eNoah Coding Standards — MySQL Version: 1.0

Scope:

This document defines MySQL-specific coding standards aligned with ANSI SQL practices and the provided Developer Code Standards (extended). It is tailored for MySQL 5.7 / 8.x features, InnoDB transactional behavior, JSON support, and MySQL-specific administration and deployment practices.

1. Purpose & Scope - Maintainable, secure, and high-performance MySQL schema, queries, and routines. - Intended for application developers, DBAs, and CI/CD pipelines validating SQL artifacts.

2. Naming Conventions - Use snake_case for all identifiers (tables, columns, indexes, constraints, triggers, routines). - Use singular nouns for table names (customer, order). Be consistent across the project. - Prefix constraint and index names with type and table: pk_orders, fk_order_customer, idx_orders_customer_id. - Keep names lowercase to avoid cross-platform case-sensitivity issues. - Prefix temporary objects with tmp_: tmp_order_import. - Use meaningful column names; avoid generic names like data or value. - Avoid reserved words; if necessary, quote with backticks (`identifier`) consistently.

3. File Organization & Versioning - Separate DDL, DML, routines, seeds, and migrations into folders: ddl/, dml/, routines/, seeds/, migrations/. - Name migration files with monotonically increasing version or timestamp prefixes: 20251010_001_create_customer.sql or 001_create_customer.sql. - Keep environment-specific configs out of version control; use templates and secret managers for credentials. - Use a migration tool (Flyway, Liquibase, pt-online-schema-change) and ensure scripts are idempotent or reversible.

4. SQL Formatting & Style - SQL keywords UPPERCASE, identifiers lowercase: SELECT id, first_name FROM customer WHERE active = 1; - Indentation: 2 or 4 spaces (team decision); never tabs. - Place SELECT columns on new lines for long lists. - Use explicit JOIN ... ON syntax; avoid comma joins. - Always specify column lists in SELECT and INSERT statements; avoid SELECT * in production. - Use AS for aliases: u.id AS user_id. - Keep line length <= 120 characters; break long expressions logically.

5. Data Types & Schema Design - Use InnoDB for transactional tables; MyISAM is deprecated for transactional needs. - Use utf8mb4 and utf8mb4_unicode_ci (or utf8mb4_general_ci if justified) for charset and collation. - Choose appropriate column types and lengths (e.g., VARCHAR(255) when needed, VARCHAR(320) for email). - Use proper date/time types and store timestamps in UTC (TIMESTAMP / DATETIME with UTC handling at application layer). - Avoid storing comma-separated lists; normalize with junction tables. - Prefer BOOLEAN / TINYINT(1) for flags rather than CHAR(1). - Use JSON column type for semi-structured data; add generated columns for values to index when required. - Define PRIMARY KEYs and FOREIGN KEY constraints to enforce integrity.

6. Indexing Strategy - Index columns used in WHERE, JOIN, ORDER BY, and GROUP BY according to query patterns. - Use composite indexes in the order matching queries (left-most prefix rule). - Avoid over-indexing; each index adds write overhead. - Use covering indexes to avoid accessing table rows when possible. - Use INVISIBLE indexes (MySQL 8+) to test/remove indexes safely. - Avoid functions on indexed columns in WHERE clauses (prevents index usage); use generated/persisted columns if normalization of data is required.

7. Query Best Practices - Avoid SELECT *; list only required columns. - Use parameterized queries/prepared statements in application code to prevent SQL injection. - Prefer set-based operations over row-by-row processing; avoid cursors unless necessary. - Use EXISTS instead of IN for subqueries when appropriate for performance with correlated subqueries. - Use window functions (ROW_NUMBER,

RANK, PARTITION BY) for ranking and partitioned aggregations instead of correlated subqueries. - Use LIMIT/OFFSET or keyset pagination for large result sets; always include ORDER BY for deterministic pagination. - Test queries with EXPLAIN and include plan checks in CI pipelines for critical queries. - Avoid DISTINCT as a band-aid for duplicate data; fix data or joins producing duplicates instead.

8. Transactions & Error Handling - Wrap multi-statement updates in transactions (START TRANSACTION; ... COMMIT; ROLLBACK;). - Use appropriate isolation levels; avoid long-running transactions to minimize locking. - In stored procedures, declare handlers for exceptions and perform rollback: DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; END; - Use SAVEPOINTS for partial rollbacks in complex procedures. - Check and handle warnings with SHOW WARNINGS during development and test phases. - Log meaningful error messages and context; do not swallow exceptions silently.

9. Stored Routines, Procedures & Triggers - Keep stored routines focused and under 200 lines; split complex logic into smaller routines. - Use DELIMITER when creating routines and triggers in scripts. - Document routine signatures (parameters, return, side effects) in header comments. - Avoid using DEFINER=root for routines; set proper DEFINER/INVOKER security contexts. - Use triggers sparingly; prefer application logic or scheduled jobs for complex processing. - Test routines with unit/integration tests where possible.

