SQL Query Runner - Technical Documentation

System Overview

The SQL Query Runner is a React-based web application that allows users to:

1. Execute SQL queries against mock data
2. Upload and query CSV files
3. Save and manage frequently used queries
4. View historical query executions
5. Browse results with pagination

Database Schema Explanation

Entities:

1. **USER**
    - Stores application user information
    - Relationships:
        - One-to-many with QUERY (a user can create many queries)
        - One-to-many with SAVED_QUERY (a user can save many queries)
2. **QUERY**
    - Core entity storing SQL query text
    - Relationships:
        - Many-to-one with USER
        - One-to-many with QUERY_HISTORY (each execution is logged)
        - One-to-many with RESULT (each execution generates results)
3. **SAVED_QUERY**
    - Extension of QUERY with additional metadata
    - Relationships:
        - Many-to-one with USER
        - One-to-one with QUERY (each saved query references one base query)
4. **QUERY_HISTORY**

    - Audit log of query executions
    - Stores performance metrics and timestamps
5. **RESULT**
    - Stores query execution outputs
    - Relationships:
        - Many-to-one with QUERY
        - Many-to-one with CSV_DATA (results may reference uploaded data)
6. **CSV_DATA**
    - Stores metadata about uploaded CSV files
    - Relationships:
        - One-to-many with RESULT (a CSV can be used in many queries)

Technical Decisions

1. Frontend Architecture

- **Component Structure**:
    - Container components: App, QueryRunner
    - Presentational components: QueryEditor, ResultTable
    - State management: React hooks (useState, useContext)

2. Mock Data Generation

- Implemented through the `generateMockResult()` function
- Uses query text analysis to determine appropriate mock data
- Generates consistent datasets for pagination demonstration

3. CSV Processing

- **PapaParse** library chosen because:
    1. Handles large files efficiently
    2. Provides streaming interface
    3. Supports complex CSV formats
- Processing workflow:
    1. File validation (type, size)
    2. Streaming parse
    3. Column structure analysis
    4. Memory-efficient storage

4. State Management

- Application state divided into:
    - UI state (current tab, theme)
    - Data state (queries, results)
    - Session state (history, saved items)
- Context API used for theme management

5. Pagination Implementation

- Client-side pagination for:
    - Performance with large result sets
    - Consistent user experience
- Key parameters:
    - `currentPage`: Tracks visible page
    - `itemsPerPage`: Fixed at 10 items
    - `totalPages`: Calculated from result length

Challenges and Solutions

1. Large CSV Handling

Challenge: Browser memory limitations with large files
Solution:

- Implemented streaming CSV parsing
- Limited preview to first 1000 rows
- Added file size validation (max 10MB)

2. Query Performance

Challenge: Mock data generation delays
Solution:

- Optimized data generation algorithms
- Added loading states
- Implemented memoization for repeated queries

3. State Synchronization

Challenge: Coordinating query, results, and history states
Solution:

- Created unified state management structure
- Implemented atomic state updates
- Added error boundaries for failed queries

System Limitations

1. **Persistence**:
    - Currently uses localStorage (volatile)
    - Production version would require backend database
2. **Security**:
    - No query validation/sanitization
    - Not suitable for real database connections
3. **Performance**:
    - Large result sets (>10,000 rows) may cause UI lag

Future Enhancements

1. **Backend Integration**:
    - Node.js service for query execution
    - MongoDB for data persistence
2. **Advanced Features**:
    - Query validation and syntax highlighting
    - Result visualization (charts, graphs)
    - Multi-table JOIN support
3. **Collaboration**:
    - Query sharing between users
    - Commenting on saved queries