

## --Index trong SQL Server 2012

Index (chỉ mục) trong SQL Server là các cấu trúc dữ liệu đặc biệt được liên kết với các bảng hoặc [view](#) giúp tăng tốc truy vấn. SQL Server cung cấp hai loại index: clustered index và non-clustered index.

### Clustered Index trong SQL Server

1. Giả sử ta tạo 1 bảng dữ liệu từ bảng sau:

--Tạo bảng khách hàng dự vào bảng khách hàng

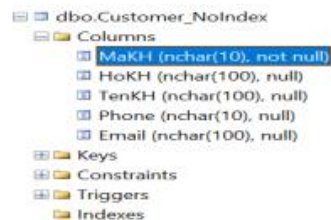
```
SELECT *
```

```
INTO Customer_NoIndex
```

```
FROM KháchHang
```

```
insert Customer_NoIndex(MaKH,HoKH,TenKH,Phone,Email) values('003',N'Võ  
Huỳnh',N'Hàng','0978888888','hang@gmail.com');
```

```
insert Customer_NoIndex(MaKH,HoKH,TenKH,Phone,Email) values('004',N'Võ  
Thị',N'Nga','0978886689','nga@gmail.com');
```



Bảng Customer\_NoIndex không có khóa chính. Do đó SQL server sẽ lưu các bản ghi trong 1 cấu trúc có thứ tự được gọi là Heap(Đống)

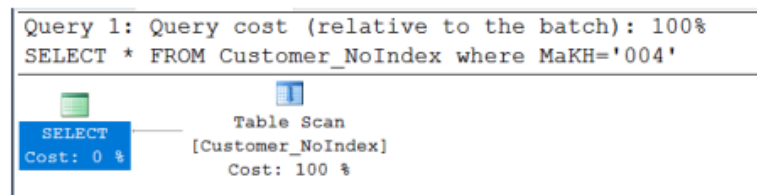
Khi chúng ta truy vấn từ bảng trình tối ưu hóa truy vấn sẽ quét toàn bộ bảng để xác định vị trí chính xác

--Tìm bản ghi có mã khách hàng 004

```
SELECT *
```

```
FROM Customer_NoIndex
```

```
where MaKH='004'
```



Xem ước lượng trong SQL server 2012 như sau:

*Lưu ý: để xem ước lượng kế hoạch thực hiện trong SQL Server Management Studio, bạn bấm vào nút **Display Estimated Execution Plan** hoặc chọn truy vấn và nhấn phím tắt **Ctrl+L**:*



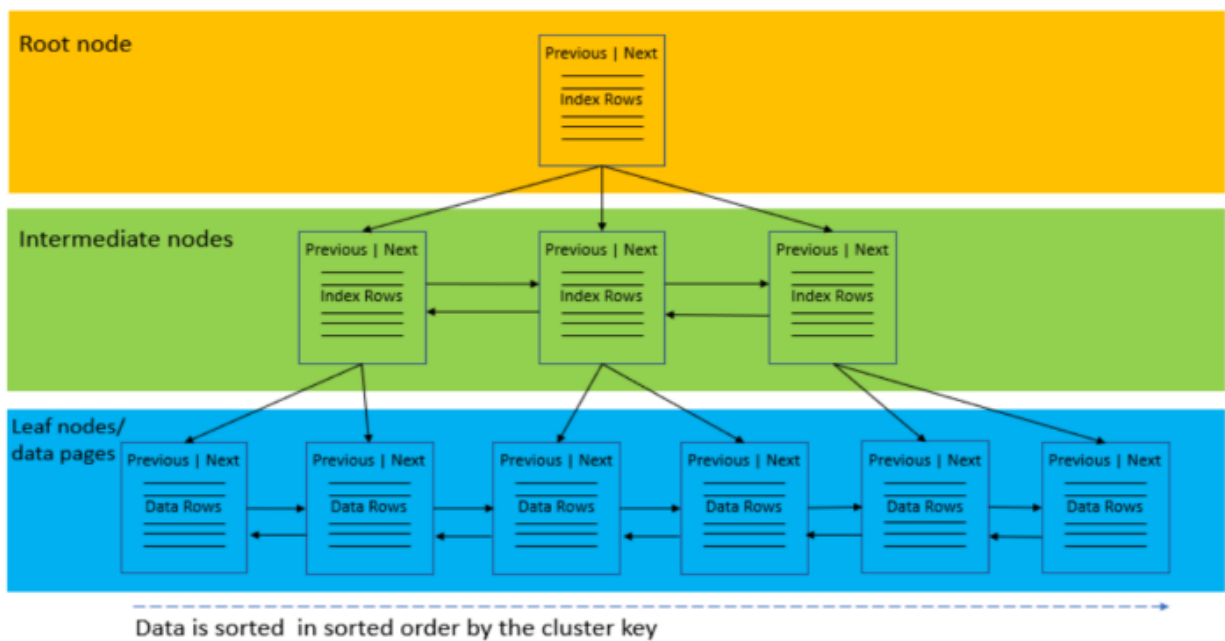
Truy vấn với số lượng lớn thì rất chậm.

Để giải quyết vấn đề này, SQL Server cung cấp một cấu trúc chuyên dụng để tăng tốc độ truy xuất các bản ghi từ một bảng được gọi là index.

SQL Server có hai loại index là clustered index và non-clustered index.

Một clustered index lưu trữ các bản ghi dữ liệu trong một cấu trúc được sắp xếp dựa trên các giá trị khóa của nó. Mỗi bảng chỉ có một clustered index vì các bản ghi dữ liệu chỉ có thể được sắp xếp theo một thứ tự. Bảng có clustered index được gọi là clustered table.

Hình ảnh sau đây minh họa cấu trúc của một clustered index:



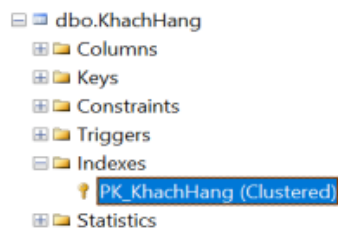
Một clustered index tổ chức dữ liệu bằng cách sử dụng một cấu trúc đặc biệt được gọi là B-tree (balanced tree - cây cân bằng) cho phép tìm kiếm, chèn, cập nhật và xóa bản ghi bất kỳ với thời gian như nhau.

Trong cấu trúc này, nút trên cùng của B-tree được gọi là **nút gốc** (root node). Các nút ở cấp độ dưới cùng được gọi là các **nút lá** (leaf nodes). Bất kỳ nút nào ở giữa các nút gốc và nút lá được gọi là nút trung gian.

