Queries auxiliares

```
SELECT
    object_name(cols.object_id) tabla
    ,cols.name columna
    ,ind.name indice
    ,ind.type_desc tipo
    ,ind.is_unique
    FROM
    sys.columns cols, sys.indexes ind , sys.index_columns ind_cols
    where
    cols.object_id = ind.object_id
    and cols.object_id = ind_cols.object_id
    and cols.column_id = ind_cols.column_id
    and ind.index_id = ind_cols.index_id
    and object_name(cols.object_id) LIKE 'Employee'
    order by object_name(cols.object_id), ind.name;
```

```
SELECT TABLE_NAME, COLUMN_NAME, IS_NULLABLE, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'WorkOrder';
```

“If a column is unique, then it will have the highest possible selectivity, and the selectivity degrades as the level of uniqueness decreases. A column such as "gender," for example, will likely have a low selectivity”

Convenciones de nomenclatura de índice de SQL Server:

- PK_ Primary Key, clustered
- AK_ unclestered, unique
- IX_ unclestered, no unique

--consulta 1

```
select NationalIDNumber, HireDate from HumanResources.Employee
where NationalIDNumber='121491555';

select NationalIDNumber, BusinessEntityID from HumanResources.Employee
where NationalIDNumber= '121491555';
```

- NationalIdNumber tiene un índice unclustered. Como el índice non-clustered no contiene toda la información de cada registro tiene que hacer un key-lookup usando el clustered para poder traer el HireDate. Hace dos lecturas: primero sobre el índice para resolver el where y después usa el operador key lookup para traerse el HireDate.

- En el segundo caso va directo a la hoja y trae los datos. No necesita hacer un key-lookup **porque la hoja además de contener el índice non-clustered, contiene el índice clustered que también se pide.**

--consulta 2

```
select NationalIDNumber, BusinessEntityID from HumanResources.Employee
where NationalIDNumber= '121491555';

select NationalIDNumber, BusinessEntityID from HumanResources.Employee
where NationalIDNumber= 121491555;
```

- En la primera se busca por un string, en la segunda por un entero.

- NationalIdNumber es un varchar. Como contraejemplo, si el string que se guarda es el ‘000123’, no lo va a obtener convirtiendo el número 123 a string.

- En la primera query hace uso del index seek dado que puede aprovecharlo.

- En la segunda query, al tener que hacer la conversión de integer a string, el motor no puede hacer uso directo del seek, así que debe escanear los índices hasta encontrar el que matchea con el valor convertido.

--consulta 3

```
select count(UnitPrice) from sales.SalesOrderDetail;

select count(CarrierTrackingNumber) from sales.SalesOrderDetail;
```

- La columna es nulleable en CarrierTrackingNumber y en UnitPrice no. Para la que no es nulleable el count va a contar cuántos registros tiene la tabla.

- Usa el índice unclustered porque es más chico.

- El motor tiene la información de que UnitPrice no puede ser null, por lo tanto. Con este dato elige hacer un index scan por el non-clustered ProductID. Al ser la cantidad de UnitPrice igual a la cantidad de ProductID, basta con contar cualquiera para obtener la cantidad de registros de la tabla.

- Esto no pasa al contar CarrierTrackingNumber porque algunos de ellos pueden ser null (con lo cual suman 0 en el COUNT). Tiene que hacer un clustered index scan, y revisar si CarrierTrackingNumber no es NULL

--consulta 4

```
select p.ProductNumber from Sales.SpecialOffer so
join Sales.SpecialOfferProduct sop on so.SpecialOfferID = sop.SpecialOfferID
join Production.Product p on sop.ProductID = p.ProductID

select * from Sales.SpecialOffer so
join Sales.SpecialOfferProduct sop on so.SpecialOfferID = sop.SpecialOfferID
join Production.Product p on sop.ProductID = p.ProductID
```

- En la primera consulta se busca una sola columna del join y en la segunda todas.

