







# Database Testing

Nov 13, 2025

Speakers:

- 22120252 - Giang Đức Nhật
- 22120368 - Phan Thanh Tiến
- 22120370 - Nguyễn Bùi Vương Tiến
- 22120371 - Lý Trọng Tín

# Table of contents

- 1 Tổng quan về Database Testing 
- 2 Các loại Kiểm thử Cơ sở dữ liệu 
- 3 Quy trình Kiểm thử CSDL 
- 4 Kỹ thuật Kiểm thử CSDL 
- 5 Thách thức trong Kiểm thử CSDL 
- 6 Công cụ Kiểm thử CSDL 

# Tổng quan

# Giới thiệu về Kiểm thử Cơ sở dữ liệu

- **Cơ sở dữ liệu (CSDL)**

- CSDL quan hệ: Là một tập hợp dữ liệu có cấu trúc được lưu trữ và quản lý bởi Hệ quản trị cơ sở dữ liệu (DBMS). Ví dụ: MySQL, PostgreSQL, SQL Server, Oracle.
- CSDL NoSQL: Là loại CSDL không sử dụng mô hình quan hệ truyền thống, thường dùng để lưu trữ dữ liệu phi cấu trúc hoặc bán cấu trúc. Ví dụ: MongoDB, Cassandra, Redis.

- **Kiểm thử CSDL**

- Là quá trình verify và validate chất lượng, chức năng, hiệu suất và bảo mật của CSDL.
- Đảm bảo việc lưu trữ, truy xuất và quản lý dữ liệu hoạt động chính xác, hiệu quả và an toàn.

# Mục đích

- **Database Testing** là quá trình verify (xác minh) và validate (xác thực) chất lượng, chức năng, hiệu suất và bảo mật của hệ thống CSDL.
- **So với UI Testing:**
  - UI Testing tập trung vào những gì người dùng thấy (giao diện, bố cục, tương tác).
  - Database Testing tập trung vào dữ liệu (logic xử lý, tính toán), đảm bảo dữ liệu chính xác, toàn vẹn, an toàn.
- **Vai trò trong quy trình phát triển:**
  - Đảm bảo tính toàn vẹn và chính xác của dữ liệu.
  - Ngăn chặn mất mát hoặc hỏng hóc dữ liệu.
  - Tối ưu hiệu suất và khả năng mở rộng thông qua benchmark/profiling.
  - Tăng cường bảo mật qua việc phát hiện lỗ hổng.

# Mục tiêu chính của Database Testing

- **Toàn vẹn & Ánh xạ Dữ liệu:**

- **Data Integrity:** Xác thực (Verify) tính chính xác, tính nhất quán của dữ liệu qua các ràng buộc (khóa, check, not null) trong DDL.
- **Data Mapping:** Đảm bảo luồng dữ liệu từ UI → API → DB được ánh xạ chính xác, không sai lệch.

- **Quy tắc Nghiệp vụ:**

- Xác minh logic trong **Stored Procedures, Triggers** và các hàm có hoạt động đúng theo yêu cầu nghiệp vụ hay không.

- **Đảm bảo thuộc tính ACID:**

- **Atomicity:** Một giao dịch phải được thực hiện hoàn toàn hoặc không gì cả (all or nothing).
- **Consistency:** Dữ liệu phải luôn ở trạng thái hợp lệ sau mỗi giao dịch.
- **Isolation:** Các giao dịch đồng thời không ảnh hưởng lẫn nhau.
- **Durability:** Dữ liệu đã commit phải được lưu trữ bền vững.

# Các loại kiểm thử

## Các loại Kiểm thử Cơ sở dữ liệu

- **Structural Testing:** Tập trung vào việc xác thực các thành phần cấu trúc của CSDL đảm bảo đúng thiết kế. Bao gồm schema, tables, columns, keys, indexes, stored procedures và triggers.
- **Functional Testing:** Kiểm tra các chức năng của CSDL từ góc độ người dùng cuối. Bao gồm: các thao tác CRUD, logic nghiệp vụ, ràng buộc dữ liệu, xử lý lỗi (error handling)/
- **Non-functional Testing:** Đánh giá các khía cạnh như hiệu suất, bảo mật và khả năng sử dụng của CSDL.



