POSTGRESQL (1)

NỘI DUNG

1. Tổng quan
2. Cơ bản về database
3. Join
4. View
5. Giới thiệu Index & Partition

1. TỔNG QUAN

- **Postgresql Server version 9.5.13:**
  https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
- **Toolui** để truy cập và thao tác với DB pgAdmin III.

- **SQL (Structured Query Language):** Ngôn ngữ truy vấn và thao tác dữ liệu
  - Thêm, sửa, xóa dữ liệu
  - Truy vấn dữ liệu
- **Định nghĩa dữ liệu DDL** – Data Definition Language:
  - Create Database, Drop Database, Create Table, Alter Table.
- **Thao tác dữ liệu DML** – Data Manipulation Language:
  - Select, Insert, Update, Delete.

2. CƠ BẢN VỀ DATABASE

- 1 cơ sở dữ liệu có thể bao gồm nhiều bảng
- 1 bảng có thể bao gồm nhiều cột
- 1 cột có thể có hoặc không có những thuộc tính (ràng buộc dữ liệu)

- Một số kiểu dữ liệu cơ bản thường dùng:
  - Kiểu số: integer, smallint, bigint, serial, smallserial, bigserial, double
  - Kiểu ký tự: character, character varying, text
  - Kiểu datetime: date, time, timestamp
  - Kiểu boolean: boolean
  - Kiểu json: json, jsonb

TẠO DATABASE

CREATE DATABASE database_name

XÓA DATABASE

DROP DATABASE [IF EXISTS] database_name

TẠO TABLE (BẢNG)

CREATE TABLE [IF NOT EXISTS] table_name (
column_name data_type [column_constraint]
| table_constraint
)

CONSTRAINT TRONG POSTGRES

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

CẬP NHẬT TABLE

- Lệnh ALTER TABLE trong SQL được sử dụng để thêm, xóa hoặc sửa đổi các cột trong một bảng đang tồn tại.
- Sử dụng lệnh ALTER TABLE để thêm và xóa các ràng buộc (Constraint) trên một bảng đang tồn tại.

- Thêm:

ALTER TABLE table_name ADD column_name data_type

- Xóa:

ALTER TABLE table_name DROP COLUMN column_name

- Sửa:

ALTER TABLE table_name ALTER COLUMN column_name TYPE data_type

XÓA TABLE

DROP TABLE [IF EXISTS] table_name

INSERT DỮ LIỆU

INSERT INTO table_name [(column_name [, ...])]
VALUES ([, ...])

UPDATE DỮ LIỆU

```
UPDATE table_name
SET { column_name = { expression } |
( column_name [, ...] ) = ( { expression } [, ...] ) } [, ...]
[ FROM from_list ]
[ WHERE condition ]
```

XÓA DỮ LIỆU

DELETE FROM table_name
[**USING** using_list]
[**WHERE** condition]

TRUY VẤN DỮ LIỆU

SELECT [* | list_column_name]
FROM table_name
[**WHERE** condition]
[**GROUP BY** expression [, ...]]
[**HAVING** condition [, ...]]
[**ORDER BY** expression [ASC | DESC]
[**LIMIT** { count | ALL }]

TOÁN TỬ SO SÁNH

Toán tử	Miêu tả	Ví dụ
=	Bằng	(a = b) là không true
!=	Khác	(a != b) là true
<>	Khác	(a <> b) là true
>	Lớn hơn	(a > b) là không true
<	Nhỏ hơn	(a < b) là true
>=	Lớn hơn hoặc bằng	(a >= b) là không true
<=	Nhỏ hơn hoặc bằng	(a <= b) là true
!<	Nhỏ hơn hoặc khác	(a !< b) là false
!>	Lớn hơn hoặc khác	(a !> b) là true