Trong B-tree, nút gốc và nút trung gian chứa các trang chỉ mục để lưu trữ các chỉ mục của các bản ghi. Các nút lá chứa các trang dữ liệu (data pages) của bảng. Các trang trong mỗi cấp của index được liên kết bằng cấu trúc khác gọi là danh sách liên kết đôi.

## Clustered Index và khóa chính trong SQL Server

Khi bạn tạo bảng có khóa chính, SQL Server sẽ tự động tạo một clustered index tương ứng dựa trên các cột có trong khóa chính.



### Tạo Clustered Index trong SQL Server

Trong trường hợp một bảng không có khóa chính (điều này rất hiếm) bạn có thể sử dụng câu lệnh **CREATE CLUSTERED INDEX** để định nghĩa một clustered index cho bảng.

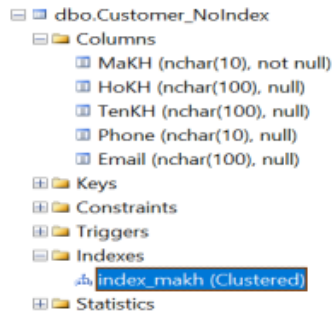
```
CREATE CLUSTERED INDEX index_name  
ON schema_name.table_name (column_list);
```

Trong cú pháp này:

- Đầu tiên, bạn sử dụng mệnh đề **CREATE CLUSTERED INDEX** để tạo clustered index.
- Thứ hai, chỉ định tên của clustered index sau mệnh đề **CREATE CLUSTERED INDEX**.
- Thứ ba, chỉ định lược đồ và tên bảng mà bạn muốn tạo index.
- Cuối cùng, liệt kê một hoặc nhiều cột có trong index.

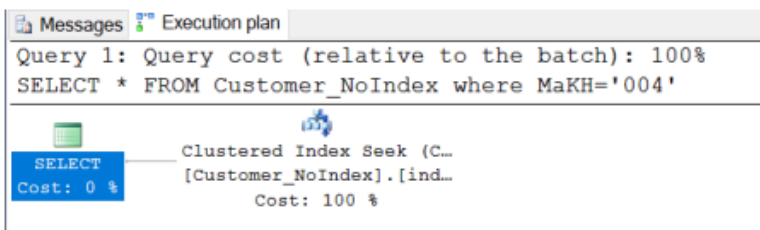
--Câu lệnh tạo index cluster cho bảng Customer\_NoIndex

```
CREATE CLUSTERED INDEX index_makh  
ON Customer_NoIndex (MakH);
```



--Thực hiện câu tìm khách hàng 004 với index clustered

```
SELECT *  
FROM Customer_NoIndex  
where MaKH='004'
```



## Non-clustered index trong SQL Server

Trong phần này, bạn sẽ tìm hiểu cách sử dụng câu lệnh SQL Server **CREATE INDEX** để tạo các non-clustered index cho các bảng.

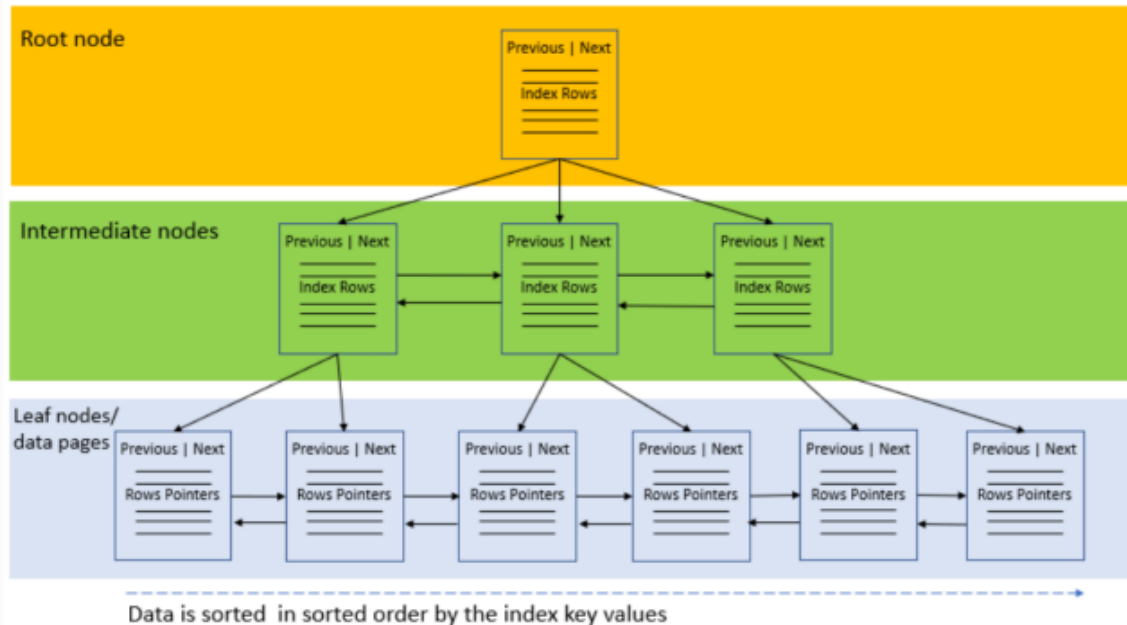
### Giới thiệu về non-clustered index trong SQL Server

Non-clustered index là một cấu trúc dữ liệu giúp cải thiện tốc độ truy xuất dữ liệu từ các bảng. Không giống như clustered index, non-clustered index sắp xếp và lưu trữ dữ liệu riêng biệt với các bản ghi trong bảng. Nó là một bản sao dữ liệu của các cột được chọn từ một bảng được liên kết.

Tương tự như clustered index, non-clustered index sử dụng cấu trúc cây B-Tree để tổ chức dữ liệu của nó.

Một bảng có thể có một hoặc nhiều non-clustered index và mỗi non-clustered index có thể bao gồm một hoặc nhiều cột của bảng.

Hình ảnh sau đây minh họa cấu trúc non-clustered index:



Bên cạnh việc lưu trữ các giá trị khóa index, các nút lá cũng lưu trữ các con trỏ trỏ tới các bản ghi có chứa các giá trị khóa. Những con trỏ bản ghi này còn được gọi là các định vị hàng (row locators).

Nếu bảng là một clustered table (bảng có clustered index), con trỏ bản ghi là khóa của clustered index. Trong trường hợp bảng không có clustered index, con trỏ bản ghi trỏ đến bản ghi của bảng.