# Kiểm thử Cấu trúc (Structural)

- **Kiểm thử Schema & Consistency:**

- Xác thực **Schema** (bảng, cột, kiểu dữ liệu) khớp với đặc tả.
- Kiểm tra **Schema Consistency** giữa các môi trường (DEV, TEST, PROD).
- Đảm bảo không có đối tượng DB thừa/thiếu.

- **Kiểm thử Stored Procedure:**

- Xác minh logic nghiệp vụ, xử lý lỗi.
- Kiểm tra kết quả trả về với các bộ dữ liệu đầu vào khác nhau (White-box).

- **Kiểm thử Keys & Indexes:**

- **Khóa (Keys):** Xác thực khóa chính/ngoại (tính duy nhất, not-null, toàn vẹn tham chiếu).
- **Chỉ mục (Indexes):** Đảm bảo index được tạo đúng, sử dụng hiệu quả và không thừa.

- **Kiểm thử Trigger:**

- Đảm bảo trigger được kích hoạt đúng sự kiện (INSERT, UPDATE, DELETE).
- Xác thực logic thực thi có đúng và không gây side-effect.

# Kiểm thử Chức năng (Functional)

## Black Box Testing

- Tập trung vào verify chức năng từ góc độ người dùng cuối, không cần biết logic bên trong.
- Được thực hiện bởi **testers**.
- Mục tiêu:
  - **Kiểm thử các thao tác CRUD:** Xác minh các thao tác **CREATE**, **READ**, **UPDATE**, **DELETE** từ giao diện/API được phản ánh chính xác trong CSDL.
  - **Kiểm tra giá trị mặc định/ràng buộc:** Đảm bảo dữ liệu nhập vào tuân thủ đúng các ràng buộc (constraints: kiểu, giới hạn, not null...)
  - **Kiểm tra xử lý lỗi:** Đảm bảo CSDL xử lý đúng các tình huống lỗi (ví dụ: vi phạm ràng buộc, dữ liệu không hợp lệ).

# Kiểm thử Chức năng (Functional)

## White Box Testing

- Tập trung vào validate logic bên trong các đối tượng CSDL.
- Được thực hiện bởi **developers**.
- Mục tiêu:
  - **Validate logic nghiệp vụ:** Kiểm tra các trigger, stored procedure, và view đảm bảo chúng thực thi đúng yêu cầu.
  - **Kiểm tra toàn vẹn dữ liệu:** Đảm bảo các ràng buộc (constraints) và quan hệ (relationships) hoạt động chính xác.

# Kiểm thử Phi chức năng (Non-functional)

## Hiệu năng (Performance)

- **Load test:** Đánh giá hiệu suất CSDL dưới tải thiết kế, đo thời gian phản hồi truy vấn.
- **Stress test:** Xác định điểm gãy (breaking point) của CSDL bằng cách tạo tải ảo rất lớn vào CSDL.

## Bảo mật (Security)

- Ngăn chặn các lỗ hổng phổ biến như **SQL Injection**.
- Xác thực quyền truy cập, đảm bảo người dùng chỉ thấy dữ liệu được phép.

## Sao lưu & phục hồi (Backup & Recovery)

- Xác minh việc sao lưu (backup) và phục hồi (restore) dữ liệu hoạt động đúng.
- Xác minh CSDL có thể được phục hồi thành công từ các bản sao lưu (backups) sau sự cố.

## Tương thích (Compatibility)

- Đảm bảo CSDL hoạt động ổn định trên các phiên bản DBMS, hệ điều hành, và nền tảng được dùng để triển khai.



Bảo mật CSDL cần ưu tiên:

- \*Đảm bảo kiểm soát truy cập chặt chẽ.
- \*Tránh các lỗi bảo mật cơ bản như SQL Injection
- \*Mã hóa dữ liệu nhạy cảm ở trạng thái nghỉ và khi truyền tải.

# Quy trình kiểm thử

# Quy trình Kiểm thử CSDL

## 1. Phân tích & Thiết kế Test cases

- Xác định yêu cầu nghiệp vụ, phạm vi kiểm thử.
- Thiết kế test case, kịch bản và bộ dữ liệu.

