Caso 3 Preliminar 3 Bases de Datos I

José Eduardo Gutiérrez Conejo – 2019073558

Emanuel Rodríguez Oviedo – 2022108678

Para crear las vistas se utilizará la tabla de container_inventories para realizar la prueba. Esta tabla debe hacer join con las tablas de container_types, producers, wastes y traductions, para verificar el tipo de container que se está utilizando, el tipo de desecho que maneja y el productor que posee el contenedor.

Una vista es un subconjunto de una base de datos y se basa en una consulta que se ejecuta en una o más tablas de la base de datos. Las vistas de la base de datos se guardan en la base de datos como consultas con nombre y se pueden usar para guardar consultas complejas de uso frecuente.

Una vista dinámica es una declaración de un SELECT, por lo que involucra un proceso de agregado, calculo, y filtrado sobre las tablas subyacentes cada vez que se ejecuta una consulta sobre la vista, lo que hace que esta operación sea bastante costosa tanto en lógica como en lectura.

Por otro lado, una vista indexada, al crear un índice agrupado sobre la vista, los datos de las tablas subyacentes son procesado automáticamente y guardado en el índice, lo que evita que se tenga que revisar todos los datos sobre las tablas subyacentes y solo se tenga que revisar los datos en el índice, lo que reduce de gran manera la cantidad de lecturas que deben realizarse, y a su vez aumentando la eficiencia del SELECT. A pesar de todo esto las vistas indexadas tienen un problema, los índices que actúan sobre la vista deben de tener actualizaciones cada vez que cambia la información de las tablas sobre las que actúa, por lo que si se desea utilizar una vista con índice es mejor analizar cuantas operaciones se van a realizar sobre las tablas que están asociadas ya que podría ser más costoso mantener este tipo de vista.

Para nuestro experimento se crearán dos vistas, una indexada y una dinámica, ambas trabajarán con los mismos campos y tablas (las mencionadas anteriormente), y se medirá el tiempo de ejecución que dura el motor en realizar un SELECT * sobre las vistas.

Realizando la prueba sobre la ejecución de una vista dinámica contra una indexada se obtiene lo siguiente:

Tabla 1: Comparación del tiempo de ejecución de una vista dinámica vs. una indexada

Tiempo de ejecución de vista dinámica (s)	Tiempo de ejecución de vista indexada (s)
1.615	1.222
1.327	1.372
1.646	1.263
1.362	1.516
1.345	1.271

Al obtener el promedio de los datos se obtiene que el tiempo de ejecución promedio de la consulta de la vista dinámica es de 1.459 s y el de la vista indexada es de 1.3288, con esto se puede ver que en promedio hay una mejoría de 0.1302 segundos o 8.924%. Al tiempo de la vista indexada se le debe agregar el tiempo de ejecución de la creación del índice el cual fue de 1.052 segundos cada vez que se realice sobre las tablas que trabaja alguna modificación.

A pesar de que la vista indexada presenta una mejoría del tiempo esto solo se da si las tablas sobre las que trabaja no son modificadas constantemente, ya que el mantenimiento del índice es bastante caro y en nuestro caso si se suma el tiempo de ejecución del índice más el tiempo de ejecución de la consulta este más bien representaría que una vista indexada es menos eficiente que una dinámica.

El plan de ejecución en SQL Server es un conjunto de información que describe cómo se ejecutará una consulta en la base de datos. Este plan incluye información detallada sobre cómo SQL Server optimiza y ejecuta una consulta específica, incluyendo qué índices y operaciones utilizará para acceder a los datos necesarios.

El plan de ejecución se genera automáticamente por SQL Server cada vez que se ejecuta una consulta. El plan se almacena en caché y se reutiliza para consultas similares en el futuro, lo que ayuda a mejorar el rendimiento de las consultas y reduce el tiempo de respuesta.

