**Báo Cáo Thu Hoạch**

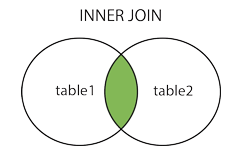
1. **SQL**

* **SQL (Ngôn ngữ truy vấn cấu trúc)** được sử dụng để thực hiện các thao tác trên các bản ghi được lưu trữ trong cơ sở dữ liệu như cập nhật bản ghi, xóa bản ghi, tạo và sửa đổi bảng, dạng xem, v.v.
* SQL chỉ là một ngôn ngữ truy vấn, **nó không phải là một cơ sở dữ liệu**. Để thực hiện truy vấn SQL, cần cài đặt bất kỳ cơ sở dữ liệu nào, ví dụ: Oracle, MySQL, MongoDB, PostGre SQL, SQL Server, DB2, v.v.
* Tất cả các DBMS như MySQL , Oracle , MS Access, Sybase, Informix, PostgreSQL và SQL Server đều sử dụng SQL làm ngôn ngữ cơ sở dữ liệu chuẩn.
* **SQL là cần thiết**:
  + Để tạo cơ sở dữ liệu, bảng và dạng xem mới
  + Để chèn bản ghi vào cơ sở dữ liệu
  + Để cập nhật bản ghi trong cơ sở dữ liệu
  + Để xóa bản ghi khỏi cơ sở dữ liệu
  + Để lấy dữ liệu từ cơ sở dữ liệu
* **SQL làm những gì?**
  + Với SQL, chúng ta có thể truy vấn cơ sở dữ liệu của mình theo nhiều cách, sử dụng các câu lệnh giống như tiếng Anh.
  + Với SQL, người dùng có thể truy cập dữ liệu từ hệ quản trị cơ sở dữ liệu quan hệ.
  + Nó cho phép người dùng mô tả dữ liệu.
  + Nó cho phép người dùng xác định dữ liệu trong cơ sở dữ liệu và thao tác khi cần.
  + Nó cho phép người dùng tạo và thả cơ sở dữ liệu và bảng.
  + Nó cho phép người dùng tạo một view, stored procedure, function trong cơ sở dữ liệu.
  + Nó cho phép người dùng thiết lập quyền trên bảng, procedures, và views.