## 2. Chuẩn bị Môi trường & Dữ liệu

- Thiết lập môi trường test riêng biệt, đảm bảo cô lập giữa các bộ test.
- Tạo bộ dữ liệu thử nghiệm (Test Data).

## 3. Thực thi

- Chạy các kịch bản kiểm thử. (*auto/manual*)
- Ghi lại kết quả và các lỗi phát sinh.

## 4. Báo cáo

- Phân tích kết quả, so sánh với kết quả mong đợi.
- Ghi nhận lỗi (log bug) và báo cáo cho đội phát triển.

## 5. Kiểm thử Hồi quy và Hoàn tất

- Thực hiện lại các bài kiểm thử sau khi lỗi được sửa.
- Hoàn tất báo cáo.
- Đóng test cycle.

# Thách thức



# Thách thức trong Kiểm thử CSDL

- **Dữ liệu lớn và phức tạp:** Việc kiểm thử với khối lượng dữ liệu lớn có thể rất khó khăn và tốn thời gian.
- **Kiến thức về SQL:** Người kiểm thử cần có hiểu biết tốt về SQL và các khái niệm CSDL.
- **Chi phí và Thời gian:** Kiểm thử CSDL có thể tốn kém và mất nhiều thời gian, đặc biệt với các hệ thống lớn.
- **Quản lý dữ liệu thử nghiệm:** Tạo và quản lý dữ liệu thử nghiệm phù hợp (realistic, với referential integrity, đủ khối lượng) là rất quan trọng nhưng cũng đầy thách thức.
- **Cô lập môi trường thử nghiệm:** Đảm bảo môi trường thử nghiệm được tách biệt hoàn toàn với môi trường sản phẩm.

# Công cụ kiểm thử

- Chọn công cụ phù hợp với mục đích kiểm thử. Không có một công cụ nào có thể phù hợp với tất cả nhu cầu kiểm thử.
- Để kiểm thử một cách toàn diện, cần kết hợp nhiều công cụ khác nhau.

## Phân loại công cụ

Để chọn đúng công cụ, chúng ta cần xem xét chúng qua các khía cạnh sau:

- **Kiểm thử cái gì?** Hiệu năng, chức năng, bảo mật... Phạm vi kiểm thử?
- **Phương pháp tiếp cận?** Black-box hay White-box?
- **Tự động hóa và Khả năng tích hợp?** Khả năng tích hợp vào CI/CD và quản lý dữ liệu test.
- **Tính tương thích?** Hỗ trợ những hệ quản trị CSDL nào? SQL hay NoSQL?
- **Thiết kế và Chức năng?** Giao diện, tính dễ sử dụng, phù hợp với đối tượng nào (dev hay tester chẳng hạn).

# I. Phạm vi & Loại hình Kiểm thử (1/2)

- **Kiểm thử Hiệu năng (Performance):**

- Mô phỏng tải lớn, đo lường thời gian phản hồi và tìm breaking point.
- Công cụ: JMeter, HammerDB, Swingbench.

- **Kiểm thử Chức năng/Unit Test:**

- Xác minh logic nghiệp vụ và thao tác CRUD.
- Công cụ: tSQLt (Unit Test), DbFit (Acceptance Test).

- **Kiểm thử Cấu trúc (Structural):**

- So sánh và xác thực schema, bảng, cột, khóa...
- Công cụ: Redgate SQL Compare.

- **Kiểm thử Dữ liệu và ETL**

- So sánh dữ liệu giữa nguồn và đích, đảm bảo tính toàn vẹn.
- Công cụ: QuerySurge.

# I. Phạm vi & Loại hình Kiểm thử (2/2)

- **Kiểm thử Bảo mật (Security)**

- Phát hiện các lỗ hổng bảo mật, đặc biệt là SQL Injection.
- Kiểm tra về phân quyền truy cập dữ liệu.
- Công cụ: SQLMap (SQL Injection), Các công cụ functional testing (dùng query kiểm tra phân quyền).

- **Tạo test data**

- Tạo dữ liệu giả lập/khối lượng lớn cho kiểm thử tải, hiệu năng.
- Công cụ: Databene Benerator, IRI RowGen.