En el caso de nuestra base de data, se genera una consulta con todos los parámetros comunes en consultas de bases de datos (fields, joins, left/right join, aggregate functions, except/intersect, group by, sort, for json, wheres sobre campos primary y non primary, igualdades y desigualdades). Dicha consulta se empleará para retornar todos los empleados con su usuario que pertenezcan a una productora, donde también se mostraran la cantidad de empleados que trabajan para esa misma productora, los inventarios que manejan según la corporación con el estado de todos sus contenedores e inclusive la posición y nombre de usuario de cada empleado. Esta consulta corresponde en código a:

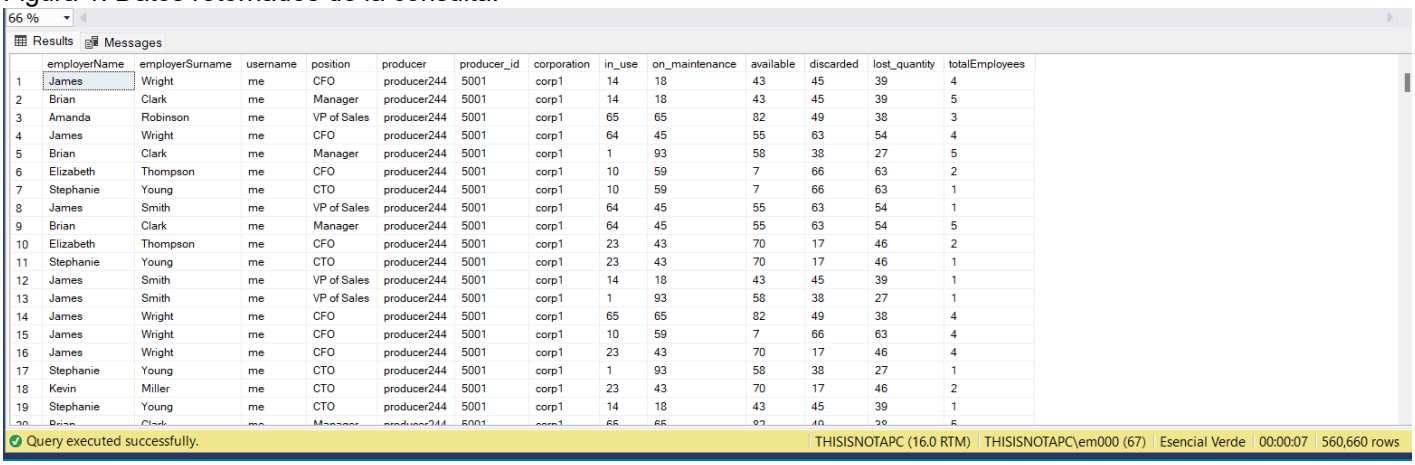
```
SELECT
    usrs.name AS employerName,
    usrs.last_name AS employerSurname,
    usrs.username,
    usrs.position,
    prd.name AS producer,
prd.producer_id,
    crp.name AS corporation,
    cont_inv.in_use,
    cont_inv.on_maintenance,
    cont_inv.available,
    cont_inv.discarded,
    cont_inv.lost_quantity,
    COUNT(usrs.user_id) AS totalEmployees
FROM
    producers prd
    LEFT JOIN users usrs ON usrs.producer_id = prd.producer_id
    RIGHT JOIN containers_inventories cont_inv ON cont_inv.producer_id = prd.producer_id
```

```
    JOIN corporations crp ON prd.corporation_id = crp.corporation_id
WHERE
    usrs.position NOT IN ('CEO', 'Director of HR')
    AND usrs.created_at BETWEEN '2023-02-01' AND '2023-12-01'

GROUP BY
    crp.name,
crp.corporation_id,
    usrs.name,
    usrs.last_name,
    usrs.username,
    usrs.position,
    prd.name,
prd.producer_id,
    cont_inv.in_use,
    cont_inv.on_maintenance,
    cont_inv.available,
    cont_inv.discarded,
    cont_inv.lost_quantity
EXCEPT
SELECT
    usrs.name AS employerName,
    usrs.last_name AS employerSurname,
    usrs.username,
    usrs.position,
    prd.name AS producer,
prd.producer_id,
    crp.name AS corporation,
    cont_inv.in_use,
    cont_inv.on_maintenance,
    cont_inv.available,
    cont_inv.discarded,
    cont_inv.lost_quantity,
```

```
    COUNT(usrs.user_id) AS totalEmployees
FROM
    producers prd
    LEFT JOIN users usrs ON usrs.producer_id = prd.producer_id
    RIGHT JOIN containers_inventories cont_inv ON cont_inv.producer_id = prd.producer_id
    JOIN corporations crp ON prd.corporation_id = crp.corporation_id
WHERE
usrs.created_at BETWEEN '2023-05-01' AND '2023-05-15'
GROUP BY
    crp.name,
crp.corporation_id,
    usrs.name,
    usrs.last_name,
    usrs.username,
    usrs.position,
    prd.name,
prd.producer_id,
    cont_inv.in_use,
    cont_inv.on_maintenance,
    cont_inv.available,
    cont_inv.discarded,
    cont_inv.lost_quantity
ORDER BY
prd.producer_id;
```