- INTEGRIDAD REFERENCIAL: el motor tiene los datos de cómo están vinculados los datos entre sí. Usa la integridad referencial como una herramienta.

- La primera consulta devuelve los ProductNumber que surgen del matcheo entre SpecialOfferProduct y Product, a través de ProductID. Pero SpecialOffer no tiene vínculo con Product. Por lo tanto, no agrega nada que se haga ese join.

- SpecialOfferID es la clave de SpecialOffer. Por lo tanto, los SpecialOfferID que tenga la tabla SpecialOfferProduct seguro van a estar en SpecialOffer.

- En SpecialOfferProduct hay dos claves foráneas: una que “viaja” desde SpeccialOffer y la otra desde Product.

- En la primer query hay un join de más. El motor interpreta que alcanza con escanear solamente SpecialOfferProduct porque tiene las columnas SpeciallOfferID y ProductID. Esto puede hacerlo gracias a la integridad referencial de la base. Aprovecha el índice non-clustered de ProductID y hace un scan con el mismo, que devuelve los resultados ordenados. Luego hace uso del índice clustered de Product para scanear dicha tabla por el ProductID que también vuelve ordenado. Como ambos scans devuelven resultados ordenados, usa el Merge Join.

- En el segundo caso no puede hacer lo mismo porque se piden más datos en el SELECT. Por lo pronto es necesario escanear las tablas completamente. El primer join (Nested) es posible hacerlo rápido porque luego del clustered index scan (sobre SpecialOffer), dispone de los índices también presentes en SpecialOffer product para poder hacer clustered index seek.

- Se finaliza con un hash match dado que los datos no vienen ordenados y tienen un volumen alto.

--consulta 5

```
select AddressID, City, StateProvinceID, ModifiedDate
from  Person.Address
where StateProvinceID = 32

select AddressID, City, StateProvinceID, ModifiedDate
from Person.Address
where StateProvinceID = 20
```

- Cambia la cantidad de registros a devolver. Hay pocos registros en la StateProvince 32. Puede usar el índice.
- En la primera query elige hacer un index scan y un key-lookup dado que la cantidad de resultados con StateProvinceID = 32 es escasa. El motor tiene ese dato porque almacena la demografía de los datos.

- En el segundo caso, al ser mucho mayor la cantidad de matches de StateProvinceID = 20, el motor decide hacer un scan por índice clustered.

- Ejecutando las consultas se obtiene que la cantidad de registros para StateProvinceID = 32 es 1 mientras que las que cumplen la segunda condición son 308.

--consulta 6

```
Select AddressLine1, AddressLine2, City from Person.Address
        where AddressLine1 like '1%';

Select AddressLine1, AddressLine2, City from Person.Address
        where AddressLine1 NOT like '1%';

Select AddressLine1, AddressLine2, City, ModifiedDate from Person.Address
        where AddressLine1 like '1%';
```

- En la primera query se están pidiendo 3 columnas que pertenecen a un índice unclustered, además como es un índice de este tipo entonces está ordenado por los valores de estas columnas y puedo rápidamente buscar un valor de AddressLine1 (que es una de estas columnas) entonces puedo hacer un seek en este índice para encontrar aquellos registros cuyo AddressLine1 empiece con 1 fácilmente.
- Ahora para hacer lo mismo, pero para aquellos registros que no empiecen con 1 no es tan fácil. Debería buscar de un modo similar para todo el resto de los dígitos posibles. Así que el optimizador resuelve que es mejor directamente hacer un scan en todo este índice unclustered y ver uno por uno que NO empiece con 1 para quedárselo
- Luego en la tercera Query se pide una cuarta columna que no se encuentra en el índice unclustered. Es decir, no existe ningún índice unclustered que tenga estas 4 columnas por lo que debo pasar sí o sí por el índice clustered. Ahora en este índice que es más grande, por más que me pidan los registros cuyo AddressLine1 empiece con 1, como ahora este índice está ordenado por la clave primaria (que no incluye a AddressLine1) entonces lo mejor es hacer un scan de todo el índice clustered y quedarme con aquellos que cumplen la restricción del where.

