Consultas

Orden sintáctico de la cláusula SELECT

```
SELECT select_list
      [INTO new_table]
      FROM table
      [WHERE search_condition]
      [GROUP BY group_by_condition]
      [HAVING search_condition]
      [ORDER BY order_expression [ASC|DESC]]
```

FROM

```
USE sample;
SELECT dept_no, dept_name, location
FROM department
```

WHERE

```
USE sample
SELECT dept_name, dept_no
   FROM department
   WHERE location = 'Dallas'

USE sample
SELECT emp_lname, emp_fname
   FROM employee
   WHERE emp_no >= 15000

USE sample
SELECT project_name
   FROM project
   WHERE budget*0.51 > 60000

USE sample
SELECT project_no, emp_no
   FROM works_on
   WHERE project_no = 'p1'
   OR project_no = 'p2'
```

Operadores booleanos (Paréntesis, AND, OR y NOT)

```
USE sample
SELECT emp_no, emp_fname, emp_lname
   FROM employee
   WHERE emp_no = 25348 AND emp_lname = 'Smith'
   OR emp_fname = 'Matthew' AND dept_no = 'd1'
```

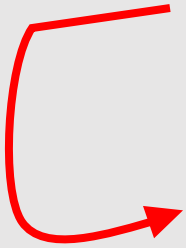
Se ejecuta 1ero el paréntesis más interno, luego el externo y luego el AND

```
USE sample
SELECT emp_no, emp_fname, emp_lname
   FROM employee
   WHERE ((emp_no = 25348 AND emp_lname = 'Smith')
   OR emp_fname ='Matthew') AND dept_no = 'd1'
```

```
USE sample
SELECT emp_no, emp_lname
   FROM employee
   WHERE NOT dept_no = 'd2'
```

Operador DISTINCT

NOTA:
Tenga en cuenta que la opción DISTINCT sólo puede utilizarse una vez en una lista SELECT, y debe preceder a todos los nombres de columna de esa lista. Por lo tanto, lo siguiente es incorrecto.



```
USE sample
SELECT emp_fname, DISTINCT emp_no
FROM employee
WHERE emp_lname = 'Moser';
```

Forma correcta

```
USE sample
SELECT DISTINCT emp_no, emp_fname
FROM employee
WHERE emp_lname = 'Moser';
```

NOTA:
Cuando hay más de una columna en la lista SELECT, la cláusula DISTINCT muestra todas las filas en las que la combinación de columnas es distinta.

Operadores IN y BETWEEN

```
USE sample
SELECT emp_no, emp_fname, emp_lname
FROM employee
WHERE emp_no IN (29346, 28559, 25348)
```

Traer a todos los clientes de Sales.IndividualCustomer cuyo apellido sea Lopez, Martin o Wood, y sus nombres empiecen con cualquier letra entre la C y la L.

```
USE AdventureWorks2019
SELECT *
FROM Sales.vIndividualCustomer
WHERE LastName IN ('Lopez','Martin','Wood')
AND LastName LIKE '[C-L]%'
```

```
USE sample
SELECT emp_no, emp_fname, emp_lname, dept_no
FROM employee
WHERE emp_no NOT IN (10102, 9031)
```

BETWEEN toma los valores 95000 y 120000, si no se quisiera tomarlos en cuenta tendríamos que utilizar los operadores > y <

```
USE sample
SELECT project_name, budget
FROM project
WHERE budget BETWEEN 95000 AND 120000

USE sample
SELECT project_name, budget
FROM project
WHERE budget >= 95000 AND budget <= 120000

USE sample
SELECT project_name
FROM project
WHERE budget NOT BETWEEN 100000 AND 150000
```

BETWEEN utilizando fechas

```
USE sample
SELECT *
FROM works_on
WHERE enter_date BETWEEN '2016-01-15' AND '2017-12-31'
```

Consultas que implican valores NULL

Consulta Correcta

```
USE sample
SELECT emp_no, project_no
FROM works_on
WHERE project_no = 'p2'
AND job IS NULL
```

Consulta Incorrecta

```
USE sample
SELECT project_no, job
FROM works_on
WHERE job <> NULL
```

```
USE sample
SELECT emp_no, project_no
FROM works_on
WHERE project_no = 'p2'
AND job IS NOT NULL
```

