SQL CODING STANDARD RULE DOCUMENT

INTRODUCTION

This document outlines a set of coding standards and best practices for writing SQL (Structured Query Language) code. These guidelines aim to ensure consistency, improve code quality, and enhance maintainability across database projects.

Table of Contents

1. Naming Conventions
- 1.1. Database Objects
- 1.2. Columns
- 1.3. Aliases

2. Formatting and Indentation
- 2.1. Indentation
- 2.2. Line Length
- 2.3. Whitespace
- 2.4. Capitalization

3. SQL Statements
- 3.1. SELECT Statements
- 3.2. INSERT Statements
- 3.3. UPDATE Statements
- 3.4. DELETE Statements
- 3.5. JOINs
- 3.6. Subqueries
- 3.7. Common Table Expressions (CTEs)
- 3.8. Comments

4. Data Types
- 4.1. Choosing Appropriate Data Types
- 4.2. Avoid Using Deprecated Data Types

5. Indexes
- 5.1. Proper Use of Indexes
- 5.2. Avoid Over-Indexing
- 5.3. Index Naming

6. Stored Procedures and Functions
- 6.1. Naming Conventions
- 6.2. Error Handling
- 6.3. Input Validation
- 6.4. Avoid Large Stored Procedures

7. Security
7.1. SQL Injection Prevention
7.2. Data Encryption
7.3. Proper Permissions and Roles

8. Performance Optimization
8.1. Query Performance
8.2. Transaction Management
8.3. Avoid Using SELECT *
8.4. Limit Data Retrieval

1. Naming Conventions

1.1. Database Objects
- Use descriptive, lowercase names for tables, views, indexes, and stored procedures.
- Separate words in names with underscores (e.g., `user_profiles`).

1.2. Columns
- Use lowercase names for columns.
- Avoid using reserved words as column names.
- Use meaningful names that reflect the data they store.

1.3. Aliases
- Use aliases for table and column names in queries to improve readability.
- Aliases should be meaningful and concise.

2. Formatting and Indentation

2.1. Indentation
- Use consistent indentation for SQL statements (e.g., two or four spaces).
- Indent subqueries and clauses (SELECT, FROM, WHERE, JOIN) for readability.

2.2. Line Length
- Limit lines to a reasonable length (e.g., 80-100 characters) to improve code readability.

2.3. Whitespace
- Use whitespace to separate keywords, operators, and expressions for clarity.

2.4. Capitalization
- Use uppercase for SQL keywords (e.g., SELECT, FROM, WHERE).
- Use lowercase for table and column names.

3. SQL Statements

3.1. SELECT Statements
- Explicitly list columns instead of using SELECT
- Avoid using subqueries when simple JOINs can achieve the same result.

3.2. INSERT Statements

- Specify column names when performing INSERT operations.

3.3. UPDATE Statements

- Always include a WHERE clause to avoid unintended updates to all rows.

3.4. DELETE Statements

- Use DELETE statements with caution and always include a WHERE clause.

3.5. JOINs

- Use appropriate JOIN types (INNER, LEFT, RIGHT, FULL) based on the desired result.
- Avoid excessive nesting of JOINs.

3.6. Subqueries

- Use subqueries judiciously for complex queries.
- Optimize subqueries for performance.

3.7. Common Table Expressions (CTEs)

- Use CTEs to simplify complex queries and improve readability.
- CTEs should have descriptive names.

3.8. Comments

- Add comments to explain complex queries, logic, and any potential issues.

4. Data Types

4.1. Choosing Appropriate Data Types

- Select data types based on the nature of the data and storage requirements.
- Avoid using VARCHAR for storing numeric data, use appropriate numeric data types instead.

4.2. Avoid Using Deprecated Data Types

- Avoid using deprecated data types or features.

5. Indexes

5.1. Proper Use of Indexes

- Use indexes to improve query performance on columns that are frequently used for filtering or joining.
- Monitor and maintain indexes to ensure optimal performance.

5.2. Avoid Over-Indexing

- Avoid creating too many indexes, as this can slow down INSERT and UPDATE operations.

5.3. Index Naming

- Use meaningful names for indexes that reflect their purpose.

6. Stored Procedures and Functions

6.1. Naming Conventions

- Use descriptive names for stored procedures and functions.
- Prefix stored procedures with "sp_" and functions with "fn_" for clarity.

6.2. Error Handling

- Implement error handling and provide meaningful error messages in stored procedures and functions.

6.3. Input Validation

- Validate input parameters to stored procedures and functions to prevent SQL injection.

6.4. Avoid Large Stored Procedures

- Keep stored procedures and functions concise and focused on specific tasks.

7. Security

7.1. SQL Injection Prevention

- Use parameterized queries or prepared statements to prevent SQL injection.
- Sanitize user inputs before using them in SQL queries.

7.2. Data Encryption

- Encrypt sensitive data, such as passwords and personal information.

7.3. Proper Permissions and Roles

- Grant minimal necessary permissions to database users and roles.
- Follow the principle of least privilege.

8. Performance Optimization

8.1. Query Performance

- Optimize queries for performance by analyzing execution plans.
- Avoid using functions or operations that hinder query optimization.

8.2. Transaction Management

- Use transactions appropriately to ensure data consistency.
- Keep transactions short and avoid long-running transactions.

8.3. Avoid Using SELECT \

- Minimize data retrieval by selecting only the required columns.

8.4. Limit Data Retrieval

- Use pagination and LIMIT/OFFSET for large result sets.