Fundamentos teóricos

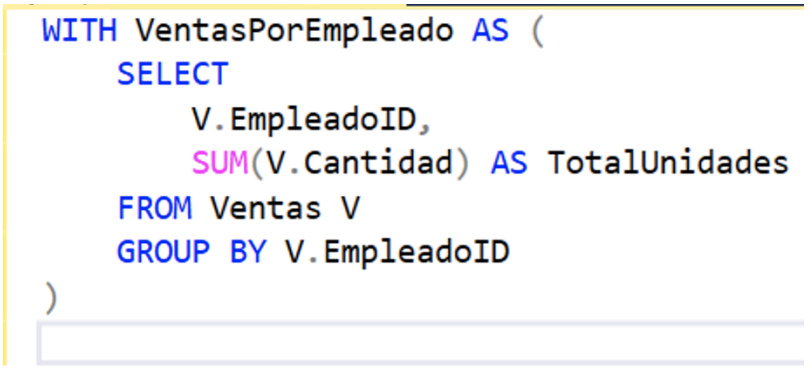
Flujo lógico de un SELECT

1. FROM (+ JOIN) - crea el *set base*.
2. WHERE - filtra filas (*filtrado vertical*).
3. GROUP BY - agrupa filas.
4. HAVING - filtra grupos.
5. SELECT - proyecta columnas / expresiones.
6. DISTINCT - elimina duplicados.
7. ORDER BY - ordena resultado.
8. TOP / OFFSET-FETCH - limita filas devueltas.

Por qué importa: entender el orden evita errores como usar un alias del SELECT en WHERE y ayuda a optimizar consultas (aplicar filtros cuanto antes).

Common Table Expression (CTE)

Una Common Table Expression (CTE) es un bloque de consulta temporal que se define al principio de un SELECT, INSERT, UPDATE o DELETE con la cláusula WITH. Piensa en ella como una “vista ad-hoc” válida solo durante la ejecución de esa sentencia.



```sql
WITH VentasPorEmpleado AS (
    SELECT
        V.EmpleadoID,
        SUM(V.Cantidad) AS TotalUnidades
    FROM Ventas V
    GROUP BY V.EmpleadoID
)
```

¿Para qué la usamos?

Uso típico	¿Por qué es útil?	Mini-ejemplo
Claridad y mantenimiento	Divide consultas muy grandes en secciones lógicas, evitando subconsultas anidadas difíciles de leer.	`sql WITH VentasPorCliente AS ( SELECT CustomerID, SUM(Total) AS Monto FROM Orders GROUP BY CustomerID ) SELECT * FROM VentasPorCliente WHERE Monto > 1000;`
Reutilizar resultados intermedios	La CTE puede referirse múltiples veces en la misma consulta sin volver a escribir la lógica.	Calcular un ranking y luego filtrar por él sin repetir el cálculo.
Recursividad	Ideal para recorrer jerarquías (categorías padre-hijo, empleados, etc.) sin procedimientos almacenados.	`sql WITH cteCat (CategoryID, CategoryName, Nivel) AS ( SELECT CategoryID, CategoryName, 1 FROM Categories WHERE ParentID IS NULL UNION ALL SELECT c.CategoryID, c.CategoryName, Nivel+1 FROM Categories c JOIN cteCat p ON c.ParentID = p.CategoryID ) SELECT * FROM cteCat;`
Mejorar rendimiento	En algunos casos reemplaza subconsultas correlacionadas costosas o evita materializar tablas temporales.	Cálculos agregados reutilizados varias veces en la misma sentencia.

