SQL for Developer

Giảng viên: Bùi Quang Đăng

Contents

SQL for Programming

THỦ TỤC - STORED PROCEDURES

Stored Procedures

- **Thủ tục (stored procedures)** là một nhóm của một hoặc nhiều câu lệnh SQL được viết để thực hiện xử lý công việc trong MySQL Server. Nó giống như cấu trúc lập trình trong các ngôn ngữ lập trình khác bao gồm các đặc điểm sau:
  - Cho phép tham số truyền vào và trả về nhiều giá trị theo tham số trả về trong chương trình.
  - Chứa các câu lệnh xử lý với cơ sở dữ liệu bao gồm cả việc gọi các thủ tục khác.
  - Trả về trạng thái của chương trình để biết thành công hay thất bại

Stored Procedures

- **Cú pháp sử dụng**: Tạo ra một T-SQL hoặc common language runtime (CLR) thủ tục trong SQL Server

```
Delimiter $$
Create PROCEDURE [Procedure_Name](in|out parameters)
Begin
        [Câu lệnh SQL];
End$$;
Delimiter ;
```

Stored Procedures

- **Cú pháp sử dụng**:
  - **Delimiter**: Là từ khoá để tạo thủ tục trong MySQL
  - **procedure_name**: Tên thủ tục phải là duy nhất trong database
  - **@ parameter**: Khai báo các tham số của thủ tục
  - [type_schema_name**.**] **data_type**: Kiểu dữ liệu của các tham số
  - default: Giá trị mặc định phải là hằng số hoặc có thể NULL
  - OUT | OUTPUT: Tham số trả về của thủ tục.

Stored Procedures

- Ví dụ:
  - Thủ tục lấy tất cả nhân viên:

```
1 Delimiter $$
2 Create PROCEDURE SP_LayDanhSachNhanVien()
3 Begin
4     Select * from NhanVien;
5 End$$;
6 DELIMITER ;
```

Stored Procedures

- Ví dụ:
  - Thủ tục lấy chi tiết nhân viên:

```
1 Delimiter $$
2 Create PROCEDURE SP_LayChiTietNhanVien(
3 ma varchar(10)
4 )
5 Begin
6     Select * from NhanVien where MaNV = ma;
7 End$$;
8 DELIMITER ;
```

Stored Procedures

- **Gọi thủ tục để sử dụng:**

```
#Lấy danh sách nhân viên
Call SP_LayDanhSachNhanVien();

#Lấy chi tiết nhân viên theo mã
Call SP_LayChiTietNhanVien('SF001');
```

Stored Procedures

- **Hiển thị các thủ tục trong db với cú pháp sau:**

```
SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search_condition]
```

  - **Ví dụ:**

```
#Xem các thủ tục trong bookstore

SHOW PROCEDURE STATUS WHERE db = 'bookstore';
```

Name	Type	Definer
SP_LayChiTietNhanVien	PROCEDURE	root@localhost
SP_LayDanhSachNguoiDung	PROCEDURE	root@localhost
SP_LayDanhSachNhanVien	PROCEDURE	root@localhost

Stored Procedures

- **Xoá thủ tục sử dụng cú pháp sau:**

```
DROP PROCEDURE [IF EXISTS] stored_procedure_name;
```

  - **Ví dụ:**

```
#Xoá thủ tục lấy danh sách nhân viên

Drop procedure if exists
SP_LayDanhSachNhanVien
```

HÀM - FUNCTION

Function

- **Hàm (function)** là một nhóm của một hoặc nhiều câu lệnh SQL được viết để xử lý công việc trong MySQL Server và có kết quả trả về sau lời gọi hàm. Thường sẽ trả về một giá trị sau khi gọi hàm.

Function

- **Hàm (function)** được sử dụng trong các trường hợp sau:
  - Trong khối SQL của câu lệnh Select
  - Gọi từ ứng dụng
  - Sử dụng trong hàm khác
  - Sử dụng trong khung nhìn (view)
  - Trong định nghĩa cột của bảng
  - Sử dụng trong từ khóa CHECK khi tạo ràng buộc

Function

Cú pháp sử dụng hàm (function)

```
DELIMITER $$
CREATE FUNCTION function_name( param1, param2,… )
RETURNS datatype
[NOT] DETERMINISTIC
BEGIN
-- statements
END $$
DELIMITER ;
```

- Mặc định dùng not DETERMINISTIC giá trị trả về khác giá trị của tham số truyền vào

Chú ý: Khi viết hàm hoặc thủ tục nếu có nhiều xử lý sẽ Khai báo và viết trong khối BEGIN…END