### Tạo non-clustered index trong SQL Server

Để tạo một non-clustered index trong SQL Server, bạn sử dụng câu lệnh **CREATE INDEX** :

```
1 CREATE [NONCLUSTERED] INDEX index_name
2 ON table_name(column_list);
```

Trong cú pháp này:

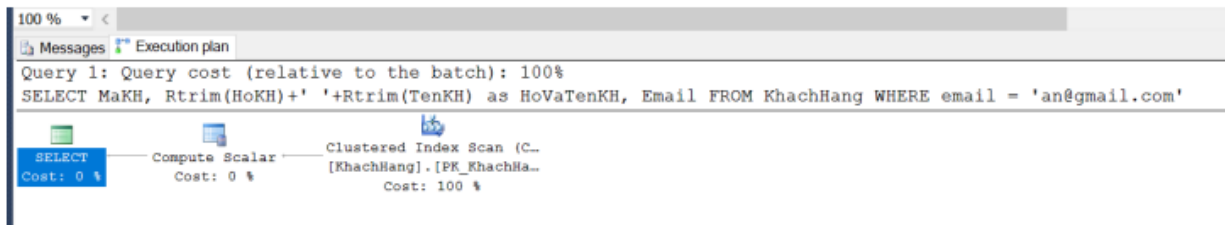
- Đầu tiên, chỉ định tên của index sau mệnh đề **CREATE NONCLUSTERED INDEX** . Lưu ý rằng từ khóa **NONCLUSTERED** là tùy chọn.
- Thứ hai, chỉ định tên bảng mà bạn muốn tạo index và danh sách các cột của bảng đó làm cột khóa index.

--Tạo non Clustered cho cột bảng KháchHang

KháchHang	
MaKH	
HoKH	
TenKH	
Phone	
Email	

--Tìm kiếm khách hàng có email 'an@gmail.com'

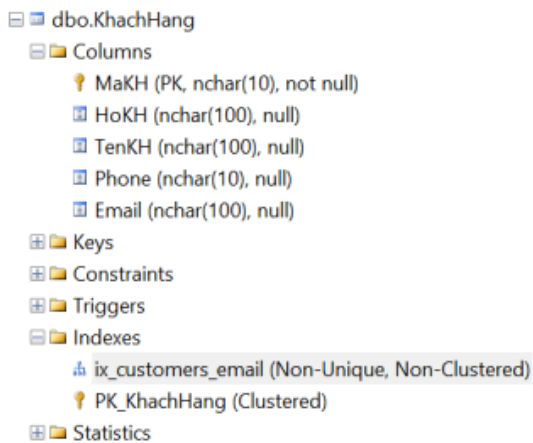
```
SELECT
    MaKH,
    Rtrim(HoKH)+' '+Rtrim(TenKH) as HoVaTenKH,
    Email
FROM
    KháchHang
WHERE
    email = 'an@gmail.com'
```



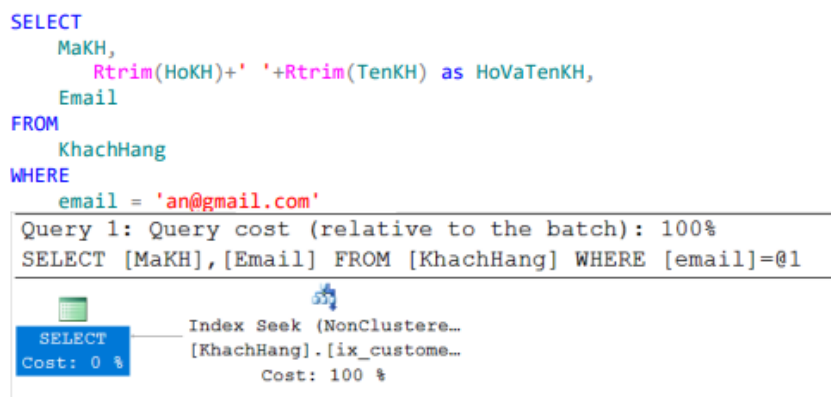
Nếu xem ước lượng kế hoạch ta thấy cột Email chưa có index clusterd nên tăng tốc độ ta tạo non cluster index cho cột Email

--non clustered cho cột email của khách hàng có 'an@gmail.com'

```
CREATE NONCLUSTERED INDEX ix_customers_email
ON KhachHang(email);
```



Xem lại ước lượng:



--Tạo Non Clustered Index cho nhiều cột:



Truy vấn các khách hàng có HoKH='Nguyễn Văn' và TenKH='An'

```
SELECT
    MaKH,
    Rtrim(HoKH)+' '+Rtrim(TenKH) as HoVaTenKH,
    Email
FROM
    KhachHang
WHERE
    HoKH=N'Nguyễn Văn' and TenKH=N'An'
```

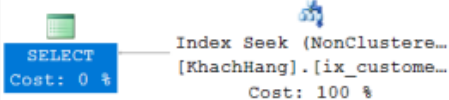
	MaKH	HoVaTenKH	Email
1	001	Nguyễn Văn An	an@gmail.com

Ta tạo Non Clustered Index cho HoKH và TenKH như sau:

```
--Tạo index non cluster index cho 2 cột HoKH và TenKH
CREATE NONCLUSTERED INDEX ix_customers_ho_tenkh
ON khachhang(HoKH, TenKH);
--Xem kết quả thực hiện
SELECT
    MaKH,
    HoKH,
    TenKH
FROM
    KhachHang
WHERE
    HoKH=N'Nguyễn Văn' and TenKH=N'An'
```

Query 1: Query cost (relative to the batch): 100%

SELECT [MaKH],[HoKH],[TenKH] FROM [KhachHang] WHERE [HoKH]=@1 AND [TenKH]=@2



Đổi tên index bằng cách sử dụng stored procedure sp\_rename

**sp\_rename** là một stored procedure hệ thống cho phép bạn đổi tên bất kỳ đối tượng nào do người dùng tạo trong cơ sở dữ liệu hiện tại bao gồm bảng, index và cột.

Câu lệnh đổi tên một index như sau:

```
1 EXEC sp_rename
2     index_name,
3     new_index_name,
4     N'INDEX';
```

Hoặc bạn có thể sử dụng các tham số rõ ràng như sau:

```
1 EXEC sp_rename
2     @objname = N'index_name',
3     @newname = N'new_index_name',
4     @objtype = N'INDEX';
```

--Câu lệnh đổi tên :




```
--Đổi tên ix_customers_email thành ix_cust_email
```

```
EXEC sp_rename
```

```
    N'khachhang.ix_customers_email',
```

```
    N'ix_cust_email' ,
```

```
    N'INDEX';
```

 Indexes

 ix\_cust\_email (Non-Unique, Non-Clustered)

## Vô hiệu hóa index trong SQL Server

Đôi khi, bạn cần phải vô hiệu hóa một index trước khi thực hiện **UPDATE** lớn trên bảng. Bằng cách vô hiệu hóa index, bạn có thể tăng tốc quá trình cập nhật bằng cách tránh chi phí ghi/cập nhật index.