--consulta 7

```
select count(EndDate) from Production.WorkOrder

select count(OrderQty) from Production.WorkOrder
```

- Parecido (pero no igual) a la consulta 3.

- EndDate en WorkOrder puede ser NULL con lo cual contar las filas no alcanza (porque NULL es ignorado por COUNT) entonces hay que recorrer todas filas e inspeccionar el valor de EndDate para saber si contarlo o no. Tiene que escanear todo el índice.

- OrderQty no es nulleable pero es un int. Hacer un COUNT sobre OrderQty es equivalente a hacerlo sobre la cantidad de filas total dado que OrderQty no puede ser NULL. El motor hace un index scan sobre ScrapReasonID en lugar de ProductID. La razón es porque ScrapReasonID es un smallint con lo cual entran más en una página entonces necesita menos páginas para scanear todos. Entonces usa un índice unclustered que tiene con un smallint, que es el IX_WorkOrder_ScrapReasonID

--consulta 8

```
Dada la siguiente consulta:

select e.* from HumanResources.Employee as e where e.Gender = 'F'


Ejecutarla y ver el plan de ejecución,

Luego crear el siguiente índice:

CREATE INDEX IX_Employee_Test ON HumanResources.Employee (Gender)

--WITH (DROP_EXISTING = ON) ;

Volver a ejecutarla. ¿Qué ocurrió con el índice?
```

- Cuando se crea el índice la consulta queda igual. El problema es la selectividad del índice. La selectividad de Gender es baja por lo cual no conviene usar un índice.

--Consulta 9

```
select soh.* from Sales.SalesOrderHeader soh
join Sales.SalesOrderDetail sod
    on soh.SalesOrderID = sod.SalesOrderID
where soh.SalesOrderID = 71832 ;

select soh.* from Sales.SalesOrderHeader soh
join Sales.SalesOrderDetail sod
    on soh.SalesOrderID = sod.SalesOrderID
```

- La PK de SalesOrderDetail es compuesta, pero la primera parte es SalesOrderID. Usa el seek clustered y un nested loop join para unir todo. Al tener una búsqueda el motor elige hacerla primero para que el join de ambas tablas no sea grande. Para encontrar dicha fila hace uso del clustered index seek que es la mejor situación posible. Luego puede hacer uso del mismo algoritmo en SalesOrderDetail porque SalesOrderID es (la primera) parte de la primary key y tiene un índice clustered por ella.

- En la segunda puede usar un merge porque la condición del join es por igualdad de PK, y entonces los registros vienen ordenados. No hace búsqueda por WHERE pero sí puede sacar ventaja de que la condición de join es por igualdad de primary key, lo cual hace que sus scans sean por índice clustered y estén ordenados, con lo cual se puede usar eficazmente el Merge Sort

--consulta 10

```
select distinct(CardType) from Sales.CreditCard;

select distinct(CardNumber) from Sales.CreditCard;
```

- En la segunda consulta usa el índice correspondiente.

- En el primer caso tiene la información que son pocos los tipos de tarjetas. Por eso usa un Hash Match.

- En la primera query tiene una gran cantidad de filas a leer (19118) pero al buscar el tipo de tarjeta los posibles valores tienen una cantidad muy reducida en comparación. El optimizador decide hacer un clustered index scan sobre la tabla y usar un hash match (para agregación) dado que hash match es eficiente para tablas grandes donde los datos no están ordenados y su cardinalidad se estima en pocos grupos.

- En el segundo caso, está haciendo distinct sobre un valor que es unique. El optimizador aprovecha el índice non-clustered sobre CardNumber para hacer un index scan y devolver ese valor. No hace filtro ni agregación porque tiene la información de que todos los valores serán distintos