



INDEX TRONG DATABASE

Index database là gì?

- ❖ Index là một cấu trúc dữ liệu được dùng để định vị và truy cập nhanh nhất vào dữ liệu trong các bảng database.
- ❖ Index là một cách tối ưu hiệu suất truy vấn database bằng việc giảm lượng truy cập vào bộ nhớ khi thực hiện truy vấn.

Ví dụ

	Customers
<input type="checkbox"/>	* (All Columns)
<input type="checkbox"/>	CustomerID
<input type="checkbox"/>	CompanyName
<input type="checkbox"/>	ContactName
<input type="checkbox"/>	ContactTitle
<input type="checkbox"/>	Address
<input type="checkbox"/>	City
<input type="checkbox"/>	Region
<input type="checkbox"/>	PostalCode
<input type="checkbox"/>	Country
<input type="checkbox"/>	Phone
<input type="checkbox"/>	Fax

```
SELECT * FROM Customers  
WHERE City LIKE 'London';
```

Khi không có Index cho cột City, truy vấn sẽ phải chạy qua tất cả các Row của bảng Customers để so sánh và lấy ra những Row thỏa mãn.

Khi số lượng bản ghi lớn, truy vấn sẽ rất chậm.

Index database là gì?

- ❖ Nói đơn giản, index trả tới địa chỉ dữ liệu trong một bảng, giống như Mục lục của một cuốn sách (Gồm tên đề mục và số trang), nó giúp truy vấn trở nên nhanh chóng như việc bạn xem mục lục và tìm đúng trang cần đọc.

I. Mục đích, yêu cầu	2
1. Mục đích	2
2. Yêu cầu	2
II. Các nội dung phải hoàn thành	3
III. Tổ chức thực hiện	3
1. Thời gian tổ chức thực tập	3
2. Địa điểm thực tập tốt nghiệp	4
3. Tiến trình thực tập tốt nghiệp.	4
3.1. Bước 1: Địa điểm thực tập	4
3.2. Bước 2: Sinh viên đến làm việc thực tế tại đơn vị thực tập	4

Sự khác biệt giữa truy vấn số và chuỗi ký tự

String -

```
SELECT TOP 100 * FROM tblScratch WHERE contactsurname = 'hoare' ORDER BY NEWID()
```

Duration 430ms

Reads 902

CPU 203

Integer -

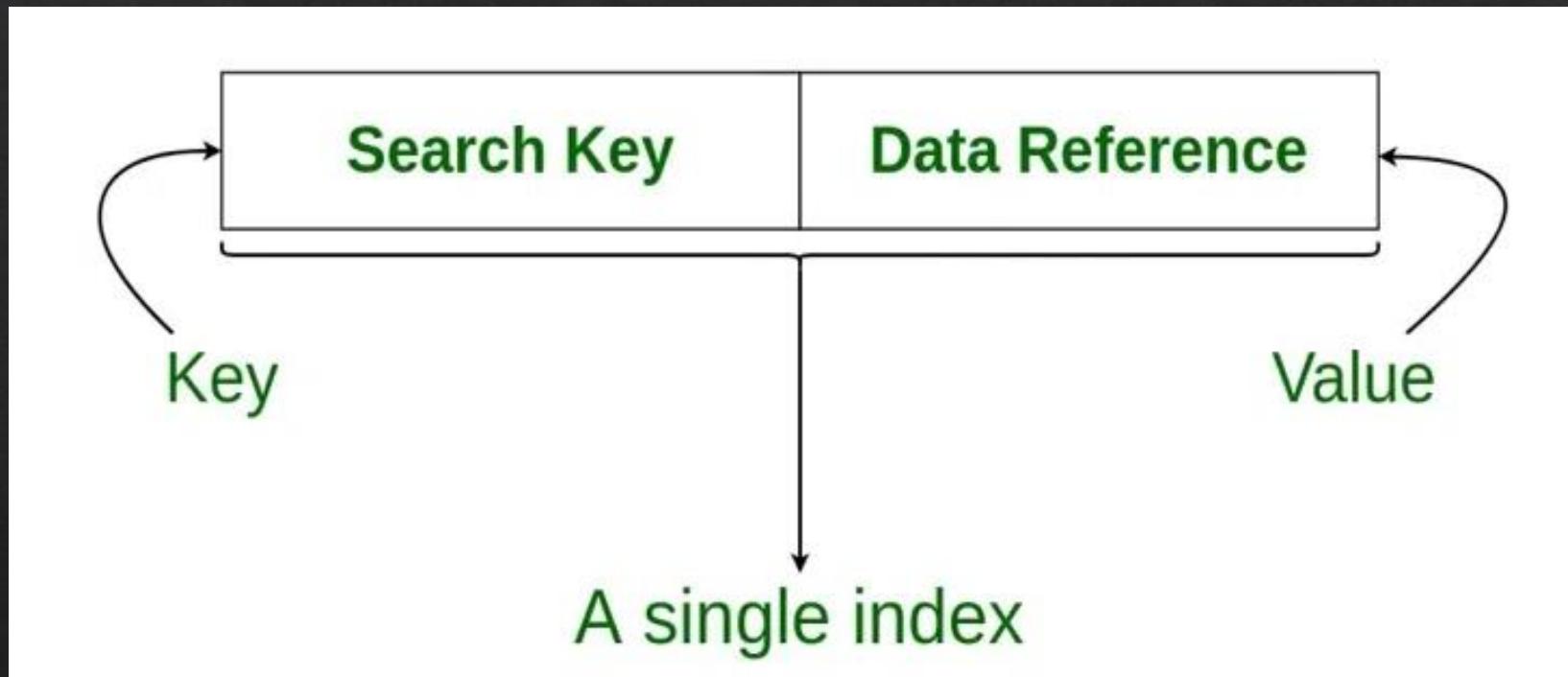
```
SELECT TOP 100 * FROM tblScratch WHERE contacttyperef = 3 ORDER BY NEWID()
```