## II. Phương pháp tiếp cận

- **Phương pháp tiếp cận:**

- **White Box Testing:** Kiểm tra trực tiếp logic thực thi bên trong CSDL.
  - VD: *tSQLt* cho phép viết test bằng T-SQL.
- **Black Box Testing:** Kiểm tra kết quả cuối cùng từ góc độ người dùng mà không cần biết logic.
  - VD: *DbFit* dùng bảng để mô tả logic.

- **Tính cô lập (Isolation):**

- Đảm bảo mỗi test case chạy độc lập, không ảnh hưởng đến CSDL hay các test khác.

VD:

- *tSQLt* tự động **ROLLBACK** sau mỗi test.
- *DBUnit* dùng **CLEAN\_INSERT** để reset dữ liệu.

### III. Tự động hóa & khả năng tích hợp

- **Tích hợp CI/CD:**

- Khả năng chạy qua dòng lệnh (CLI) và xuất báo cáo để tích hợp vào Jenkins, GitLab CI, Azure DevOps.
- Điều này giúp tự động hóa việc kiểm thử trong quy trình phát triển phần mềm.

- **Mocking:**

- Thay thế các đối tượng phụ thuộc (bảng, procedure) bằng đối tượng giả để cô lập test.
- Công cụ: *tSQLt* cung cấp **FakeTable** và **SpyProcedure**.

- **Quản lý dữ liệu test:**

- Hỗ trợ thiết lập dữ liệu mẫu (dataset) trước khi test và dọn dẹp (teardown) sau đó.
- Công cụ: *DBUnit*, *NoSQLUnit* sử dụng file XML/JSON để quản lý và xuất báo cáo.
- Data generation tools (*Databene Benerator*, *IRI RowGen*) giúp tạo dữ liệu test chất lượng cao.
- Việc hỗ trợ này giúp đội ngũ dễ dàng trong việc quan sát và phân tích kết quả test để nhanh chóng sửa lỗi



## Ứng dụng Database Testing trong CI/CD

1. **Commit Code:** Developer commit thay đổi CSDL (schema, SP, migrations) lên Git.
2. **Build & Deploy:** CI server deploy CSDL phiên bản mới lên môi trường test.
3. **Test Execution:**
  - Unit Tests (*tSQLt*): Kiểm tra logic bên trong SP, Function, Trigger.
  - Acceptance Tests (*DbFit*): Xác minh các luồng nghiệp vụ end-to-end.
  - Performance Tests (*JMeter*, *HammerDB*): Đánh giá hiệu năng dưới tải.
4. **Report & Feedback:**
  - Xuất báo cáo (XML, HTML, JUnit format).
  - Nếu có test FAIL → Dừng build và gửi thông báo.

## IV. Tương thích

- **Tính tương thích:** Tùy vào nhu cầu kiểm thử trên DBMS nào mà chọn công cụ phù hợp.
  - Một số công cụ chỉ hỗ trợ một (vài) DBMS cụ thể:
    - MS SQL Server: *tSQLt*, *DBFit*.
    - Oracle DB: *Swingbench*
  - Một số công cụ có thể hỗ trợ đa nền tảng:
    - *JMeter* (các DBMS có hỗ trợ JDBC)
    - *NoSQLUnit* (MongoDB, Cassandra...).

- **Giao diện và Tính dễ sử dụng:**

- Kịch bản test có dễ đọc, dễ hiểu không?
- Viết một test mới có đơn giản không?

**Ví dụ:** *DbFit* dùng bảng, dễ sử dụng cho nhiều đối tượng, từ dev, QA/QC đến cả BA.

## V. Khả năng thiết kế test case

- Hỗ trợ Kỹ thuật Thiết kế Test Case:
  - Hỗ trợ các kỹ thuật như **Data-Driven Testing (DDT)**.
  - **Ví dụ:** *JMeter* có thể đọc dữ liệu từ file CSV để thực hiện nhiều kịch bản test.

# Structural Testing

- **Redgate SQL Compare**

- Là công cụ trả phí mạnh mẽ để so sánh schema CSDL.
- **Chức năng:** Tìm và đồng bộ hóa sự khác biệt về schema (bảng, cột, view...) giữa các môi trường (VD: DEV vs. PROD).
- **Chi phí:** Trả phí (Commercial).

- **ApexSQL Diff**

- Là công cụ trả phí tương tự Redgate.
- **Chức năng:** So sánh, phát hiện sự khác biệt trong cả schema và dữ liệu.
- **Chi phí:** Trả phí (Commercial).

