: Options include SQL, MySQL, or Oracle (via a dropdown list).
  + Database Connection String or Database Creation Script with sample data.
* Submit Button:
  + **Connection String Provided**: Application connects to the hosted database, analyzes it, and saves the connection details.
  + **Creation Script Provided**: Application creates the database, analyzes it, and saves both the creation script and connection details.

**2. Database Management (Admin Features):**

* Display all connected databases in a dropdown list.
* For each database, show:
  + Tables.
  + Columns.
  + Relationships between tables.
* Admin can:
  + Describe each table and column.
    - Example: Table "Emps" -> "Employees"; Column "Emp\_Name" -> "Employee Name."
  + Define/modify relationships if not properly set in the database.
  + Highlight lookup columns for the model to understand references.
* Save the database structure and its description for use by the LLM model.

**3. Query Dashboard (User Features):**

* Users can:
  + Select a database from a dropdown list.
  + Enter questions in a text box.
    - Example: "Show employees with salaries above $1000 who joined in the last 3 months."
  + Receive results with descriptive explanations:
    - Example: "This result shows employees earning more than $1000 who joined in the last 3 months."
  + Confirm or decline descriptions:
    - **Confirm**: Show results in a well-designed table or chart.
    - **Decline**: Suggest 3 similar questions for user selection.
* Users can:
  + Ask for insights (e.g., "Compare employee attendance for the last 2 months with 2023").
  + Export results and raw data to Excel.

**4. Insights and Visualizations:**

* Predefined dashboards to display insights with:
  + Multiple datasets.
  + Dynamic charts and tables.
* Future Phase: Add customization features for user preferences and saved settings.

**5. Dynamic Question Suggestions:**

* Suggest question formats using:
  + Predefined templates.
  + Sample data (if available) for contextual suggestions.
* If a question is unclear, provide guidance with examples and topics.

**6. Comparison Module (Admin Features):**

* Upload an Excel file containing columns:
  + Question.
  + Corresponding Query.
* Compare the results of:
  + Queries generated externally.
  + Queries generated by the application model.
* Ensure comparison ignores row/column order differences.

#### **Key Features**

**1. Offline Mode:**

* Fully functional without internet access.
* Use open-source models (e.g., Llama or DeepSeek) hosted locally.

**2. Deployment Environment:**

* Local server hosting for offline mode.
* Development can temporarily use online hosting for faster iterations.

**3. Database Flexibility:**

* Connect to any structured database: SQL, MySQL, or Oracle.
* No tuning required for specific databases.

**4. Roles and Permissions:**

* **Admins**:
  + Manage database connections and configurations.
  + View and edit database metadata.
* **Normal Users**:
  + Access the Query Dashboard.
  + Connect only to assigned databases.

**5. Data Export:**

* Export raw data or results (charts and tables) to Excel.

**6. Data Cleaning (Admin Feature):**

* Highlight missing values in datasets.
* Allow manual handling and editing by admins.

**7. Error Handling:**

* User-friendly error messages for issues like:
  + Incorrect database configurations.
  + Connection failures.
* Admin debugging tools:
  + Validate connection strings.
  + Highlight issues for resolution.

**8. Dashboard Design:**

* Modern and visually appealing (e.g., Metronic theme).
* Responsive and user-friendly with light/dark modes.

#### **Technical Recommendations**

**1. Model Selection:**

* Deep Seek

**2. Security:**

* Implement Role-Based Access Control (RBAC) for user permissions.
* Add future support for row-level security.

**3. Development Hosting:**

* Use online services during development for faster testing.
* Transition to offline environments for final deployment.

**4. FrontEnd:**

* To ensure that the implementation is modular, secure, scalable, performant, and has proper exception handling, here are some key considerations based on the document:
* **Modularity**: The backend architecture will prioritize efficiency and security, leveraging .NetCore's extensive libraries and frameworks. The WebAPI structure will be designed to ensure modularity and scalability, incorporating best practices for maintaining code quality and facilitating seamless integration with other services1.
* **Security**: Implement Role-Based Access Control (RBAC) for user permissions and add future support for row-level security2. This will ensure that only authorized users have access to specific data and functionalities.
* **Scalability**: The SQL DB Structure will define the schema, tables, relationships, and constraints necessary for the efficient storage and retrieval of application data. It will be designed to support the application's requirements, including scalability, performance, and security3.
* **Performance**: Indexing strategies will be employed to optimize query performance, while stored procedures and triggers will handle complex business logic3. Additionally, the front-end will use ApexCharts for visualization with modern Bootstrap for design, allowing for sleek and interactive data presentations4.
* **Exception Handling**: Implement robust error handling mechanisms to manage database connection issues, query execution failures, and other database-related errors. This will improve the application's reliability and user experience5. User-friendly error messages will be provided for issues like incorrect database configurations and connection failures

