

# Physical Database Design

## Indexes Part 2 : Heap Table and Table Scan



## Heap Table



```
graph LR; A[Heap Table] --- B[Table does not contain any clustered index]; A --- C[But heap table can contain non-clustered indexes or any another type of indexes]; A --- D[heap table store data on disk without any order]; A --- E[Usually data stored physically on the disk as they insert first.]; A --- F[but some time database engine can split , shift or shrink data pages based on the amount of data and available space for write operations like : insert , update , delete.];
```

Table does not contain any clustered index

But heap table can contain non-clustered indexes or any another type of indexes

heap table store data on disk without any order

Usually data stored physically on the disk as they insert first.

but some time database engine can split , shift or shrink data pages based on the amount of data and available space for write operations like : insert , update , delete.

## Create Product Heap Table

```
graph LR; A[Create Product Heap Table] --- B[Create Product Table Without any primary key or clustered index]; A --- C[CREATE TABLE [dbo].[Product] ([ID] [int] NOT NULL, [Name] [nvarchar](255) NULL, [Price] [int] NULL, [AvailableQuantity] [int] NULL) GO];
```

Create Product Table Without any primary key or clustered index

```
CREATE TABLE [dbo].[Product] (  
[ID] [int] NOT NULL,  
[Name] [nvarchar](255) NULL,  
[Price] [int] NULL,  
[AvailableQuantity] [int] NULL  
)  
GO
```

## Insert Rows To Product Heap Table :

```
graph LR; A[Insert Rows To Product Heap Table :] --- B[Insert into Product ( 5 , 'Product 5' , 100 , 20 )]; A --- C[Insert into Product ( 6 , 'Product 6' , 120 , 17)]; A --- D[Insert into Product ( 4 , 'Product 4' , 50 , 26)]; A --- E[Insert into Product ( 1 , 'Product 1' , 90 , 30)]; A --- F[Insert into Product ( 3 , 'Product 3' , 80 , 36)]; A --- G[Insert into Product ( 2 , 'Product 2' , 35 , 14)];
```

Insert into Product ( 5 , 'Product 5' , 100 , 20 )

Insert into Product ( 6 , 'Product 6' , 120 , 17)

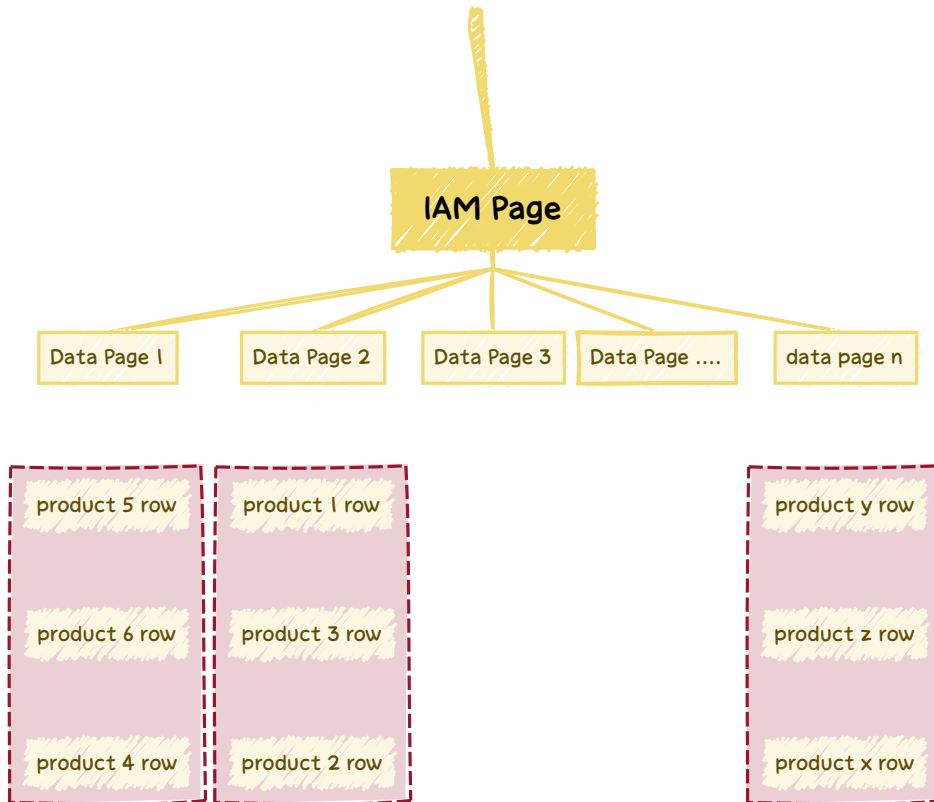
Insert into Product ( 4 , 'Product 4' , 50 , 26)

Insert into Product ( 1 , 'Product 1' , 90 , 30)

Insert into Product ( 3 , 'Product 3' , 80 , 36)

Insert into Product ( 2 , 'Product 2' , 35 , 14)

# Product Heap Table Physical Representation On Disk



Data is Stored in The Disk Without Any Order In Heap Table

Select \* From Product

## Product Heap Table Logical Representation On Memory

	ID	Name	Price	Available Quantity
1	5	product 5	100	20
2	6	product 6	120	17
3	4	product 4	50	26
4	1	product 1	90	30
5	3	product 3	80	36
6	2	product 2	35	14
.....	....	.....	.....	.....