Repaso Consultas Avanzadas I

Subconsultas

Tipo	Devuelve	Dónde se usa	Ejemplo
Escalar	1 valor	SELECT, WHERE, HAVING	`SELECT (SELECT AVG(UnitPrice) FROM Products)`
Columna	1 columna	SELECT	`SELECT (SELECT MAX(Stock) FROM Products) AS MaxStock`
Tabla (in-line)	N filas/col	FROM, JOIN	`FROM (SELECT …) AS t`
Correlacionada	depende de fila externa	WHERE, SELECT	`WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products p2 WHERE p2.CategoryID = p.CategoryID)`

Funciones de ventana

Concepto	Detalle
Ventana	Conjunto de filas “visibles” para la función. Definida con `PARTITION BY`, `ORDER BY`, y un marco (`ROWS
Row vs. Range	`ROWS BETWEEN 1 PRECEDING AND CURRENT ROW` mira exactamente dos filas; `RANGE` usa valores de orden (útil en fechas).
Uso Classics	Ranking (`ROW_NUMBER`, `RANK`), acumulados (`SUM(...) OVER(ORDER BY)`), *running averages* (`AVG` con marco móvil), comparaciones temporales (`LAG`, `LEAD`).
Ventaja clave	No colapsa filas: permite calcular métricas por fila sin perder detalle.

JOINs y teoría de conjuntos

JOIN	Equivalencia en teoría de conjuntos	¿Cuándo usar?
INNER	Intersección (∩)	Relacionar datos obligatorios de ambos lados.
LEFT OUTER	A ∪ (A∩B)	Reportar todo de la tabla principal + coincidencias.
RIGHT OUTER	B ∪ (A∩B)	Inverso del anterior (poco común si controlas el orden).
FULL OUTER	A ∪ B	Auditorías: "qué falta en cada lado".
CROSS	Producto cartesiano	Generar combinaciones (¡cuidado con tamaño!).
SELF	Intersección de una tabla consigo misma	Comparaciones internas (jerarquías, productos similares).

Demostraciones guiadas con historias reales

Demo	Historia	Consulta clave
INNER JOIN	"Listar pedidos con su cliente para generar facturas."	`SELECT ... FROM Customers C JOIN Orders O ON ...`
FULL OUTER	"Auditoría: detectar pedidos huérfanos o clientes sin pedidos."	`FULL JOIN`
SELF JOIN	"Mostrar accesorios de la misma categoría que un producto elegido."	`SELECT P1..., P2... FROM Products P1 JOIN Products P2 ...`
UNION ALL	"Construir mailing combinando clientes y leads (sin deduplicar)."	`(SELECT Email FROM Customers) UNION ALL (SELECT Email FROM Leads)`
INTERSECT	"Clientes que además son proveedores (programa de partners)."	`SELECT ... FROM Customers INTERSECT SELECT ... FROM Suppliers`

Ejercicios prácticos sobre JOINs

INNER JOIN — Facturación diaria

Caso real

El área de facturación necesita ver, para cada pedido, la fecha y el nombre del cliente para emitir la factura.

1. Identifica las tablas

- Orders contiene la fecha, Customers el nombre.
  Define la columna vinculante

- Orders.CustomerID = Customers.CustomerID.
  Escoge el JOIN

- Necesitas solo las coincidencias (un pedido sin cliente no es posible): INNER JOIN.
  Agrega lo que quieres proyectar

```sql
SELECT   o.OrderID,
         o.OrderDate,
         c.FirstName + ' ' + c.LastName AS Cliente
FROM     Orders o
INNER JOIN Customers c ON c.CustomerID = o.CustomerID
ORDER BY o.OrderDate;
```

LEFT JOIN — Productos sin ventas

Caso real

Compras quiere saber qué productos jamás se han vendido para liquidarlos con descuento.

Pasos:

1. Tabla primaria: Products (queremos todos).

2. Tabla secundaria: OrderItems (solo si hubo ventas).

3. Clave: ProductID.

4. JOIN que conserve no coincidentes ⇒ LEFT JOIN.

Agrega totales con GROUP BY:

```sql
SELECT  p.ProductName,
        ISNULL(SUM(i.Quantity),0) AS Vendidos
FROM    Products p
LEFT JOIN OrderItems i
       ON i.ProductID = p.ProductID
GROUP BY p.ProductName
HAVING ISNULL(SUM(i.Quantity),0)=0;    -- jamás vendidos
```

RIGHT JOIN — Ítems huérfanos (control de integridad)

Caso real

Un bug creó líneas en OrderItems sin cabecera en Orders. Debes detectarlas.

1. Tabla *derecha* imprescindible: OrderItems (queremos todos sus registros).
2. Tabla izquierda: Orders.
3. Un RIGHT JOIN mantendrá los ítems aunque no haya Order.