TOÁN TỬ LOGIC

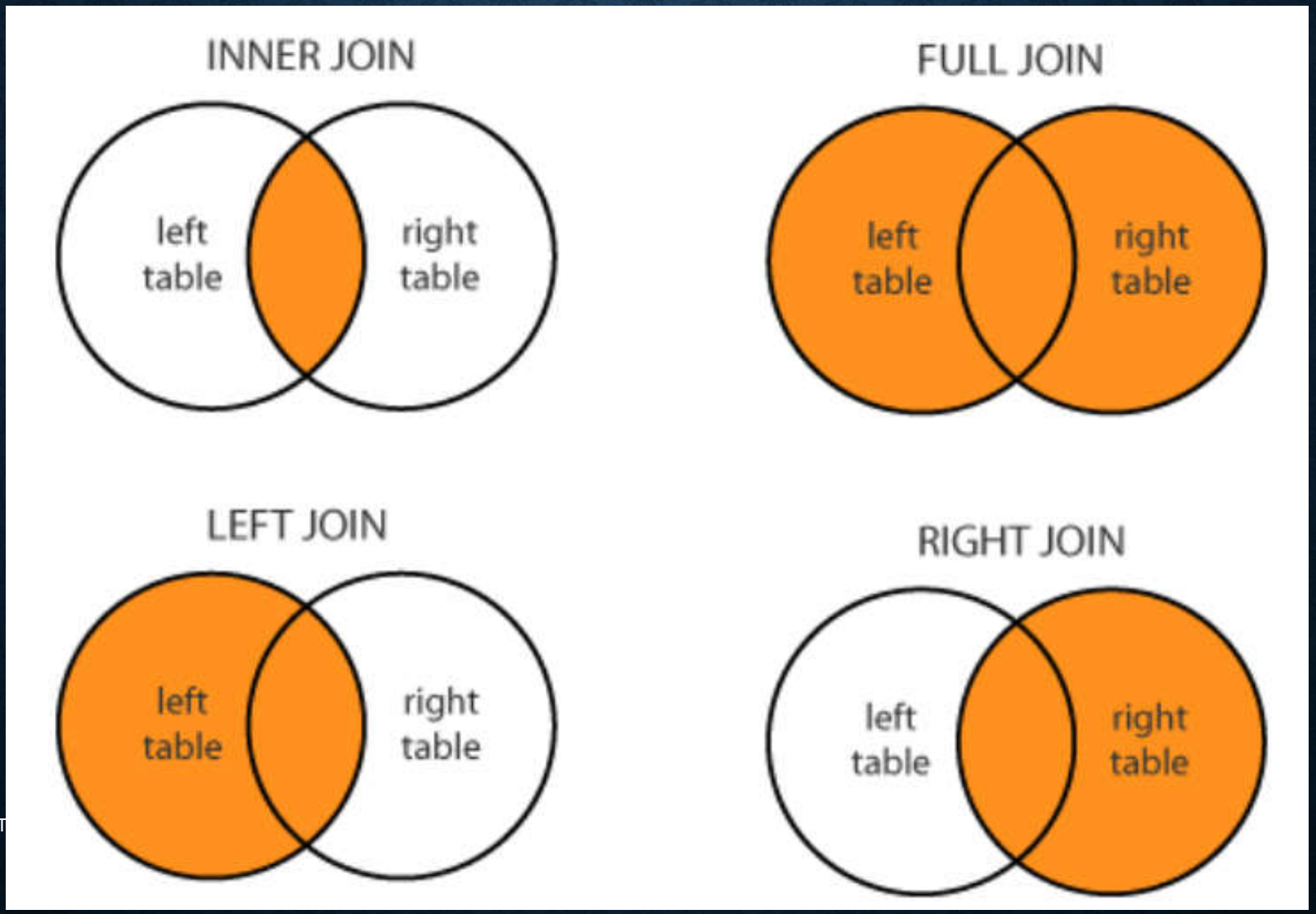
Toán tử	Miêu tả
AND	Toán tử AND cho phép sự tồn tại của nhiều điều kiện trong mệnh đề WHERE của một lệnh SQL
BETWEEN	Toán tử BETWEEN được sử dụng để tìm các giá trị mà là trong một tập giá trị, được cung cấp giá trị nhỏ nhất và giá trị lớn nhất
IN / NOT IN	Toán tử IN được sử dụng để so sánh một giá trị với một danh sách các giá trị hằng mà đã được xác định
LIKE/ NOT LIKE	Toán tử LIKE được sử dụng để so sánh một giá trị với các giá trị tương tự bởi sử dụng các toán tử Wildcard
NOT	Toán tử NOT đảo ngược ý nghĩa của toán tử logic khi được sử dụng cùng với toán tử logic đó. Ví dụ: **NOT EXISTS, NOT BETWEEN, NOT IN**, ... **Đây là một toán tử phủ định**
OR	Toán tử OR được sử dụng để kết hợp nhiều điều kiện trong mệnh đề WHERE của một lệnh SQL
IS NULL	Toán tử IS NULL được sử dụng để so sánh một giá trị với một giá trị NULL