Para esta primera consulta se puede observar su tiempo de ejecución en la siguiente imagen y en la tabla de tiempos:

Figura 1: Datos retornados de la consulta.



66 %

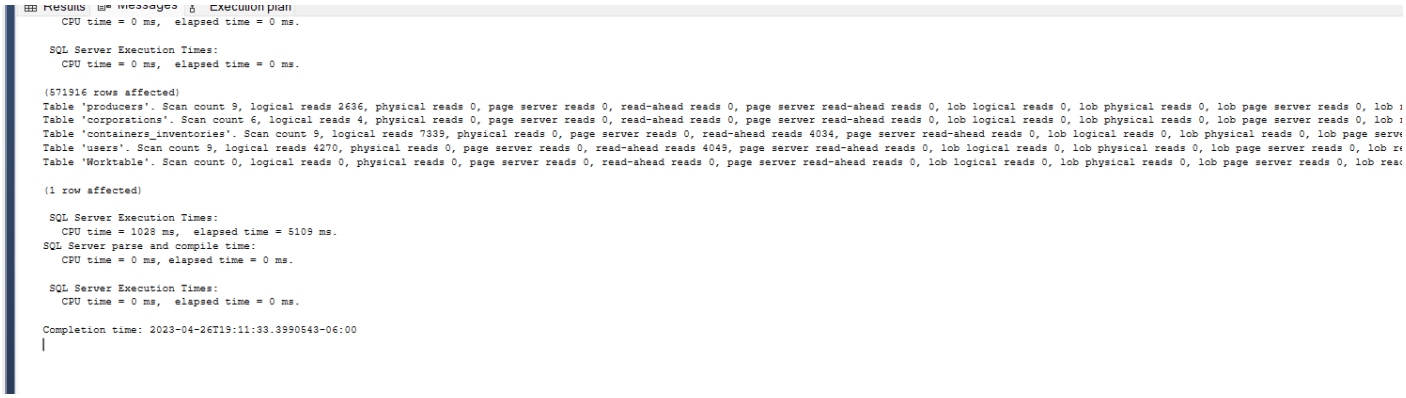
Results | Messages

	employerName	employerSurname	username	position	producer	producer_id	corporation	in_use	on_maintenance	available	discarded	lost_quantity	totalEmployees
1	James	Wright	me	CFO	producer244	5001	corp1	14	18	43	45	39	4
2	Brian	Clark	me	Manager	producer244	5001	corp1	14	18	43	45	39	5
3	Amanda	Robinson	me	VP of Sales	producer244	5001	corp1	65	65	82	49	38	3
4	James	Wright	me	CFO	producer244	5001	corp1	64	45	55	63	54	4
5	Brian	Clark	me	Manager	producer244	5001	corp1	1	93	58	38	27	5
6	Elizabeth	Thompson	me	CFO	producer244	5001	corp1	10	59	7	66	63	2
7	Stephanie	Young	me	CTO	producer244	5001	corp1	10	59	7	66	63	1
8	James	Smith	me	VP of Sales	producer244	5001	corp1	64	45	55	63	54	1
9	Brian	Clark	me	Manager	producer244	5001	corp1	64	45	55	63	54	5
10	Elizabeth	Thompson	me	CFO	producer244	5001	corp1	23	43	70	17	46	2
11	Stephanie	Young	me	CTO	producer244	5001	corp1	23	43	70	17	46	1
12	James	Smith	me	VP of Sales	producer244	5001	corp1	14	18	43	45	39	1
13	James	Smith	me	VP of Sales	producer244	5001	corp1	1	93	58	38	27	1
14	James	Wright	me	CFO	producer244	5001	corp1	65	65	82	49	38	4
15	James	Wright	me	CFO	producer244	5001	corp1	10	59	7	66	63	4
16	James	Wright	me	CFO	producer244	5001	corp1	23	43	70	17	46	4
17	Stephanie	Young	me	CTO	producer244	5001	corp1	1	93	58	38	27	1
18	Kevin	Miller	me	CTO	producer244	5001	corp1	23	43	70	17	46	2
19	Stephanie	Young	me	CTO	producer244	5001	corp1	14	18	43	45	39	1