Calcula cuántos hay por "pedido fantasma":

```sql
SELECT  o.OrderID,
        COUNT(i.OrderItemID) AS Lineas
FROM    Orders o
RIGHT JOIN OrderItems i
       ON i.OrderID = o.OrderID
WHERE   o.OrderID IS NULL
GROUP BY o.OrderID;
```

FULL OUTER JOIN — Auditoría de contactos duplicados

Caso real

Marketing quiere un listado único de emails, indicando si la persona es cliente, proveedor o ambos.

1. Necesitamos todos los emails de ambas tablas.
2. Clave: Email.
3. JOIN: FULL OUTER.

Usa COALESCE para elegir el email no nulo; clasifica el tipo con CASE.

```sql
SELECT  COALESCE(c.Email, s.Email) AS Email,
        CASE WHEN c.Email IS NOT NULL AND s.Email IS NOT NULL THEN 'Cliente+Proveedor'
             WHEN c.Email IS NOT NULL THEN 'Cliente'
             ELSE 'Proveedor' END AS Tipo
FROM    Customers  c
FULL OUTER JOIN Suppliers s
ON s.Email = c.Email;
```

CROSS JOIN — Matriz de márgenes potenciales

Caso real

Dirección quiere un listado de todas las combinaciones Categoría-Proveedor para analizar futuras alianzas y cuántos productos ofrece cada par.

Pasos:

1. Categories × Suppliers → CROSS JOIN (producto cartesiano).
2. A ese resultado, hazle una LEFT JOIN con Products para contar.

Agrupa y cuenta:

```sql
SELECT   c.CategoryName,
         s.SupplierName,
         COUNT(p.ProductID) AS Productos
FROM     Categories c
CROSS JOIN Suppliers s
LEFT JOIN Products  p
        ON p.CategoryID = c.CategoryID
       AND p.SupplierID = s.SupplierID
GROUP BY c.CategoryName, s.SupplierName
ORDER BY Productos DESC;
```

SELF JOIN — Recomendador de productos similares

Caso real

Al ver un producto, la web debe sugerir otro del mismo proveedor y precio ± 5 USD, evitando mostrar el mismo artículo.

1. Dos alias de Products: p1 (base) y p2 (sugerido).
2. Condiciones de unión:
    - p1.SupplierID = p2.SupplierID
    - p1.ProductID <> p2.ProductID
    - ABS(p1.UnitPrice - p2.UnitPrice) <= 5

Para cada p1 obtén 1 sugerencia usando, p.ej., TOP 1 o ROW_NUMBER().

```sql
WITH Sug AS (
    SELECT  p1.ProductID  AS ProductoBase,
            p1.ProductName,
            p2.ProductID  AS ProductoSugerido,
            p2.ProductName AS NombreSugerido,
            ROW_NUMBER() OVER (PARTITION BY p1.ProductID
                               ORDER BY ABS(p1.UnitPrice - p2.UnitPrice)) AS rn
    FROM    Products p1
    JOIN    Products p2
          ON p1.SupplierID = p2.SupplierID
         AND p1.ProductID <> p2.ProductID
         AND ABS(p1.UnitPrice - p2.UnitPrice) <= 5
)

SELECT ProductName, NombreSugerido FROM Sug WHERE rn = 1; -- una sugerencia por producto
```

Operadores de Conjunto — definiciones y reglas clave

Operador	Definición (qué hace)	Duplicados	Sintaxis mínima
`UNION`	Devuelve la unión matemática de dos result-sets.	**Se eliminan** (aplica `DISTINCT` implícito).	`SELECT … FROM A UNION SELECT … FROM B`
`UNION ALL`	Igual que `UNION`, pero **conserva** los duplicados (más veloz, no requiere ordenamiento).	Se mantienen.	`… UNION ALL …`
`INTERSECT`	Devuelve solo las filas que aparecen **en ambos** result-sets (intersección ∩).	Se eliminan.	`… INTERSECT …`
`EXCEPT`	Devuelve las filas del primer result-set que **no** están en el segundo (diferencia de conjuntos A – B).	Se eliminan.	`… EXCEPT …`

Reglas que deben cumplir los SELECT involucrados

Mismo número de columnas

```sql
-- OK (3 columnas cada uno)
SELECT ProductID, ProductName, UnitPrice FROM Products
UNION
SELECT ProductID, ProductName, UnitPrice FROM Products2024;
```

Si no coinciden, SQL Server arrojará: *"All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists."*

1. Compatibilidad de tipos por posición

    - La columna 1 del primer SELECT debe ser implícitamente convertible al tipo de la columna 1 del segundo SELECT, y así sucesivamente.

