**Báo cáo**

1. **Sql**

* Là 1 ngôn ngữ tiêu chuẩn để lưu trữ, thao tác và truy xuất dữ liệu trong cơ sở dữ liệu
* Sử dụng mô hình dữ liệu quan hệ với các bảng và liên kết ràng buộc giữa các bảng

1. **No sql**

2.1 Tổng quan

* Là 1 thuật ngữ để mô tả bất kỳ hệ thống thay thế nào cho cơ sở dữ liệu sql truyền thống
* Cơ sở dữ liệu noSQL có cấu trúc khác với mô hình bảng, hàng và cột truyền thống được sử dụng với Hệ thống quản lý cơ sở dữ liệu quan hệ (RDBMS).
* Tốc độ phát triển với cơ sở dữ liệu NoSQL có thể nhanh hơn nhiều so với cơ sở dữ liệu SQL
* Cấu trúc của nhiều hình thức dữ liệu khác nhau được xử lý và phát triển dễ dàng hơn với cơ sở dữ liệu NoSQL
  1. Các loại cơ sở dữ liệu noSql

Có 4 loại cơ sở dữ liệu noSql chính

* + 1. Document databases
* Được lưu trữ dưới dạng JSON, BSON hoặc XML
* Phổ biến với các nhà phát triển vì có tính linh hoạt để làm lại các cấu trúc tài liệu để phù hợp với ứng dụng, định hình cấu trúc dữ liệu khi các yêu cầu ứng dụng thay đổi theo thời gian.
* Tốc độ linh hoạt này tăng tốc độ phát triển bởi vì, trên thực tế, dữ liệu trở nên giống như mã và nằm dưới sự kiểm soát của các nhà phát triển. Còn trong SQL thì sẽ phải thay đổi cấu trúc của cơ sở dữ liệu
  + 1. Key-value stores
* Được lưu trữ dưới dạng key – value
* Sử dụng trong các trường hợp như giỏ hàng, thông tin người dùng, …
  + 1. Column-orented databases
    2. Graph databases
* Tập trung vào mối quan hệ giữa các yếu tố dữ liệu. Mỗi yếu tố được lưu trữ dưới dạng node. Các kết nối giữa các yếu tố được gọi là liên kết hoặc các mối quan hệ. Khác với cơ sở dữ liệu quan hệ, các liên kết được sử dụng ngầm để thể hiện mối quan hệ thì trong Graph databases các liên kết được lưu trữ trực tiếp
* Được tối ưu hóa để ghi lại và tìm kiếm các kết nối giữa các thành phần dữ liệu, khắc phục chi phí việc kết nối nhiều bảng trong SQL

1. **Join**

3.1 Cú pháp

*table\_references:*

*escaped\_table\_reference* [, *escaped\_table\_reference*] ...

*escaped\_table\_reference*: {

*table\_reference*

| { OJ *table\_reference* }

}

*table\_reference*: {

*table\_factor*

| *joined\_table*

}

*table\_factor*: {

*tbl\_name* [PARTITION (*partition\_names*)]

[[AS] *alias*] [*index\_hint\_list*]

| [LATERAL] *table\_subquery* [AS] *alias* [(*col\_list*)]

| ( *table\_references* )

}

*joined\_table*: {

*table\_reference* {[INNER | CROSS] JOIN | STRAIGHT\_JOIN} *table\_factor* [*join\_specification*]

| *table\_reference* {LEFT|RIGHT} [OUTER] JOIN *table\_reference* *join\_specification*

| *table\_reference* NATURAL [INNER | {LEFT|RIGHT} [OUTER]] JOIN *table\_factor*

}

*join\_specification*: {

ON *search\_condition*

| USING (*join\_column\_list*)

}

*join\_column\_list*:

*column\_name* [, *column\_name*] ...

*index\_hint\_list*:

*index\_hint* [, *index\_hint*] ...

*index\_hint*: {

USE {INDEX|KEY}

[FOR {JOIN|ORDER BY|GROUP BY}] ([*index\_list*])

| {IGNORE|FORCE} {INDEX|KEY}

[FOR {JOIN|ORDER BY|GROUP BY}] (*index\_list*)

}

*index\_list*:

*index\_name* [, *index\_name*] ...

* 1. Các loại join
* Inner join

Trả về giá trị có ít nhất 1 giá trị ở 2 bảng

Ảnh có chứa biểu đồ

Mô tả được tạo tự động

* Left join

Trả lại tất cả các dòng từ bảng bên trái, và các dòng đúng với điều kiện từ bảng bên phải

Ảnh có chứa biểu đồ

Mô tả được tạo tự động

* Right join

Trả lại tất cả các hàng từ bảng bên phải, và các dòng thỏa mãn điều kiện từ bảng bên trái

Ảnh có chứa biểu đồ

Mô tả được tạo tự động

1. **Function**

4.1 Cú pháp

CREATE

[DEFINER = *user*]

FUNCTION [IF NOT EXISTS] *sp\_name* ([*func\_parameter*[,...]])

RETURNS *type*

[*characteristic* ...] *routine\_body*

*func\_parameter*:

*param\_name* *type*

*type*:

*Any valid MySQL data type*

*characteristic*: {

COMMENT '*string*'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

}

*routine\_body*:

*Valid SQL routine statement*

* 1. Ví dụ

CREATE FUNCTION hello (s CHAR(20))

RETURNS CHAR(50) DETERMINISTIC

RETURN CONCAT('Hello, ',s,'!');

1. **Procedure**

5.1 Cú pháp

CREATE

[DEFINER = *user*]

PROCEDURE [IF NOT EXISTS] *sp\_name* ([*proc\_parameter*[,...]])

[*characteristic* ...] *routine\_body*

*proc\_parameter*:

[ IN | OUT | INOUT ] *param\_name* *type*

*type*:

*Any valid MySQL data type*

*characteristic*: {

COMMENT '*string*'

| LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

}

* 1. Ví dụ

delimiter //

CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)

BEGIN

SELECT COUNT(\*) INTO cities FROM world.city

WHERE CountryCode = country;

END//

delimiter ;

1. **Transaction**

6.1 Cú pháp

START TRANSACTION

[*transaction\_characteristic* [, *transaction\_characteristic*] ...]

*transaction\_characteristic*: {

WITH CONSISTENT SNAPSHOT

| READ WRITE

| READ ONLY

}

BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET autocommit = {0 | 1}Index

* 1. ví dụ

START TRANSACTION;

SELECT ***@A***:=SUM(salary) FROM table1 WHERE type=1;

UPDATE table2 SET summary=***@A*** WHERE type=1;

COMMIT;

1. **Index**

Chỉ mục được sử dụng để tìm các hàng có giá trị cột cụ thể một cách nhanh chóng. Không có chỉ mục, MySQL phải bắt đầu với hàng đầu tiên và sau đó đọc qua toàn bộ bảng để tìm các hàng có liên quan. Bảng càng lớn, chi phí này càng cao. Nếu bảng có chỉ mục cho các cột được đề cập, MySQL có thể nhanh chóng xác định vị trí cần tìm ở giữa tệp dữ liệu mà không cần phải xem tất cả dữ liệu. Điều này nhanh hơn nhiều so với đọc từng hàng một cách tuần tự.