Query executed successfully. | THISISNOTAPC (16.0 RTM) | THISISNOTAPC\em000 (67) | Esencial Verde | 00:00:07 | 560,660 rows

En la tabla y en la siguiente imagen es posible observar el tiempo estimado de ejecución para este query sin ninguna mejoría en performance:

Figura 2: Tiempo de ejecución de la consulta.

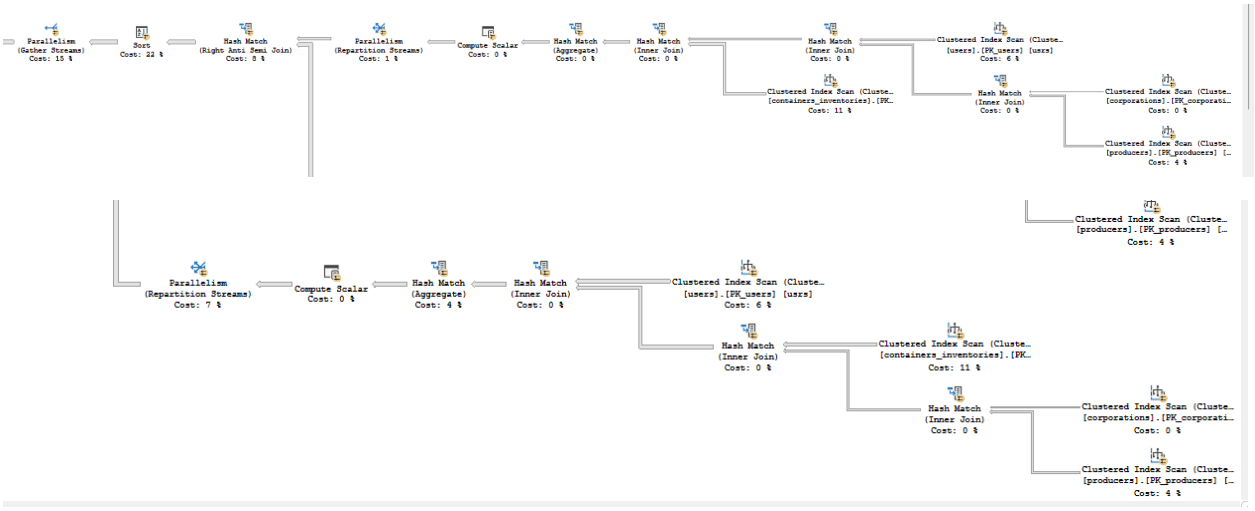


Results | Messages | Execution plan

```
   CPU time = 0 ms,  elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

(571916 rows affected)
Table 'producers'. Scan count 9, logical reads 2636, 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
Table 'corporations'. Scan count 6, logical reads 4, 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
Table 'containers_inventories'. Scan count 9, logical reads 7339, physical reads 0, page server reads 0, read-ahead reads 4034, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page serve
Table 'users'. Scan count 9, logical reads 4270, physical reads 0, page server reads 0, read-ahead reads 4049, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob re
Table 'Worktable'. Scan count 0, logical reads 0, 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 reac

(1 row affected)

 SQL Server Execution Times:
   CPU time = 1028 ms,  elapsed time = 5109 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

Completion time: 2023-04-26T19:11:33.3990543-06:00
```

Ahora, observamos el árbol del plan de ejecución estimado:

Figura 3: Árbol de plan de ejecución de la consulta.