1. **NoSQL**

* Cơ sở dữ liệu có thể được chia thành 3 loại:
  + RDBMS (Hệ thống quản lý cơ sở dữ liệu quan hệ)
  + OLAP (Xử lý phân tích trực tuyến)
  + NoSQL (\*cơ sở dữ liệu được phát triển gần đây\*)
* Cơ sở dữ liệu NoSQL được sử dụng để chỉ một cơ sở dữ liệu non-SQL hoặc cơ sở dữ liệu không quan hệ.
* Nó cung cấp một cơ chế lưu trữ và truy xuất dữ liệu khác với mô hình quan hệ bảng được sử dụng trong cơ sở dữ liệu quan hệ. Cơ sở dữ liệu NoSQL không sử dụng bảng để lưu trữ dữ liệu. Nó thường được sử dụng để lưu trữ dữ liệu lớn và các ứng dụng web thời gian thực.
* Ưu điểm của NoSQL
  + Nó hỗ trợ ngôn ngữ truy vấn.
  + Nó cung cấp hiệu suất nhanh chóng.
  + Nó cung cấp khả năng mở rộng theo chiều ngang.
* **Phân loại NoSQL Database**
  + **Key-value stores**: Lưu trữ kiểu key-value là kiểu lưu trữ dữ liệu NoSQL đơn giản nhất sử dụng từ một API. Chúng ta có thể nhận được giá trị cho khóa, đặt một giá trị cho một khóa, hoặc xóa một khóa từ dữ liệu.
  + **Column-oriented databases (column-family):** Cơ sở dữ liệu column-family lưu trữ dữ liệu trong nhiều cột trong mỗi dòng với key cho từng dòng. Column families là một nhóm các dữ liệu liên quan được truy cập cùng với nhau.
  + **Graph databases:** Kiểu đồ thị này cho phép bạn lưu trữ các thực thể và quan hệ giữa các thực thể. Các đối tượng này còn được gọi là các nút, trong đó có các thuộc tính. Mỗi nút là một thể hiện của một đối tượng trong ứng dụng. Quan hệ được gọi là các cạnh, có thể có các thuộc tính. Cạnh có ý nghĩa định hướng; các nút được tổ chức bởi các mối quan hệ. Các tổ chức của đồ thị cho phép các dữ liệu được lưu trữ một lần và được giải thích theo nhiều cách khác nhau dựa trên các mối quan hệ. Thông thường, khi chúng ta lưu trữ một cấu trúc đồ thị giống như trong RDBMS, nó là một loại duy nhất của mối quan hệ. Việc tăng thêm một mối quan hệ có nghĩa là rất nhiều thay đổi sơ đồ và di chuyển dữ liệu, mà không phải là trường hợp khó khi chúng ta đang sử dụng cơ sở dữ liệu đồ thị. Trong cơ sở dữ liệu đồ thị, băng qua các thành phần tham gia hoặc các mối quan hệ là rất nhanh. Các mối quan hệ giữa các node không được tính vào thời gian truy vấn nhưng thực sự tồn tại như là một mối quan hệ. Đi qua các mối quan hệ là nhanh hơn so với tính toán cho mỗi truy vấn. Có rất nhiều cơ sở dữ liệu đồ thị có sẵn, chẳng hạn như Neo4J, Infinite Graph, OrientDB, hoặc FlockDB (đó là một trường hợp đặc biệt: một cơ sở dữ liệu đồ thị mà chỉ hỗ trợ các mối quan hệ duy nhất chuyên sâu hoặc danh sách kề, nơi mà bạn không thể đi qua nhiều hơn một mức độ sâu sắc đối với các mối quan hệ ).
  + **Document Oriented databases:** Tài liệu là nguyên lý chính của cơ sở dữ liệu kiểu dữ liệu. Dữ liệu lưu trữ và lấy ra là các tài liệu với định dạng XML, JSON, BSON,… Tài liệu miêu tả chính nó, kế thừa từ cấu trúc dữ liệu cây. Có thể nói cơ sở dữ liệu tài liệu là 1 phần của key-value. Cơ sở dữ liệu kiểu tài liệu như MongoDB cung cấp ngôn ngữ truy vấn đa dạng và cúc trúc như là cơ sở dữ liệu như đánh index,… Một số cơ sở dữ liệu tài liệu phổ biến mà chúng ta hay gặp là MongoDB, CouchDB, Terastore, OrientDB, RavenDB.

1. **JOIN**

* JOIN là phép kết nối dữ liệu từ nhiều bảng lại với nhau, nối 2 bảng, 3 bảng.. với nhau. Khi cần truy vấn các cột dữ liệu từ nhiều bảng khác nhau để trả về trong cùng một tập kết quả, cần dùng JOIN. Hai bảng kết nối được với nhau khi có 1 trường chung giữa 2 bảng này.
* **INNER JOIN (Hoặc JOIN):** Trả về tất cả các hàng khi có ít nhất một giá trị ở cả hai bảng.
* **Cú pháp:**
* **INNER JOIN nhiều table**

SELECT column\_name(s)

FROM table1

INNER JOIN table2

ON table1.column\_name = table2.column\_name;

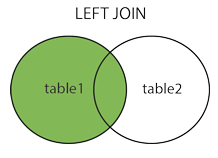
* **INNER JOIN nhiều table**

SELECT column\_list

FROM table1

INNER JOIN table2 ON join\_condition1

INNER JOIN table3 ON join\_condition2

* **LEFT OUTER JOIN (Hoặc LEFT JOIN):** Trả lại tất cả các dòng từ bảng bên trái, và các dòng đúng với điều kiện từ bảng bên phải.
* **Cú pháp:**
  + **LEFT JOIN 2 table**

SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name = table2.column\_name;

* **LEFT JOIN cũng có thể join nhiều table**

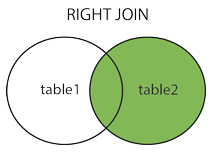
SELECT column\_list

FROM table1

LEFT JOIN table2 ON join\_condition1

LEFT JOIN table3 ON join\_condition2

* **RIGHT OUTER JOIN (Hoặc RIGHT JOIN):** Trả lại tất cả các hàng từ bảng bên phải, và các dòng thỏa mãn điều kiện từ bảng bên trái.