Duration 136ms

Reads 902

CPU 79

Cấu trúc của index

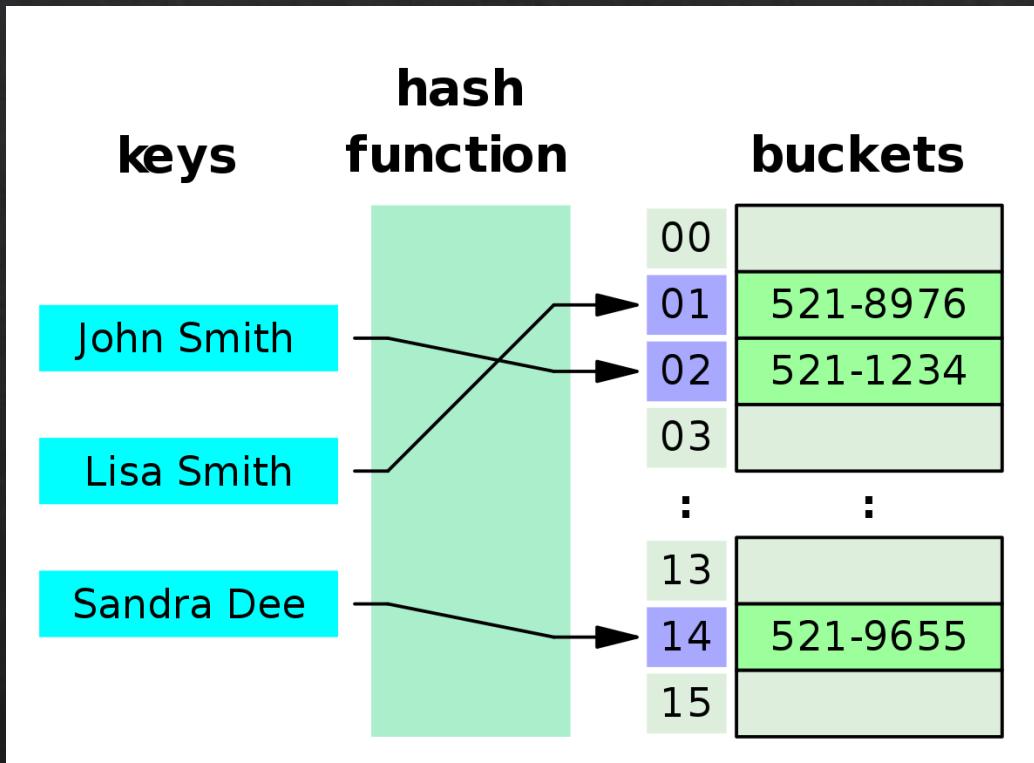


<https://www.geeksforgeeks.org/indexing-in-databases-set-1/>

Các kiểu index

❖ Hash index

- ❖ Được tổ chức dưới dạng key-value với key là kết quả hash value của column được đánh index và value sẽ chứa 1 con trỏ đến chính xác row tương ứng