## Functional Testing (Unit test)

- **tSQLt**: Framework mã nguồn mở cho SQL Server để thực hiện Unit Test.
- **DbFit**: Hỗ trợ kiểm thử CSDL qua giao diện các bảng quyết định (decision tables), dễ dàng sử dụng với các đối tượng khác nhau.
- **DbUnit**: Mở rộng của JUnit, chuyên dùng để quản lý trạng thái CSDL giữa các lần chạy test.
- **NoSQLUnit**: Tương tự DbUnit nhưng dành cho CSDL NoSQL như MongoDB, Cassandra.

# Non-functional: Performance & Security

- **Apache JMeter**

- Kiểm thử tải và hiệu năng, bằng cách tạo ra lượng lớn truy vấn đồng thời để đo thời gian phản hồi và khả năng chịu tải của CSDL.
- Open source, miễn phí.

- **HammerDB**

- Công cụ benchmark và load testing.
- Chuyên đo lường hiệu suất của nhiều loại CSDL (Oracle, SQL Server, PostgreSQL...) dưới các kịch bản tải khác nhau.
- Open source, miễn phí.

- **SQLMap**

- Chuyên dò tìm và khai thác lỗ hổng SQL Injection.
- Tự động hóa việc phát hiện các điểm yếu bảo mật liên quan đến SQL Injection trong ứng dụng.
- Open source, miễn phí.

# Data Generation Tools

## Tầm quan trọng của Data Generation:

- Tạo dữ liệu kiểm thử chất lượng cao là nền tảng cho việc kiểm thử CSDL hiệu quả.
- Dữ liệu cần đảm bảo: tính đa dạng, đúng ràng buộc, phân phối gần thực tế, và khối lượng phù hợp với mục đích kiểm thử.

## Mục đích sử dụng:

- **Load/Performance Testing:** Tạo khối lượng lớn dữ liệu để kiểm thử hiệu năng.
- **Functional Testing:** Tạo dữ liệu đa dạng để kiểm tra các trường hợp edge cases.
- **Security & Compliance:** Dùng dữ liệu tổng hợp thay vì dữ liệu production để tránh rò rỉ thông tin nhạy cảm.

# Công cụ thử nghiệm



## tSQLt - Framework Unit Test cho SQL Server (1/2)

- tSQLt là framework **Unit Test** mã nguồn mở dành riêng cho **Microsoft SQL Server**.
- Cho phép lập trình viên viết và thực thi test case tự động bằng ngôn ngữ **T-SQL**.
- Phương pháp **White Box Testing**, tập trung xác minh logic bên trong code CSDL.

### Nguyên lí: Tính Cô lập (Isolation)

- Mỗi test được bọc trong transaction và tự động **ROLLBACK** sau khi chạy, đảm bảo CSDL luôn ở trạng thái chuẩn để tiếp tục chạy các bộ test khác.
- Các đối tượng đặc trưng:
  - **FakeTable**: Tạo bảng giả không có ràng buộc, khóa ngoại, trigger.
  - **SpyProcedure**: Thay thế SP thật bằng một SP mock để kiểm tra các thao tác gọi SP.

# tSQLt - Mô hình Arrange-Act-Assert

## Quy trình kiểm thử AAA

- **Arrange:**

- Giả lập bảng/SP bằng FakeTable và SpyProcedure.
- Chuẩn bị dữ liệu test và kết quả mong đợi.

- **Act:**

- Thực thi SP hoặc Function cần kiểm thử.

- **Assert:**

- So sánh kết quả thực tế với kỳ vọng bằng AssertEqualsTable.

# tSQLt - Ưu điểm & Nhược điểm

## Ưu điểm

- Mã nguồn mở, hoàn toàn miễn phí.
- Hỗ trợ regression test dễ dàng.
- Phát hiện lỗi ngay tại tầng CSDL, trước khi backend gọi đến.
- **Hỗ trợ CI/CD:**
  - Hỗ trợ CLI để build script pipeline.
  - Tích hợp test report vào pipeline.

## Nhược điểm

- **Chỉ hỗ trợ SQL Server:** Không dùng được cho các CSDL khác Oracle, MySQL, PostgreSQL.

