

Optimización - SQL Server

Dr. Gerardo Rossel



Bases de Datos

2025

Entornos

Azure Data Studio (INSTALADO EN LOS LABOS)

The screenshot displays the Azure Data Studio interface. On the left, the 'SERVERS' pane shows a tree view of the 'AdventureWorks2017' database, including tables like 'HumanResources.Department' and 'HumanResources.Employee'. The central editor shows a SQL query: `select * from HumanResources.Department`. The 'RESULTADOS' pane at the bottom displays the query results in a table format.

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and ...	2008-04-30 00...
2	2	Tool Design	Research and ...	2008-04-30 00...
3	3	Sales	Sales and Mar...	2008-04-30 00...
4	4	Marketing	Sales and Mar...	2008-04-30 00...
5	5	Purchasing	Inventory Man...	2008-04-30 00...
6	6	Research and ...	Research and ...	2008-04-30 00...

The 'MESSAGES' pane at the bottom shows the execution status: 'Comenzó la ejecución de la consulta en Línea 1 (16 rows affected)' and 'Tiempo total de ejecución: 00:00:00.017'.

Problema: deja de tener continuidad en febrero de 2026

Visual Studio Code

The screenshot shows the Visual Studio Code interface with the Explorer pane on the left displaying a list of SQL files. The main editor shows the content of Test.sql, which contains a SQL query. The bottom pane displays the Query Results for the executed query.

EXPLORER

- ✓ **QUERIES**
 - > Evaluacion2020
 - 1-Indices.sql
 - 2-Agregates.sql
 - 3-Juntas.sql
 - 4-IntegridadReferencial.sql
 - 5-estadisticas.sql
 - 6-sargable.sql
 - ColumnStore.sql
 - MetaDatos.sql
 - Sorts.sql
 - SQLQuery_1.sql
 - SQLQuery_1a.sql
 - SQLQuery_2.sql
 - SQLQuery_3.sql
 - SQLQuery_3a.sql
 - SQLQuery_4.sql
 - Test.sql

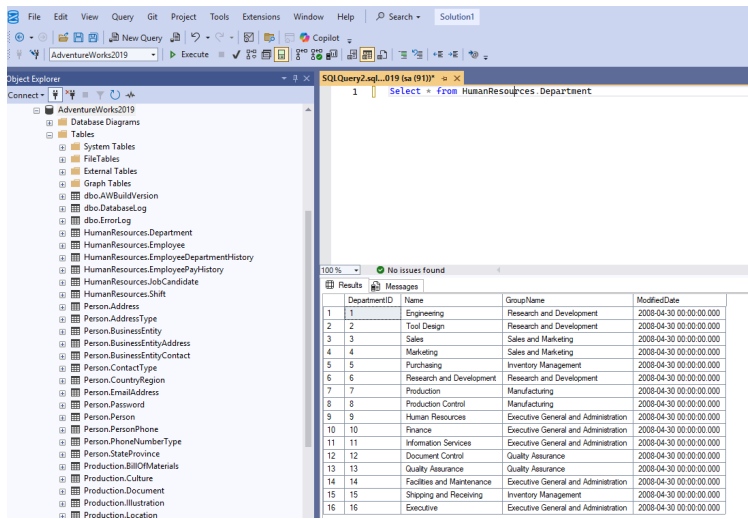
Test.sql

```
PCMINI | AdventureWorks2019
1 Select * from HumanResources.Department;
```

Query Results

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2008-04-30 00:00:00.000
2	2	Tool Design	Research and Development	2008-04-30 00:00:00.000
3	3	Sales	Sales and Marketing	2008-04-30 00:00:00.000
4	4	Marketing	Sales and Marketing	2008-04-30 00:00:00.000
5	5	Purchasing	Inventory Management	2008-04-30 00:00:00.000
6	6	Research a...	Research and Development	2008-04-30 00:00:00.000
7	7	Production	Manufacturing	2008-04-30 00:00:00.000
8	8	Production...	Manufacturing	2008-04-30 00:00:00.000
9	9	Human Reso...	Executive General and A...	2008-04-30 00:00:00.000
10	10	Finance	Executive General and A...	2008-04-30 00:00:00.000
11	11	Informatio...	Executive General and A...	2008-04-30 00:00:00.000
12	12	Document C...	Quality Assurance	2008-04-30 00:00:00.000
13	13	Quality As...	Quality Assurance	2008-04-30 00:00:00.000
14	14	Facilities...	Executive General and A...	2008-04-30 00:00:00.000
15	15	Shipping a...	Inventory Management	2008-04-30 00:00:00.000
16	16	Executive	Executive General and A...	2008-04-30 00:00:00.000

SSMS - Ideal si se trabaja en Windows



The screenshot displays the Microsoft SQL Server Enterprise Manager (SSMS) interface. The top menu bar includes File, Edit, View, Query, Git, Project, Tools, Extensions, Window, and Help. The main window is divided into three panes:

- Object Explorer:** Shows the database structure for 'AdventureWorks2019'. The 'Tables' folder is expanded, listing various tables such as 'System Tables', 'FileTables', 'External Tables', 'Graph Tables', and 'HumanResources.Department'.
- SQL Query Editor:** Contains a query titled 'SQLQuery2.sql...019 (sa (91))'. The query text is:

```
1 | Select * from HumanResources.Department
```
- Results:** Displays the execution results of the query. A status bar at the top indicates '100%' zoom and 'No issues found'. The results are shown in a table with columns: DepartmentID, Name, GroupName, and ModifiedDate.

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2008-04-30 00:00:00.000
2	2	Tool Design	Research and Development	2008-04-30 00:00:00.000
3	3	Sales	Sales and Marketing	2008-04-30 00:00:00.000
4	4	Marketing	Sales and Marketing	2008-04-30 00:00:00.000
5	5	Purchasing	Inventory Management	2008-04-30 00:00:00.000
6	6	Research and Development	Research and Development	2008-04-30 00:00:00.000
7	7	Production	Manufacturing	2008-04-30 00:00:00.000
8	8	Production Control	Manufacturing	2008-04-30 00:00:00.000
9	9	Human Resources	Executive General and Administration	2008-04-30 00:00:00.000
10	10	Finance	Executive General and Administration	2008-04-30 00:00:00.000
11	11	Information Services	Executive General and Administration	2008-04-30 00:00:00.000
12	12	Document Control	Quality Assurance	2008-04-30 00:00:00.000
13	13	Quality Assurance	Quality Assurance	2008-04-30 00:00:00.000
14	14	Facilities and Maintenance	Executive General and Administration	2008-04-30 00:00:00.000
15	15	Shipping and Receiving	Inventory Management	2008-04-30 00:00:00.000
16	16	Executive	Executive General and Administration	2008-04-30 00:00:00.000