## When To Use Heap Table ?

microsoft docs

<https://docs.microsoft.com/en-us/sql/relational-databases/indexes/heap-tables-without-clustered-indexes>

Usually Most Cases Clustered Index Table is Better Than Heap Table , But there some cases when heap table can be better than clustered index table :

## When To Use Heap Table ?

2. insert operation usually more faster than cluster index table , because heap table dose not maintenance the order , so heap table can use as staging or temp table to insert a huge amount of unsorted data



## When To Use Heap Table ?

```
graph LR; A[When To Use Heap Table ?] --- B["1. most of select queries read all or most rows on the table and dose not enforcing a strict order to return rows , so heap table in this case will be faster than cluster index table because cluster index table has extra pages."]; A --- C[good : select * from product]; A --- D[bad : select * from product order by ID]; A --- E[bad : select * from product Where ID = 666666];
```

1. most of select queries read all or most rows on the table and dose not enforcing a strict order to return rows , so heap table in this case will be faster than cluster index table because cluster index table has extra pages.

good : select \* from product

bad : select \* from product order by ID

bad : select \* from product Where ID = 666666

## When To Not Use Heap Table ?

```
graph LR; A[When To Not Use Heap Table ?] --- B[Do not use a heap when the data is frequently returned in a sorted order. A clustered index on the sorting column could avoid the sorting operation.]; A --- C[good with heap table : select * from product]; A --- D[bad with heap table : select * from product order by ID];
```

Do not use a heap when the data is frequently returned in a sorted order. A clustered index on the sorting column could avoid the sorting operation.

good with heap table : `select * from product`

bad with heap table : `select * from product order by ID`

## When To Not Use Heap Table ?

```
graph LR; A[When To Not Use Heap Table ?] --- B[Do not use a heap when the data is frequently grouped together. Data must be sorted before it is grouped, and a clustered index on the sorting column could avoid the sorting operation.]; A --- C[bad with heap table :]; A --- D["select count(name) as numberOfProductsHasSamePrice , Price from Product group by Price"];
```

Do not use a heap when the data is frequently grouped together. Data must be sorted before it is grouped, and a clustered index on the sorting column could avoid the sorting operation.

bad with heap table :

```
select count(name) as  
numberOfProductsHasSamePrice , Price  
from Product  
group by Price
```

## When To Not Use Heap Table ?

```
graph LR; A[When To Not Use Heap Table ?] --- B[Do not use a heap when ranges of data are frequently queried from the table. A clustered index on the range column will avoid sorting the entire heap.]; A --- C[bad with heap table :]; A --- D["select * from Product where id > 1000 and id < 2000"];
```

Do not use a heap when ranges of data are frequently queried from the table. A clustered index on the range column will avoid sorting the entire heap.

bad with heap table :

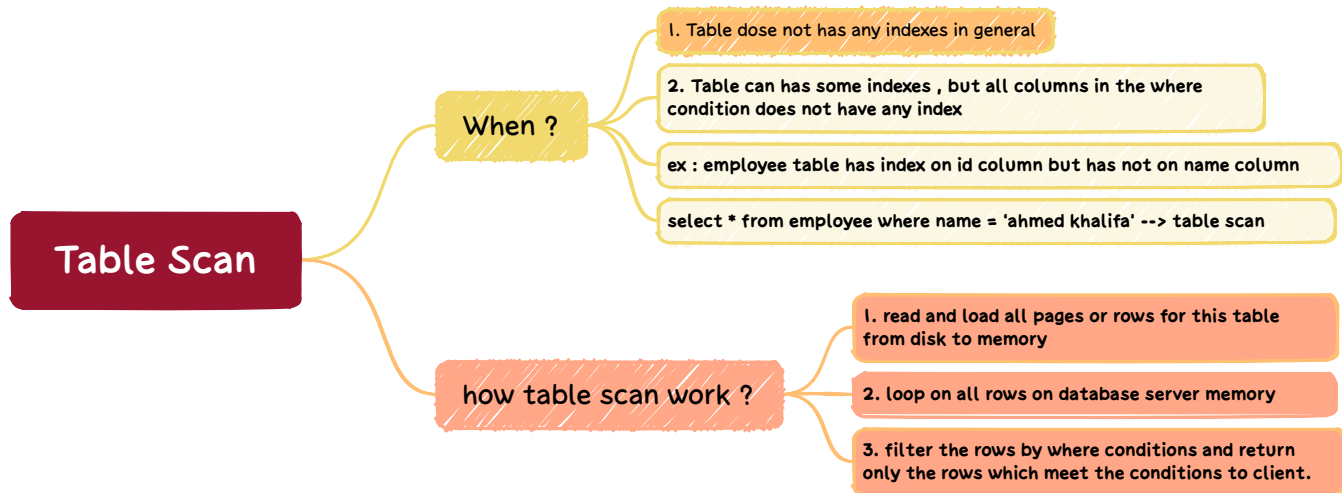
```
select * from Product
where id > 1000 and id < 2000
```