<https://www.sqlpipe.com/blog/b-tree-vs-hash-index-and-when-to-use-them>

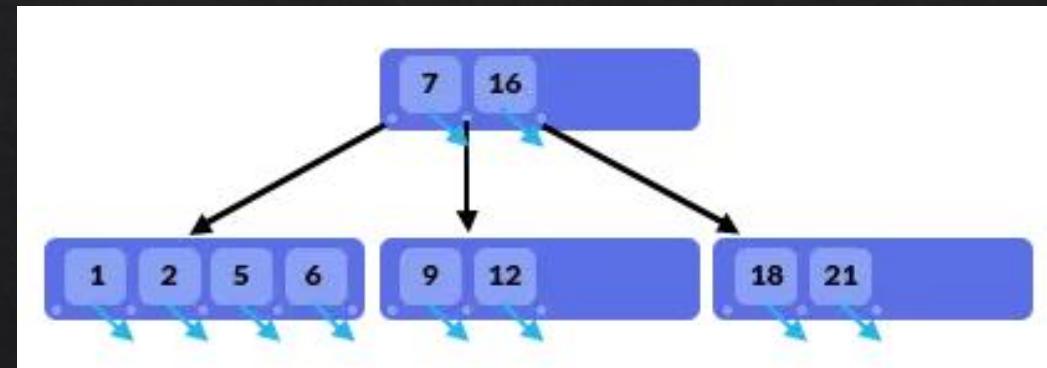
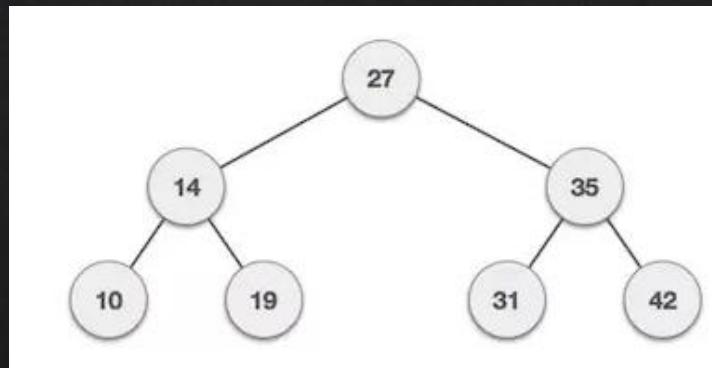
Các kiểu index

- ❖ Hash index
 - ❖ Hash index mạnh mẽ khi thực hiện các phép truy vấn với toán tử = hay <> (IN, NOT IN) (có thể nhanh hơn cả B-tree)
 - ❖ Tuy nhiên lại không hữu ích khi gặp các trường truy vấn với điều kiện như >, <, like
 - ❖ Không thể tối ưu hóa toán tử ORDER BY bằng việc sử dụng Hash index bởi vì nó không thể tìm kiếm được phần tử tiếp theo trong Order

Các kiểu index

❖ B-Tree

- ❖ Dữ liệu index trong B-Tree được tổ chức và lưu trữ theo dạng cây (tree):
 - ❖ Root node – node đầu tiên đứng vị trí cao nhất trong cây
 - ❖ Child nodes – nodes con được trả từ Parent nodes (vị trí cao hơn)
 - ❖ Parent nodes – nodes cha trong cây mà có trả sang các Child nodes
 - ❖ Leaf nodes – nodes lá, không trả đến bất kì nodes nào khác, có vị trí thấp nhất trong nhánh của cây.
 - ❖ Internal nodes – tất cả các nodes không phải là nodes lá
 - ❖ External nodes – tên gọi khác của nodes lá.
- ❖ B-Tree index được sử dụng trong các biểu thức so sánh dạng: =, >, >=, <, <=, BETWEEN và LIKE
- ❖ B-Tree index được sử dụng cho những column trong bảng khi muốn tìm kiếm 1 giá trị nằm trong khoảng nào đó