Estructura de las Tablas e Indices

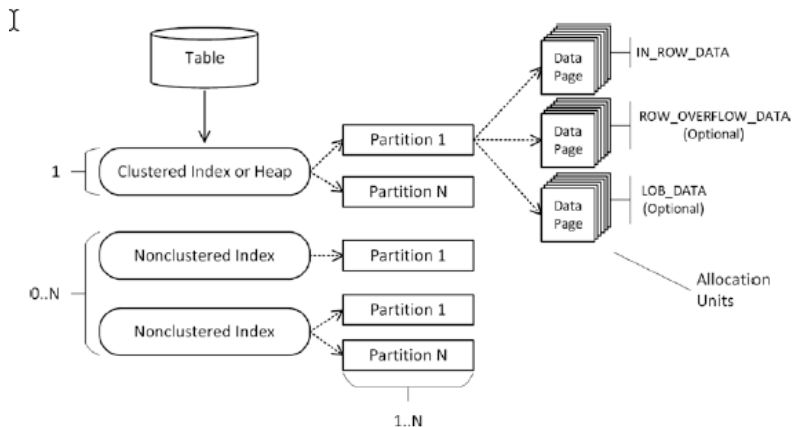
SQL Server almacena datos en tablas e índices.

- Heap Tables
- Clustered Indexes
- Nonclustered Indexes
- Columnstore
- In Memory

Heap vs Clustered

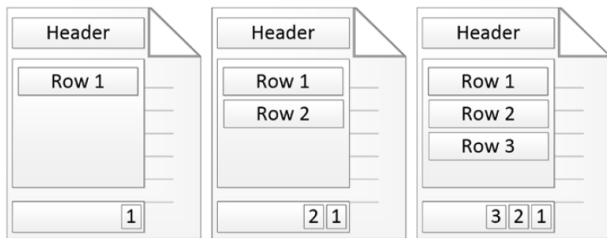
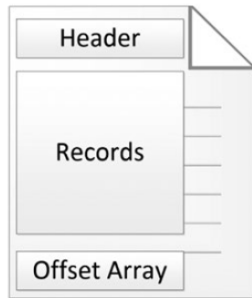
- Una tabla se puede organizar de una de dos maneras: como un *heap* o como un *árbol B* (B-Tree). Técnicamente, la tabla se organiza como un *árbol B* cuando tiene un *clustered index* definido y como un *heap* cuando no.
- Cuando agrega una restricción de *Primary Key* a una tabla, SQL Server la aplicará utilizando un *clustered index* a menos que especifique explícitamente la palabra clave *NONCLUSTERED* o ya exista un *clustered index* en la tabla.
- Cuando agrega una restricción única a una tabla, SQL Server la aplicará utilizando un *nonclustered index* a menos que especifique la palabra clave *CLUSTERED*

Vista general de una tabla

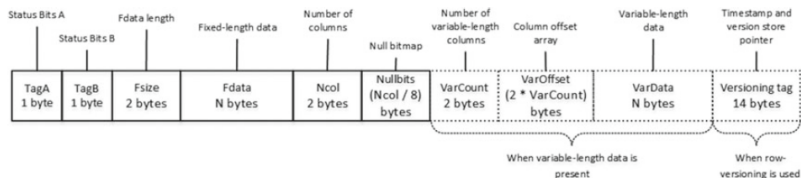


Páginas

- El área de almacenamiento más básica es una página.
- Cada página ocupa 8KB.
- Cuando SQL Server interactúa con los archivos de la base de datos, la unidad más pequeña en la que puede ocurrir una operación de E / S está en el nivel de página
- Las páginas se agrupan de ocho en ocho en estructuras llamadas extents.



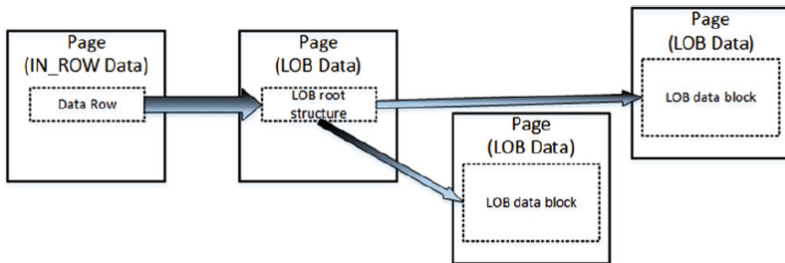
- SQL Server opera con páginas almacenadas en el *buffer pool*
- La interacción con las páginas ocurre principalmente dentro del *buffer pool*, no directamente en el disco.
- Cuando SQL Server lee una página, verifica si ya está en la caché de datos (*buffer pool*).
 - Si la página está en memoria, SQL Server realiza una lectura lógica (lee desde la memoria).
 - Si la página no está en memoria, SQL Server realiza una lectura física (extrae la página del disco a la memoria) seguida de una lectura lógica.
- Cuando SQL Server escribe una página, verifica si ya está en la caché de datos.
 - Si la página está en memoria, SQL Server realiza una escritura lógica.
 - El indicador *dirty* en el encabezado de la página indica que la página en memoria es más reciente que en el disco.



- Status Bits A y B contiene información sobre la fila: tipo de fila, si fue borrada lógicamente (ghosted), si tiene valores nulos, si tiene datos de longitud variable, etc.

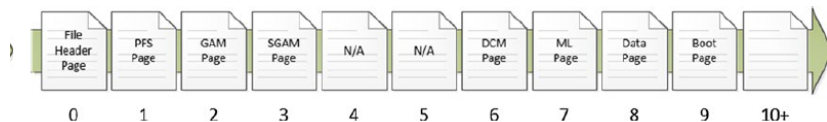
- **LOB Root Structure:** Al trabajar con tipos de datos grandes como `VARBINARY(MAX)`, `VARCHAR(MAX)`, `TEXT`, `IMAGE`, etc., se utiliza una estructura especial llamada **LOB root structure**. Esta estructura contiene punteros a diversas partes de los datos LOB.
- **Small LOB Data:** Si los datos LOB son relativamente pequeños (menos de 32 KB y caben en cinco páginas de datos), la *LOB root structure* puede hacer referencia directa a las páginas que contienen los fragmentos de datos LOB.
- **Large LOB Data:** Si los datos LOB son demasiado grandes, se introducen niveles intermedios adicionales para gestionar la jerarquía de almacenamiento. Esta estructura se asemeja a un **árbol B** (*B-Tree*), donde los nodos intermedios contienen punteros a nodos secundarios, y los nodos hoja contienen los fragmentos de datos reales.

LOB



Páginas

Diferentes tipos de páginas



- File header page. Contiene información de metadatos para el file en cuestión.
- Boot page. Idem File Header pero para toda la Base de Datos.
- Page Free Space (PFS)
 - Es la segunda página y después se ubica cada 8088 páginas.
 - Cada byte en la página PFS representa una página subsiguiente en el archivo de datos.
 - Uso del Espacio: Cuánto espacio libre hay en la página.
 - Estado de la Página: Indica si la página está llena, vacía o parcialmente llena.
 - Condiciones Especiales: Puede incluir indicadores para páginas que están en proceso de ser truncadas o que contienen datos LOB.