Trong phần này, bạn sẽ học cách sử dụng câu lệnh **ALTER TABLE** để vô hiệu hóa các index của bảng.

### Cú pháp vô hiệu hóa index trong SQL Server

Để vô hiệu hóa một index, bạn sử dụng câu lệnh **ALTER INDEX** như sau:

```
1 ALTER INDEX index_name
2 ON table_name
3 DISABLE;
```

Để vô hiệu hóa tất cả các index của bảng, bạn sử dụng mẫu câu lệnh **ALTER INDEX ALL** như sau:

```
1 ALTER INDEX ALL ON table_name
2 DISABLE;
```

Nếu bạn vô hiệu hóa một index, trình tối ưu hóa truy vấn sẽ không sử dụng index bị vô hiệu hóa đó để tạo các kế hoạch thực hiện truy vấn.

Khi bạn vô hiệu hóa một index trên một bảng, SQL Server sẽ giữ định nghĩa chỉ mục trong siêu dữ liệu và thống kê index trong các non-clustered index. Tuy nhiên, nếu bạn vô hiệu hóa một non-clustered index hoặc clustered index trên một view, SQL Server sẽ xóa tất cả dữ liệu index.

Nếu bạn vô hiệu hóa một clustered index của một bảng, bạn không thể truy cập vào dữ liệu bảng sử dụng ngôn ngữ thao tác dữ liệu như **SELECT**, **INSERT**, **UPDATE** và **DELETE** cho đến khi bạn xây dựng lại hoặc xóa clustered index.

### Ví dụ vô hiệu hóa index trong SQL Server

```
--Vô hiệu hóa index ix_cust_email
```

```
ALTER INDEX ix_cust_email
```

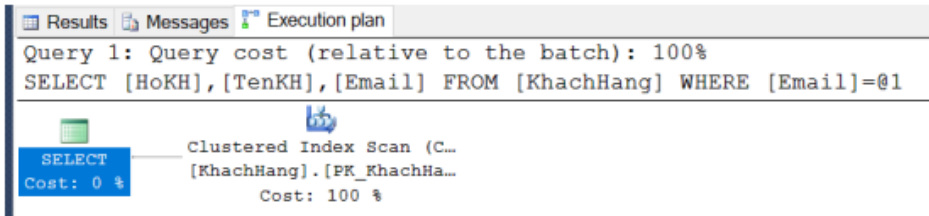
```
ON KháchHang
```

```
DISABLE;
```

```
SELECT
```

```
    HoKH,
```

```
TenKH,  
Email  
FROM  
KhachHang  
WHERE  
Email = 'an@gmail.com';  
Do hiệu hóa nên không còn tối ưu nữa
```



Vô hiệu hóa tất cả các index

```
--Vô hiệu hóa tất cả các index trên bảng khachhang
```

```
ALTER INDEX ALL ON KhachHang  
DISABLE;
```

Do đã vô hiệu hóa clustered index nên câu lệnh truy vấn trên bảng sẽ không sử dụng được

```
select * from KhachHang;
```

Messages

Msg 8655, Level 16, State 1, Line 1  
The query processor is unable to produce a plan because the index 'PK\_KhachHang' on table or view 'KhachHang' is disabled.

## Kích hoạt index trong SQL Server

Trong phần này, bạn sẽ tìm hiểu cách sử dụng các câu lệnh khác nhau để kích hoạt một hoặc tất cả các index bị vô hiệu hóa trong bảng.

Đôi khi, bạn cần phải vô hiệu hóa một index trước khi thực hiện **UPDATE** lớn trên bảng. Bằng cách vô hiệu hóa index, bạn có thể tăng tốc quá trình cập nhật bằng cách tránh chi phí ghi/cập nhật index.

Sau khi hoàn thành cập nhật vào bảng, bạn cần kích hoạt lại các index. Vì index đã bị vô hiệu hóa, bạn có thể xây dựng lại index nhưng không thể chỉ kích hoạt nó. Bởi vì sau khi cập nhật, index cần được xây dựng lại để phản ánh dữ liệu mới trong bảng.

Trong SQL Server, bạn có thể xây dựng lại một chỉ mục bằng cách sử dụng lệnh **ALTER INDEX** hoặc lệnh **DBCC DBREINDEX**.

```
--Kích hoạt lại các index đã vô hiệu hóa
```

```
ALTER INDEX PK_KhachHang ON KhachHang  
REBUILD;
```

```
ALTER INDEX ix_cust_email ON KhachHang  
REBUILD;  
ALTER INDEX ix_customers_ho_tenkH ON KhachHang  
REBUILD;  
select * from KhachHang;
```

	MaKH	HoKH	TenKH	Phone	Email
1	001	Nguyễn Văn	An	098899999	an@gmail.com
2	002	Lưu Bình	Nguyễn	0978977777	nguyen@gmail.com

--Kích hoạt lại tất cả các index bị vô hiệu hóa

```
ALTER INDEX ALL ON table_name  
REBUILD;
```

--Hoặc sử dụng để kích hoạt lại toàn bộ index bảng khách hàng

```
ALTER INDEX ALL ON KhachHang  
REBUILD;
```

## Kích hoạt index bằng câu lệnh DBCC DBREINDEX trong SQL Server

Câu lệnh sau đây sử dụng lệnh **DBCC DBREINDEX** để kích hoạt một index trên bảng:

```
1 | DBCC DBREINDEX (table_name, index_name);
```

Câu lệnh sau đây sử dụng lệnh **DBCC DBREINDEX** để kích hoạt tất cả các index trên một bảng:

```
1 | DBCC DBREINDEX (table_name, " ");
```

```
DBCC DBREINDEX (KhachHang, ix_cust_email);  
DBCC DBREINDEX (KhachHang, ix_customers_ho_tenkH);  
DBCC DBREINDEX (KhachHang, PK_KhachHang);  
3 câu lệnh này tương đương  
DBCC DBREINDEX (KhachHang, " ");
```

## Xóa index trong SQL Server

Trong phần này, bạn sẽ tìm hiểu cách sử dụng câu lệnh `DROP INDEX` trong SQL Server để xóa index.