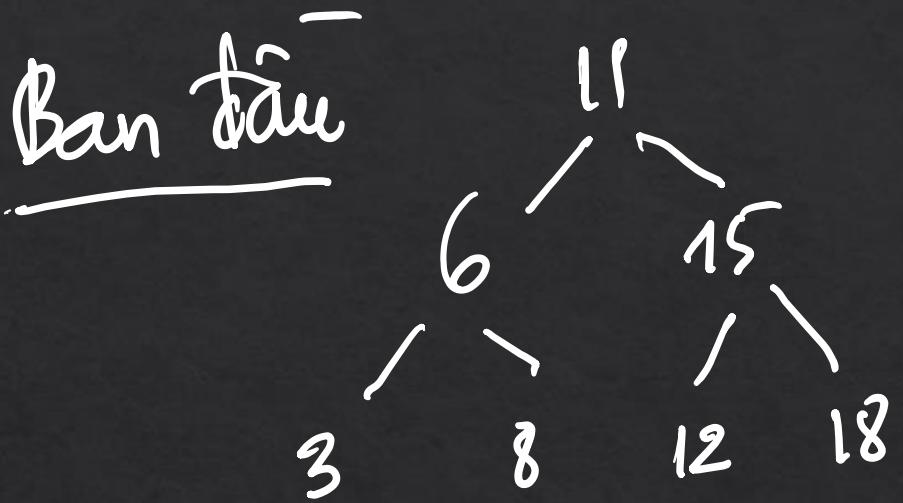


Các kiểu index

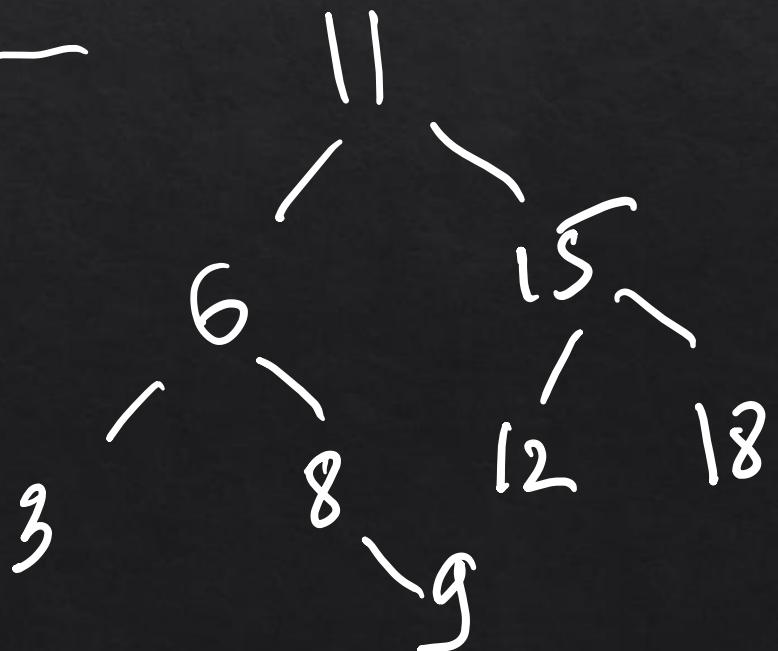
- ❖ **B-Tree (tt)**
 - ❖ Cơ chế đánh index (cho 1 trường):
 - ❖ Bước 1: sắp xếp các value của cột theo thứ tự từ a-z
 - ❖ Bước 2: Gán index tương ứng từ 0, 1, 2, ... tăng dần.
 - ❖ Bước 3: Điền vào cây nhị phân