Al utilizar la función ISNULL de una columna en la lista de SELECT, va a necesitar 2 argumentos: la columna y el valor con el que se reemplazará los NULL que tenga la columna.

```
USE sample
SELECT emp_no, ISNULL(job, 'Job unknown') AS task
FROM works_on
WHERE project_no = 'p1'
```

Operador LIKE y comodines (%, _, [,], ^ y ESCAPE)

- **El signo de porcentaje (%) especifica cualquier secuencia de cero o más caracteres.**
- **El guion bajo (_) especifica cualquier carácter individual.**

```
SELECT emp_no,emp_fname
FROM employee
WHERE emp_fname LIKE 'Ja%'
```

Results | Messages

	emp_no	emp_fname
1	29346	James

```sql
SELECT emp_no,emp_fname
FROM employee
WHERE emp_fname LIKE '%am%'
```

Results | Messages

	emp_no	emp_fname
1	29346	James

Que el nombre tenga una "o" y una "d"

```sql
USE AdventureWorks2019
SELECT DISTINCT FirstName
FROM Person.Person
WHERE FirstName LIKE '%o%' AND FirstName LIKE '%d%'
```

Nombre contenga la letra "a" como segundo carácter

```sql
USE AdventureWorks2019
SELECT *
FROM Person.Person
WHERE FirstName LIKE '_a%'
```

Results | Messages

	BusinessEntityID	PersonType	NameStyle	Title	FirstName	Middle
1	5	EM	0	Ms.	Gail	A
2	13	EM	0	Ms.	Janice	M
3	16	EM	0	NULL	David	M
4	19	EM	0	NULL	Mary	A
5	20	EM	0	NULL	Wanida	M
6	22	EM	0	NULL	Sariya	E

Nombre que comience con la cadena "Lock Nut" y despues de un espacio contenga otro carácter

```sql
USE AdventureWorks2019
SELECT *
FROM Production.Product
WHERE Name LIKE 'LOCK NUT _'
```

Results | Messages

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color
1	438	Lock Nut 5	LN-1024	0	0	NUL
2	439	Lock Nut 6	LN-1032	0	0	NUL
3	442	Lock Nut 7	LN-1224	0	0	NUL
4	443	Lock Nut 8	LN-1420	0	0	NUL

Personas con un nombre de 4 digitos y que terminen en "ean"

```sql
USE AdventureWorks2019
SELECT *
FROM Person.Person
WHERE FirstName LIKE '_ean'
```

Results | Messages

	BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastN
1	1101	SC	0	Ms.	Jean	J.	Hollo
2	1183	SC	0	Ms.	Jean	NULL	Jorda
3	17055	IN	0	NULL	Sean	B	King
4	16554	IN	0	NULL	Sean	F	Scott
5	2342	GC	0	NULL	Jean	E	Tren

Personas con un nombre de 4 caracteres y que luego del espacio contengan otros caracteres

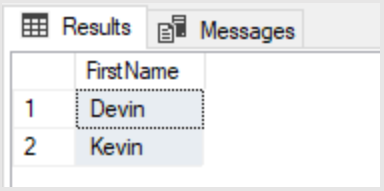
```sql
USE AdventureWorks2019
SELECT *
FROM Person.Person
WHERE FirstName LIKE '____ %'
```

Results | Messages

	BusinessEntityID	PersonType	NameStyle	Title	FirstName	Middle
1	1649	VC	0	Ms.	Mary Lou	M.
2	1813	SC	0	Ms.	Liza Marie	N.
3	1827	SC	0	Ms.	Ruby Sue	R.
4	146	EM	0	NULL	Jian Shuo	NULL
5	2213	GC	0	NULL	Jian Shuo	NULL

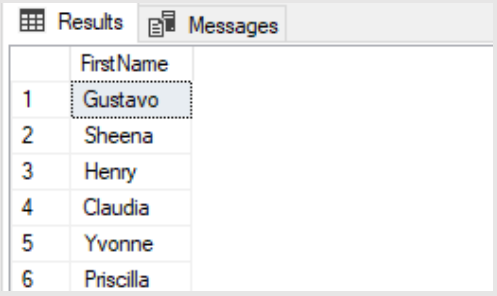
Personas con un nombre de 4 caracteres que comiencen con letras en el rango C y P y terminen en "evin"

```
USE AdventureWorks2019
SELECT DISTINCT FirstName
FROM Person.Person
WHERE FirstName LIKE '[C-P]evin'
```



Obtener los datos completos de todos los empleados cuyo nombre NO termine en el carácter n.

```
USE AdventureWorks2019
SELECT DISTINCT FirstName
FROM Person.Person
WHERE FirstName NOT LIKE '%n'
```



Obtenga los detalles completos de todos los departamentos cuya ubicación comienza con un carácter en el rango de C a F.

```
USE sample
SELECT dept_no, dept_name, location
FROM department
WHERE location LIKE '[C-F]%'
```

El carácter ^ especifica la negación de un rango o una lista de caracteres. Este carácter sólo tiene este significado dentro de un par de corchetes.