HÀM HỮU ÍCH TRONG SQL

- Hàm **COUNT** trong SQL - Hàm tập hợp COUNT trong SQL được sử dụng để đếm số hàng trong một bảng của cơ sở dữ liệu.
- Hàm **MAX** trong SQL - Hàm tập hợp MAX trong SQL cho phép chúng ta lựa chọn giá trị lớn nhất cho một cột cụ thể.
- Hàm **MIN** trong SQL - Hàm tập hợp MIN trong SQL cho phép chúng ta lựa chọn giá trị nhỏ nhất cho một cột cụ thể.
- Hàm **AVG** trong SQL - Hàm tập hợp AVG trong SQL cho giá trị trung bình cho một cột cụ thể trong bảng.
- Hàm **SUM** trong SQL - Hàm tập hợp SUM trong SQL tính tổng một cột dạng giá trị số.
- Hàm **SQRT** trong SQL - Hàm này được sử dụng để cho căn bậc hai của một số đã cho.
- Hàm **RAND** trong SQL - Hàm này được sử dụng để tạo một số ngẫu nhiên bởi sử dụng lệnh SQL.
- Hàm **CONCAT** trong SQL - Hàm này được sử dụng để nối chuỗi bên trong bất kỳ lệnh SQL nào.
- **Từ khóa DISTINCT được sử dụng kết hợp với lệnh SELECT để loại tất cả các bản sao của bản ghi và chỉ lấy các bản ghi duy nhất.**

3. JOIN

- **INNER JOIN** trả về các bản ghi có giá trị phù hợp giữa hai bảng
- **LEFT JOIN** mọi bản ghi bảng bên trái trả về kết hợp với bản ghi phù hợp với bên phải nếu có (nếu không có thì nhận NULL)
- **RIGHT JOIN** mọi bản ghi bảng bên phải trả về có kết hợp với giá trị phù hợp nếu có ở bảng trái
- **OUTER JOIN** (full join) mọi bản ghi ở bảng trái và bảng phải kết hợp lại


INNER JOIN
left table
right table
FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table

INNER JOIN

- Sử dụng INNER JOIN để kết nối dữ liệu hai bảng với nhau
- Cú pháp:

```
SELECT column_list
FROM t1 INNER JOIN t2 ON join_condition1
WHERE where_conditions;
```

VÍ DỤ 1

LopID	TenLop
1	CNTT
2	TOAN
3	HOA HOC

MaSV	TenSV	NamSinh	LopID
1	Nguyen Van A	1990	1
2	Nguyen Van B	1991	1
3	Nguyen Van C	1980	2
4	Nguyen Van D	1976	3
5	Nguyen Van E	1990	2
6	Nguyen Van F	1954	3
7	Nguyen Van G	1967	1
8	Nguyen Van H	1978	1
9	Nguyen Van I	1991	2
10	Nguyen Van K	2000	3

- Giả sử chúng ta cần viết một câu truy vấn xem danh sách sinh viên và lớp mà sinh viên đó đang học thì chúng ta dựa vào khóa ngoại (foreign key) của bảng **sinhvien** và khóa chính của bảng **lop** để truy vấn.
- Có thể sử dụng tích đề cát hoặc **INNER JOIN**.