Páginas

Global Allocation Map (GAM)



GAM (Global Allocation Map) Indica qué *extents* están libres y disponibles para asignación.

- Cada página GAM cubre un intervalo de **64,000 extents** (≈ 4 GB).
- Cada bit representa un *extent* dentro del intervalo:
 - Bit = 0 \rightarrow extent libre.
 - Bit = 1 \rightarrow extent ocupado.

Páginas

Diferentes tipos de páginas



SGAM (Shared Global Allocation Map) Página casi idéntica a la GAM, pero indica si un *extent* es mixto.

- Bit = 1 → *extent* mixto con al menos una página libre.

DCM (Differential Changed Map) Cada bit representa un *extent* y su estado para backups diferenciales.

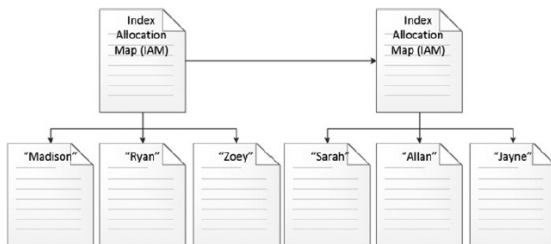
- Bit = 1 → al menos una página del *extent* cambió desde la última copia de seguridad diferencial.
- Bit = 0 → ninguna página cambió.

ML (Minimally Logged) Indica extensiones modificadas por operaciones de registro mínimo (bulk insert, select into, etc.).

- Aplica a *extents* dentro de intervalos GAM.

- SQL Server utiliza páginas especiales llamadas **mapas de asignación** para gestionar el uso de *extents* y páginas en los archivos de la base de datos.
- **Index Allocation Map (IAM):**
 - Cada página IAM representa una partición de un índice o tabla.
 - Su función es indicar qué *extents* pertenecen a esa partición.
 - Cada página IAM es un mapa de bits: - Bit = 1 → el *extent* pertenece a la partición. - Bit = 0 → el *extent* no pertenece a la partición.
- Cada página IAM cubre aproximadamente 64,000 *extents* (≈ 4 GB). Para archivos más grandes, varias páginas IAM se enlazan formando **cadena IAM**.
- Cada página IAM en la cadena cubre un intervalo GAM específico.

Heap Tables



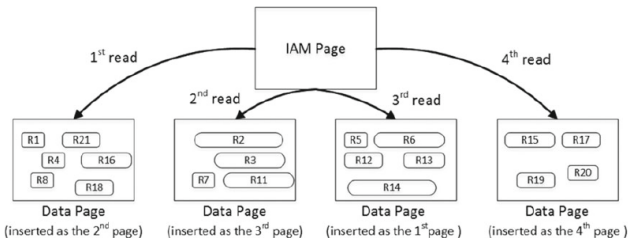
- Un *HEAP* está compuesto por una o más páginas de **Index Allocation Map (IAM)** que apuntan a las páginas de datos que conforman el *heap*.
- La primera página disponible de un *heap* es la que se encuentra primero en el archivo de datos correspondiente.

Selección de datos en una tabla heap

Uso del Mapa de Asignación de Índices (IAM)

Al seleccionar datos de una tabla *heap*, SQL Server utiliza el **Mapa de Asignación de Índices (IAM)** para localizar las páginas y *extents* que deben ser escaneados. El motor analiza qué *extents* pertenecen a la tabla y los procesa según su **orden de asignación**, no según el orden en que se insertaron los datos.

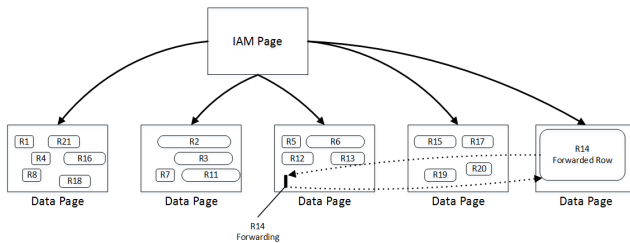
- **Orden de asignación:** el orden en que los *extents* se reservaron en el archivo de datos.
- **Orden de inserción:** el orden cronológico en que se insertaron los registros en la tabla.



Actualización de filas en una tabla heap

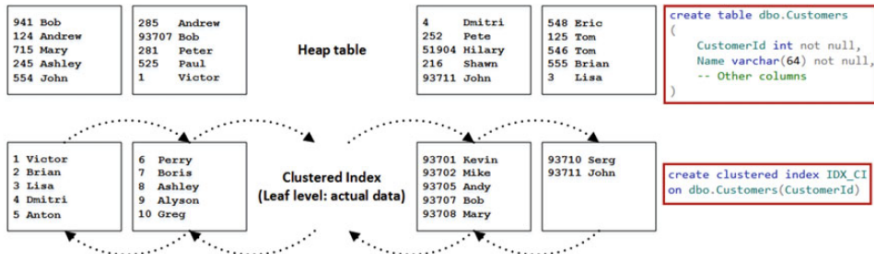
Manejo de actualizaciones de filas

Cuando se actualiza una fila en la tabla heap, SQL Server intenta acomodarla en la misma página. Si no hay espacio disponible, mueve la nueva versión de la fila a otra página y reemplaza la fila anterior con una fila especial de 16 bytes llamada **puntero de redireccionamiento**. La nueva versión de la fila se denomina **fila redireccionada**.

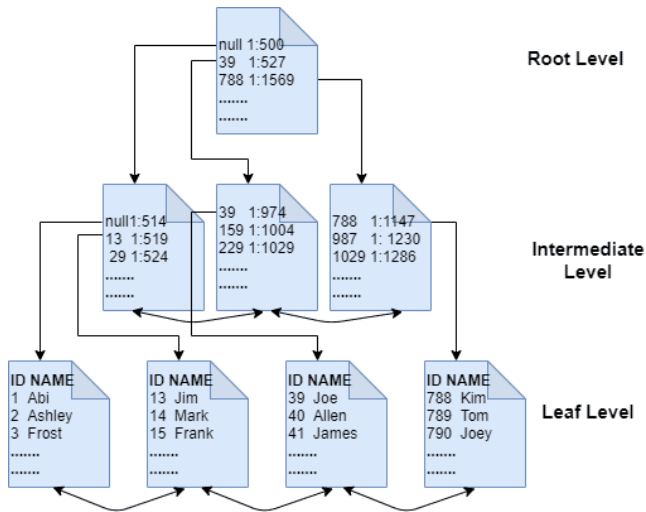


Clustered Indexes

- 1 Un *clustered index* dicta el **orden físico** de los datos en una tabla
- 2 Una tabla puede tener **sólo un *clustered index*** definido