Function

- Ví dụ: Viết 1 hàm đếm số lượng nhân viên trong bảng NhanVien

```
1  Delimiter $$
2  Create function F_TongSoNhanVien()
3  Returns INT
4  BEGIN
5  #Khai báo biến
6  Declare tongSo int;
7  #Lấy giá trị của câu lệnh gán cho biến
8  set tongSo = (Select count(*) from NhanVien);
9  #Trả về kết quả
10 return tongSo;
11 End$$
12 Delimiter;
```

Function

- Ví dụ: Gọi hàm tính tổng số nhân viên sau khi đã tạo hàm

```
#Gọi hàm tính tổng nhân viên
Select F_TongSoNhanVien();
```

Function

- **Hiển thị các hàm trong MySQL với cú pháp sau:**

```
SHOW FUNCTION STATUS [LIKE 'pattern' | WHERE search_condition];
```

  - **Ví dụ:**

```
#Hiển thị các hàm trong db bookstore
SHOW FUNCTION STATUS
WHERE db = 'bookstore';
```

Function

- **Xoá hàm trong MySQL với cú pháp sau:**

```
DROP FUNCTION [IF EXISTS] function_name;
```

  - **Ví dụ:**

```
Drop function if exists F_TongSoNhanVien
```

KHUNG NHÌN - VIEW

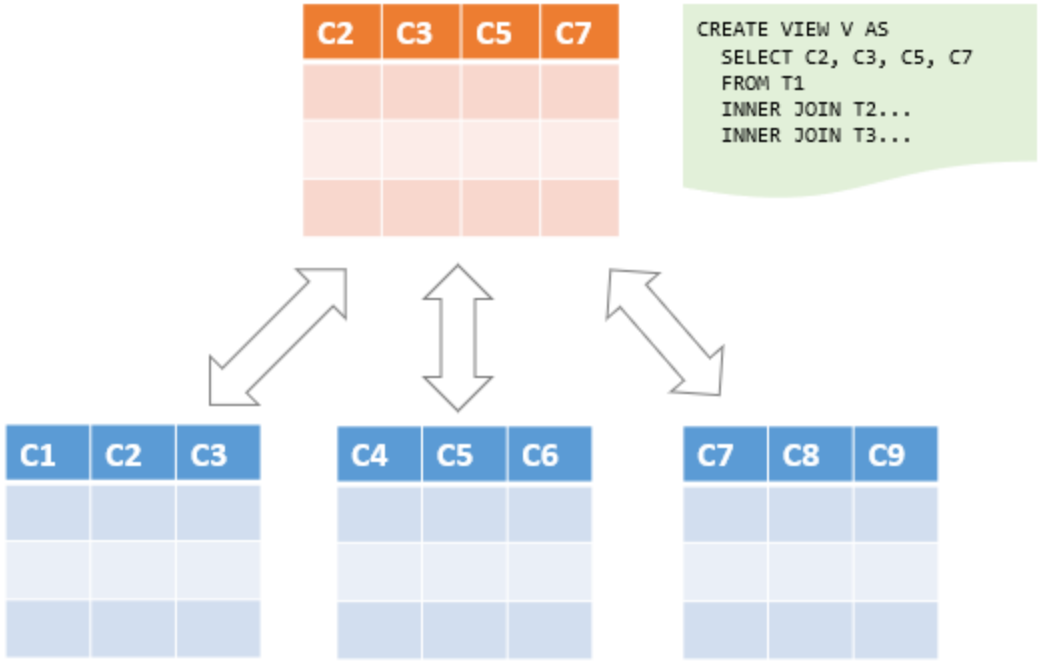
View

- **Khung nhìn (view)** là tạo ra một bảng ảo (columns và rows) khi thực hiện truy vấn lấy thông tin và nó có thể lấy từ nhiều bảng khác nhau trong cơ sở dữ liệu.
- **Cú pháp khai báo:**

```
CREATE [OR REPLACE] VIEW [db_name.]view_name
[(column_list)]
AS select-statement;
```

View

- **Khung nhìn (view)** là tạo ra một bảng ảo (columns và rows) khi thực hiện truy vấn lấy thông tin và nó có thể lấy từ nhiều bảng khác nhau trong cơ sở dữ liệu.



