Test Strategy for standalone database

/Project name/

[Date]

1. Objective

Ensure the standalone database is functional, reliable, secure, and performs as expected under different conditions.

2. Scope

- Database schema validation
- Data integrity
- DDL, DML, DQL, DCL, TCL statements
- Performance testing
- Security testing
- Backup & recovery testing

3. Testing Types & Approach

3.1 Functional Testing

Schema Validation

- Verify tables, columns, data types, constraints (PK, FK, Unique, Not Null, Default, Check, Auto-Increment).
- Make sure proper relationships between tables are implemented.

Data definition statements and Operations

- Ensure CREATE, ALTER, DROP operations work as expected.
- Validate stored procedures, triggers, and functions.

Data Integrity Testing

- Check if data remains consistent after INSERT, UPDATE, DELETE operations.
- Test referential integrity (foreign key constraints).
- Retrieve data using SELECT queries with filtering, sorting, aggregations, grouping, pagination, joining and uniting tables, subqueries etc. and make sure results meet expectations.

3.2 Performance Testing

Query Performance

- Measure execution time for SELECT, INSERT, UPDATE, DELETE queries.
- Test with large datasets to identify slow queries.

Index Performance

- Validate effectiveness of indexes in optimizing queries.

Concurrency Testing

- Simulate multiple users accessing the database to detect deadlocks.

3.3 Security Testing

User Access Control

- Verify role-based access and permissions (e.g., read-only, admin).

SQL Injection Prevention

- Test for SQL injection vulnerabilities in queries and stored procedures.

Data Encryption

- Validate encryption for sensitive data.

3.4 Backup & Recovery Testing

Backup Verification

- Check if backup and recovery strategy has been implemented.

Disaster Recovery

- Restore database from backups and check data consistency.

4. Test Data Strategy

- Types of Data Used: Synthetic randomized test data
- Data Volume: Sufficient to simulate real-world scenarios.
- Data Management: Periodic cleanup of test data to avoid cluttering the database.

5. Entry & Exit Criteria

5.1 Entry Criteria

- Test environment setup is complete.
  Database schema is finalized.
  Test data is prepared.
  Test cases are prepared.

5.2 Exit Criteria

- All test cases have been executed.
- No Critical or High severity defects remain open.
- Test results are documented.

6. Defect Management

- Defect Tracking Tool: Jira
- Defect Life Cycle: New → Assigned → In Progress → Resolved → Verified → Closed
- Defect Severity Levels:
  - Critical – Database crashes, data corruption, architectural gaps
  - High – Incorrect query results, missing data, sufficient performance leakage
  - Medium – insufficient performance leakage, minor inconsistencies
  - Low – Misprints, suggestions for improvement

7. Roles & Responsibilities

Role	**Responsibility**
Test Manager	Defines strategy, oversees execution
Database Tester	Writes and executes test cases, logs defects
Performance Tester	Runs query optimization and load tests
Security Tester	Conducts security audits, SQL injection tests
Developers	Fix defects, optimize database queries

8. Schedule & Timeline

Phase	**Timeline**
Test Planning	Week 1
Environment Setup	Week 2
Test Case Development	Weeks 3-4
Test Execution	Weeks 5-7

Defect Fixing & Retesting	Weeks 6-8
Test Closure & Reporting	Week 9

9. Test Environment & Tools

- Database: MySQL
- Tools: SQL scripts, MySQL Workbench, DBeaver, Jira for bug tracking

10. Reporting & Metrics

- Test Execution Report – Number of test cases passed/failed.
- Defect Report – Number of defects found, resolved, and open.
- Performance Report – Query execution times and system resource usage.
- Coverage Report – Percentage of requirements covered by tests.