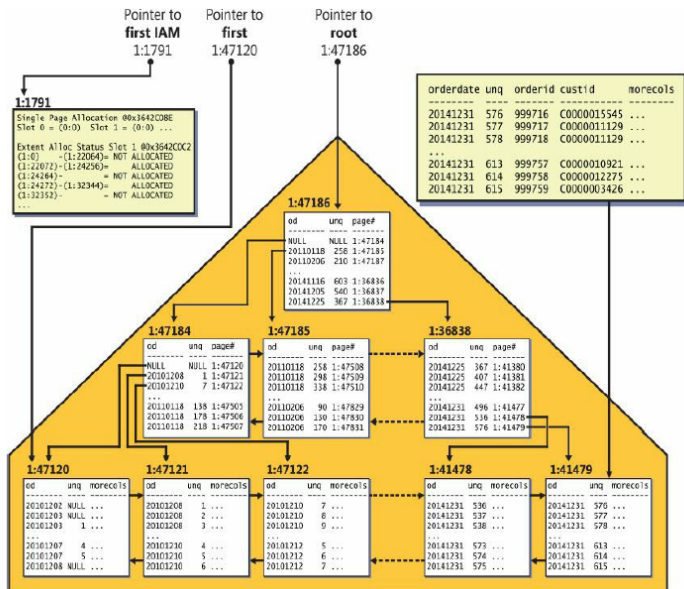


Árbol B



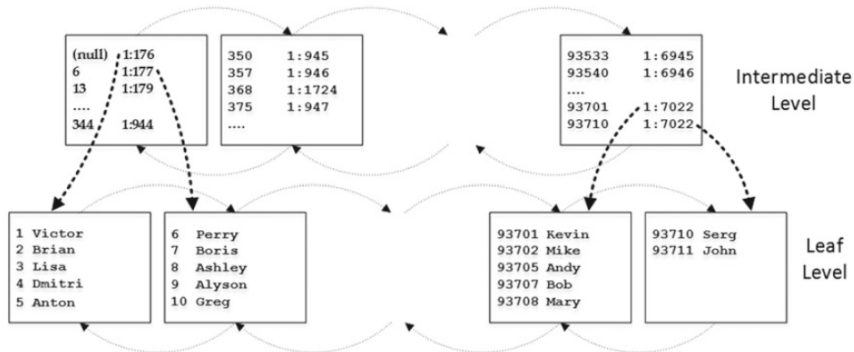
B-Tree Clustered Index on ID

Clustered Indexes

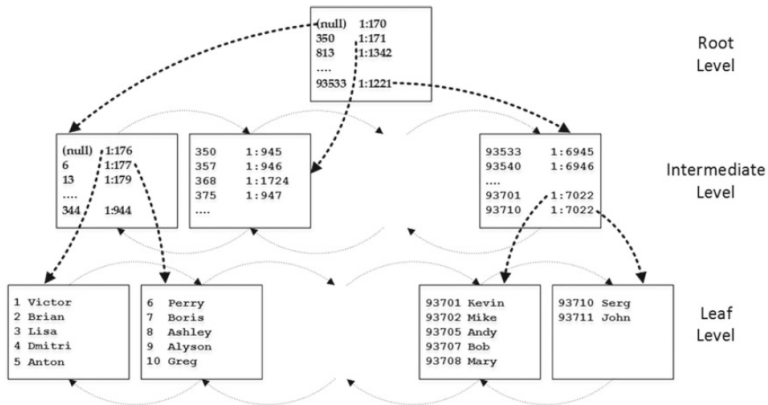


Clustered Indexes

- 1 El nivel intermedio almacena una fila por cada página del nivel de hoja.
- 2 Almacena: dirección física de la página y el valor mínimo de la clave del índice de la hoja referenciada, con excepción del primera fila de la primer página donde almacena NULL (optimización para insertar una fila con clave mas baja en la tabla)



Clustered Indexes



Siempre hay un nivel hoja, cero o mas niveles intermedios y un nivel raíz. Excepto cuando la tabla entra en una única página, en este caso solo hay una página con los datos.

Non Clustered Indexes

- ① *Non clustered* indexes definen un orden que es almacenado en una estructura separada de los datos.
- ② Es similar al clustered index en su estructura, pero las páginas del nivel de hoja incluyen el valor de la clave y un *rowid*
- ③ El ***rowid***
 - Para *heap tables* el rowid representa la locación física de la página
 - Para tablas con *clustered index* representa el *clustered index key* de la fila.
- ④ Los nodos intermedios almacenan la dirección física del página y el valor mínimo de la clave.

NonClustered Indexes

```
SELECT * FROM dbo.Customers WHERE Name = 'Boris'
```

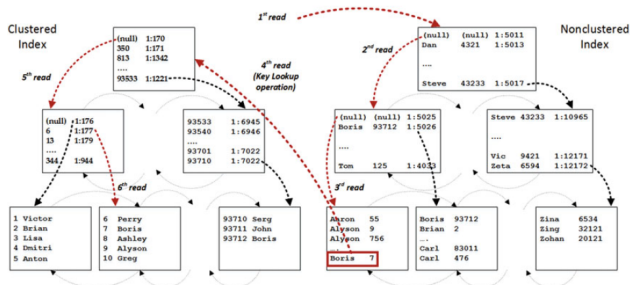


Figura: 1er Paso

NonClustered Indexes

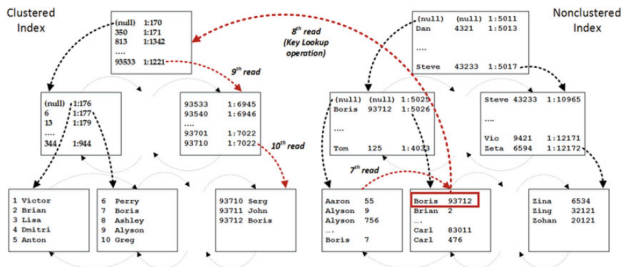
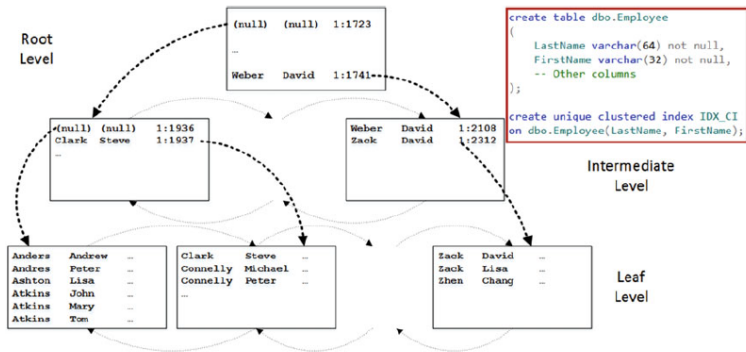
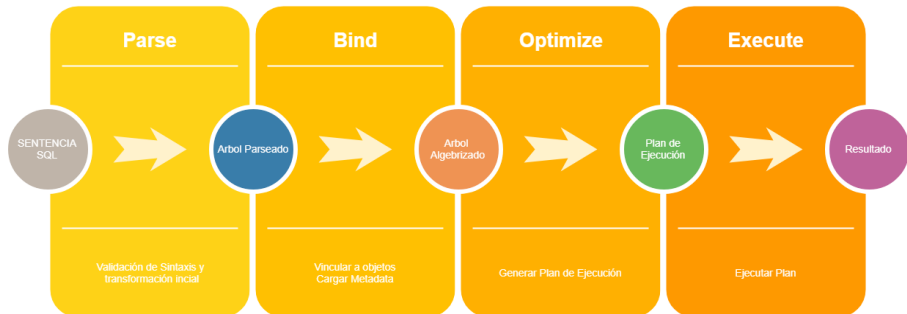