Nombre no comience con ninguna letra que se encuentre en el rango entre J y O, y que su apellido no comience con "ma"

```
USE AdventureWorks2019
SELECT *
FROM Person.Person
WHERE FirstName LIKE '[^J-O]%'
AND LastName LIKE '[^MA]%'
```

Ambas sentencias SELECT buscan el guión bajo como un carácter real en la columna project_name

```
USE sample
SELECT project_no,project_name
FROM project
WHERE project_name LIKE '%[_]%'
```

```
USE sample
SELECT project_no,project_name
FROM project
WHERE project_name LIKE '%!_%' ESCAPE '!'
```

GROUP BY

NOTA

Existe una restricción en cuanto al uso de columnas en la cláusula GROUP BY. Cada columna que aparece en la lista SELECT de la consulta debe aparecer también en la cláusula GROUP BY. Esta restricción no se aplica a las constantes ni a las columnas que forman parte de una función agregada.

```
USE sample
SELECT job AS job_agrupado
FROM works_on
GROUP BY job
```

```
USE sample
SELECT project_no, job
FROM works_on
GROUP BY project_no, job
```

NOTA

Las funciones de agregación aparecen en la lista SELECT, que puede incluir una cláusula GROUP BY. Si no hay cláusula GROUP BY en la sentencia SELECT, y la lista SELECT incluye al menos una función de agregación, no se pueden incluir columnas simples en la lista SELECT (salvo como argumentos de una función de agregación). Por lo tanto, el ejemplo siguiente es erróneo.

```
USE sample
SELECT emp_lname, MIN(emp_no)
FROM employee
```

Forma correcta

```
USE sample
SELECT emp_lname, MIN(emp_no)
FROM employee
GROUP BY emp_lname
```

Clausula HAVING

La cláusula HAVING define la condición que luego se aplica a los grupos de filas. Por lo tanto, esta cláusula tiene el mismo significado para los grupos de filas que la cláusula WHERE para el contenido de la tabla correspondiente.

Se recomienda utilizarlo solo con las funciones de agregación. El HAVING actúa como un WHERE, pero solamente en las funciones de agregación.

Obtener el número de proyectos de todos los proyectos que emplean a menos de cuatro personas

Forma incorrecta

```
USE sample
SELECT project_no, COUNT(*) AS Cuenta
FROM works_on
GROUP BY project_no
HAVING Cuenta < 4
```

Forma correcta

```
USE sample
SELECT project_no
FROM works_on
GROUP BY project_no
HAVING COUNT(*) < 4
```

Agrupar las filas de la tabla works_on por job y eliminar aquellos jobs que no comienzan con la letra M.

```
USE sample
SELECT job
FROM works_on
GROUP BY job
HAVING job LIKE 'M%'

/* */

USE AdventureWorks2019
SELECT TerritoryID, MIN(OrderDate), MAX(OrderDate)
FROM Sales.SalesOrderHeader
GROUP BY TerritoryID
HAVING MIN(OrderDate)>='2011-06-01'
```

Selecciona aquellas ordenes que tengan una cantidad > 10 y que su promedio de precio unitario sea mayor a 100.

```
USE AdventureWorks2019
SELECT SalesOrderID, AVG(UnitPrice)'Promedio Precio Unitario'
FROM Sales.SalesOrderDetail
WHERE OrderQty > 10
GROUP BY SalesOrderID
HAVING AVG(UnitPrice) > 100
```

Otra explicación que encontré

La cláusula HAVING se utiliza en combinación con la cláusula GROUP BY para restringir los grupos de filas devueltos a sólo aquellos cuya condición es TRUE.

```
SELECT ColumnNames, aggregate_function(expression)
FROM tables
[WHERE conditions]
GROUP BY column1
HAVING condition;
```

Ejemplos que no corresponden a ninguna base de datos

```
SELECT region, COUNT(customer_id) AS customer_count
FROM customer
GROUP BY region
HAVING COUNT(customer_id) > 200
```

Cláusula ORDER BY

Orden ascendente por defecto

NOTA

Las columnas de la cláusula ORDER BY no tienen por qué aparecer en la lista SELECT. Sin embargo, las columnas ORDER BY deben aparecer en la lista SELECT si se especifica SELECT DISTINCT. Además, esta cláusula no puede hacer referencia a columnas de tablas que no aparezcan en la cláusula FROM.