## DbFit - Kiểm thử Chấp nhận cho CSDL

- **DbFit** là framework mã nguồn mở cho Acceptance Testing CSDL.
- Được xây dựng trên **FIT/FitNesse**.
- Hỗ trợ **đa nền tảng**: Oracle, SQL Server, MySQL, DB2, PostgreSQL, HSQLDB, Derby.
- Phương pháp Black Box
  - **Không quan tâm logic bên trong SP**, chỉ quan tâm đầu vào/đầu ra.
  - Mô hình **Given-When-Then**:
    - **Given**: Chuẩn bị CSDL với dữ liệu.
    - **When**: Thực thi nghiệp vụ, store procedure.
    - **Then**: Xác minh trạng thái của CSDL.

# DbFit - Đặc điểm nổi bật

Test case dạng Bảng (Table-based)

Dễ đọc, dễ hiểu

Kết quả trực quan

[đợi thêm ảnh]

# DbFit - Ưu điểm & Nhược điểm

## Ưu điểm

- **Mã nguồn mở (GPL):** Hoàn toàn miễn phí.
- **Đa nền tảng:** Hỗ trợ nhiều loại CSDL khác nhau (Oracle, SQL Server, MySQL, DB2, PostgreSQL...).
- **Dễ tiếp cận:** Không yêu cầu kỹ năng lập trình sâu, phù hợp cho nhiều vai trò trong team.
- **Tích hợp CI/CD:** Có thể chạy qua command-line, JUnit, Maven.

## Nhược điểm

- **Yêu cầu nhiều môi trường Runtime:** Cần Java và .NET 2.0 (tùy cấu hình).
- **Cấu hình phức tạp:** Phải tự cài JDBC drivers, setup FitNesse.