Figura: 2do Paso

Composite Indexes



El procesamiento de consultas



Estadísticas


```
SET STATISTICS TIME ON;
```

```
SELECT soh.AccountNumber, sod.LineTotal,  
sod.OrderQty, sod.UnitPrice, p.Name  
FROM Sales.SalesOrderHeader soh  
JOIN Sales.SalesOrderDetail sod  
ON soh.SalesOrderID = sod.SalesOrderID  
JOIN Production.Product p  
ON sod.ProductID = p.ProductID  
WHERE sod.LineTotal > 1000 ;
```

```
SET STATISTICS TIME OFF;
```

Liberar cache:

```
DBCC FREEPROCCACHE
```

```
SET STATISTICS IO ON;
```

```
SELECT soh.AccountNumber, sod.LineTotal,  
sod.OrderQty, sod.UnitPrice, p.Name  
FROM Sales.SalesOrderHeader soh  
JOIN Sales.SalesOrderDetail sod  
ON soh.SalesOrderID = sod.SalesOrderID  
JOIN Production.Product p  
ON sod.ProductID = p.ProductID  
WHERE sod.LineTotal > 1000 ;
```

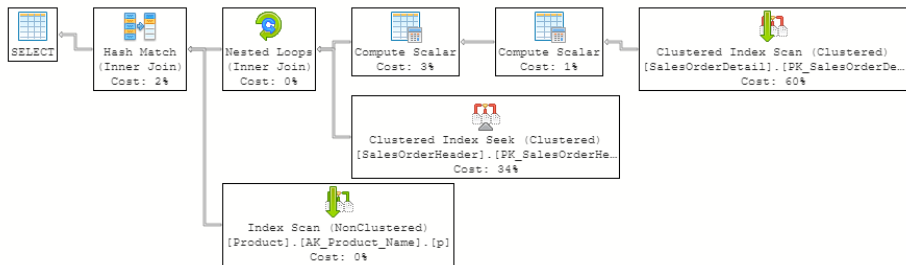
```
SET STATISTICS IO OFF;
```

Liberar cache:

```
DBCC dropcleanbuffers  
DBCC freeproccache
```

Planes de Ejecución

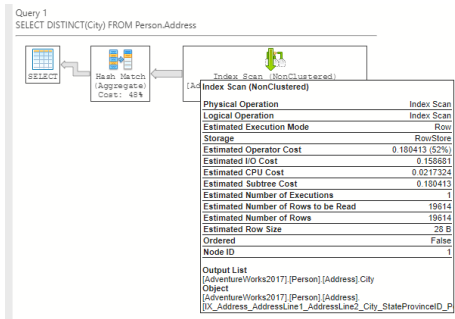
Ejecución de Consultas



Ejecución de Consultas

Cada nodo en un plan de ejecución implementa al menos tres métodos

- Open(): se inicializa el operador y se configuran las estructuras de datos requeridas
- GetRow() : Requiere una fila del operador
- Close() : finaliza el operador limpiando estructuras y datos que sean necesarios.



Ejecución de Consultas: Secuencial vs Paralela

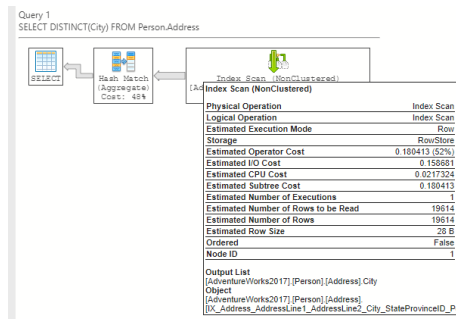
Cada nodo en un plan de ejecución implementa al menos tres métodos:

- **Open():** inicializa el operador y configura estructuras de datos.
- **GetRow():** devuelve una fila del operador.
- **Close():** finaliza el operador y libera recursos.






Ejecución secuencial: cada operador se ejecuta en un solo hilo, siguiendo un flujo padre-hijo.

Ejecución paralela:

- SQL Server puede usar varios hilos para un mismo operador o fragmentos del plan.
- Se introducen operadores **Exchange** para distribuir (Distribute/Repartition) y recolectar filas (Gather) entre hilos.
- Cada hilo ejecuta su propia instancia de *Open/GetRow/Close*, de manera concurrente y thread-safe.



Operadores de Acceso a Datos

Estructura	Scan	Seek
Heap	 Table Scan	
Clustered Index	 Clustered Index Scan	 Clustered Index Seek
Nonclustered Index	 Index Scan	 Index Seek

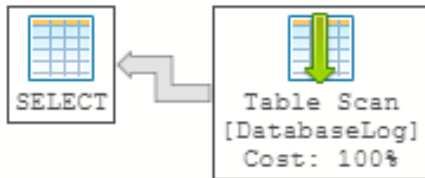
bookmark lookup: RID lookup or Keylookup





Table Scan

Query 1
`SELECT * FROM DatabaseLog`



La tabla Databaselog no tiene índice clustered.



Cluster Index Scan

- Unordered Clustered Index Scan
- Ordered Clustered Index Scan

```
1 select * from Sales.SalesOrderHeader|
```

Results Messages Query Plan (Preview) Top Operations

Try 1: Query cost (relative to the script): 100.00%
select * from Sales.SalesOrderHeader





PROPERTIES	
Clustered Index Scan (Clustered)[SalesOrderHeader].[PK_SalesOrderHeader_Sal...	
NAME	VALUE
Column	PurchaseOrderNumber
Database	[AdventureWorks2019]
Schema	[Sales]
Table	[SalesOrderHeader]
[9]	[AdventureWorks2019].[Sales].[SalesOrder...
Column	AccountNumber
Database	[AdventureWorks2019]
Schema	[Sales]
Table	[SalesOrderHeader]
Forced Index	False
Ordered	False
Parallel	False

Comparemos el uso de índices

- ❶ `SELECT * FROM Person.Address ORDER BY AddressID;`
- ❷ `SELECT AddressID, City, StateProvinceID FROM Person.
Address ORDER by AddressID;`
- ❸ `SELECT AddressID, City, StateProvinceID FROM Person.
Address;`
- ❹ `SELECT AddressID, City, StateProvinceID FROM Person.
Address ORDER by City;`

Operadores de Agregación

SQL Server utiliza dos operadores:

- 1  Stream aggregate
- 2  Hash aggregate

Se usan para implementar agregados (SUM, AVG, MAX, etc) y para GROUP BY y DISTINCT.