10. Security Practices - Principle of Least Privilege: create dedicated MySQL users/roles for applications with minimal required privileges. - Avoid using root for application connections; restrict SUPER, FILE, PROCESS, and GRANT OPTION privileges. - Use strong authentication plugin (caching_sha2_password for MySQL 8+) and enforce password policies. - Use TLS for client-server connections; enforce require_secure_transport where applicable. - Configure secure_file_priv to restrict data import/export locations. - Do not build SQL by concatenating untrusted input; whitelist and validate inputs for any dynamic SQL. - Store sensitive secrets (DB credentials) in secret managers; never hard-code in scripts. - Enable and monitor audit logs and slow query logs; mask sensitive data in logs. - Encrypt sensitive columns at application layer or use MySQL Enterprise TDE where required by compliance.

11. Performance & Operations - Regularly run ANALYZE TABLE and OPTIMIZE TABLE as part of maintenance depending on workload. - Maintain up-to-date statistics for the optimizer. - Monitor InnoDB buffer pool usage, slow query log, table locks, and replication lag. - Use connection pooling at application layer; tune max_connections and thread_cache_size. - Consider partitioning for very large tables only after benchmarking and understanding query patterns. - For large schema changes, use online schema change tools (pt-online-schema-change, gh-ost) to minimize downtime. - Use batching for large DML operations to reduce lock pressure and logging overhead.

12. Backups & Disaster Recovery - Use logical (mysqldump) and physical backups (Percona XtraBackup) as appropriate. - Test backups regularly by performing restore drills in staging environments. - Keep backup retention and encryption policies aligned with organizational requirements. - Document rollback procedures for schema changes and major data migrations.

13. Character Set, Collation & Internationalization - Use utf8mb4 for all text columns to support full Unicode (including emojis). - Define collation explicitly in DDL where string comparison ordering matters. - Be consistent with character set and collation across database, tables, and connections.

14. Replication & High Availability - Understand row-based vs statement-based vs mixed replication modes; prefer row-based for accuracy in many cases. - Monitor replication lag and handle drift with checksums (pt-table-checksum) when using replication. - Secure replication channels and authentication;

avoid using root for replication users. - Document failover procedures and test them periodically.

15. CI/CD, Reviews & Testing - Include EXPLAIN/EXPLAIN ANALYZE checks for heavy queries in CI pipelines. - Lint SQL scripts with tools (sqlfluff, linter-specific rules) and run tests for stored routines. - Code review database changes; review indexes, schema, and migration scripts as part of PRs. - Maintain automated tests for data migrations and stored procedure behavior.

16. Comments & Documentation - Add file headers in scripts: purpose, author, date, migration id, and brief description. - Use -- for single-line comments and /* ... */ for blocks. Comment intent, not obvious mechanics. - Document decisions such as engine choices, partitioning strategy, and index rationale in DDL headers or PRs. - Maintain an architecture-level document listing critical tables and their purpose.

17. Miscellaneous Best Practices - Use GENERATED columns for computed values and to support indexing of expressions. - Prefer surrogate integer primary keys (BIGINT UNSIGNED) unless natural keys are stable and small. - Use ENUM sparingly — prefer lookup tables for extensibility. - Regularly review and remove unused indexes and obsolete columns. - Use INVISIBLE indexes to test index removal without immediate drop impact (MySQL 8+).

Appendix A — Quick Checklist (for code analysis) - No SELECT * in production queries. - Parameterized queries only; no concatenated user input. - Transactions for multi-statement mutations. - Proper naming for constraints and indexes (pk_, fk_, idx_). - Stored routines have error handlers and documentation. - Charset is utf8mb4 and collation defined. - Indexes analyzed and EXPLAIN included for slow/critical queries. - Migration files versioned and idempotent where possible.

Appendix B — Examples (MySQL-specific) -- Avoid SELECT * SELECT id, title, author_id FROM books WHERE status = 'available';

```
-- Upsert (INSERT ... ON DUPLICATE KEY UPDATE)
INSERT INTO users (email, name) VALUES (?, ?)
ON DUPLICATE KEY UPDATE name = VALUES(name);

-- Transaction with handler
DELIMITER $$
CREATE PROCEDURE transfer_funds(IN p_from INT, IN p_to INT, IN p_amount DECIMAL(10,2))
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
  END;

START TRANSACTION;
  UPDATE accounts SET balance = balance - p_amount WHERE id = p_from;
  UPDATE accounts SET balance = balance + p_amount WHERE id = p_to;
  COMMIT;
END$$
DELIMITER ;

-- Generated column example
CREATE TABLE users (
  id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(320) NOT NULL,
  normalized_email VARCHAR(320) GENERATED ALWAYS AS (LOWER(email)) VIRTUAL,
  INDEX idx_users_normalized_email (normalized_email)
);

-- JSON indexed access (use generated column for indexing)
CREATE TABLE events (
  id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  payload JSON,
  event_type VARCHAR(50) GENERATED ALWAYS AS (JSON_UNQUOTE(JSON_EXTRACT(payload, '$.type'))) VIRTUAL,
```

```
    INDEX idx_events_type (event_type)
);
```

End of document.