****

* **Cú pháp**
* **RIGHT JOIN 2 table**

SELECT column\_name(s)

FROM table1

RIGHT JOIN table2

ON table1.column\_name = table2.column\_name;

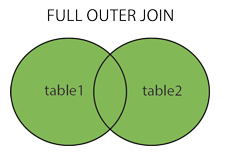
* **RIGHT JOIN cũng có thể join nhiều table**

SELECT column\_list

FROM table1

RIGHT JOIN table2 ON join\_condition1

RIGHT JOIN table3 ON join\_condition2

* **FULL OUTER JOIN (Hoặc OUTER JOIN):** Trả về tất cả các dòng đúng với 1 trong các bảng.

**FULL OUTER JOIN** là sự kết hợp của LEFT JOIN và RIGHT JOIN

* **Cú Pháp**:

SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name = table2.column\_name

WHERE condition;

1. **Function**

#### **Định nghĩa:**

Là một đối tượng trong cơ sở dữ liệu (CSDL) sử dụng trong các câu lệnh SQL, được biên dịch sẵn và lưu trong CSDL nhằm mục đích thực hiện xử lý nào đó như tính toán phức tạp và trả về kết quả là giá trị nào đó.

#### **Đặc điểm**:

* Luôn trả về giá trị
* Gồm 2 loại: Function hệ thống và Function do người dùng tự định nghĩa
* Function người dùng tự định nghĩa gồm 2 loại:
  + **Scalar-valued**: Trả về giá trị vô hướng của các kiểu dữ liệu T-SQL
  + **Table-valued**: Trả về bảng, là kết quả của một hoặc nhiều lệnh
* **Cách định nghĩa Function**
  + 1. **Tạo Function trả về giá trị loại Scalar-valued**

CREATE FUNCTION <Tên function>

([@<tên tham số> <kiểu dữ liệu> [= <giá trị mặc định>], …,[...]])

RETURNS <kiểu dữ liệu>

[WITH ENCRYPTION]

[AS]

BEGIN

[Thân của hàm]

RETURN <Biểu thức giá trị đơn>

END

**Trong đó:**

* **Tên function**: Tên của hàm chúng ta sẽ tạo
* **Tên tham số**: Là các tham số Input cho hàm. Khai báo báo gồm tên của tham số (trước tên tham số sử dụng tiền tố @), kiểu dữ liệu của tham số, chúng ta có thể chỉ định giá trị mặc định cho tham số. Có thể chỉ định nhiều tham số đầu vào
* **RETURNS**: từ khóa này chỉ định kiểu dữ liệu hàm sẽ trả về. Kiểu dữ liệu phải được chỉ định kiểu độ dài dữ liệu. Ví dụ: varchar(100)
* **WITH ENCRYPTION**: Từ khóa chỉ định code của hàm sẽ được mã hóa trong bảng syscomments.
* **AS**: Từ khóa cho biết code của hàm bắt đầu.
* **BEGIN**: Đi cùng với END để tạo thành bao khối bao các câu lệnh trong thân hàm.
* **RETURN**: Từ khóa này sẽ gửi giá trị tới thủ tục gọi hàm. **Một số lưu ý:**
* Tên function phải là duy nhất trong 1 CSDL. Function được tạo/định nghĩa trong CSDL nào thì chỉ sử dụng trong CSDL đó. Khác với Function có sẵn của SQL được truy cập ở bất cứ đâu.
* Danh sách tham số tối đa 1024 tham số.
  + 1. **Tạo Function trả về giá trị loại Table-valued**

Function **Table-valued** có 2 loại:

* **Hàm giá trị bảng đơn giản**: Trả về bảng, là kết quả của một câu lệnh SELECT đơn
* **Hàm giá trị bảng đa câu lệnh**: Trả về bảng, là kết quả của nhiều câu lệnh

1. Hàm giá trị bảng đơn giản

CREATE FUNCTION <Tên function>

([@<tên tham số> <kiểu dữ liệu> [= <giá trị mặc định>], …,[...]])