**5. BackEnd:**

* To ensure that the implementation is modular, secure, scalable, performant, and has proper exception handling, here are some key considerations based on the document:
* **Modularity**: The backend architecture will prioritize efficiency and security, leveraging .NetCore's extensive libraries and frameworks. The WebAPI structure will be designed to ensure modularity and scalability, incorporating best practices for maintaining code quality and facilitating seamless integration with other services1.
* **Security**: Implement Role-Based Access Control (RBAC) for user permissions and add future support for row-level security2. This will ensure that only authorized users have access to specific data and functionalities.
* **Scalability**: The SQL DB Structure will define the schema, tables, relationships, and constraints necessary for the efficient storage and retrieval of application data. It will be designed to support the application's requirements, including scalability, performance, and security3.
* **Performance**: Indexing strategies will be employed to optimize query performance, while stored procedures and triggers will handle complex business logic3. Additionally, the front-end will use ApexCharts for visualization with modern Bootstrap for design, allowing for sleek and interactive data presentations4.
* **Exception Handling**: Implement robust error handling mechanisms to manage database connection issues, query execution failures, and other database-related errors. This will improve the application's reliability and user experience5. User-friendly error messages will be provided for issues like incorrect database configurations and connection failures

|  |  |
| --- | --- |
| **Component** | **Details** |
| BackEnd | .NetCore v8 WebAPI Project |

**6. Application DB:**

* The application database will utilize SQL Server, initially hosted through Visual Studio as LocalDB.

7. WEBAPI (Backend) Structure:

* The WebAPI structure will be designed to ensure modularity and scalability, incorporating best practices for maintaining code quality and facilitating seamless integration with other services. The backend architecture will prioritize efficiency and security, leveraging .NetCore's extensive libraries and frameworks. As the application evolves, additional features and optimizations will be incrementally introduced, ensuring adherence to robust development methodologies and industry standards.

8. (FrontEnd) Structure:

The front-end design will need to be modern and eye-catching, leveraging the latest Bootstrap design principles to ensure a sleek, professional appearance similar to metronic style. enhanced by proper code refactoring to maintain high standards of code quality and readability. This approach will allow for the efficient incorporation of new features and ensure that the design remains at the forefront of modern web development trends.

The front-end structure will embrace a responsive and user-friendly approach, featuring intuitive navigation and dynamic content rendering to enhance user interaction. Blaozr's integration with metronic design will facilitate a cohesive and modern user experience, ensuring consistency across various devices and screen sizes. The front-end will seamlessly interact with the back-end, supported by robust API endpoints and efficient data processing. We will use ApexCharts for visualization with modern Bootstrap for design, allowing for sleek and interactive data presentations. Additionally, the chart has features to export, providing flexibility in data handling. Future enhancements will focus on customizable dashboards and visualizations, enhanced query learning from user history, integration with data cleaning and transformation tools, and row-level security for granular data access control.

9. Common Project

The Common Project will include the classes and models used across different projects, ensuring consistent development practices throughout the application. This approach will enhance collaboration among teams and allow for efficient resource allocation and the swift incorporation of new features.

10. SQL DB Structure

:

The SQL DB Structure will define the schema, tables, relationships, and constraints necessary for the efficient storage and retrieval of application data. It will be designed to support the application's requirements, including scalability, performance, and security. The schema will encompass various entities, such as users, roles, permissions, and application-specific data, ensuring data normalization and integrity. Indexing strategies will be employed to optimize query performance, while stored procedures and triggers will handle complex business logic.

# Missing Enhancements and Additions

Considering the application's ability to connect to different types of databases, the following enhancements and additions are suggested:

* Database Abstraction Layer: Implement a database abstraction layer to facilitate seamless integration with different database types. This layer will handle database-specific queries and operations, ensuring consistency across various database systems.
* Dynamic Connection Strings: Incorporate dynamic connection strings to allow the application to connect to different databases based on configuration settings or user preferences. This will enable flexibility in database selection and connectivity.
* Enhanced Error Handling: Implement robust error handling mechanisms to manage database connection issues, query execution failures, and other database-related errors. This will improve the application's reliability and user experience.
* Database Migration Tools: Include tools and scripts for database migration and versioning, allowing for smooth transitions and upgrades between different database systems. This will ensure data consistency and integrity during migrations.

#### **Future Enhancements**

* Customizable dashboards and visualizations.
* Enhanced query learning from user history.
* Integration with data cleaning and transformation tools.
* Row-level security for granular data access control.