Các kiểu index

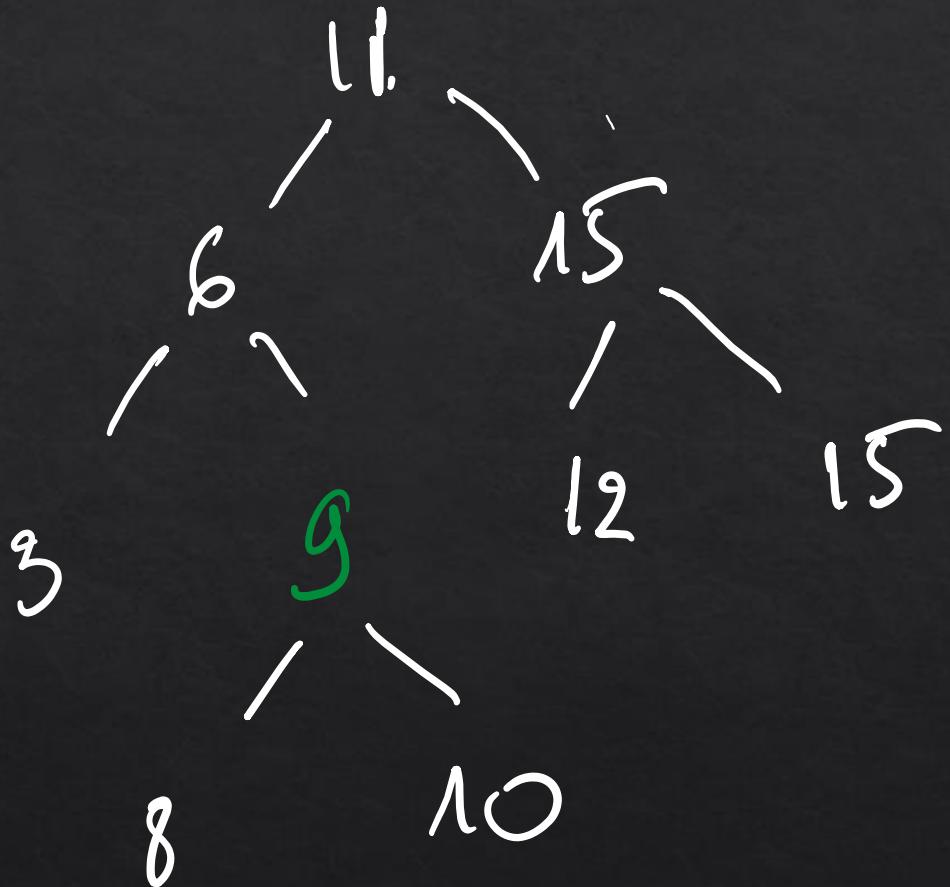
- ❖ B-Tree (tt)
 - ❖ Cơ chế đánh index (cho 1 trường):
 - ❖ Tìm vị trí thích hợp để chèn dữ liệu mới
 - ❖ Chèn dữ liệu mới vào lá phù hợp
 - ❖ Phân chia nút lá tràn (nếu cần)
 - ❖ Lặp lại quá trình nếu cần thiết



Chèn 9:



Chèn nút lá trán: 10



Các kiểu index

- ❖ B-Tree (tt)
 - ❖ Cơ chế đánh index (cho nhiều trường):
 - ❖ Tương tự nhưng sẽ đánh index cho từng trường theo thứ tự

Tóm tắt các loại đánh index

❖ Hash Index

- ❖ Hash Index sử dụng hàm băm để ánh xạ các giá trị cột thành các khóa. Các khóa này sau đó được lưu trữ trong một bảng băm, nơi chúng có thể được tìm thấy nhanh chóng bằng cách sử dụng hàm băm ngược.
- ❖ Hash Index có hiệu suất cao cho các truy vấn khớp chính xác, chẳng hạn như `SELECT * FROM table WHERE column = value`. Tuy nhiên, chúng không hiệu quả cho các truy vấn khớp khoảng, chẳng hạn như `SELECT * FROM table WHERE column > value`.

❖ B-tree Index

- ❖ B-tree Index sử dụng một cấu trúc cây để lưu trữ các giá trị cột. Cây này được tổ chức theo thứ tự tăng dần, giúp các giá trị có thể được tìm thấy nhanh chóng bằng cách duyệt qua cây.
- ❖ B-tree Index có hiệu suất cao cho cả truy vấn khớp chính xác và khớp khoảng. Chúng thường được coi là loại index hiệu quả nhất.

Cú pháp

```
CREATE INDEX index_name  
ON table_name (column1, column2, ...);
```

```
CREATE UNIQUE INDEX index_name  
ON table_name (column1, column2, ...);
```

↳ giá trị trong cột này
không được trùng lặp

Ví dụ

- ❖ Đánh index cho cột "CustomerName" trong bảng "Customers"

```
CREATE INDEX idx_CustomerName ON Customers (CustomerName);
```

- ❖ Đánh index cho cột "ProductName" trong bảng "Products"

```
CREATE INDEX idx_ProductName ON Products (ProductName);
```

Ví dụ

- ❖ Tạo composite index cho cột "CustomerID" và "OrderDate" trong bảng "Orders"

```
CREATE INDEX idx_CustomerID_OrderDate ON Orders (CustomerID, OrderDate);
```

Lưu ý về việc dùng Index trong Database

- ❖ Cẩn trọng:
 - ❖ Index sẽ làm tốn thêm bộ nhớ để lưu trữ.
 - ❖ Làm chậm các hoạt động khác, khi insert hay update column sử dụng index, index cần được điều chỉnh (reindex) sẽ tiêu tốn một khoảng thời gian.
 - ❖ Việc đánh index bừa bãi, lộn xộn, không những không tăng tốc độ truy vấn mà còn làm giảm hiệu năng hoạt động.