View

- Ví dụ: Hiển thị bảng thông tin nhân viên có thông tin tên phòng như sau:

```
#Tạo view thông tin nhân viên phòng
Create view vNhanVienPhong
AS
Select MaNV, HoTen, DienThoai, Email, DiaChi, TenPhong
from NhanVien nv inner join PhongBan pb on nv.MaPhong =
pb.MaPhong;
```

Lấy thông tin từ khung nhìn: **Select * from vNhanVienPhong**;

View

- **Xem khung nhìn bằng câu lệnh:**

```
Show full tables
WHERE table_type = 'VIEW';
#Xem của db nào đó
SHOW FULL TABLES [{FROM | IN } database_name]
WHERE table_type = 'VIEW';
```

Hiển thị thông tin có chứa các view

ChuDe	BASE TABLE
HeSoLuong	BASE TABLE
MonHoc	BASE TABLE
NguoiDung	BASE TABLE
NhanVien	BASE TABLE
NhanVien1	BASE TABLE
PhongBan	BASE TABLE
Sach	BASE TABLE
Sach1	BASE TABLE
vnhanvienphong	VIEW

View

- **Đổi tên của khung nhìn với cú pháp sau:**

```
RENAME TABLE original_view_name TO
new_view_name;
```

  - Ví dụ:

```
Rename table vNhanVienPhong to vNhanVienVaPhong
```

View

- **Xoá khung nhìn với cú pháp sau:**

```
DROP VIEW [IF EXISTS] view_name;

#Xoá nhiều view
DROP VIEW [IF EXISTS] view_name1 [,view_name2]...;
```

  - Ví dụ:

```
drop view if exists vNhanVienPhong
```

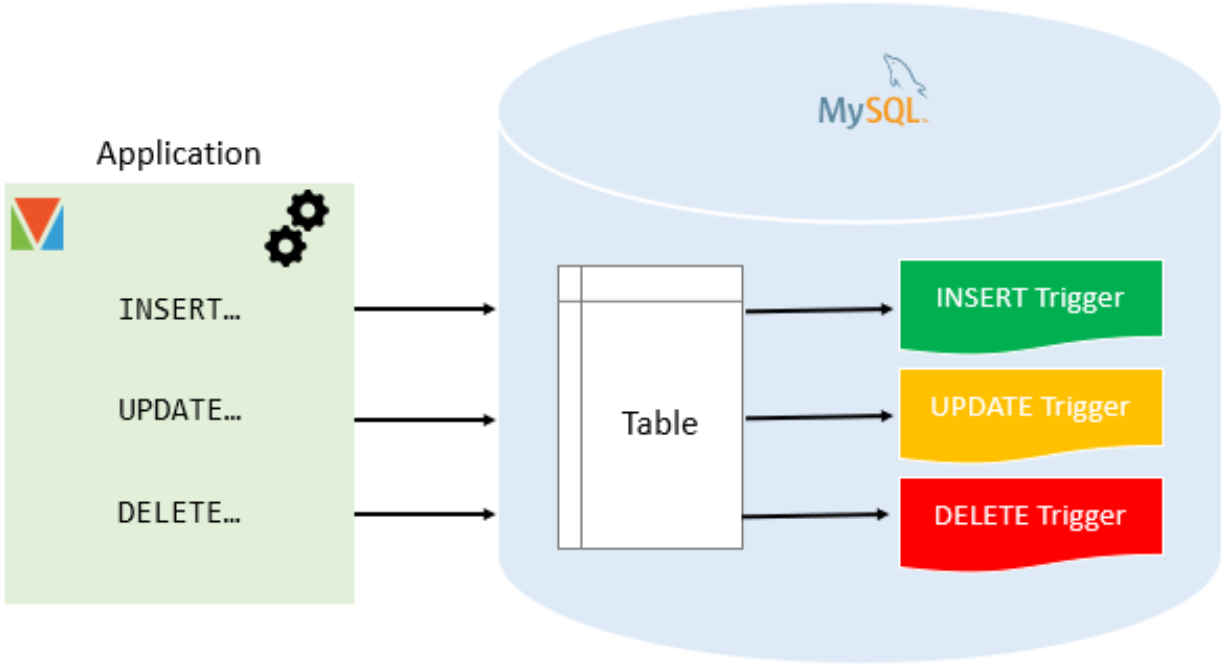
TRIGGER

Trigger

- **Trigger** là một thủ tục không có tham số được thực hiện tự động khi người dùng sử dụng các câu lệnh insert, update, delete thay đổi dữ liệu trong bảng trên SQL.
  - Trigger có thể sử dụng để kiểm tra ràng buộc, đúng đắn của số liệu
  - Thực hiện tự động một công việc nào đó khi có thay đổi từ bảng

Trigger

- **Trigger** là một thủ tục không có tham số được thực hiện tự động khi người dùng sử dụng các câu lệnh insert, update, delete thay đổi dữ liệu trong bảng trên SQL.