Ahora podemos observar en la siguiente tabla varias unidades de workload que aumentan en gran medida el tiempo de ejecución, ante las cuales se proponen normas que ayuden a mejorar el rendimiento de futuros queries.

Tabla 2: Explicación de las operaciones y propuestas de norma.

Unidad de Workload	Explicación	Norma
Clustered Index Scan (Clustered) Scanning a clustered index, entirely or only a range. **Physical Operation** Clustered Index Scan **Logical Operation** Clustered Index Scan **Estimated Execution Mode** Batch **Storage** RowStore **Estimated Operator Cost** 5.22168 (11%) **Estimated I/O Cost** 5.15572 **Estimated Subtree Cost** 5.22168 **Estimated CPU Cost** 0.0659596 **Estimated Number of Executions** 1 **Estimated Number of Rows to be Read** 480000 **Estimated Number of Rows for All Executions** 48000 **Estimated Number of Rows Per Execution** 48000 **Estimated Row Size** 31 B **Ordered** False **Node ID** 14 **Predicate** PROBE([Opt_Bitmap1068],[Esencial Verde].[dbo].[containers_inventories].producer_id) as [cont_inv].[producer_id]) **Object** Esencial Verde].[dbo].[containers_inventories].PK_containers_inventories] [cont_inv] **Output List** Esencial Verde].[dbo].[containers_inventories].producer_id, [Esencial Verde].[dbo].[containers_inventories].in_use, [Esencial Verde].[dbo].containers_inventories].on_maintenance, [Esencial Verde].[dbo].containers_inventories].available, [Esencial Verde].[dbo].containers_inventories].discarded, [Esencial Verde].[dbo].containers_inventories].lost_quantity	Se puede observar que se están extrayendo todos los fk correspondientes a producer_id para poder hacerle probe y encontrar los usuarios con corporaciones y producers unidos que les corresponda ese inventario de contenedores.	Crear un non-clustered index cuando se observe que existe un clustered index scan sobre un fk en un join, mientras la cantidad de lectura de filas sea lo suficientemente considerable. Esto termina usando un hash join con S3b de algoritmo para encontrar la unión de los elementos anteriormente mencionados.