RETURNS TABLE

[WITH ENCRYPTION]

[AS]

RETURN <Câu lệnh SQL>

END

* **Lưu ý** Hàm giá trị bảng đơn còn được gọi là hàm giá trị bảng nội tuyến. Có thể được dùng trong câu lệnh truy vấn thay thế cho tên bảng hoặc tên View.

**b) Hàm giá trị bảng đa câu lệnh**

CREATE FUNCTION <Tên function>

([@<tên tham số> <kiểu dữ liệu> [= <giá trị mặc định>], …,[...]])

RETURNS @<tên biến trả về> TABLE (<tên cột 1> <kiểu dữ liệu> [tùy chọn thuộc tính], ..., <tên cột n> <kiểu dữ liệu> [tùy chọn thuộc tính])

[AS]

BEGIN

<Câu lệnh SQL>

RETURN

END

**Thay đổi, xóa, xem nội dung Function**

1. **Thay đổi Function**

Để thay đổi các hàm đã khai báo ta sử dụng câu lệnh ALTER FUNCTION. Cú pháp tương tự như tạo mới Function, chỉ thay từ khóa CREATE bằng từ khóa ALTER

ALTER FUNCTION <Tên function>

([@<tên tham số> <kiểu dữ liệu> [= <giá trị mặc định>], …,[...]])

RETURNS <kiểu dữ liệu> | TABLE

[WITH ENCRYPTION]

[AS]

BEGIN

[Thân của hàm]

RETURN <Biểu thức giá trị đơn> | Câu lệnh SQL

END

1. Xóa Function

Để xóa hàm ta dùng câu lệnh DROP FUNCTION.

DROP FUNCTION [schema\_name.] <Tên function>

1. Xem nội dung Function

Để xem nội dung function ta sử dụng Store Procedure (Thủ tục) có sẵn của SQL là sp\_helptext (Transact-SQL)

EXEC sp\_helptext 'FunctionName'

| **Lợi ích** | **Giải thích** |
| --- | --- |
| Module hóa | Bạn chỉ cần viết Stored Procedure 1 lần, sau đó có thể gọi nó nhiều lần ở trong ứng dụng. |
| Hiệu suất | Stored Procedure thực thi mã nhanh hơn và giảm tải băng thông. - **Thực thi nhanh hơn**: Stored Procedure sẽ được biên dịch và lưu vào bộ nhớ khi được tạo ra. Điều đó có nghĩa rằng nó sẽ thực thi nhanh hơn so với việc gửi từng đoạn lệnh SQL tới SQL Server. Vì nếu bạn gửi từng đoạn lệnh nhiều lần thì SQL Server cũng sẽ phải biên dịch lại nhiều lần, rất mất thời gian so với việc biên dịch sẵn. - **Giảm tải băng thông**: Nếu bạn gửi nhiều câu lệnh SQL thông qua network đến SQL Server sẽ ảnh hưởng tới hiệu suất đường truyền. Thay vì gửi nhiều lần thì bạn có thể gom các câu lệnh SQL vào 1 Stored Procedure và chỉ phải gọi đến 1 lần duy nhất qua network. |
| Bảo mật | Trong SQL Server có các tác vụ cấp cao mà người dùng bình thường không thể truy cập vào được. Bằng việc cung cấp các Stored Procedure đã truy cập tới các tác vụ này cho người dùng thường thì không sao hết. Vì làm vậy thì người dùng thường sẽ truy cập gián tiếp mà không ảnh hưởng tới vấn đề bảo mật của SQL Server. |

1. **STORE PROCEDURE**

**Lợi ích của Stored Procedure**

* **Tạo Stored Procedure**

Bạn tạo Stored Procedure được lưu trữ trong SQL Server Management Studio bằng cách sử dụng câu lệnh CREATE PROCEDURE:

CREATE PROCEDURE StoredProcedureName AS

...