Lưu ý về việc dùng Index trong Database

- ❖ Các trường hợp Index sẽ được tạo tự động
 - ❖ Khóa chính
 - ❖ Khóa ngoại
 - ❖ Các cột UNIQUE

Lưu ý về việc dùng Index trong Database

- ❖ Các trường hợp nên sử dụng index:
 - ❖ Những bảng có dữ liệu vừa và lớn (>100 nghìn dòng)
 - ❖ Các column thường xuyên sử dụng trong mệnh đề WHERE, JOIN và ORDER BY
- ❖ Các trường hợp không nên sử dụng index:
 - ❖ Cơ sở dữ liệu nhỏ hoặc cần sử dụng tài nguyên ít
 - ❖ Dữ liệu thay đổi thường xuyên
 - ❖ Cột chứa dữ liệu không đa dạng
 - ❖ Cột chứa dữ liệu text quá dài (ví dụ như description)
- ❖ Cần phải lưu ý về thứ tự columns trong một index nhiều trường

Thực hành 1

Bước 1: Hãy restore lại cơ sở dữ liệu theo hướng dẫn bên dưới

<https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms>

Bước 2: Thử truy cập câu truy vấn trước khi đánh index

```
1 -- Bật hiển thị thống kê về tài nguyên I/O
2 SET STATISTICS IO ON;
3
4 -- Viết truy vấn của bạn ở đây
5 SELECT *
6 FROM [Sales].[SalesOrderDetail]
7 WHERE [CarrierTrackingNumber] = '1B2B-492F-A9';
8
9 -- Tắt hiển thị thống kê
10 SET STATISTICS IO OFF;
```

(22 rows affected)

Table 'SalesOrderDetail'. Scan count 1, logical reads 1238, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0

Thực hành

Bước 3: Chạy lại câu lệnh

```
1 -- Bật hiển thị thống kê về tài nguyên I/O
2 SET STATISTICS IO ON;
3
4 -- Viết truy vấn của bạn ở đây
5 SELECT *
6 FROM [Sales].[SalesOrderDetail]
7 WHERE [CarrierTrackingNumber]= '1B2B-492F-A9';
8
9 -- Tắt hiển thị thống kê
10 SET STATISTICS IO OFF;
```

(22 rows affected)

Table 'SalesOrderDetail'. Scan count 1, logical reads 69, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0