# Demo

# User Story & Lược đồ CSDL

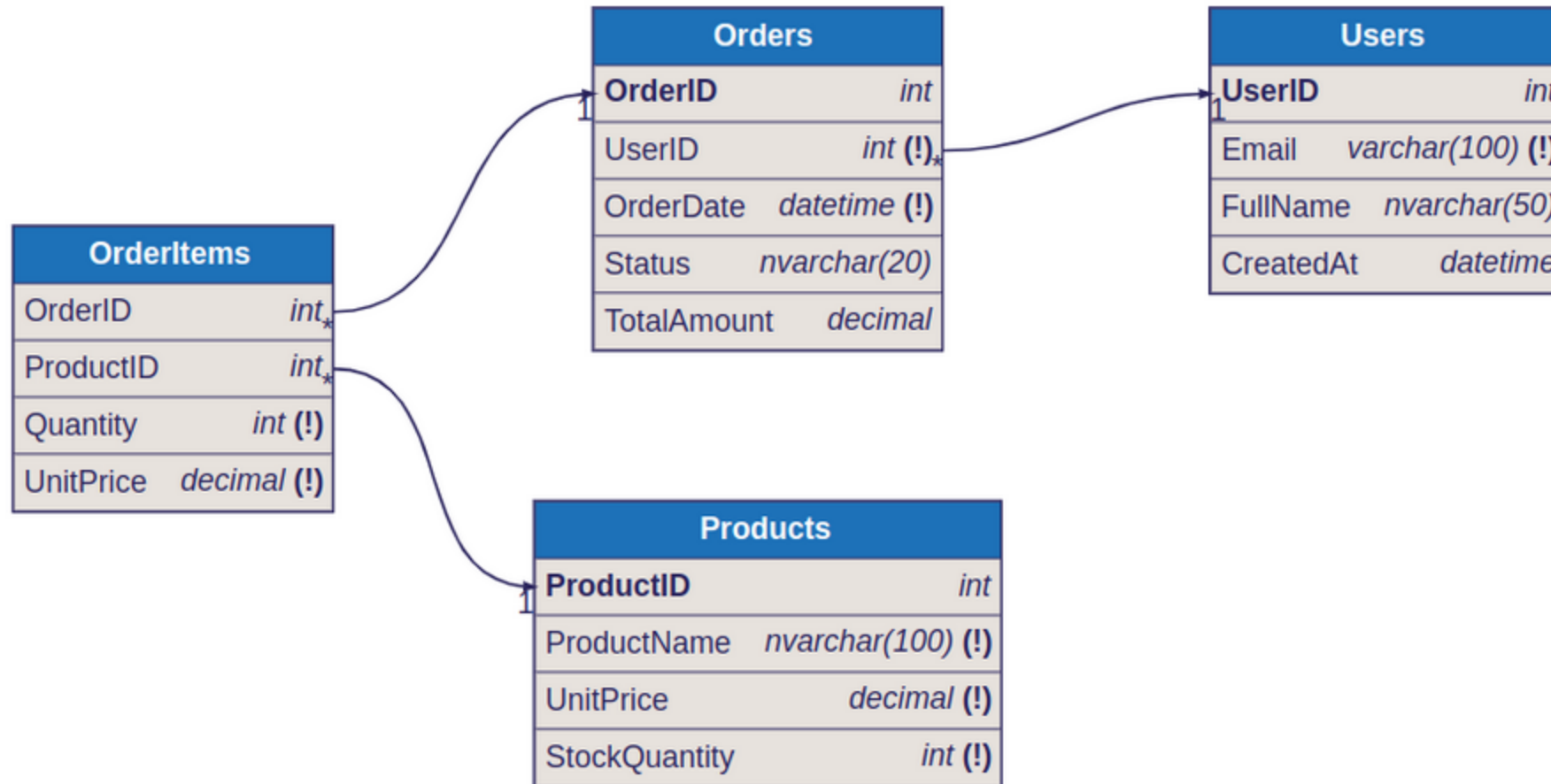
## User Story

Demo tập trung vào kịch bản xử lý đơn hàng, bao gồm:

- Xác thực dữ liệu đầu vào
- Kiểm tra tính toàn vẹn ràng buộc (constraints)
- Kiểm tra tồn kho (Products)
- Cập nhật trạng thái đơn hàng (Orders)
- Đảm bảo quy tắc bảo mật



# Schema



# Kịch bản (1/2)

## 1. Functional Testing

- **Setup:** Store Procedures
  - `usp_CalculateOrderTotal`: Tính tổng tiền
  - `usp_CompleteOrder`: Hoàn tất đơn hàng
- **Test Cases:**
  - Pending → Completed, trừ tồn kho, tính tổng tiền
  - Hết hàng → Trả lỗi, giữ nguyên trạng thái
  - Đã hoàn tất → Ngăn xử lý trùng

## 2. Constraint Testing

- **Setup:** Bổ sung ràng buộc vào schema
- **Test Cases:**
  - **CHECK:** `StockQuantity < 0`, `Quantity = 0`, `UnitPrice < 0`
  - **UNIQUE:** Email trùng lặp
  - **FOREIGN KEY:** UserID không tồn tại, xóa User có Orders
  - **DEFAULT/NULL:** `Status = 'Pending'`, `ProductName NOT NULL`

## Kịch bản (2/2)

### 3. Security Testing

- **Setup:**

3 roles: Admin, User, Read-only, grant quyền tương ứng. Kích hoạt Row-Level Security (RLS) trên Orders

- **Test Cases:**

- Admin: SELECT tất cả, UPDATE Products
- User RLS USING: Chỉ thấy đơn hàng của mình
- User RLS WITH CHECK: Bị chặn INSERT cho UserID khác
- Read-only: SELECT được, bị chặn UPDATE/DELETE

### 4. Regression Testing

- **Setup:**



- Bảng ProductPriceHistory
- v1: usp\_AddOrderItem\_v1 – Giá truyền thủ công
- v2: usp\_AddOrderItem\_v2 – Giá tự động từ Products

- **Test Cases:**

- v2 tự động lấy UnitPrice, ghi lịch sử
- Gọi v2 với tham số v1 → Lỗi "too many arguments"
- Giá thay đổi → v2 lấy đúng giá mới

## Link Youtube Demo

Thực hiện test trên kịch bản với hai công cụ:

- dbFit: <https://youtu.be/s4oSxgitSgA> 
- tSQLt: [https://youtu.be/DRxOTRa\\_DXs](https://youtu.be/DRxOTRa_DXs) 

# Best practices & kết luận

## Best Practices (1/4)

- Document chi tiết, đầy đủ:
  - Test case: input, output, expected result, steps.
  - Test log: kết quả, lỗi, thời gian. Phục vụ cho debug và kiểm tra lại.
  - Test coverage: đảm bảo bao phủ các thành phần quan trọng.
- Kiểm tra dữ liệu & metadata:
  - Xác minh cấu trúc bảng: column, type, default value, constraints, indexes.
  - Kiểm tra giới hạn dữ liệu (length, range, enum), giá trị mặc định và NULL handling.
  - Kiểm tra cả functional data và metadata (migrations, schema versions).

