eNoah Coding Standards — SQL Version: 1.0

Scope: This document defines ANSI SQL coding standards based on the Developer Code Standards (Full Extended) and enhanced with eNoah conventions. It applies to all SQL code written for relational database systems, including DDL, DML, DCL, and TCL statements.

1. Purpose & Scope - Ensure readable, consistent, and portable SQL code across platforms. - Promote secure, maintainable, and optimized SQL scripts for applications and reports. - Apply to stored procedures, scripts, and embedded SQL in application logic.

2. Naming Conventions - Use snake_case for all database identifiers: tables, columns, views, indexes, and constraints. - Table names should be singular and descriptive: employee, department. - Prefix constraints and indexes appropriately: - Primary Key: pk_table - Foreign Key: fk_child_parent - Unique: uq_table_column - Index: idx_table_column - Avoid abbreviations unless well-known (cust for customer). - Use UPPERCASE for SQL keywords and lowercase for object names. - Use clear, descriptive names for variables and parameters in stored routines.

3. File Organization - Organize scripts into logical folders: ddl/, dml/, functions/, procedures/, migrations/. - Version migration scripts using timestamps or incremental numbers: 20251010_create_employee_table.sql. - Include headers in each script with author, purpose, version, and date. - Store configuration and environment-specific SQL outside of source-controlled code.

4. SQL Syntax & Structure - Use ANSI-standard JOIN syntax, not legacy comma joins. - Always include explicit column lists in SELECT and INSERT statements; avoid SELECT *. - Use meaningful aliases and include the AS keyword: e.first_name AS employee_name. - Maintain consistent clause ordering: SELECT → FROM → JOIN → WHERE → GROUP BY → HAVING → ORDER BY. - Use parentheses in complex conditions for clarity. - Break long queries across multiple lines with indentation for logical blocks. - Example: SELECT e.id, e.first_name, d.department_name FROM employee e INNER JOIN department d ON e.dept_id = d.id WHERE e.status = 'ACTIVE' ORDER BY e.last_name;

5. Comments & Documentation - Use -- for single-line comments and /* ... */ for multi-line comments. - Comment intent, not syntax. - Provide detailed comments for complex logic, stored procedures, and triggers. - Each script must include a file header: -- == -- Script: create_employee_table.sql -- Purpose: Defines structure for employee table -- Author: John Doe -- Date: 2025-10-10 -- Version: 1.0 -- ==

6. Indentation & Formatting - Use 2 or 4 spaces per indentation level; do not use tabs. - Align keywords vertically and use new lines for major clauses. - Keep line length ≤ 120 characters. - Align column lists in SELECT or INSERT vertically for readability. - Example: INSERT INTO employee (id, first_name, last_name, hire_date) VALUES (101, 'John', 'Smith', CURRENT_DATE);

7. Data Types & Constraints - Use standard ANSI SQL data types (INTEGER, DECIMAL, VARCHAR, DATE, TIMESTAMP). - Choose appropriate precision and scale for numeric types. - Avoid vendor-specific extensions unless required. - Always define PRIMARY KEYs, FOREIGN KEYs, and constraints explicitly. - Enforce data integrity using CHECK constraints. - Use DEFAULT values for non-nullable columns where appropriate.

8. Query Optimization & Performance - Analyze query plans (EXPLAIN) to identify bottlenecks. - Avoid subqueries when a JOIN can achieve the same result efficiently. - Use WHERE clauses to limit result sets early. - Avoid functions on indexed columns in WHERE clauses. - Use appropriate indexing strategies for

frequently queried columns. - Use EXISTS instead of IN for correlated subqueries where possible. - Remove redundant joins and ensure filters are selective. - Periodically review indexes to remove unused ones.

9. Error Handling & Transactions - Always wrap multiple DML statements in transactions: BEGIN / COMMIT / ROLLBACK. - Use error-handling constructs available in the specific SQL dialect (e.g., TRY/CATCH or EXCEPTION blocks). - Example: BEGIN TRANSACTION; UPDATE accounts SET balance = balance - 500 WHERE id = 101; UPDATE accounts SET balance = balance + 500 WHERE id = 202; COMMIT; - Use SAVEPOINTS for partial rollbacks when necessary. - Log errors and rollbacks in audit or error tables where appropriate.

10. Security Practices - Apply the Principle of Least Privilege: limit database access to only necessary roles. - Use parameterized queries or prepared statements to prevent SQL injection. - Validate all input parameters before executing queries. - Avoid dynamic SQL unless absolutely necessary; sanitize all dynamic inputs. - Mask or encrypt sensitive data (e.g., personally identifiable information). - Never hard-code credentials in SQL scripts. - Implement row-level and column-level security policies when supported.

11. Stored Procedures & Functions - Keep procedures small, modular, and well-documented. - Use consistent naming for stored objects: sp_calculate_bonus, fn_get_employee_age. - Avoid business logic duplication across stored procedures. - Validate all input parameters inside the procedure. - Include error-handling sections and meaningful return codes. - Comment procedure purpose, parameters, and examples of usage.

12. Control Flow - Use CASE expressions instead of nested IFs where possible. - Use COALESCE and NULLIF for null-safe operations. - Example: SELECT id, COALESCE(phone, 'N/A') AS phone_display FROM customer;

13. Security & Compliance - Encrypt connections and data where supported. - Audit DDL and DML operations for sensitive tables. - Mask confidential data in logs and views. - Ensure GDPR/PII compliance when dealing with user data. - Use secure schema separation for environments (e.g., staging vs production).

14. File Execution & Deployment - Test all scripts in a staging environment before production deployment. - Include rollback scripts for destructive operations. - Maintain version history in change logs or migration tables. - Avoid dependent scripts without clear sequencing.

15. Testing & Validation - Validate schema and constraints after migrations. - Test stored procedures and functions with edge cases and invalid inputs. - Include EXPLAIN plan analysis in query reviews. - Conduct peer code reviews for all SQL scripts before merging to main branch.

Appendix A — Quick Checklist (for static analysis) - No SELECT * usage. - All scripts include file headers and comments. - Transactions used for multi-step DML. - JOIN syntax is explicit and correct. - Data types are standardized (INTEGER, VARCHAR, etc.). - Proper constraint and index prefixes (pk_, fk_, idx_). - No dynamic SQL without input sanitization. - Performance validated using EXPLAIN or equivalent.

Appendix B — Examples -- Table creation CREATE TABLE department (id INTEGER PRIMARY KEY, name VARCHAR(100) NOT NULL, location VARCHAR(100));

```
-- Query with join and aliasing
SELECT
    e.first_name,
    e.last_name,
    d.name AS department_name
FROM employee e
INNER JOIN department d ON e.dept_id = d.id
WHERE e.status = 'ACTIVE';

-- Example transaction
BEGIN TRANSACTION;
UPDATE products SET stock = stock - 10 WHERE id = 5;
UPDATE inventory_log SET last_updated = CURRENT_TIMESTAMP WHERE product_id = 5;
COMMIT;
```

End of document.