Thực hành 2

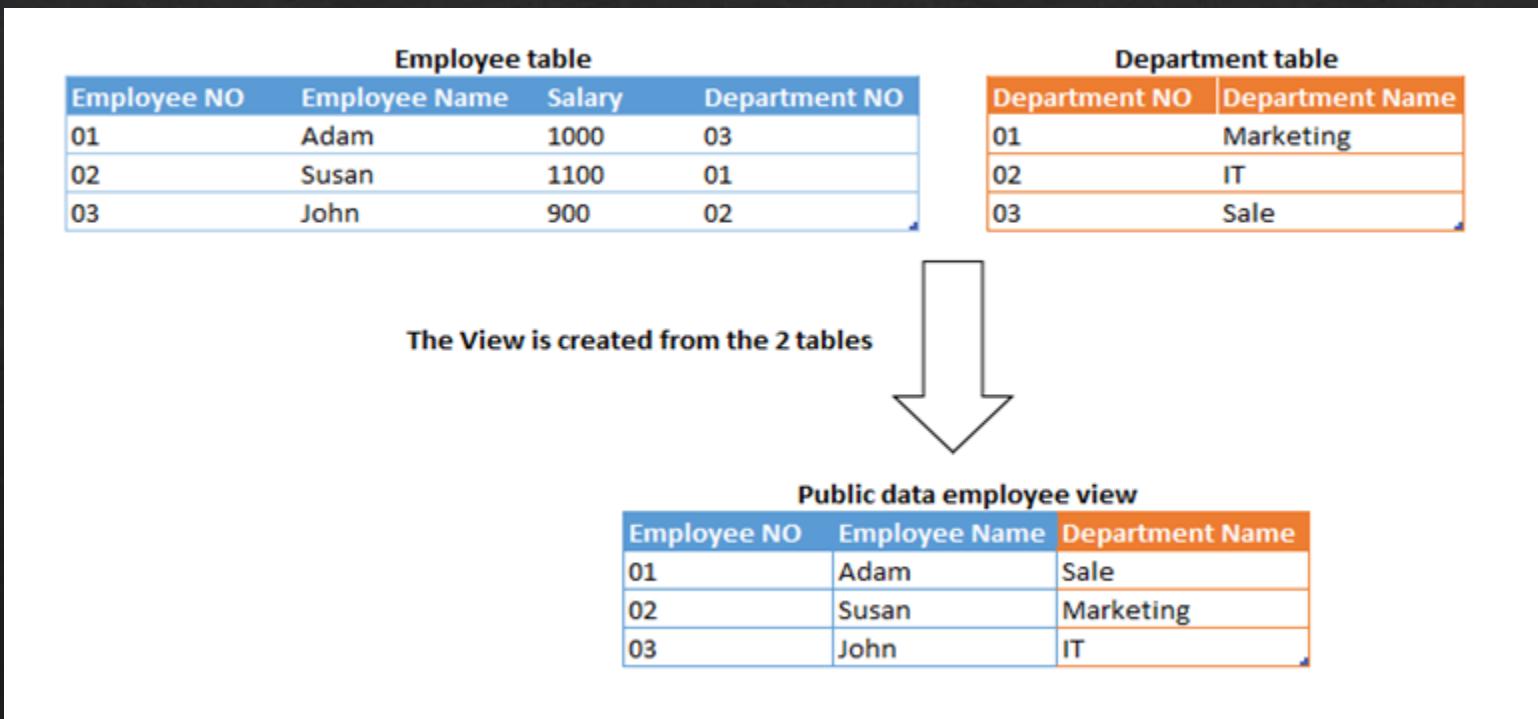
- ❖ Tạo index trên bảng Person.Address, cột AddressLine1 và đánh giá hiệu suất truy vấn.
- ❖ Tạo index cho cột "ProductName" trong bảng "Production.Product" và đánh giá hiệu suất truy vấn.



VIEW TRONG DATABASE

View là gì?

- ❖ Database View là sự trình bày data theo ý muốn được trích xuất từ một hoặc nhiều table/view khác. View không lưu data nên nó còn được biết đến với cái tên "bảng ảo(virtual tables)".



View là gì?

- ❖ Các thao tác select, insert, update và delete với view tương tự như table bình thường.
- ❖ Vì không lưu data nên tất cả những thao tác được thực hiện trên view thì đều được phản ánh đến base table mà được trích xuất dữ liệu.

Cú pháp

```
-- Syntax for SQL Server and Azure SQL Database
```

```
CREATE [ OR ALTER ] VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement
[ WITH CHECK OPTION ]
[ ; ]
```

```
<view_attribute> ::=
{
    [ ENCRYPTION ]
    [ SCHEMABINDING ]
    [ VIEW_METADATA ]
}
```

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Ví dụ 1

- ❖ Tạo View có tên "MonthlySales" để hiển thị tổng doanh số bán hàng hàng tháng từ bảng "Orders."

```
CREATE VIEW MonthlySales AS  
SELECT YEAR(OrderDate) AS Year, MONTH(OrderDate) AS  
Month, SUM(Total) AS SalesTotal  
FROM Orders  
GROUP BY YEAR(OrderDate), MONTH(OrderDate);
```

Ví dụ 2

- ❖ Tạo View kết hợp thông tin về khách hàng và đơn hàng

```
CREATE VIEW CustomerOrders AS  
SELECT  
    C.CustomerID,  
    C.CompanyName,  
    C.ContactName,  
    O.OrderID,  
    O.OrderDate,  
    O.ShipCountry  
FROM Customers C  
JOIN Orders O ON C.CustomerID = O.CustomerID;
```

Ví dụ 3

- ❖ Tạo View hiển thị tổng giá trị của từng đơn hàng

```
CREATE VIEW OrderTotalValue AS  
SELECT  
    O.OrderID, O.CustomerID,  
    O.OrderDate,  
    SUM(OD.Quantity * (OD.UnitPrice - (OD.UnitPrice *  
        OD.Discount))) AS TotalOrderValue  
FROM Orders O JOIN [Order Details] OD ON O.OrderID = OD.OrderID  
GROUP BY O.OrderID, O.CustomerID, O.OrderDate;
```

Check option

- ❖ CHECK OPTION trong View là một điều kiện cho phép bạn xác định ràng buộc về việc cập nhật hoặc chèn dữ liệu vào View. Nó đảm bảo rằng các dòng dữ liệu được chèn hoặc cập nhật thông qua View sẽ tuân theo một điều kiện cụ thể.