Trigger

- **Cú pháp khai báo:**

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE } ON
table_name FOR EACH ROW
trigger_body;
```

Trigger

- **Một số chú ý khi sử dụng Trigger:**
  - Các khối lệnh được viết trong **Begin** …**End**
  - Sử dụng **before**, **after** để xử lý trước, sau khi hành động thực hiện trên bang dung trigger
  - Sử dụng **OLD** để lấy thông tin cũ ví dụ OLD.MaNV
  - Sử dụng **NEW** để lấy thông tin mới ví dụ NEW.MaNV

Trigger

- **Tạo Trigger trong MySQL:**

```
#Tạo bảng NhatKy
Create table NhatKy
(
 Id int primary key AUTO_INCREMENT,
 NoiDung varchar(1000),
 HanhDong varchar(20),
 NgayTao date
);
```

Trigger

- **Tạo Trigger trong MySQL:**

```
DELIMITER $$
Create trigger Auto_NhatKy_Trigg
Before insert on NhanVien
For each ROW
Begin
Insert into NhatKy(NoiDung, HanhDong, NgayTao)
values(Concat('Thêm nhân viên có mã: ', NEW.MaNV, ', họ
tên: ', NEW.HoTen, ', Điện thoại: ', NEW.DienThoai, ',
Email: ', NEW.Email), 'ThemMoi', NOW());
End$$

```

Trigger

- **Xem các Trigger trong MySQL:**

```
SHOW TRIGGERS [{FROM | IN} database_name]
[LIKE 'pattern' | WHERE search_condition];
```

  - Ví dụ:

```
SHOW TRIGGERS
#Xem trigger trong db bookstore
SHOW TRIGGERS
from bookstore;
#Xem trigger của bảng NhanVien
SHOW TRIGGERS FROM bookstore WHERE table =
'NhanVien';
```

Trigger

- **Xoá Trigger trong MySQL:**

```
DROP TRIGGER [IF EXISTS]
[schema_name.]trigger_name;
```

  - Ví dụ:

```
#Xoá thông tin trigger

Drop trigger Auto_NhatKy_Trigg
```

INDEX

Index

- **Index** là một kiểu cấu trúc B-Tree, lưu dữ liệu dưới dạng các node có sắp xếp theo thứ tự nhất định trên SQL.
  - Sử dụng index để đánh chỉ mục, tăng hiệu xuất tìm kiếm trên bảng với các thông tin thường xuyên sử dụng tìm kiếm, không thay đổi dữ liệu nhiều
  - Các thông tin primary key, unique trong bảng sẽ được tự động đánh index

Index

- **Cú pháp tạo Index:**

```
CREATE TABLE t(
c1 INT PRIMARY KEY,
c2 INT NOT NULL,
c3 INT NOT NULL,
c4 VARCHAR(10),
INDEX (c2,c3)
);

Hoặc

CREATE INDEX index_name ON table_name (column_list)
```

Index

- **Tạo index cho trường:**

```
1 #Tạo chỉ mục cho trường thông tin Họ
  tên
2 Create index idx_hoten on
  NhanVien(HoTen);
```

Index

- **Xem thông tin Index:**

```
SHOW INDEXES FROM table_name
IN database_name;
```

- **Ví dụ:**

```
SHOW INDEXES FROM NhanVien
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Null	Index_type
nhanvien	0	PRIMARY	1	MaNV		BTREE
nhanvien	0	UIQ_DienThoai	1	DienThoai	YES	BTREE
nhanvien	1	FK_PhongBan	1	MaPhong	YES	BTREE
nhanvien	1	idx_hoten	1	HoTen	YES	BTREE

Index

- **Xoá Index:**

```
DROP INDEX index_name ON table_name
[algorithm_option | lock_option];
```

  - **Ví dụ:**

```
DROP INDEX idx_hoten ON NhanVien;
```

Index

- **Xoá Index:**
  - ALGORITHM [=] {DEFAULT|INPLACE|COPY}
    - COPY: Copy ra bảng mới sau đó xoá các index ở bang gốc
    - INPLACE: Bảng sẽ được rebuild lại và copy ra dòng mới. Các metadata sẽ được khoá khi xoá index.

Index

- **So sánh khi chưa và sau khi đánh index:**

```
EXPLAIN
Select MaNV, HoTen, DiaChi
from NhanVien
where HoTen like '%Nguyễn%';
```

Chưa đánh index cho Họ tên

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	NhanVien	ALL	*NULL*	*NULL*	*NULL*	*NULL*	7	Using where

Sau đánh index cho Họ tên

Exercises


Stanford
nâng tầm tri thức
Thank You !
www.stanford.com.vn