## When To Not Use Heap Table ?

Do not use a heap when there are no nonclustered indexes and the table is large, unless you intend to return the entire table content without any specified order. In a heap, all rows of the heap must be read to find any row

## When To Not Use Heap Table ?

Do not use a heap if the data is frequently updated. If you update a record and the update uses more space in the data pages than they are currently using, the record has to be moved to a data page that has enough free space. This creates a forwarded record pointing to the new location of the data, and forwarding pointer has to be written in the page that held that data previously, to indicate the new physical location. This introduces fragmentation in the heap. When scanning a heap, these pointers must be followed which limits read-ahead performance, and can incur additional I/O which reduces scan performance.



## Table Scan Complexity

```
graph LR; A[Table Scan Complexity] --- B[algorithm : Linear Search]; A --- C["Time Complexity : O(n)  
Because The Algorithm Need To Loop on All rows on Memory"]; A --- D["Space Complexity : O(n)  
Because the engine need to read and load all rows and pages from Disk to Memory"];
```

algorithm : Linear Search

Time Complexity :  $O(n)$   
Because The Algorithm Need To Loop on All rows on Memory

Space Complexity :  $O(n)$   
Because the engine need to read and load all rows and pages from Disk to Memory



## Table Scan With one Million Rows Example

```
graph LR; A[Table Scan With one Million Rows Example] --- B[Insert 1 Million Rows By Data Generator Tools Like Spawner]; A --- C[all Tools and Scripts are attached on github repository : https://github.com/Ahm7dKhalifa/Database-Design]; A --- D[Note To Remember : Product Table is Heap , and Dose Not Has Cluster Index]; A --- E[SELECT * FROM [dbo].[Product] WHERE ID = 666666]; A --- F[The Result as you see on The Next Image , Read and Load All Rows To Return Only One Row];
```

Insert 1 Million Rows By Data Generator Tools Like Spawner

all Tools and Scripts are attached on github repository : <https://github.com/Ahm7dKhalifa/Database-Design>

Note To Remember : Product Table is Heap , and Dose Not Has Cluster Index

```
SELECT * FROM [dbo].[Product]
WHERE ID = 666666
```

The Result as you see on The Next Image , Read and Load All Rows To Return Only One Row



Object Explorer

Connect

- DESKTOP-GNFE269\MSSQLSERVER01 (DESKTOP-GNFE269\lenovo (55))
  - Databases
    - System Databases
    - Database Snapshots
    - Indexes\_Tutorials
      - Database Diagrams
      - Tables
        - System Tables
        - FileTables
        - External Tables
        - Graph Tables
        - dbo.Product
      - Views
      - External Resources
      - Synonyms
      - Programmability
      - Service Broker
      - Storage
      - Security
    - Security
    - Server Objects
    - Replication
    - PolyBase
    - Always On High Availability
    - Management
    - Integration Services Catalogs
    - SQL Server Agent (Agent XPs disabled)
    - XEvent Profiler

SQLQuery2.sql - DE...FE269\lenovo (55))\* SQLQuery1.sql - DE...FE269\lenovo (53))\*

```

/*
select product with id = 666666 when there is no any index
*/
SELECT * FROM [dbo].[Product]
WHERE ID = 666666
    
```

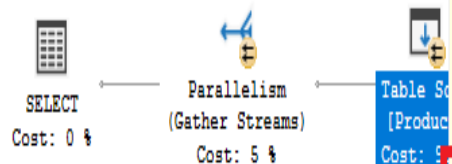
100 %

Messages Execution plan

Query 1: Query cost (relative to the batch)

SELECT \* FROM [dbo].[Product] WHERE ID = 666666

Missing Index (Impact 99.439): CREATE NONCLUSTERED INDEX [IX\_Product\_ID] ON [dbo].[Product] ([ID])



### Table Scan

Scan rows from a table.	
Physical Operation	Table Scan
Logical Operation	Table Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	5.50435 (95%)
Estimated I/O Cost	4.95431
Estimated Subtree Cost	5.50435
Estimated CPU Cost	0.550039
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1000000
Estimated Number of Rows for All Executions	1.01348
Estimated Number of Rows Per Execution	1.01348
Estimated Row Size	278 B
Ordered	False
Node ID	1

### Predicate

[Indexes\_Tutorials].[dbo].[Product].[ID]=[@1]

### Object

[Indexes\_Tutorials].[dbo].[Product]

### Output List

[Indexes\_Tutorials].[dbo].[Product].ID, [Indexes\_Tutorials].[dbo].[Product].Name, [Indexes\_Tutorials].[dbo].[Product].Price, [Indexes\_Tutorials].[dbo].[Product].AvailableQuantity

Query executed successfully.