### Tổng quan về câu lệnh DROP INDEX trong SQL Server

Câu lệnh `DROP INDEX` xóa một hoặc nhiều index khỏi cơ sở dữ liệu hiện tại. Đây là cú pháp của câu lệnh `DROP INDEX`:

```
1 DROP INDEX [IF EXISTS] index_name
2 ON table_name;
```

Trong cú pháp này:

- Đầu tiên, chỉ định tên của index mà bạn muốn xóa sau mệnh đề `DROP INDEX`.
- Thứ hai, chỉ định tên của bảng chứa index.

Xóa một index không tồn tại sẽ dẫn đến một lỗi. Tuy nhiên, bạn có thể sử dụng tùy chọn `IF EXISTS` để xóa index một cách có điều kiện và tránh lỗi.

*Lưu ý: tùy chọn `IF EXISTS` đã có sẵn kể từ SQL Server 2016 (13.x).*

Câu lệnh `DROP INDEX` không thể xóa các index được tạo bởi `PRIMARY KEY` hoặc các ràng buộc `UNIQUE`. Để xóa các index liên quan đến các ràng buộc này, bạn sử dụng câu lệnh `ALTER TABLE DROP CONSTRAINT`.

Để xóa nhiều index khỏi một hoặc nhiều bảng cùng một lúc, bạn chỉ định danh sách tên index được phân tách bằng dấu phẩy với tên bảng tương ứng sau mệnh đề `DROP INDEX` như trong truy vấn sau:

```
1 DROP INDEX [IF EXISTS]
2     index_name1 ON table_name1,
3     index_name2 ON table_name2,
4     ...;
```

```
--Xóa index ix_cust_email
DROP INDEX ix_cust_email
ON KháchHang;
```

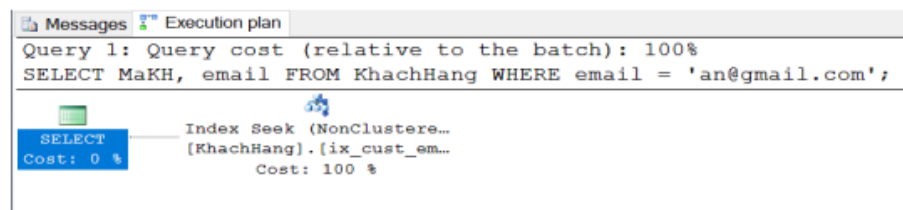
## Index với các cột được bao gồm trong SQL Server

Trong phần này, bạn sẽ tìm hiểu cách sử dụng các index với các cột được bao gồm để cải thiện tốc độ truy vấn.

### Giới thiệu về index với các cột được bao gồm trong SQL Server

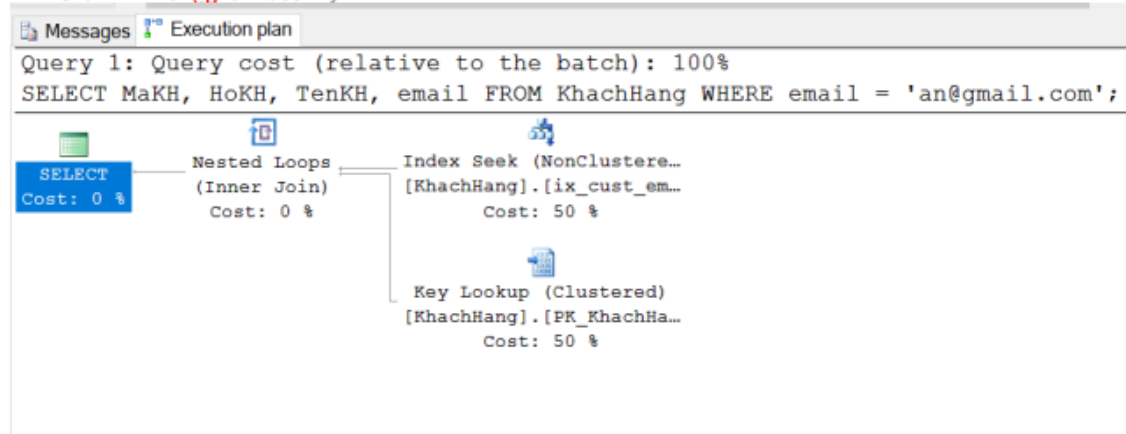
--Tạo index cho cột email

```
CREATE UNIQUE INDEX ix_cust_email  
ON KháchHang(email);  
--Truy vấn  
SELECT  
    MaKH,  
    email  
FROM  
    KháchHang  
WHERE  
    email = 'an@gmail.com';
```



--Tuy nhiên khi sử dụng câu truy vấn này thì sẽ plan kết hợp

```
SELECT  
    MaKH,  
    HoKH,  
    TenKH,  
    email  
FROM  
    KháchHang  
WHERE  
    email = 'an@gmail.com';
```



Trong kế hoạch thực hiện này:

Đầu tiên trình tối ưu hóa truy vấn sử dụng index seek trên non clustered index ix\_cust\_email để tìm MaKH và Email

Index Seek (NonClustered)	
Scan a particular range of rows from a nonclustered index.	
Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0032831 (50%)
Estimated Subtree Cost	0.0032831
Estimated CPU Cost	0.0001581
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	227 B
Ordered	True
Node ID	1
<b>Object</b>	
[QLSACH].[dbo].[KhachHang].[ix_cust_email_unique]	
<b>Output List</b>	
[QLSACH].[dbo].[KhachHang].MaKH, [QLSACH].[dbo].[KhachHang].Email	
<b>Seek Predicates</b>	
Seek Keys[1]: Prefix: [QLSACH].[dbo].[KhachHang].Email = Scalar Operator (CONVERT_IMPLICIT(nvarchar(4000),[@1],0))	

Thứ hai, trình tối ưu hóa truy vấn sử dụng tra cứu khóa (key lookup) trên clustered index của bảng KhachHang để tìm HoKH và TenKH theo MaKH của KhachHang

Key Lookup (Clustered)	
Uses a supplied clustering key to lookup on a table that has a clustered index.	
Physical Operation	Key Lookup
Logical Operation	Key Lookup
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0032831 (50%)
Estimated Subtree Cost	0.0032831
Estimated CPU Cost	0.0001581
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	407 B
Ordered	True
Node ID	3
<b>Object</b>	
[QLSACH].[dbo].[KhachHang].[PK_KhachHang]	
<b>Output List</b>	
[QLSACH].[dbo].[KhachHang].HoKH, [QLSACH].[dbo].[KhachHang].TenKH	
<b>Seek Predicates</b>	
Seek Keys[1]: Prefix: [QLSACH].[dbo].[KhachHang].MaKH = Scalar Operator([QLSACH].[dbo].[KhachHang].[MaKH])	

Thứ ba, mỗi bản ghi được tìm thấy trong non-clustered index sẽ khớp với các bản ghi được tìm thấy trong clustered index bằng các vòng lặp lồng nhau.