Para algunos operadores se requiere que los datos vengan ordenados, el *Query Optimizer* puede usar un índice existente o puede introducir un *Sort Operator* explícitamente.

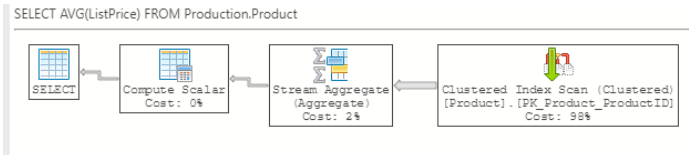
Tanto el hashing como el sort usan memoria, pero si no encuentran espacio suficiente pueden usar *tempdb database*.



Stream Aggregate

Consultas que usan un agregado y no tienen una clausula GROUP BY (***scalar aggregates***) siempre se implementan con Stream Aggregate. En caso de usar GROUP BY los datos deben venir ordenados.

```
SELECT AVG(ListPrice) FROM Production.Product
```

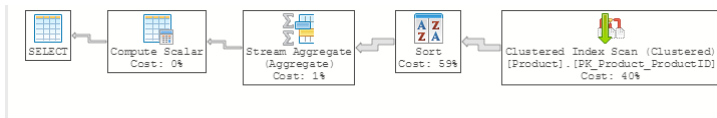


```
--Compute Scalar(DEFINE:([Expr1002]=CASE WHEN [Expr1003]=(0) THEN NULL ELSE [Expr1004]/CONVERT_IMPLICIT(money,[Expr1003],0) END  
--Stream Aggregate(DEFINE:([Expr1003]=Count(*), [Expr1004]=SUM([AdventureWorks2017].[Production].[Product].[ListPrice])))  
--Clustered Index Scan(OBJECT:([AdventureWorks2017].[Production].[Product].[PK_Product_ProductID]))
```



Stream Aggregate

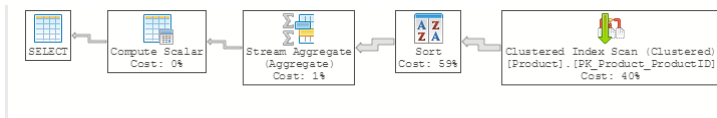
```
SELECT ProductLine, COUNT(*) FROM Production.Product  
GROUP BY ProductLine
```



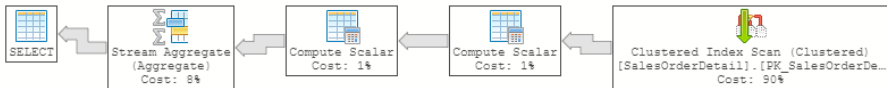


Stream Aggregate

```
SELECT ProductLine, COUNT(*) FROM Production.Product  
GROUP BY ProductLine
```



```
SELECT SalesOrderID, SUM(LineTotal) FROM Sales.  
SalesOrderDetail GROUP BY SalesOrderID
```





Hash Aggregate

Hash Match

El Hash Aggregate cómo el Hash Join se implementa con el operador físico

Hash Match



Hash Aggregate

Hash Match

El Hash Aggregate cómo el Hash Join se implementa con el operador físico

Hash Match

```
SELECT City, COUNT(City) AS CityCount
FROM   Person.Address
GROUP BY City
```

El optimizador de consultas puede seleccionar un *Hash Aggregate* para tablas grandes donde los datos no están ordenados, *no es necesario* ordenarlos y su cardinalidad se estima en solo unos pocos grupos.



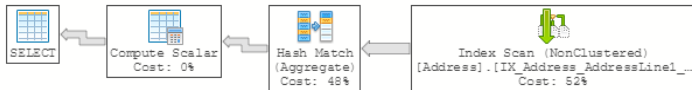
Hash Aggregate

Hash Match

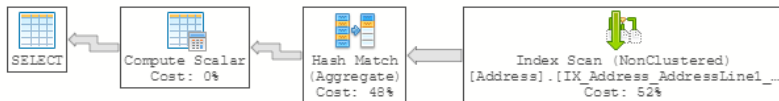
El Hash Aggregate cómo el Hash Join se implementa con el operador físico **Hash Match**

```
SELECT City, COUNT(City) AS CityCount
FROM Person.Address
GROUP BY City
```

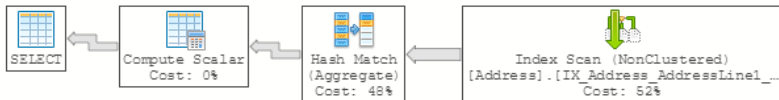
El optimizador de consultas puede seleccionar un *Hash Aggregate* para tablas grandes donde los datos no están ordenados, *no es necesario* ordenarlos y su cardinalidad se estima en solo unos pocos grupos.



Hash Keys Build



Hash Keys Build



```
<HashKeysBuild>  
  <ColumnReference Database="[AdventureWorks2017]" Schema="[Person]" Table="[Address]" Column="City">  
  </ColumnReference>  
</HashKeysBuild>
```

Hash vs Stream Aggregate

- *Hash Aggregate* ayuda cuando los datos no están ordenados. Si se crea un índice que pueda proporcionar datos ordenados, entonces el optimizador de consultas puede seleccionar un *Stream Aggregate*.
- Si la entrada no está ordenada pero se pide el orden explícitamente en una consulta, el optimizador de consultas puede introducir un *Sort operator* y un *Stream Aggregate* o puede decidir usar un *Hash Aggregate* y luego ordenar los resultados

Una consulta que usa la palabra clave **DISTINCT** puede ser implementada por *Stream Aggregate*, *Hash Aggregate* o por un operador *Distinct Sort*.