NOTA

Por lo general, puede ordenar por una columna que no esté en la lista SELECT, pero para ello debe nombrar explícitamente la columna. Sin embargo, si se utiliza GROUP BY o DISTINCT en la consulta, no se puede ordenar por columnas que no estén en la lista SELECT.

Obtener los números de departamento y los nombres de los empleados con números de empleados < 20000, en orden ascendente de apellidos y nombres.

```
SELECT dept_no, emp_fname, emp_lname
FROM employee
WHERE emp_no < 20000
ORDER BY emp_lname, emp_fname
```

También es posible identificar las columnas en la cláusula ORDER BY por la posición ordinal de la columna en la lista SELECT.

```
SELECT dept_no, emp_fname, emp_lname
FROM employee
WHERE emp_no < 20000
ORDER BY 3,2
```

Para cada número de proyecto, obtenga el número de proyecto y el número de todos los empleados, en orden descendente del número de empleado.

```
SELECT project_no, COUNT(emp_no) AS N_emp
FROM works_on
GROUP BY project_no
ORDER BY N_emp DESC
```

Podemos dar un orden personalizado a las distintas columnas.

```
USE AdventureWorks2019
SELECT EmailPromotion,PersonType, COUNT(*) 'Cantidad'
FROM Person.Person
GROUP BY EmailPromotion,PersonType
ORDER BY Cantidad DESC, EmailPromotion DESC, PersonType ASC
```

Cláusula ORDER BY, OFFSET, FETCH NEXT, FETCH FIRST

Nos devuelve filas intercaladas

Obtenga el business entity ID, job title y birthday de los empleadas con BusinessEntityID = 6 y 7.

Se debe utilizar ORDER BY, OFFSET y FETCH NEXT juntos

```
USE AdventureWorks2019
SELECT BusinessEntityID, JobTitle, BirthDate
FROM HumanResources.Employee
ORDER BY BusinessEntityID ASC
OFFSET 5 ROWS
FETCH NEXT 2 ROWS ONLY
```

Results | Messages

	BusinessEntityID	JobTitle	BirthDate
1	6	Design Engineer	1959-03-11
2	7	Research and Development Manager	1987-02-24

Operadores de conjunto: INTERSECT, EXCEPT y UNION

NOTA

Los tres operadores de conjunto discutidos en esta sección tienen diferentes prioridades de evaluación: el operador INTERSECT tiene la prioridad más alta, EXCEPT es evaluado después, y el operador UNION tiene la prioridad más baja. Si no presta atención a estas diferentes prioridades, obtendrá resultados inesperados cuando utilice varios operadores de conjunto juntos.

UNION y UNION ALL

Dos tablas pueden conectarse con el operador UNION si son compatibles entre sí. Esto significa que ambas listas SELECT deben tener el mismo número de columnas, y las columnas correspondientes deben tener tipos de datos compatibles. (Por ejemplo, INT y SMALLINT son tipos de datos compatibles).

Nos devuelve la unión de valores ÚNICOS (sin duplicados)

```
USE sample
SELECT domicile
FROM employee_enh
UNION
SELECT location
FROM department
```

Nos devuelve la unión de valores DUPLICADOS

```
USE sample
SELECT domicile
FROM employee_enh
UNION ALL
SELECT location
FROM department
```

La ordenación del resultado de la unión sólo puede realizarse si se utiliza la cláusula ORDER BY con la última sentencia SELECT.

```
USE sample
SELECT domicile
FROM employee_enh
UNION ALL
SELECT location
FROM department
ORDER BY 1 <------ Lo aplicamos al final, con la última sentencia SELECT
```

Las cláusulas GROUP BY y HAVING pueden utilizarse con las sentencias SELECT particulares, pero no con la unión en sí.

```
USE sample
SELECT domicile
FROM employee_enh
GROUP BY domicile <---
UNION ALL
SELECT location
FROM department
GROUP BY location <---
ORDER BY 1
```

INTERSECT

NOTA

Transact-SQL no soporta el operador INTERSECT con la opción ALL (Lo mismo ocurre con el operador EXCEPT).