Parallelism Repartition streams. **Physical Operation** Parallelism **Logical Operation** Repartition Streams **Estimated Execution Mode** Row **Estimated I/O Cost** 0 **Estimated Operator Cost** 3.37336 (9%) **Estimated CPU Cost** 3.37123 **Estimated Subtree Cost** 12.1302 **Estimated Number of Executions** 1 **Estimated Number of Rows Per Execution** 1281480 **Estimated Number of Rows for All Executions** 1281480 **Estimated Row Size** 102 B **Partitioning Type** Hash **Node ID** 19 **Output List** [Esencial Verde].[dbo].[producers].producer_id, [Esencial Verde].[dbo].[producers].name, [Esencial Verde].[dbo].[users].name, [Esencial Verde].[dbo].[users].last_name, [Esencial Verde].[dbo].[users].position, [Esencial Verde].[dbo].[users].username, [Esencial Verde].[dbo].[containers_inventories].in_use, [Esencial Verde].[dbo].[containers_inventories].on_maintenance, [Esencial Verde].[dbo].[containers_inventories].available, [Esencial Verde].[dbo].[containers_inventories].discarded, [Esencial... **Partition Columns** [Esencial Verde].[dbo].[users].name, [Esencial Verde].[dbo].[users].last_name, [Esencial Verde].[dbo].[users].username, [Esencial Verde].[dbo].[users].position, [Esencial Verde].[dbo].[producers].name, [Esencial Verde].[dbo].[producers].producer_id, [Esencial Verde].[dbo].[corporations].name, [Esencial Verde].[dbo].[containers_inventories].in_use, [Esencial Verde].[dbo].[containers_inventories].on_maintenance, [Esencial Verde].[dbo].[containers_inventories].available, [Esencial Verde].[dbo].[...	El uso de un except ramifica el plan de ejecución el cual repite el proceso de selección y unión de registros excepto en un factor pequeño que se usa para descartar ciertos registros en el select, en este caso se trata de los usuarios creados entre '2023-05-01' y '2023-05-15'	Tratar en lo posible de usar WHERE NOT EXISTS en vez de usar excepts pues producen una repetición en la ejecución de los algoritmos del select que alarga en gran medida el tiempo de ejecución del query.
Hash Match Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows. **Physical Operation** Hash Match **Logical Operation** Aggregate **Estimated Execution Mode** Batch **Estimated Operator Cost** 1.5024 (5%) **Estimated I/O Cost** 0 **Estimated Subtree Cost** 14.9502 **Estimated CPU Cost** 1.50249 **Estimated Number of Executions** 1 **Estimated Number of Rows Per Execution** 937658 **Estimated Number of Rows for All Executions** 937658 **Estimated Row Size** 102 B **Node ID** 3 **Output List** [Esencial Verde].[dbo].[producers].producer_id, [Esencial Verde].[dbo].[producers].name, [Esencial Verde].[dbo].[users].name, [Esencial Verde].[dbo].[users].last_name, [Esencial Verde].[dbo].[users].position, [Esencial Verde].[dbo].[users].username, [Esencial Verde].[dbo].[containers_inventories].in_use, [Esencial Verde].[dbo].[containers_inventories].on_maintenance, [Esencial Verde].[dbo].[containers_inventories].available, [Esencial Verde].[dbo].[containers_inventories].discarded, [Esencial... **Build Residual** [Esencial Verde].[dbo].[users].[name] as [usrs].[name] = [Esencial Verde].[dbo].[users].[name] as [usrs].[name] AND [Esencial Verde].[dbo].[users].[last_name] as [usrs].[last_name] = [Esencial Verde].[dbo].[users].[last_name] as [usrs].[last_name] AND [Esencial Verde].[dbo].[users].[username] as [usrs].[username] = [Esencial Verde].[dbo].[users].[username] as [usrs].[username] AND [Esencial Verde].[dbo].[users].[position] as [usrs].[position] = [Esencial Verde].[dbo].[users].[position] as [us...	El hash match presentado hace una unión de todos los datos pertinentes del producer, relacionado a los datos unidos previamente del user, junto con los datos sacados del inventario de contenedores, este hash match toma los resultados acumulados de left joins y right joins y los une con dicho hash, sin embargo el no utilizar un inner join termina afectando el rendimiento del query, pues este restringe la devolución de filas a solo las que coinciden en ambas tablas.	Usar inner joins cuando el propósito del uso de joins termine siendo equivalente para los left, right and inner joins, pues estos pueden terminar restringiendo el acceso de datos con tablas que no coinciden y mejorar el rendimiento final del query. Este utiliza entonces un algoritmo J2

Tabla 3: Tiempos de ejecución según la versión de la consulta.

Version del query	Cantidad de registros retornados	Tiempo de Ejecución	Cambio Realizado:
1	560660	6977 ms	Ninguno
2	560660	6702 ms	Añadir clustered index sobre el container_inventories incluyendo el producer_id y múltiples columnas de cantidades disponibles y en distintos estados.
3	212929	3889 ms.	Utilizar un where not exist en lugar de un except que genera un gran peso al tener que repetir las operaciones de la consulta
4	212929	3379 ms	Usar inner joins en vez de aplicar left y right joins cuando el resultante sea equivalente a nivel de selección de registros.
CTE	212929	3293 ms.	Implementar una simplificación por medio del uso de CTE en el select.

En conclusión, podemos observar que por medio de la aplicación de las normas anteriormente nombradas se logra obtener una diferencia entre el tiempo de ejecución de la consulta de 3684 ms, lo cual corresponde a un total de la consulta optimizada de 3293 en su tiempo de ejecución en contraste a los 6977 de la consulta inicial sin ninguna norma aplicada. Por lo tanto, se puede afirmar que estas normas logran de manera efectiva aumentar la eficiencia de queries en el contexto de la base de datos de Esencial Verde.