Như bạn có thể thấy chi phí cho việc tra cứu khóa là khoảng 50% truy vấn, khá tốn kém.

Để giúp giảm chi phí tra cứu khóa này, SQL Server cho phép bạn mở rộng chức năng của một non-clustered index bằng cách bao gồm các cột không phải khóa.

Bằng cách bao gồm các cột không phải khóa trong các non-clustered index, bạn có thể tạo các non-clustered index bao trùm cho nhiều truy vấn hơn.

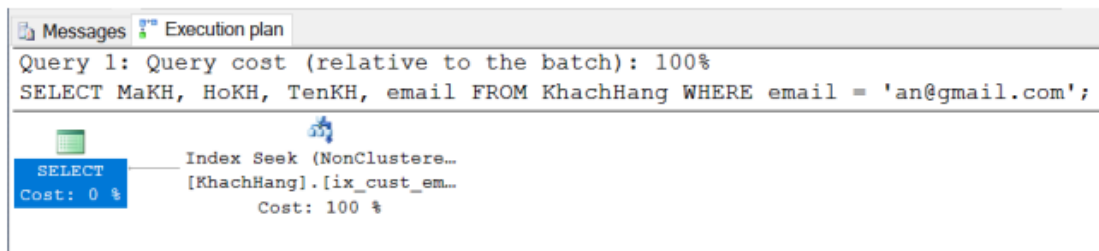
*Lưu ý: khi một index chứa tất cả các cột được tham chiếu bởi một truy vấn, chỉ mục thường được xem là bao trùm truy vấn.*

```
--drop index ix_cust_email
DROP INDEX ix_cust_email
ON KhachHang;
```



```
--Tạo index với 2 cột kết hợp
CREATE UNIQUE INDEX ix_cust_email_inc
ON KhachHang(email)
INCLUDE(HoKH,TenKH);

-----
SELECT
    MaKH,
    HoKH,
    TenKH,
    email
FROM
    KhachHang
WHERE
    email = 'an@gmail.com';
```



Một index với các cột được bao gồm có thể cải thiện đáng kể hiệu năng truy vấn vì tất cả các cột trong truy vấn đều được bao gồm trong index; Trình tối ưu hóa truy vấn có thể định vị tất cả các giá trị cột trong index mà không cần truy cập vào bảng hoặc clustered index dẫn đến ít hoạt động I/O trên đĩa hơn.

## Cú pháp tạo index với các cột được bao gồm trong SQL Server

Dưới đây minh họa cú pháp để tạo một non-clustered index với các cột được bao gồm:

```
1 CREATE [UNIQUE] INDEX index_name
2 ON table_name(key_column_list)
3 INCLUDE(included_column_list);
```

Trong cú pháp này:

- Đầu tiên, chỉ định tên của index sau mệnh đề **CREATE INDEX**. Nếu index là duy nhất, bạn cần thêm từ khóa **UNIQUE**.
- Thứ hai, chỉ định tên của bảng và danh sách danh sách cột chính cho index sau mệnh đề **ON**.
- Thứ ba, liệt kê một danh sách các cột được bao gồm bằng dấu phẩy trong mệnh đề **INCLUDE**.

---Phần nâng cao index (chỉ tham khảo thêm)

## Filtered index trong SQL Server

Trong phần này, bạn sẽ tìm hiểu cách sử dụng các filtered index trong SQL Server để tạo các non-clustered index được tối ưu hóa cho các bảng.

### Giới thiệu về filtered index trong SQL Server

Một non-clustered index, khi được sử dụng đúng cách, có thể cải thiện đáng kể hiệu năng của các truy vấn. Tuy nhiên, lợi ích của các non-clustered index có chi phí: lưu trữ và bảo trì.

- Đầu tiên, nó cần bộ nhớ bổ sung để lưu trữ bản sao dữ liệu của các cột khóa của index.
- Thứ hai, khi bạn **INSERT**, **UPDATE** hoặc **DELETE** các bản ghi khỏi bảng, SQL Server cần cập nhật non-clustered index được liên kết.

Nó sẽ không hiệu quả nếu các ứng dụng chỉ truy vấn một phần các bản ghi của bảng. Đây là đất dụng võ của các filtered index.

Một filtered index là một non-clustered index với một biểu thức cho phép bạn chỉ định những bản ghi nào sẽ được thêm vào index.

Cú pháp sau minh họa cách tạo filtered index:

```
1 CREATE INDEX index_name
2 ON table_name(column_list)
3 WHERE predicate;
```

Trong cú pháp này:

- Đầu tiên, chỉ định tên của filtered index sau lệnh **CREATE INDEX**.
- Thứ hai, liệt kê tên bảng với danh sách các cột sẽ được bao gồm trong index.
- Thứ ba, sử dụng mệnh đề **WHERE** với một biểu thức để chỉ định các bản ghi của bảng sẽ được đưa vào index.

### Ví dụ về filtered index trong SQL Server

Chúng tôi sẽ sử dụng bảng **sales.customers** từ **cơ sở dữ liệu mẫu** để minh họa cho câu lệnh filtered index trong SQL Server.

sales.customers
* customer_id
first_name
last_name
phone
email
street
city
state
zip_code

Bảng `sales.customers` có cột `phone`, trong đó có nhiều giá trị `NULL`:

```
1 SELECT
2     SUM(CASE
3         WHEN phone IS NULL
4         THEN 1
5         ELSE 0
6     END) AS [Has Phone],
7     SUM(CASE
8         WHEN phone IS NULL
9         THEN 0
10        ELSE 1
11    END) AS [No Phone]
12 FROM
13     sales.customers;
```

Đây là kết quả:

1	Has Phone	No Phone
2	-----	-----
3	1267	178

Cột `phone` là một ứng cử viên tốt cho filtered index.

Câu lệnh này tạo ra một filtered index cho cột `phone` của bảng `sales.customers`:

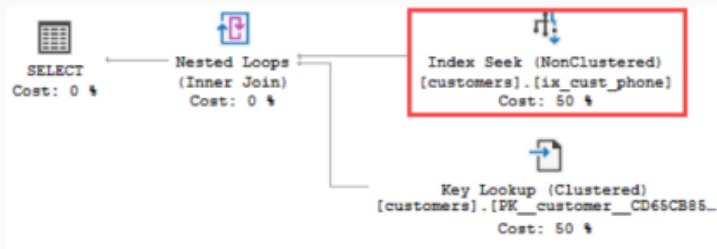
Câu lệnh này tạo ra một filtered index cho cột `phone` của bảng `sales.customers` :

```
1 CREATE INDEX ix_cust_phone
2 ON sales.customers(phone)
3 WHERE phone IS NOT NULL;
```

Truy vấn sau đây tìm kiếm khách hàng có số điện thoại là `(281) 363-3309` :

```
1 SELECT
2     first_name,
3     last_name,
4     phone
5 FROM
6     sales.customers
7 WHERE phone = '(281) 363-3309';
```

Dưới đây là kế hoạch thực hiện ước tính:



Trình tối ưu hóa truy vấn có thể tận dụng filtered index `ix_cust_phone` để tìm kiếm.