## Best Practices (2/4)

- Chú ý ETL operations:
  - Kiểm thử các bước extract/transform/load riêng biệt và toàn bộ quá trình (end-to-end).
  - So sánh dữ liệu nguồn và đích (row counts, checksums, sample records).
  - Kiểm tra incremental loads (chỉ thực hiện thay đổi trên dữ liệu mới so với lần chạy trước đó), error handling và idempotency.
- Cô lập môi trường thực thi testcases:
  - Mỗi test chạy độc lập (setup → execute → teardown) để tránh side-effects.
  - Xác định thời điểm chạy setup/teardown và dùng transaction/rollback hoặc snapshot để reset.
  - Sử dụng môi trường test tách biệt (sandbox) hoặc containerized DB.

## Best Practices (3/4)

- Sử dụng dữ liệu đầu vào:
  - Validate input test data trước khi chạy.
  - Dùng mock/seed data gần thực tế: sử dụng data generation tools (Databene Benerator, IRI RowGen), anonymized production samples.
  - Thực hiện data-driven tests với bộ dữ liệu đại diện cho các phân vùng và biên.
  - Đảm bảo dữ liệu test tuân thủ referential integrity và realistic distributions.
- Tự động hóa test execution:
  - Nếu khả thi, tự động hóa để hỗ trợ regression và CI.
  - Công cụ gợi ý: DBUnit, tSQLt, SQLTest, DbFit; hoặc scripting (bash/python) cho orchestration.
  - Nếu tự động hóa không khả thi: duy trì checklist, template logs và execution plan.










## Best Practices (4/4)

- Kiểm tra các yếu tố vận hành (operational checks):
  - DB logs: xác nhận logs được ghi và rotate đúng.
  - Cron/jobs: kiểm tra jobs chạy đúng lịch và hoàn thành thành công.
  - Backup: kiểm tra backup tồn tại, validate integrity và thử restore định kỳ.
  - Distributed DB: kiểm tra dữ liệu sync/replication, detect lag/consistency issues.
  - Giám sát Recovery Time Object/Recovery Point Object và tài liệu quy trình khôi phục.

## Kết luận

- Database testing giúp phát hiện các vấn đề về **độ tin cậy** (reliability), **tính toàn vẹn** (data integrity) và các vấn đề khác khi **vận hành** CSDL (security, performance, scalability...).
- **Tự động hóa và AI** đang định hình lại tương lai của kiểm thử CSDL, giúp tạo test case thông minh một cách tự động. Tuy nhiên, vai trò của con người vẫn rất quan trọng trong việc thiết kế kịch bản kiểm thử và phân tích kết quả.
- **Kết hợp** các **kỹ thuật** truyền thống với **công cụ** hiện đại sẽ giúp đảm bảo việc kiểm thử trở nên đầy đủ hơn, giúp phát hiện sớm các vấn đề tiềm ẩn.

# Tài liệu tham khảo

- Divesh Mehta — Database testing: What is it? Why & best practices  [truy cập: Oct. 22, 2025]
- Indaacademy — Database Testing Tutorial – Hướng dẫn kiểm thử Cơ sở dữ liệu (P2)  [truy cập: Oct. 22, 2025]
- Thomas Hamilton — Database Testing Tutorial (Guru99)  [truy cập: Oct. 22, 2025]
- Gunashree RS — Database Tests: Guide to Ensuring Data Integrity and Performance  [truy cập: Oct. 22, 2025]
- David Eketé — Advanced Test Data Management: Techniques and Best Practices  [truy cập: Oct. 23, 2025]
- HammerDB Documentation  [truy cập: Oct. 23, 2025]
- DbFit Tutorial  [truy cập: Oct. 23, 2025]

# Q&A