    - El resultado adopta el tipo de mayor precedencia.
      Ejemplo: INT ∪ DECIMAL → resultado DECIMAL; VARCHAR(100) ∪ CHAR(20) → VARCHAR(100).

Posición	Tipo en SELECT 1	Tipo en SELECT 2	¿Compatible?	Resultado final
1	`INT`	`BIGINT`	✅ Sí	`BIGINT`
2	`INT`	`DECIMAL(10,2)`	✅ Sí	`DECIMAL(10,2)`
3	`CHAR(10)`	`VARCHAR(50)`	✅ Sí	`VARCHAR(50)`
4	`VARCHAR(100)`	`NVARCHAR(100)`	✅ Sí	`NVARCHAR(100)`
5	`DATETIME`	`DATE`	✅ Sí	`DATETIME`
6	`INT`	`VARCHAR(10)`	❌ No	❌ Error (incompatibles)
7	`BIT`	`INT`	✅ Sí	`INT`
8	`FLOAT`	`DECIMAL(18,4)`	✅ Sí	`FLOAT`
9	`UNIQUEIDENTIFIER`	`VARCHAR(36)`	❌ No	❌ Error (no implícita)
10	`TEXT`	`VARCHAR(MAX)`	✅ Sí (con conversión)	`VARCHAR(MAX)`

2. Alias y orden

    - El alias que “sobrevive” para cada columna es el del primer SELECT.

    - ORDER BY sólo se permite una vez, al final de toda la expresión combinada.

Si necesitas ordenar por una columna no proyectada, debes usar la posición:

```
(SELECT Country FROM Customers
 UNION
 SELECT Country FROM Suppliers)
ORDER BY 1;          -- 1ª columna
```

3. Precedencia
    - INTERSECT se evalúa antes que UNION y EXCEPT.

Usa paréntesis para forzar el orden deseado:

```
SELECT … FROM A
UNION
(SELECT … FROM B INTERSECT SELECT … FROM C);
```

4. NULL y Collations
    - NULL cuenta como “valor válido” en los operadores: dos filas NULL se consideran duplicadas y se eliminarán con UNION.
    - Para cadenas, ambas columnas deben compartir collation o ser convertibles; de lo contrario, usa COLLATE.

Errores típicos (y cómo leerlos)

Mensaje	Causa	Solución
"number of expressions..."	Diferente # columnas	Ajusta SELECT para igualar columnas
"Types varchar and int are incompatible..."	Tipos no convertibles	`CAST` / `CONVERT` columnas
"ORDER BY items must appear in the select list if the SELECT statement contains a UNION..."	Intentar ordenar por columna no proyectada	Ordena por posición o incluye la columna

Ejemplo completo contextualizado

Escenario real: Marketing necesita un único mailing combinando clientes y potenciales leads, pero sin duplicados.

```sql
/* Clientes */
SELECT FirstName + ' ' + LastName AS Nombre, Email
FROM   Customers
UNION        -- quita duplicados
/* Leads */
SELECT LeadName, Email
FROM   Leads;
```

Si sí se quieren duplicados (para ver volumen total de envíos):

... UNION ALL ...

Escenario auditoría: Proveedores que también son clientes (programa "Marketplace Partner"):

```sql
SELECT  Email
FROM    Customers
INTERSECT
SELECT  Email
FROM    Suppliers;      -- solo corre si Suppliers.Email existe
```

Escenario segmentación: Clientes que aún no compraron nada:

```sql
SELECT CustomerID
FROM   Customers
EXCEPT
SELECT DISTINCT CustomerID
FROM   Orders;
```