THIẾT KẾ HỆ CSDL QUAN HỆ - XÂY DỰNG CSDL VẬT LÝ

Nguyễn Đình Hóa

dinhhoa@gmail.com

094-280-7711

Sử dụng VIEW

- **CREATE TABLE** t (qty **INT**, price **INT**);
- **INSERT INTO** t **VALUES**(3, 50);
- **CREATE VIEW** v **AS**

SELECT qty, price, qty*price **AS value**

FROM t;

- **SELECT** * **FROM** v;

```
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

Sử dụng VIEW

- VIEW được dùng như một bảng dữ liệu trong một CSDL
- VIEW không được trùng tên với các bảng có trong cùng CSDL
- VIEW được tạo ra bằng bất kỳ cấu trúc câu lệnh SELECT nào
- VIEW có thể gọi bảng, hoặc các VIEW khác trong CSDL
- VIEW có thể sử dụng các lệnh kết nối, hợp, và câu truy vấn phụ
- Câu lệnh SELECT trong VIEW không nhất thiết phải gọi các bảng trong CSDL

Ví dụ về VIEW sử dụng 2 bảng

```
CREATE VIEW myview AS
SELECT a.*, b.extra_data
FROM main_table a LEFT OUTER JOIN other_table b
ON a.id = b.id;
```

Xoá VIEW

- **CREATE VIEW** few_rows_from_t1 **AS**

SELECT * **FROM** t1 **LIMIT** 10;

- **DROP VIEW** few_rows_from_t1;

- **CREATE VIEW** table_from_other_db **AS**

SELECT x **FROM** db1.foo **WHERE** x **IS NOT NULL**;

- **DROP VIEW** table_from_other_db;

Trigger

- **CREATE TABLE** account (acct_num **INT**, amount **DECIMAL**(10,2));
- **CREATE TRIGGER** ins_sum BEFORE **INSERT ON** account

  FOR EACH ROW SET @sum = @sum + NEW.amount;
- **SET** @sum = 0;
- **INSERT INTO** account **VALUES**(137,14.98),(141,1937.50),(97,-100.00);
- **SELECT** @sum **AS** 'Total amount inserted';

```
+----------------------+
| Total amount inserted |
+----------------------+
| 1852.48              |
+----------------------+
```

Trigger

- **BEFORE:** trigger được kích hoạt trước yêu cầu dữ liệu
- **AFTER:** trigger được kích hoạt sau thao tác dữ liệu.
- Các thao tác dữ liệu đối với Trigger: INSERT, UPDATE, DELETE
- Xoá Trigger:
- **DROP TRIGGER** test.ins_sum;

Ví dụ về Trigger

- DELIMITER $$
- **CREATE TRIGGER** insert_date
  BEFORE **INSERT ON** stack
  FOR EACH ROW
  BEGIN

-- set the insert_date field in the request before the insert

  SET NEW.insert_date = NOW();
  END; $$

- DELIMITER ;

Ví dụ về Trigger

- DELIMITER $$
- **CREATE TRIGGER** update_date
  BEFORE **UPDATE ON** stack
  FOR EACH ROW
  BEGIN

-- set the update_date field in the request before the update

  SET NEW.update_date = NOW();
  END; $$
- DELIMITER ;

Ví dụ về Trigger

- DELIMITER $$
- **CREATE TRIGGER** deletion_date

  AFTER DELETE ON stack

  FOR EACH ROW

  BEGIN

-- add a log entry after a successful delete

  INSERT INTO log_action(stack_id, deleted_date)
  VALUES(OLD.id, NOW());

  END; $$

- DELIMITER ;

Transaction – Giao dịch dữ liệu

- Transaction là tập hợp một chuỗi các câu lệnh SQL như SELECT, INSERT, UPDATE, DELETE. Mỗi Transaction là một khối lệnh thống nhất.
- Một TRANSACTION sẽ không thực hiện được nếu như một trong số các câu lệnh bên trong nó không được thực hiện.
- Nếu có bất kỳ lỗi nào xảy ra bên trong TRANSACTION, hệ thống sẽ trở về trạng thái trước TRANSACTION.

Các tính chất của TRANSACTION

- ***Tính nguyên tố (Atomicity):*** tất cả các thao tác của giao dịch được thực hiện hoặc không thao tác nào được thực hiện.
- ***Tính nhất quán (Consistency):*** CSDL sẽ được thay đổi một cách nhất quán sau khi TRANSACTION được thực hiện thành công.
- ***Tính biệt lập (Isolation):*** các giao dịch được thực hiện một cách độc lập, không liên quan đến nhau về mặt dữ liệu cũng như thao tác dữ liệu.
- ***Tính duy trì (Durability):*** những thay đổi tới CSDL bởi một giao dịch sẽ không bị mất đi ngay cả khi hệ thống có lỗi ngay sau khi giao dịch hoàn thành.

TRANSACTION

- Một Transaction được khởi tạo bằng
- **START TRANSACTION** hoặc **BEGIN WORK**
- Kết thúc bằng câu lệnh **COMMIT** hoặc **ROLLBACK**
- Ví dụ
- **START TRANSACTION**;
- **SET** @transAmt = '500';
- **SELECT** @availableAmt:=ledgerAmt **FROM** accTable **WHERE** customerId=1 FOR **UPDATE**;
- **UPDATE** accTable **SET** ledgerAmt=ledgerAmt-@transAmt **WHERE** customerId=1;
- **UPDATE** accTable **SET** ledgerAmt=ledgerAmt+@transAmt **WHERE** customerId=2;
- **COMMIT**;

Ví dụ về TRANSACTION

- Trong ví dụ trước, FOR **UPDATE** chỉ định (và khoá) bản ghi được yêu cầu trong suốt quá trình thực hiện giao dịch.
- Khi giao dịch vẫn đang diễn ra (chưa được commit) thì giao dịch đó không được sử dụng bởi người dùng khác.
- Với lệnh **START TRANSACTION**, chức năng autocommit sẽ bị vô hiệu hoá cho đến khi thực hiện lệnh **COMMIT** hoặc **ROLLBACK**. Sau lệnh này, chức năng autocommit lại được kích hoạt về trạng thái ban đầu.

Trình tự thực hiện TRANSACTION

- Bắt đầu một giao dịch bằng câu lệnh SQL **BEGIN WORK** hoặc **START TRANSACTION**.
- Chạy tất cả các câu lệnh SQL có bên trong TRANSACTION.
- Kiểm tra xem mọi yêu cầu có được thực hiện không.
- Nếu mọi thứ đều ổn, thực hiện lệnh **COMMIT**, nếu không sẽ thực hiện lệnh **ROLLBACK** để đưa hệ thống về trạng thái trước đó.

AUTOCOMMIT

- MySQL sẽ tự động thực hiện tất cả các câu lệnh không thuộc TRANSACTION.
- Kết quả của các câu lệnh **UPDATE**,**DELETE** hoặc **INSERT** không nằm trong **TRANSACTION** sẽ được thể hiện trong CSDL và có thể sử dụng cho các câu lệnh khác.
- AUTOCOMMIT được mặc định là TRUE. Ta có thể thay đổi như sau
- **SET** AUTOCOMMIT=**false**;

Hoặc

- **SET** AUTOCOMMIT=0;

AUTOCOMMIT

thiết lập autocommit về **true**

- **SET** AUTOCOMMIT=**true**;

hoặc

- **SET** AUTOCOMMIT=1;
- Để xem trạng thái hiện tại của AUTOCOMMIT:
- **SELECT** @@autocommit;