Lưu ý rằng để cải thiện tra cứu khóa, bạn có thể sử dụng một index với các cột được bao gồm (được trình bày ở phần trước trong bài viết này) để bao gồm cả hai cột `first_name` và `last_name` :

```
1 CREATE INDEX ix_cust_phone
2 ON sales.customers(phone)
3 INCLUDE (first_name, last_name)
4 WHERE phone IS NOT NULL;
```

## Các lợi ích của filtered index trong SQL Server

Như đã đề cập trước đó, các filtered index có thể giúp bạn tiết kiệm không gian đặc biệt là khi giá trị của các cột khóa của index còn rải rác (có nhiều giá trị NULL).

Ngoài ra, các filtered index làm giảm chi phí bảo trì vì chỉ một phần của các bản ghi có dữ liệu, không phải tất cả, cần được cập nhật khi dữ liệu trong bảng liên kết thay đổi.

## Tạo index trên các cột được tính toán trong SQL Server

Trong phần này, bạn sẽ tìm hiểu cách mô phỏng các index dựa trên chức năng trong SQL Server bằng cách sử dụng các index trên các cột được tính toán.

### Giới thiệu về index trên các cột được tính toán

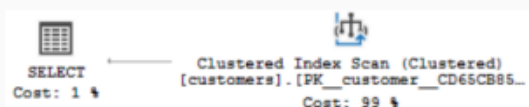
Xem bảng `sales.customers` từ [cơ sở dữ liệu mẫu](#):

sales.customers
* customer_id
first_name
last_name
phone
email
street
city
state
zip_code

Truy vấn này tìm kiếm khách hàng có phần đầu (phần trước @) của địa chỉ email là `'garry.espinoza'` ;

```
1  SELECT
2      first_name,
3      last_name,
4      email
5  FROM
6      sales.customers
7  WHERE
8      SUBSTRING(email, 0,
9          CHARINDEX('@', email, 0)
10     ) = 'garry.espinoza';
```

Dưới đây là kế hoạch thực thi của truy vấn:



Như được thể hiện rõ ràng trong kế hoạch thực thi, trình tối ưu hóa truy vấn cần quét toàn bộ clustered index để định vị khách hàng, không hiệu quả.

Nếu bạn đã làm việc với Oracle hoặc PostgreSQL, bạn có thể biết rằng Oracle hỗ trợ **các index dựa trên chức năng** và PostgreSQL có **các index dựa trên biểu thức**. Các loại index này cho phép bạn lập index cho kết quả của hàm hoặc biểu thức sẽ cải thiện hiệu năng của các truy vấn có mệnh đề **WHERE** chứa hàm và biểu thức.

Trong SQL Server, bạn có thể sử dụng một index trên một cột được tính toán để đạt được hiệu quả tương tự của một index dựa trên chức năng:

- Đầu tiên, tạo một cột được tính toán dựa trên biểu thức trên mệnh đề **WHERE**.
- Thứ hai, tạo một non-clustered index cho cột được tính toán.

Ví dụ: để tìm kiếm khách hàng dựa trên các phần đầu của địa chỉ email của họ, bạn sử dụng các bước sau:

Đầu tiên, thêm một cột được tính toán vào bảng **sales.customers**:

Đầu tiên, thêm một cột được tính toán vào bảng **sales.customers**:

```
1 ALTER TABLE sales.customers
2 ADD
3     email_local_part AS
4         SUBSTRING(email,
5             0,
6             CHARINDEX('@', email, 0)
7         );
```

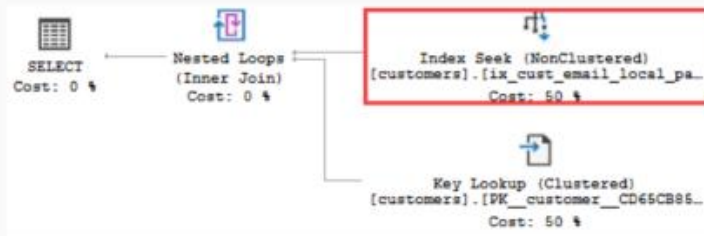
Sau đó, tạo một index trên cột **email\_local\_part**:

```
1 CREATE INDEX ix_cust_email_local_part
2 ON sales.customers(email_local_part);
```

Bây giờ, bạn có thể sử dụng cột **email\_local\_part** thay vì biểu thức trong mệnh đề **WHERE** để tìm khách hàng theo phần đầu của địa chỉ email:

```
1 SELECT
2     first_name,
3     last_name,
4     email
5 FROM
6     sales.customers
7 WHERE
8     email_local_part = 'garry.espinoza';
```

Trình tối ưu hóa truy vấn sử dụng thao tác tìm kiếm index seek trên index `ix_cust_email_local_part` như trong hình sau:



### Yêu cầu đối với index trên các cột được tính toán

Để tạo một index trên một cột được tính toán, các yêu cầu sau phải được đáp ứng:

- Các hàm liên quan đến biểu thức cột được tính toán phải có cùng chủ sở hữu với bảng.
- Biểu thức cột được tính toán phải có tính xác định. Điều đó có nghĩa là biểu thức luôn trả về cùng một kết quả cho một tập hợp đầu vào đã cho.
- Cột được tính toán phải chính xác, có nghĩa là biểu thức của nó không được chứa bất kỳ kiểu dữ liệu `FLOAT` hoặc `REAL` nào.
- Kết quả của biểu thức cột được tính toán không thể là kiểu dữ liệu không thể đánh giá như `TEXT`, `NTEXT` hoặc `IMAGE`.
- Các tùy chọn `ANSI_NULLS` phải được thiết lập `ON` để khi cột được tính toán được định nghĩa bằng cách sử dụng lệnh `CREATE TABLE` hoặc `ALTER TABLE`. Bên cạnh đó, các tùy chọn `ANSI_PADDING`, `ANSI_WARNINGS`, `ARITHABORT`, `QUOTED_IDENTIFIER`, và `CONCAT_NULL_YIELDS_NULL` cũng phải được thiết lập là `ON` và `NUMERIC_ROUNDABORT` phải được thiết lập là `OFF`.