- Sử dụng tích đề cát:

```
SELECT *
FROM sinhvien, lop
WHERE sinhvien.LopID = lop.LopID
```

- Sử dụng INNER JOIN:

```
SELECT *
FROM sinhvien INNER JOIN lop ON sinhvien.LopID = lop.LopID
```

MaSV	TenSV	NamSinh	LopID	LopID	TenLop
1	Nguyen Van A	1990	1	1	CNTT
2	Nguyen Van B	1991	1	1	CNTT
7	Nguyen Van G	1967	1	1	CNTT
8	Nguyen Van H	1978	1	1	CNTT
3	Nguyen Van C	1980	2	2	TOAN
5	Nguyen Van E	1990	2	2	TOAN
9	Nguyen Van I	1991	2	2	TOAN
4	Nguyen Van D	1976	3	3	HOA HOC
6	Nguyen Van F	1954	3	3	HOA HOC
10	Nguyen Van K	2000	3	3	HOA HOC

- Với phép tích thì sau khi tích hai bảng lại với nhau nó sẽ có tổng cộng là 10 x 3 = 30 records, sau đó ở mỗi record nó sẽ kiểm tra điều kiện nếu sinhvien.LopID = lop.LopID đúng thì record đó sẽ được chọn, ngược lại thì không được chọn.
- Với INNER JOIN thì khác, trong quá trình thực hiện tích hai bảng nó sẽ kiểm tra điều kiện ở ON (tức là sinhvien.LopID = lop.LopID), nếu đúng thì được chọn và sai thì bỏ qua.

VÍ DỤ 2

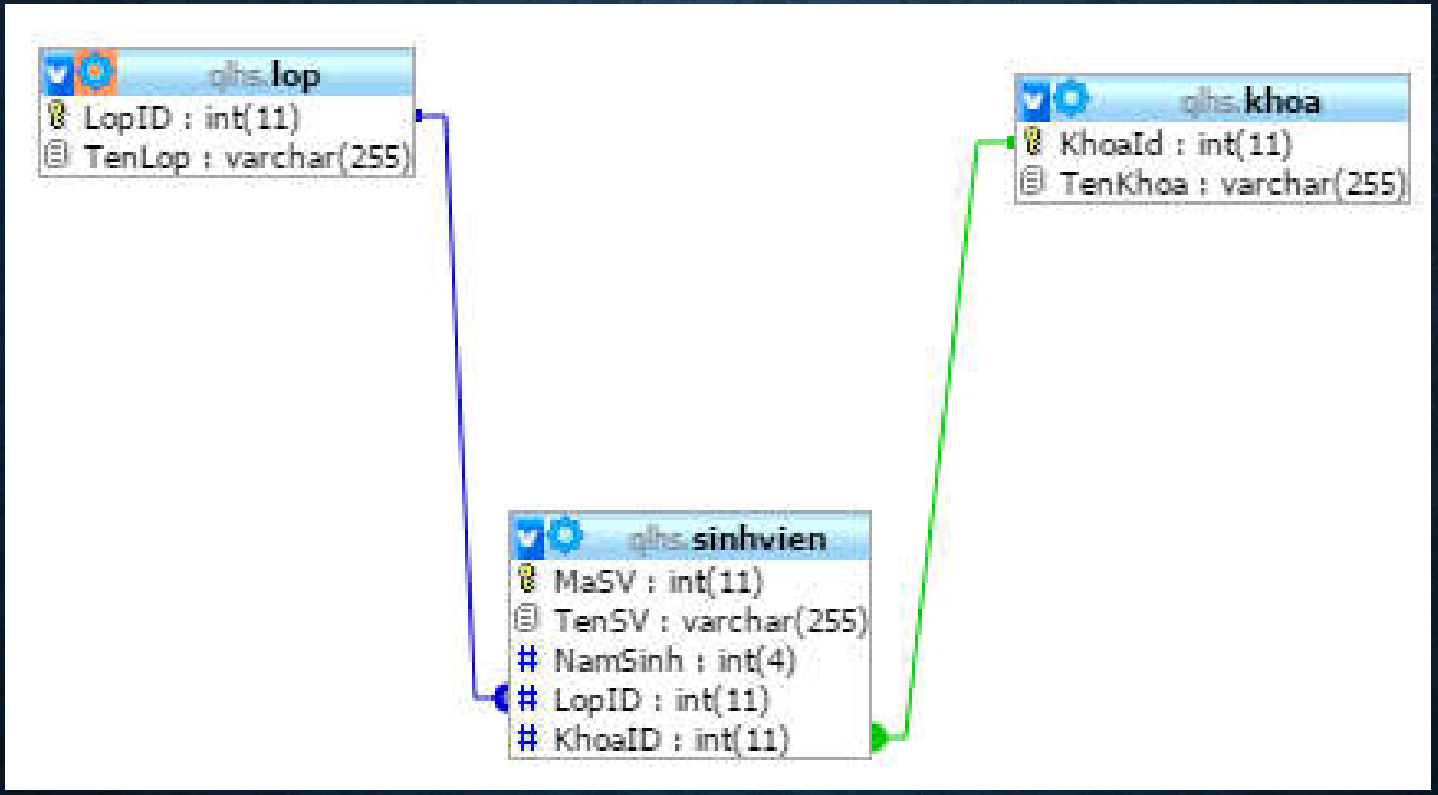
- Join Item và OrderDetail

```
SELECT t2.MaHang, t1.TenHang, t2.SoLuong
FROM Item t1 INNER JOIN OrderDetail t2 ON t1.MaHang = t2.MaHang
```