Nos devuelve la intersección de valores ÚNICOS (No duplicados)

Nos devuelve los valores distintos devueltos por las consultas situadas a los lados izquierdo y derecho del operador INTERSECT.

```
USE sample
SELECT domicile
FROM employee_enh
INTERSECT
SELECT location
FROM department
```

```
USE AdventureWorksDWH2016
SELECT CustomerKey
FROM FactInternetSales
INTERSECT
SELECT CustomerKey
FROM DimCustomer
WHERE DimCustomer.Gender = 'F'
ORDER BY CustomerKey;
# Result: 9133 Rows (Sales to customers that are female.)
```

EXCEPT

El operador EXCEPT se utiliza para devolver todas las filas de la primera sentencia SELECT que no son devueltas por la segunda sentencia SELECT.

Nos devuelve los valores distintos de la consulta del lado izquierdo del operador EXCEPT que tampoco se encuentran en la consulta derecha.

```
USE sample
SELECT domicile
FROM employee_enh
EXCEPT
SELECT location
FROM department
```

```
USE AdventureWorksDWH2016
SELECT CustomerKey
FROM FactInternetSales
EXCEPT
SELECT CustomerKey
FROM DimCustomer
WHERE DimCustomer.Gender = 'F'
ORDER BY CustomerKey;
# Result: 9351 Rows (Sales to customers that are not female.)
```

Expresión CASE

NOTA

CASE no representa una sentencia (como en la mayoría de los lenguajes de programación) sino una expresión. Por lo tanto, la expresión CASE puede utilizarse (casi) en todos los casos en los que el lenguaje Transact-SQL permite el uso de una expresión.

Creamos una nueva columna "Category" donde sus valores van a depender de los valores de la columna "ProductLine".

```
USE AdventureWorks2019

SELECT ProductNumber, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,
   Name
FROM Production.Product
```

Results | Messages

	ProductNumber	Category	Name
1	AR-5381	Not for sale	Adjustable Race
2	BA-8327	Not for sale	Bearing Ball
3	BE-2349	Not for sale	BB Ball Bearing
4	BE-2908	Not for sale	Headset Ball Bearings
5	BL-2036	Not for sale	Blade
6	CA-5965	Not for sale	LL Crankarm
7	CA-6738	Not for sale	ML Crankarm
8	CA-7457	Not for sale	HL Crankarm
9	CB-2903	Not for sale	Chainring Bolts

Lo mismo que el anterior, pero se colocó el nombre de la nueva columna junto a END

```
USE AdventureWorks2019;

SELECT ProductNumber,
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END Category,
      Name
FROM Production.Product;
```

Devuelve los project_name de acuerdo a la condición del Budget

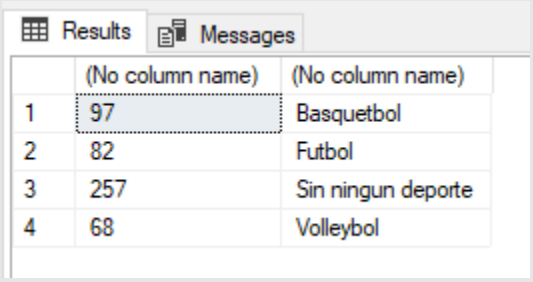
```
USE sample;
SELECT project_name,
    CASE
      WHEN budget > 0 AND budget < 100000  THEN 1
      WHEN budget >= 100000 AND budget < 200000  THEN 2
      WHEN budget >= 200000 AND budget < 300000  THEN 3
      ELSE 4
    END budget_weight
  FROM project;


USE sample
SELECT project_name,
      CASE
        WHEN p1.budget < (SELECT AVG(p2.budget) FROM project p2)
            THEN 'below average'
        WHEN p1.budget = (SELECT AVG(p2.budget) FROM project p2)
            THEN 'on average'
        WHEN p1.budget > (SELECT AVG(p2.budget) FROM project p2)
            THEN 'above average'
    END budget_category
FROM project p1
```

Results | Messages

	project_name	budget_category
1	Apollo	below average
2	Gemini	below average
3	Mercury	above average