## Unique Index trong SQL Server

Trong phần này, bạn sẽ tìm hiểu về unique index trong SQL Server và cách sử dụng chúng để thực thi tính duy nhất của các giá trị trong một hoặc nhiều cột của bảng.

### Tổng quan về unique index trong SQL Server

Unique index đảm bảo các cột khóa của chỉ mục không chứa bất kỳ giá trị trùng lặp nào.

Một unique index có thể bao gồm một hoặc nhiều cột. Nếu một unique index có một cột, các giá trị trong cột này sẽ là duy nhất. Trong trường hợp unique index có nhiều cột, sự kết hợp các giá trị trong các cột này là duy nhất.

Mọi nỗ lực INSERT hoặc UPDATE dữ liệu vào các cột khóa của unique index gây ra trùng lặp sẽ dẫn đến lỗi.

Một unique index có thể là clustered index hoặc non-clustered index.

Để tạo một unique index, bạn sử dụng câu lệnh `CREATE UNIQUE INDEX` như sau:



```
1 CREATE UNIQUE INDEX index_name
2 ON table_name(column_list);
```

Trong cú pháp này:

- Đầu tiên, chỉ định tên của unique index sau mệnh đề **CREATE UNIQUE INDEX**.
- Sau đó chỉ định tên của bảng mà index được liên kết và danh sách các cột sẽ được bao gồm trong index.

## Ví dụ về unique index trong SQL Server

Hãy lấy một số ví dụ về việc sử dụng các unique index.

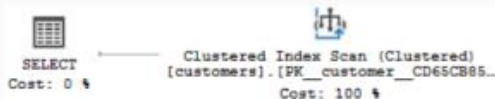
### Tạo unique index cho một cột trong SQL Server

Truy vấn sau đây tìm kiếm khách hàng có email **'caren.stephens@msn.com'**:

### Tạo unique index cho một cột trong SQL Server

Truy vấn sau đây tìm kiếm khách hàng có email **'caren.stephens@msn.com'**:

```
1 SELECT
2     customer_id,
3     email
4 FROM
5     sales.customers
6 WHERE
7     email = 'caren.stephens@msn.com';
```



Trình tối ưu hóa truy vấn phải quét toàn bộ clustered index để tìm các bản ghi phù hợp.

Để tăng tốc độ truy xuất truy vấn, bạn có thể thêm một non-clustered index vào cột **email**.

Tuy nhiên, với giả định rằng mỗi khách hàng sẽ có một email duy nhất, bạn có thể tạo một unique index cho cột **email**.

Vì bảng `sales.customers` đã có dữ liệu, trước tiên bạn cần kiểm tra các giá trị trùng lặp trong cột `email`:

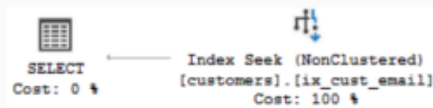
```
1 SELECT
2     email,
3     COUNT(email)
4 FROM
5     sales.customers
6 GROUP BY
7     email
8 HAVING
9     COUNT(email) > 1;
```

Truy vấn trả về một tập kết quả trống. Nó có nghĩa là không có giá trị trùng lặp trong cột `email`.

Do đó, bạn có thể tạo một unique index cho cột `email` của bảng `sales.customers`:

```
1 CREATE UNIQUE INDEX ix_cust_email
2 ON sales.customers(email);
```

Từ giờ trở đi, trình tối ưu hóa truy vấn sẽ tận dụng index `ix_cust_email` và sử dụng phương thức `index seek` để tìm kiếm khách hàng theo email.



### Tạo unique index cho nhiều cột trong SQL Server

Đầu tiên, chúng ta sẽ tạo một bảng có tên `t1` có hai cột để minh họa cho ví dụ này:

```
1 CREATE TABLE t1 (
2     a INT,
3     b INT
4 );
```

Tiếp theo, tạo một unique index bao gồm cả hai cột `a` và `b`:

```
1 CREATE UNIQUE INDEX ix_uniq_ab
2 ON t1(a, b);
```

Tiến hành **INSERT** một bản ghi mới vào bảng **t1** :

```
1 INSERT INTO t1(a,b)
2 VALUES(1,1);
```

Sau đó, **INSERT** thêm một bản ghi khác vào bảng **t1** . Lưu ý rằng giá trị 1 được lặp lại trong cột **a** , nhưng sự kết hợp của các giá trị trong cột **a** và **b** không trùng lặp:

```
1 INSERT INTO t1(a,b)
2 VALUES(1,2);
```

Cuối cùng, chèn một bản ghi đã tồn tại vào bảng **t1** :

```
1 INSERT INTO t1(a,b)
2 VALUES(1,2);
```

## Unique index so với UNIQUE constraint

Cả unique index và **UNIQUE** constraint (ràng buộc duy nhất) đều thực thi tính duy nhất của các giá trị trong một hoặc nhiều cột. SQL Server xác thực sự trùng lặp theo cùng một cách cho cả unique index và unique constraint.

Khi bạn tạo một unique constraint, SQL Server sẽ tạo một unique index liên kết với unique constrain này.

Tuy nhiên, việc tạo một unique constraint trên các cột làm cho mục tiêu của unique index rõ ràng.

Máy chủ SQL gặp lỗi :

```
1 Cannot insert duplicate key row in object 'dbo.t1' with unique index 'ix_ab'. The duplicate key value is (1,1).
```

## Unique index và NULL trong SQL Server

**NULL** thật đặc biệt. Nó đánh dấu cho biết thông tin còn thiếu hoặc không áp dụng.

NULL thậm chí không bằng chính nó. Tuy nhiên, khi nói đến unique index, SQL Server xử lý các giá trị NULL như nhau. Điều đó có nghĩa là nếu bạn tạo một unique index trên một cột NULL, bạn chỉ có thể có một giá trị NULL trong cột này.

Các câu lệnh sau đây tạo một bảng mới có tên **t2** và định nghĩa một unique index trên cột **a** :

```
1 CREATE TABLE t2(  
2     a INT  
3 );  
4  
5 CREATE UNIQUE INDEX a_uniq_t2  
6 ON t2(a);
```

Truy vấn này chèn giá trị NULL vào cột **a** của bảng **t2**:

```
1 INSERT INTO t2(a)  
2 VALUES(NULL);
```

Tuy nhiên, khi thực hiện lại truy vấn trên, SQL Server báo lỗi do các giá trị NULL trùng lặp:

```
1 INSERT INTO t2(a)  
2 VALUES(NULL);
```