INNER JOIN MULTI TABLE

- Có thể JOIN nhiều bảng lại với nhau và tuân theo quy luật chạy từ trái qua phải, nếu bảng nào khai báo trước thì chạy trước và ngược lại sẽ chạy sau.
- Cú pháp:

```
SELECT column_list
FROM t1
INNER JOIN t2 ON join_condition1
INNER JOIN t3 ON join_condition2
```


qlhs.lop
LopID : int(11)
TenLop : varchar(255)
qlhs.khoa
KhoaId : int(11)
TenKhoa : varchar(255)
qlhs.sinhvien
MaSV : int(11)
TenSV : varchar(255)
NamSinh : int(4)
LopID : int(11)
KhoaID : int(11)

- **Ví dụ 1**: Hãy liệt kê danh sách sinh viên, thông tin lớp và khoa mà sinh viên đó đang học.

```
SELECT *
FROM sinhvien
INNER JOIN lop ON sinhvien.LopID = lop.LopID
INNER JOIN khoa ON sinhvien.KhoaID = khoa.KhoaId
```

- Ví dụ 2: Liệt kê danh sách sinh viên học lớp TOAN gồm các thông tin (thông tin sinh viên + thông tin khoa mà sinh viên đang học)

```
SELECT *
FROM sinhvien
INNER JOIN lop ON sinhvien.LopID = lop.LopID
INNER JOIN khoa ON sinhvien.KhoaID = khoa.KhoaId
WHERE lop.TenLop = 'TOAN'
```

LỖI AMBIGUOUS KHI THỰC HIỆN INNER JOIN

- Câu truy vấn này chạy sẽ bị lỗi vì lý do ở SELECT nó không hiểu LopID từ bảng nào.

```
SELECT TenSV, TenLop, LopID
FROM sinhvien INNER JOIN lop ON sinhvien.LopID = lop.LopID
```

- Câu truy vấn đúng:

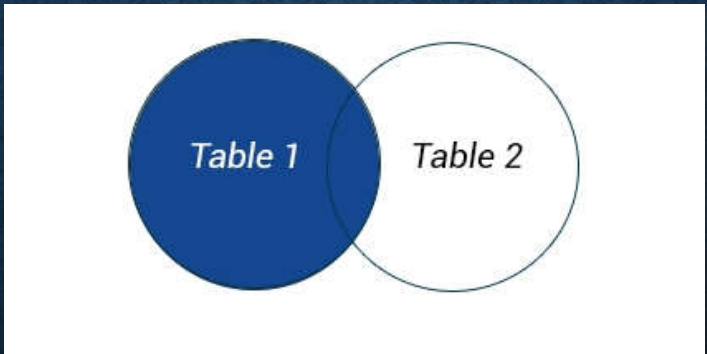
```
SELECT TenSV, TenLop, lop.LopID
FROM sinhvien INNER JOIN lop ON sinhvien.LopID = lop.LopID
```

INNER JOIN VỚI ALIAS

```
SELECT TenSV, TenLop, l.LopID
FROM sinhvien AS sv INNER JOIN lop AS l ON sv.LopID = l.LopID
```

LEFT JOIN

- LEFT JOIN trả về tất cả bản ghi bảng bên trái kể cả bản ghi đó không tương ứng với bảng bên phải, còn bảng bên phải những bản ghi nào phù hợp với bảng trái thì dữ liệu bản ghi đó được dùng để kết hợp với bản ghi bảng trái, nếu không có dữ liệu sẽ NULL



```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

CustomerName	OrderID
Alfreds Futterkiste	*null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365

Kết quả thấy mọi **CustomerName** ở bảng bên trái được lấy ra, sau đó **OrderID** ở bảng bên phải phù hợp được dùng để kết hợp với **CustomerName** nếu không có giá trị nào phù hợp thì nhận giá trị **null.**

VÍ DỤ

```
SELECT t1.MaHang, t1.TenHang, t2.SoLuong
FROM Item t1 LEFT JOIN OrderDetail t2 ON t1.MaHang = t2.MaHang
```

RIGHT JOIN

- Trái ngược với LEFT JOIN, RIGHT JOIN lấy toàn bộ dữ liệu của bảng bên phải và dữ liệu của bảng bên trái nhưng giá trị cột JOIN cũng tồn tại trong bảng bên phải.
- Ví dụ 1:

```
SELECT t2.MaHang, t2.TenHang, t1.SoLuong
FROM OrderDetail t1 RIGHT JOIN Item t2 ON t1.MaHang = t2.MaHang
```

- Ví dụ 2:

```
SELECT t1.MaHang, t1.TenHang, t2.SoLuong
FROM Item t1 RIGHT JOIN OrderDetail t2 ON t1.MaHang = t2.MaHang
```

FULL JOIN

- FULL OUTER JOIN trong SQL Server chính là kết quả gộp lại của cả hai table bên trái và bên phải.
- Ví dụ:

```
SELECT t1.MaHang, t1.TenHang, t2.SoLuong
FROM Item t1 FULL JOIN OrderDetail t2 ON t1.MaHang = t2.MaHang
```

4. VIEW

- **View là bảng ảo** giúp giới hạn truy cập một số cột dòng trên các bảng dữ liệu.
- Thuận lợi khi sử dụng view:
  - Cung cấp dữ liệu cần thiết cho người dùng
  - Che dấu đi sự phức tạp của dữ liệu
  - Tổ chức dữ liệu từ nhiều tài nguyên không đồng nhất
  - Giảm kích cỡ của đối tượng

- Hạn chế của view:
  - Không sử dụng biến local, user hay session.
  - Bảng tạm hay các bảng ảo khác cũng không được sử dụng làm nguồn dữ liệu cho câu truy vấn này (không thể sử dụng một view làm bảng nguồn cho một view khác).
  - Không thể tạo trigger cho bảng ảo.

- Cú pháp:

```
CREATE VIEW ViewName
AS
//Query
```

- Ví dụ:

```
CREATE VIEW SalePerOrder
  AS
  SELECT orderNumber,
  SUM  (quantityOrdered * priceEach) total
  FROM orderDetails
  GROUP by orderNumber
  ORDER BY total DESC
```

5.1. INDEX

- Tại sao nên tạo index?
- Khi nào thì cần sử dụng index?
- Tạo index như thế nào?

- Tạo index để khi truy vấn dữ liệu theo index sẽ làm tăng hiệu năng hoạt động của database, do dữ liệu được đánh index sẽ tìm kiếm theo chỉ số index mà nó được đánh. Giống như một danh sách không sắp xếp và một danh sách có số thứ tự.
- Cú pháp:

```
CREATE INDEX index_name ON table_name (column_1, column_2,…);
```

- Ví dụ:

```
CREATE INDEX part_of_name ON customer (name(10));
```

BẢNG NÀO DÙNG INDEX

- Khi các lệnh truy vấn nhỏ hơn 5 phần trăm số rows trong bảng
- Không đánh index cho bảng thường xuyên thực hiện update
- Nên chọn các bảng không có các rows có giá trị trùng nhau trên các cột thường được đặt điều kiện trong mệnh đề WHERE
- Đánh index cho các bảng có câu lệnh truy vấn với các mệnh đề WHERE đơn.
- Số cột đánh index không nên vượt quá 3 cột, trừ trường hợp đặc biệt.

CỘT NÀO DÙNG INDEX

- Chọn những cột thường được xuất hiện trong mệnh đề where.
- Không đánh index cho những cột mà không có nhiều giá trị duy nhất.
- Những cột được đánh index tự động là cột có ràng buộc khó PK và UK
- Đánh index cho những cột mà thường được sử dụng để join bảng
- Không nên đánh index cho những cột thường xuyên được update, vì sẽ làm giảm hiệu năng do phải update index.
- Sử dụng composite Indexes
- Sử dụng hints nếu như muốn hoặc không muốn sử dụng index.

5.2. PARTITION

- Partition table là 1 loại table đặc biệt, có tên bảng như bảng thông thường, nhưng cách tổ chức dữ liệu theo chủ ý của người thiết kế ứng dụng. Dữ liệu có thể được đặt ở các partition theo một rule nào đó để tăng tốc độ truy vấn dữ liệu và được lọc theo một trường dữ liệu trong bảng, giống như việc phân loại dữ liệu để dễ tìm kiếm và quản lý.

- Cú pháp:

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (1995),
        PARTITION p2 VALUES LESS THAN (2000),
    PARTITION p3 VALUES LESS THAN (2005)
);
```

ƯU ĐIỂM PARTITION

- Giảm lượng data bị lỗi do đặt multiple partitions, nếu lỗi chỉ lỗi riêng partition đó.
- Back up và recover riêng từng partition
- Quản lý partitions với riêng từng disk drivers (cân bằng tải I/O)
- Dễ dàng quản lý dữ liệu, và tăng hiệu năng.

PHÂN LOẠI PARTITION

- **Range partition**:
  - Là loại partition có dạng chia theo khoảng thời gian, hoặc khoảng dữ liệu liên tiếp như dãy số…. Đa số sử dụng cho các bảng dữ liệu có dữ liệu truy cập chỉ theo một khoảng thời gian nhất định theo một chu kỳ, số lượng bản ghi rất nhiều.
- **List partition**:
  - Là loại partiton có dạng chia theo một trường dữ liệu có tính chất phân danh sách. Ví dụ vùng miền các tỉnh thành phố trong nước, hoặc một danh sách các lớp học trong một trường, danh sách các phòng ban trong một công ty…

PHÂN LOẠI PARTITION (2)

- **Hash partition**:
  - Là loại partition có dạng chia theo hàm băm. Loại này sử dụng khi dữ liệu quá lớn, không có quy luật nào ngoài list và range nên sẽ được chia theo một trường dữ liệu bất kỳ, thường là khóa chính và là trường số. Dữ liệu được băm ra các partition và khi thực hiện truy vấn Oracle tự quản lý các partition này.
- **Subpartition**: là chia partition con của 3 loại partition trên.

- THANKS FOR ATTENDING!