Ví dụ 1

- ❖ Sử dụng CHECK OPTION để chỉ cho phép chèn dữ liệu thỏa mãn điều kiện. Giả sử bạn có một View có tên "HighValueProducts" để hiển thị sản phẩm có giá trị cao hơn \$500.

```
CREATE VIEW HighValueProducts AS  
SELECT ProductID, ProductName, UnitPrice  
FROM Products WHERE UnitPrice > 500  
WITH CHECK OPTION;
```

Updatable view

- ❖ Một View có thể được cấu hình để cho phép cập nhật nếu nó được tạo dựa trên một bảng cơ sở dữ liệu và tuân theo một số yêu cầu. Điều này bao gồm:
 - ❖ View phải được tạo bằng cách sử dụng SELECT, không được chứa các phép toán SET, UNION, DISTINCT, hoặc GROUP BY phức tạp.
 - ❖ View không được chứa các cột tính toán (computed columns).
 - ❖ View phải có đủ các trường cần thiết để cập nhật (ví dụ: PRIMARY KEY).

Read only view

- ❖ Cách chặn UPDATE/DELETE/INSERT với view:
 - ❖ Cách 1: Phân quyền USER
 - ❖ Cách 2: Làm cho việc update data qua view luôn vi phạm 1 trong những điều kiện

Ví dụ 1

```
CREATE VIEW employee_view_read_only
AS SELECT E.EmployeeNo, E.EmployeeName, E.DepartmentNo
FROM Employee E
UNION ALL
SELECT NULL, NULL, NULL
```

Ưu điểm và nhược điểm của View

- ❖ **Ưu điểm:**
 - ❖ **View giúp đơn giản hóa những query phức tạp:** Trong trường hợp cần truy xuất dữ liệu từ nhiều table với logic phức tạp. Lúc này ta có thể tạo view với logic tương tự và thông qua view chúng ta sẽ chỉ cần sử dụng những câu query đơn giản để truy xuất dữ liệu.
 - ❖ **Giới hạn data có thể truy cập với nhóm người dùng được chỉ định:** Đôi khi ta không muốn những dữ liệu nhạy cảm được truy cập bởi nhóm user nào đó. View có thể giúp ta giới hạn row/column của những table theo điều kiện ta muốn lấy.
 - ❖ **View cung cấp thêm 1 lớp bảo mật cho database:** Có những dữ liệu mà một nhóm người dùng có thể truy cập nhưng lại không muốn họ có thể thao tác thay đổi. Option VIEW READ ONLY sẽ giải quyết được vấn đề này.
 - ❖ **Cung cấp khả năng tương thích khi thay đổi cấu trúc dữ liệu**

Ưu điểm và nhược điểm của View

- ❖ Nhược điểm
 - ❖ **Performance:** Vì bản chất không lưu trữ dữ liệu nên tất cả những thao tác được thực hiện trên view thì đều được phản ánh đến base table. Do đó query processor phải translate view thành query đến base table nên hiệu năng truy xuất dữ liệu có thể giảm nếu view phức tạp được tạo từ nhiều table hoặc view được tạo từ view khác
 - ❖ **Phụ thuộc vào base table:** Mỗi khi base table có sự thay đổi cấu trúc điều đó có thể khiến view trở thành invalid

Thực hành về view

1. Tạo một View có tên "HighValueProducts" để hiển thị danh sách các sản phẩm có giá trị cao hơn \$50.
2. Tạo một View có tên "CustomerOrders" để hiển thị thông tin về khách hàng và số lượng hàng của họ.
3. Tạo một View có tên "EmployeeSalesByYear" để hiển thị tổng doanh số bán hàng của từng nhân viên theo năm.
4. Tạo một View có tên "CategoryProductCounts" để hiển thị số lượng sản phẩm trong mỗi danh mục sản phẩm.
5. Tạo một View có tên "CustomerOrderSummary" để hiển thị tổng giá trị đặt hàng của mỗi khách hàng