Distinct Sort

Una consulta que usa la palabra clave **DISTINCT** puede ser implementada por *Stream Aggregate*, *Hash Aggregate* o por un operador *Distinct Sort*.

```
SELECT DISTINCT(JobTitle) FROM HumanResources.Employee;
```

```
SELECT JobTitle FROM HumanResources.Employee  
GROUP BY JobTitle;
```

Query 1

```
SELECT JobTitle FROM HumanResources.Employee GROUP BY JobTitle
```



Juntas

El optimizador de consultas debe tomar dos decisiones importantes con respecto a las juntas:

- la selección de un orden de las juntas
- la selección del algoritmo de junta



- Nested Loops Join



- Merge Join

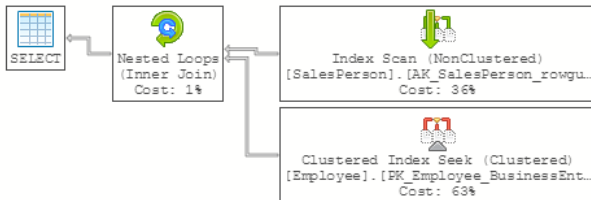


- Hash Join



Nested Loop Join

```
SELECT e.BusinessEntityID FROM HumanResources.Employee AS e
      INNER JOIN Sales.SalesPerson AS s ON
      e.BusinessEntityID = s.BusinessEntityID
```



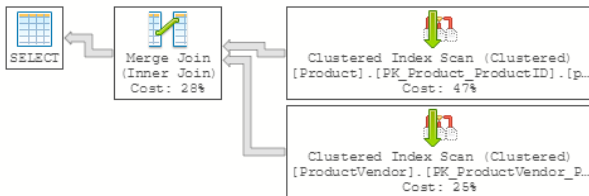
Es muy efectivo si el *outer input* es suficientemente pequeña y el *inner input* es grande pero indexada.

Merge Join y Hash Join requieren que la junta tenga al menos un predicado de junta basado sobre un operador de igualdad (ej. equi join)



Merge Join

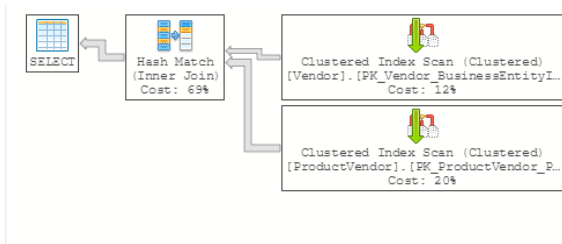
```
SELECT p.ProductID, v.BusinessEntityID
FROM Production.Product AS p
      JOIN Purchasing.ProductVendor AS v
      ON (p.ProductID = v.ProductID);
```





Hash Join

```
SELECT pv.ProductID, v.BusinessEntityID, v.Name
FROM Purchasing.ProductVendor pv JOIN Purchasing.Vendor v
ON (pv.BusinessEntityID = v.BusinessEntityID)
WHERE StandardPrice > 10
```



Hash Join

El optimizador lo usa para procesar entradas largas, no ordenadas, no indexadas eficientemente.

JOIN Resumen

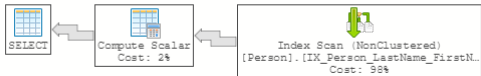
Join Type	Index on Joining Columns	Usual Size of Joining Tables	Presorted	Join Clause
Hash	Inner table: Not indexed Outer table: Optional Optimal condition: Small outer table, large inner table	Any	No	Equi-join
Merge	Both tables: Must Optimal condition: Clustered or covering index on both	Large	Yes	Equi-join
Nested loop	Inner table: Must Outer table: Preferable	Small	Optional	All



Compute Scalar

```
SELECT p.LastName + ', ' + p.FirstName  
FROM Person.Person as p
```

SELECT p.LastName + ', ' + p.FirstName FROM Person.Person as p



Results		Top Operations	Query Plan
4 RESULTS			
	XML Showplan		
1	<ShowPlanXML xmlns="http:...		



Compute Scalar

```
<RelOp NodeId="0" PhysicalOp="Compute Scalar" LogicalOp="Compute Scalar" EstimateRows="19972" EstimateIO="0" EstimateCPU="0.0019972" AvgRowSize="113" EstimatedTotalSubtreeCost="0.104285" Parallel="0">
  <OutputList>
    <ColumnReference Column="Expr1001"></ColumnReference>
  </OutputList>
  <ComputeScalar>
    <DefinedValues>
      <DefinedValue>
        <ColumnReference Column="Expr1001"></ColumnReference>
        <ScalarOperator ScalarString="[AdventureWorks2017].[Person].[Person].[LastName] as [p].[LastName]+N&apos;, &apos;+[AdventureWorks2017].[Person].[Person].[FirstName] as [p].[FirstName]">
          <Arithmetic Operation="ADD">
            <ScalarOperator>
              <Arithmetic Operation="ADD">
                <ScalarOperator>
                  <Identifier>
                    <ColumnReference Database="[AdventureWorks2017]" Schema="[Person]" Table="[Person]" Alias="[p]" Column="LastName"></ColumnReference>
                  </Identifier>
                </ScalarOperator>
                <ScalarOperator>
                  <Const ConstValue="N&apos;, &apos;"></Const>
                </ScalarOperator>
              </Arithmetic>
            </ScalarOperator>
            <ScalarOperator>
              <Identifier>
                <ColumnReference Database="[AdventureWorks2017]" Schema="[Person]" Table="[Person]" Alias="[p]" Column="FirstName"></ColumnReference>
              </Identifier>
            </ScalarOperator>
          </Arithmetic>
        </ScalarOperator>
      </DefinedValue>
    </DefinedValues>
  </ComputeScalar>
</RelOp>
```



Compute Scalar

```
SELECT COUNT(*) cRows  
FROM HumanResources.Shift;
```





Compute Scalar

```
SELECT COUNT(*) cRows  
FROM HumanResources.Shift;
```



```
SELECT COUNT_BIG(*) cRows  
FROM HumanResources.Shift;
```





Filter

```
SELECT City, COUNT(City) AS CityCount
FROM Person.Address
GROUP BY City
HAVING COUNT(City) > 1
```



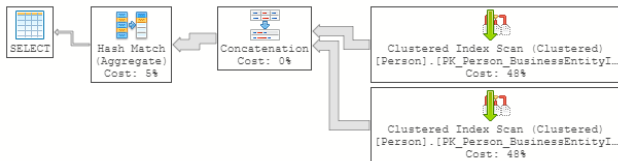


UNION - Concat

OPERADORES

Hay varios operadores que el optimizador utiliza para resolver la unión: merge, hash, concat

```
select Suffix from Person.Person
UNION
select Suffix from Person.Person
```



¿Que es más costoso UNION o UNION ALL?