Contar los ProductID, categorizarlos y agruparlos.

```
USE AdventureWorks2019
SELECT
COUNT(ProductID),
CASE
      WHEN Class = 'H' THEN 'Futbol'
      WHEN Class = 'L' THEN 'Basquetbol'
      WHEN Class = 'M' THEN 'Volleybol'
      ELSE 'Sin ningun deporte'
END
FROM Production.Product
GROUP BY
CASE
      WHEN Class = 'H' THEN 'Futbol'
      WHEN Class = 'L' THEN 'Basquetbol'
      WHEN Class = 'M' THEN 'Volleybol'
      ELSE 'Sin ningun deporte'
END
```



Results | Messages

	(No column name)	(No column name)
1	97	Basquetbol
2	82	Futbol
3	257	Sin ningun deporte
4	68	Volleybol

M: Montaña
R: Carretera
T: Turismo
S: Otros
NULL: 'Sin Linea'

```
USE AdventureWorks2019
SELECT
ProductLine,
Categoria =
CASE
      WHEN ProductLine = 'M' THEN 'Montaña'
      WHEN ProductLine = 'R' THEN 'Carretera'
      WHEN ProductLine = 'T' THEN 'Turismo'
      WHEN ProductLine = 'S' THEN 'Otros'
      ELSE 'Sin Linea'
END
FROM Production.Product
```

Uso de varios de CASE

```
USE AdventureWorks2019
SELECT
ProductLine,
Categoria =
CASE
      WHEN ProductLine = 'M' THEN
        CASE
            WHEN Class = 'H' THEN 'Carretera'
            WHEN Class = 'L' THEN 'Turismo'
            WHEN Class = 'M' THEN 'Otros'
            ELSE 'Sin Montaña'
        END
      WHEN ProductLine = 'R' THEN
        CASE
            WHEN Color = 'Black'  THEN 'Modelo Oscuridad'
            WHEN Color = 'Silver' THEN 'Modelo Plateado'
            WHEN Color = 'Yellow' THEN 'Modelo Radiante'
            WHEN Color = 'Blue'   THEN 'Modelo Oceano'
            ELSE 'Sin Modelo'
        END
      ELSE 'Sin Linea'
END
FROM Production.Product
```

Expresión TOP

```
USE Registro

SELECT * FROM employee
```

Results | Messages

	emp_no	emp_fname	emp_lname	dept_no
1	1	Matthew	Smith	d3
2	2	Ann	Jones	d1
3	3	John	Barrimore	d1
4	4	James	James	d1
5	5	Elsa	Bertoni	d2
6	6	Elke	Hansel	d2
7	7	Sybill	Moser	d3

Ejemplo 1

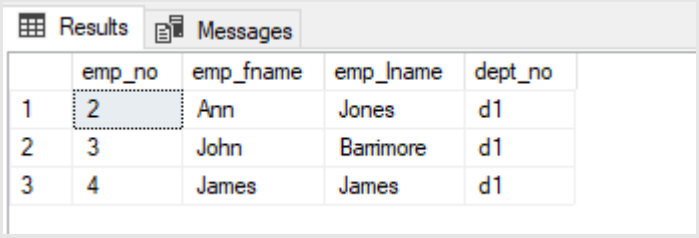
```
SELECT TOP 3 * FROM employee
```

Results | Messages

	emp_no	emp_fname	emp_lname	dept_no
1	1	Matthew	Smith	d3
2	2	Ann	Jones	d1
3	3	John	Barrimore	d1

Ejemplo 2

```
SELECT TOP 3 * FROM employee
WHERE dept_no = 'd1'
```



Results | Messages

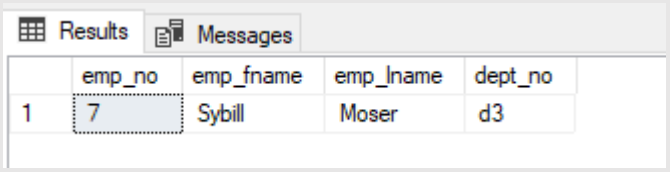
	emp_no	emp_fname	emp_lname	dept_no
1	2	Ann	Jones	d1
2	3	John	Barrimore	d1
3	4	James	James	d1

Los 3 primeros registros del dept_no = d1

Ejemplo 3

El último registro de la tabla

```
SELECT TOP 1 * FROM employee
ORDER BY emp_no DESC
```



Results | Messages

	emp_no	emp_fname	emp_lname	dept_no
1	7	Sybill	Moser	d3