Đoạn mã sau tạo một Stored Procedure gọi là "MyStoredProcedure":

CREATE PROCEDURE MyStoredProcedure AS

SET ROWCOUNT 10

SELECT Products.ProductName AS TenMostExpensiveProducts, Products.UnitPrice

FROM Products

ORDER BY Products.UnitPrice DESC

* **Chỉnh sửa Stored Procedure**

Nếu bạn cần sửa đổi một Stored Procedure, bạn chỉ cần thay CREATE bằng ALTER.

ALTER PROCEDURE MyStoredProcedure AS

...

* **Thực thi Stored Procedure**

Bạn có thể chạy một Stored Procedure bằng cách sử dụng EXECUTE hoặc EXEC. Ví dụ: để chạy Stored Procedure ở trên, hãy nhập như sau:

EXEC MyStoredProcedure

Nếu Stored Procedure có khoảng trắng bên trong tên của nó, hãy đặt nó giữa các dấu ngoặc kép:

EXEC "My Stored Procedure"

Nếu Stored Procedure của bạn cần truyên thêm các param:

EXEC MyStoredProcedure @ParameterName="MyParameter"

Sử dụng qua GUI

Bạn có thể thực thi Stored Procedure trên trực tiếp giao diện của SQL Server mà không dùng câu lệnh:

Di chuyển đến "Stored Procedures" của Database mà bạn đang làm việc.

Chọn Stored Procedure của bạn đã tạo và nhấn chuộn phải chọn "Execute Stored Procedure..."

Nếu như Stored Procedure của bạn cần param thì 1 hộp thoại sẽ xuất hiện để bạn nhập.

Nhấp "OK".

Sau đó SQL Server sẽ show kết quả thực thi cho bạn.

**Parameters**

**Parameters(tham số)** là giá trị mà Stored Procedure của bạn sử dụng để thực hiện tác vụ của nó. Khi bạn viết một Stored Procedure, bạn có thể chỉ định các Parameters cần được cung cấp từ người dùng.

Ví dụ: Nếu bạn viết một Stored Procedure để show ra chi tiết địa chỉ về một ái đó, thì Stored Procedure của bạn cần biết thông tin nào đó để tìm ra người đó. Trong trường hợp này, người dùng có thể cung cấp UserId để cho Stored Procedure lọc dữ liệu từ Database và trả về cho người dùng.

**System Stored Procedures**

SQL Server bao gồm một số lượng lớn các System Stored Procedure để hỗ trợ các việc quản trị cơ sở dữ liệu.

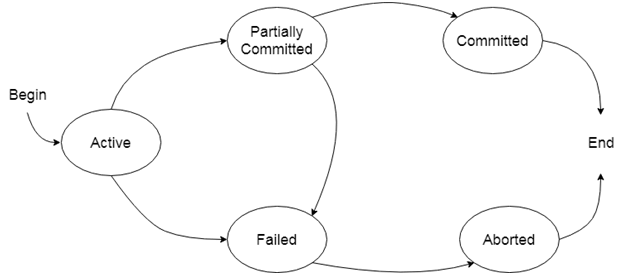
* **Một số điều bạn có thể làm với các System Stored Procedure bao gồm:**
  + Cấu hình tài khoản bảo mật.
  + Thiết lập liên kết các máy chủ.
  + Tạo kế hoạch bảo trì cơ sở dữ liệu.