Integridad Referencial

```
SELECT a.AddressID, sp.StateProvinceID
FROM Person.Address AS a
    JOIN Person.StateProvince AS sp
    ON a.StateProvinceID = sp.StateProvinceID
WHERE a.AddressID = 27234;
```

```
SELECT a.AddressID, a.StateProvinceID
FROM Person.Address AS a
    JOIN Person.StateProvince AS sp
    ON a.StateProvinceID = sp.StateProvinceID
WHERE a.AddressID = 27234;
```

```
ALTER TABLE Person.Address
DROP CONSTRAINT FK_Address_StateProvince_StateProvinceID;
```

```
ALTER TABLE Person.Address WITH CHECK
ADD CONSTRAINT FK_Address_StateProvince_StateProvinceID
    FOREIGN KEY(StateProvinceID)
REFERENCES Person.StateProvince (StateProvinceID);
```

```
SELECT ...  
OPTION (<hint>,<hint>...)
```

- **HASH GROUP / ORDER GROUP** – Fuerza el método de agregación (hash o sort) para GROUP BY.
- **MERGE — HASH — CONCAT UNION** – Indica cómo ejecutar un UNION (merge, hash o concatenación).
- **LOOP — MERGE — HASH JOIN** – Fuerza el tipo de join entre tablas.
- **FAST N** – Optimiza la consulta para devolver las primeras N filas más rápido.
- **FORCE ORDER** – Obliga a SQL Server a respetar el orden de joins especificado en la query.
- **RECOMPILE** – Fuerza recompilación del plan de ejecución en cada ejecución.
- **FROM TableName WITH (INDEX ([IndexName]))** – Fuerza el uso de un índice específico en una tabla.

Parameter Sniffing / OPTIMIZE FOR

```
SELECT * FROM Person.Address  
WHERE City = 'Mentor';
```

```
SELECT * FROM Person.Address  
WHERE City = 'London';
```

Parameter Sniffing / OPTIMIZE FOR

```
SELECT * FROM Person.Address  
WHERE City = 'Mentor';
```

```
SELECT * FROM Person.Address  
WHERE City = 'London';
```

```
DECLARE @City NVARCHAR(30)  
SET @City = 'Mentor'  
SELECT * FROM Person.Address  
WHERE City = @City
```

```
SET @City = 'London'  
SELECT * FROM Person.Address  
WHERE City = @City;
```

```
OPTION( OPTIMIZE FOR (@City = 'Mentor') )
```

SARGable / search-ARGument-able

Comparar

```
SELECT sod.*  
FROM Sales.SalesOrderDetail AS sod  
WHERE sod.SalesOrderID IN ( 51825, 51826, 51827, 51828 );
```

```
SELECT sod.*  
FROM Sales.SalesOrderDetail AS sod  
WHERE sod.SalesOrderID BETWEEN 51825 AND 51828 ;
```

```
SELECT sod.*  
FROM Sales.SalesOrderDetail AS sod  
WHERE sod.SalesOrderID = 51825  
      OR sod.SalesOrderID = 51826  
      OR sod.SalesOrderID = 51827  
      OR sod.SalesOrderID = 51828;
```

Started executing query at Line 1

(150 rows affected)

Table 'SalesOrderDetail'. Scan count 4, logical reads 18, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

(150 rows affected)

Table 'SalesOrderDetail'. Scan count 1, logical reads 6, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Total execution time: 00:00:00.021

Composite Indexes: SARGable

SARGable predicates	Non-SARGable predicates
<code>LastName = 'Clark' and FirstName = 'Steve'</code>	<code>LastName <> 'Clark' and FirstName = 'Steve'</code>
<code>LastName = 'Clark' and FirstName <> 'Steve'</code>	<code>LastName LIKE '%ar%' and FirstName = 'Steve'</code>
<code>LastName = 'Clark'</code>	<code>FirstName = 'Steve'</code>
<code>LastName LIKE 'Cl%'</code>	

La *SARGability* de un índice compuesto depende de la *SARGability* del predicado sobre la primer columna del índice.

```
SELECT *  
FROM Purchasing.PurchaseOrderHeader AS poh  
WHERE poh.PurchaseOrderID * 2 = 3400;
```

```
SELECT *  
FROM Purchasing.PurchaseOrderHeader AS poh  
WHERE poh.PurchaseOrderID * 2 = 3400;
```

```
SELECT *  
FROM Purchasing.PurchaseOrderHeader AS poh  
WHERE poh.PurchaseOrderID = 3400/2;
```

```
SELECT a.PostalCode  
FROM Person.Address AS a  
WHERE a.StateProvinceID = 42;
```

```
CREATE NONCLUSTERED INDEX IX_Address_StateProvinceID  
ON Person.Address(StateProvinceID ASC)  
INCLUDE(PostalCode)  
WITH (DROP_EXISTING = ON);
```

Indice con filtros

```
SELECT soh.PurchaseOrderNumber ,  
       soh.OrderDate ,  
       soh.ShipDate ,  
       soh.SalesPersonID  
FROM Sales.SalesOrderHeader AS soh  
WHERE PurchaseOrderNumber LIKE 'P05%'  
AND soh.SalesPersonID IS NOT NULL;
```

```
CREATE NONCLUSTERED INDEX IX_Test  
ON Sales.SalesOrderHeader (PurchaseOrderNumber ,SalesPersonID)  
INCLUDE (OrderDate ,ShipDate)
```

```
CREATE NONCLUSTERED INDEX IX_Test  
ON Sales.SalesOrderHeader (PurchaseOrderNumber ,SalesPersonID)  
INCLUDE (OrderDate ,ShipDate)  
WHERE PurchaseOrderNumber IS NOT NULL  
AND SalesPersonID IS NOT NULL  
WITH (DROP_EXISTING = ON);
```

Almacenamiento físico: SQL Server vs PostgreSQL

SQL Server

- Usa **Heap** (sin índice clusterizado) o **Clustered Index** (árbol B+).
- El índice clusterizado define el orden físico de las filas.
- Páginas de 8 KB, organizadas en extents (64 KB).

PostgreSQL

- Cada tabla es siempre un **heap** (orden de inserción).
- No existe índice clusterizado nativo.
- Soporta CLUSTER para reordenar físicamente según un índice, pero no se mantiene automáticamente.
- Páginas de 8 KB por defecto.

Índices: SQL Server vs PostgreSQL

SQL Server

- B+Tree (clustered y non-clustered).
- Columnstore (orientado a columnas).
- XML Index, Full-text Index.
- Spatial Index.

PostgreSQL

- B-Tree (por defecto).
- Hash Index.
- GiST (para datos geométricos y búsqueda avanzada).
- SP-GiST (para particionamiento).
- GIN (texto, JSON, arrays).
- BRIN (grandes tablas con correlación).

- Grant Fritchey. **SQL Server 2017 Query Performance Tuning Troubleshoot and Optimize Query Performance** (Fifth Edition). Apress, Berkely, CA, USA.
- Jason Strate and Grant Fritchey. 2015. **Expert Performance Indexing in SQL Server** (2nd ed.). Apress, Berkely, CA, USA.
- Benjamin Nevarez **Inside the SQL Server Query Optimizer**
- Grant Fritchey 2012 **SQL Server Execution Plans** -Second Edition Simple Talk Publishing September
- Dimitri Korotkevich **Pro SQL Server Internals** -Second Edition. Apress, Berkely, CA, USA.