1. **TRANSACTION**

* Có thể hiểu Transaction là một tiến trình xử lý có xác định điểm đầu và điểm cuối, được chia nhỏ thành các operation (phép thực thi) , tiến trình được thực thi một cách tuần tự và độc lập các operation đó theo nguyên tắc hoặc tất cả đều thành công hoặc một operation thất bại thì toàn bộ tiến trình thất bại. Nếu việc thực thi một operation nào đó bị fail đồng nghĩa với việc dữ liệu phải rollback về trạng thái ban đầu.. Transaction được xem là thành công khi và chỉ khi tất cả thao tác sửa đổi dữ liệu trong một giao dịch được thực hiện và được lưu trong cơ sở dữ liệu vĩnh viễn (dữ liệu đã được commit). Nếu Transaction được quay lui (rollback) hoặc hủy bỏ, khi đó nó có nghĩa là Transaction đã gặp các lỗi và không thực hiện bất kỳ thay đổi nào trong nội dung của cơ sở dữ liệu. Do đó, Transaction có thể được triển khai hoặc rollback.
* Transaction có 2 kiểu:
  + Flat Transaction – Transaction ngang hàng: Việc chia các operation là ngang hàng nhau. Thực thi các operation là tuần tự từ trái sang phải hoặc từ trên xuống dưới.
  + Nested Transaction – Transaction lồng nhau: Việc thực thi các operation dựa theo nguyên tắc từ trong ra ngoài. Như vậy khi nhìn vào hình vẽ chúng ta thấy các operation ở dạng này có vẻ phụ thuộc vào nhau nhưng khi thực thi thì là độc lập theo nguyên tắc operation trong thực thi xong thì mới đến operation ngoài.
* Một transaction có 4 thuộc tính:
  + Atomicity(Tính nguyên tử): Nếu transaction có nhiều phép tính thì tất cả phải được thực hiện. Nếu bất kỳ phép tính nào trong nhóm thất bại thì nó nên được rollback (trở lại trạng thái trước khi transaction bắt đầu).
  + Consistency(Tính nhất quán):
    - Các ràng buộc toàn vẹn được duy trì để cơ sở dữ liệu nhất quán trước và sau giao dịch.
    - Việc thực hiện một giao dịch sẽ để lại cơ sở dữ liệu ở trạng thái ổn định trước đó hoặc trạng thái ổn định mới.
    - Thuộc tính nhất quán của cơ sở dữ liệu nói rằng mọi giao dịch đều thấy một phiên bản cơ sở dữ liệu nhất quán.
    - Giao dịch được sử dụng để chuyển đổi cơ sở dữ liệu từ trạng thái nhất quán này sang trạng thái nhất quán khác.
  + Isolation(Sự cô lập):
    - Nó cho thấy rằng dữ liệu được sử dụng tại thời điểm thực hiện một giao dịch không thể được sử dụng bởi giao dịch thứ hai cho đến khi giao dịch đầu tiên hoàn thành.
    - Riêng biệt, nếu giao dịch T1 đang được thực hiện và sử dụng mục dữ liệu X, thì mục dữ liệu đó không thể được truy cập bởi bất kỳ giao dịch T2 nào khác cho đến khi giao dịch T1 kết thúc.
    - Hệ thống con kiểm soát đồng thời của DBMS đã thực thi thuộc tính cách ly.
  + Durability(Độ bền):
    - Thuộc tính độ bền được sử dụng để chỉ ra hiệu suất của trạng thái nhất quán của cơ sở dữ liệu. Nó nói rằng giao dịch đã thực hiện các thay đổi vĩnh viễn.
    - Chúng không thể bị mất do hoạt động sai của một giao dịch bị lỗi hoặc do lỗi hệ thống. Khi một giao dịch được hoàn thành, cơ sở dữ liệu đạt đến trạng thái được gọi là trạng thái nhất quán. Trạng thái nhất quán đó không thể bị mất, ngay cả trong trường hợp hệ thống bị lỗi.
    - Hệ thống con phục hồi của DBMS có trách nhiệm thuộc tính Độ bền.
* Cú pháp:



Các từ khóa cung cấp quyền kiểm soát đối với việc sử dụng các TRANSACTION :

* **START TRANSACTION** hoặc **BEGIN** bắt đầu một giao dịch mới.
* **COMMIT** cam kết giao dịch hiện tại, thực hiện các thay đổi của nó là vĩnh viễn.
* **ROLLBACK** quay trở lại giao dịch hiện tại, hủy các thay đổi của nó.
* **SET** autocommit tắt hoặc bật chế độ tự động gửi mặc định cho phiên hiện tại.
* Nhiều API được sử dụng để viết các ứng dụng khách MySQL (chẳng hạn như JDBC) cung cấp các phương thức riêng của chúng để bắt đầu các giao dịch có thể (và đôi khi nên) được sử dụng thay vì gửi một START TRANSACTION câu lệnh từ máy khách.
* Trạng thái TRANSACTION
* Trong cơ sở dữ liệu, giao dịch có thể ở một trong các trạng thái sau:
  + Trạng thái **ACTIVE**
    - Trạng thái **ACTIVE** là trạng thái đầu tiên của mọi giao dịch. Ở trạng thái này, giao dịch đang được thực hiện.
    - Ví dụ: Việc chèn hoặc xóa hoặc cập nhật bản ghi được thực hiện tại đây. Nhưng tất cả các bản ghi vẫn không được lưu vào cơ sở dữ liệu.
  + **Partially committed**
    - Trong trạng thái được cam kết một phần, một giao dịch thực hiện hoạt động cuối cùng của nó, nhưng dữ liệu vẫn không được lưu vào cơ sở dữ liệu.
    - Trong ví dụ tính tổng điểm, hiển thị cuối cùng của bước tổng điểm được thực hiện ở trạng thái này.
  + **Committed**
    - Một giao dịch được cho là ở trạng thái cam kết nếu nó thực hiện thành công tất cả các hoạt động của nó. Ở trạng thái này, tất cả các hiệu ứng hiện được lưu vĩnh viễn trên hệ thống cơ sở dữ liệu.
  + Trạng thái **Failed**
    - Nếu bất kỳ kiểm tra nào được thực hiện bởi hệ thống khôi phục cơ sở dữ liệu không thành công, thì giao dịch được cho là ở trạng thái không thành công.
    - Trong ví dụ về tính tổng điểm, nếu cơ sở dữ liệu không thể kích hoạt một truy vấn để tìm nạp các dấu, thì giao dịch sẽ không thực hiện được.
  + **Aborted**
    - Nếu bất kỳ lần kiểm tra nào không thành công và giao dịch đạt đến trạng thái không thành công thì hệ thống khôi phục cơ sở dữ liệu sẽ đảm bảo rằng cơ sở dữ liệu ở trạng thái nhất quán trước đó. Nếu không, nó sẽ hủy bỏ hoặc khôi phục giao dịch để đưa cơ sở dữ liệu về trạng thái nhất quán.
    - Nếu giao dịch không thành công ở giữa giao dịch thì trước khi thực hiện giao dịch, tất cả các giao dịch đã thực hiện được khôi phục về trạng thái nhất quán của nó.
    - Sau khi hủy giao dịch, mô-đun khôi phục cơ sở dữ liệu sẽ chọn một trong hai thao tác:
    - Bắt đầu lại giao dịch.
    - Hủy giao dịch.

1. **INDEX**

* Khái niệm về chỉ mục (index) trong mysql
* Chỉ mục (**Index**) là bảng tra cứu đặc biệt mà Database Search Engine có thể sử dụng để tăng nhanh thời gian và hiệu suất thu thập dữ liệu. Hiểu đơn giản, một chỉ mục là một con trỏ tới dữ liệu trong một bảng. Một chỉ mục trong một Database là tương tự như một chỉ mục trong Mục lục của cuốn sách.
* Ví dụ, nếu bạn muốn tham chiếu tất cả các trang trong một cuốn sách về một chủ đề nào đó, đầu tiên bạn nghĩ ngay đến mục lục của nó, mà liệt kê tất cả các chương, chủ đề theo thứ tự và sau đó được tham chiếu tới một hoặc nhiều trang cụ thể. Khi đã có mục lục của cuốn sách, việc tìm kiếm trang sách đó sẽ nhanh chóng và đỡ tốn công hơn rất nhiều so với việc bạn phải lật từng trang.
* **Cú pháp**