BASE DE DATOS 2

Resumen de Errores y Cómo Evitarlos

3 Parciales de Práctica — Northwind

Este documento resume todos los errores cometidos en los tres parciales de práctica, organizados por ejercicio y por tipo de error, con explicaciones claras de por qué ocurrieron y cómo evitarlos en el parcial real.

Ejercicio 1 — VISTA

Errores por parcial

Parcial 1 — Primer intento

Error 1: Pusiste BEGIN/END en la vista
Las vistas NO llevan BEGIN ni END. Solo va CREATE VIEW ... AS y el SELECT directamente. Los BEGIN/END son de procedimientos y funciones, no de vistas.

Error 2: Mezclaste dos enfoques en el mismo FROM
Usaste subconsultas correlacionadas en el SELECT y al mismo tiempo una subconsulta en el FROM. Hay que elegir uno solo. El mas limpio es la subconsulta en el FROM.

Error 3: El CASE referenciaba un alias mal conectado
El CASE usaba promediocat.Precio pero ese alias no estaba correctamente vinculado con el resto de la consulta exterior.

Parcial 2 — Segundo intento

Error: Repetiste la tabla Categories adentro y afuera de la subconsulta
La subconsulta ya calculaba todo lo necesario de Categories. Al ponerla tambien afuera, generabas un JOIN innecesario que podia multiplicar filas o romper el resultado.

✓ Sin BEGIN/END esta vez. Aprendiste del error anterior.

Parcial 3 — Tercer intento

✓ Vista practicamente perfecta. Solo faltaba quitar la tabla Categories del FROM externo, ya que la subconsulta la resuelve sola.

La regla de oro de las vistas

Tip para el parcial: Si tenes que clasificar un TOTAL o PROMEDIO en categorias: 1) calculalo en una subconsulta con GROUP BY, 2) ponela en el FROM con un alias, 3) aplicale el CASE afuera sobre el resultado ya calculado. Nunca mezcles dos enfoques en la misma consulta.

```
CREATE VIEW vw_Ejemplo

AS

SELECT datos.Columna,

CASE

WHEN datos.Total > 1000 THEN 'Alto'

WHEN datos.Total BETWEEN 500 AND 1000 THEN 'Medio'
```

```
ELSE 'Bajo'

END as Categoria

FROM (SELECT columna, SUM(valor) as Total

FROM Tabla

GROUP BY columna) as datos
```

Ejercicio 2 — FUNCION

Errores por parcial

Parcial 1 — Primer intento

Error 1: Tipo de retorno incorrecto (int en vez de money)
Cuando la funcion retorna un importe de dinero, el tipo debe ser RETURNS money, no RETURNS int. Con int se pierden los decimales y el resultado es incorrecto.

Error 2: Faltó la consulta final que usa la funcion
El enunciado pedia ademas un SELECT que invoque la funcion para todos los productos/clientes. Hay que leer el enunciado completo hasta el final.

Parcial 2 — Segundo intento

Error: No terminaste la funcion (te trabaste)
El enfoque que intentabas era correcto pero mas complejo de lo necesario. La solucion mas limpia era un SELECT TOP 1 con GROUP BY y ORDER BY SUM(...) DESC directamente.

Error: Faltó la consulta final otra vez
El mismo error que en el parcial anterior: hay que leer el enunciado completo.

Parcial 3 — Tercer intento

Error: Funcion de fecha incorrecta (DAY() en vez de DATEDIFF())
DAY(fecha) extrae solo el numero del dia del mes (1-31), no calcula diferencia entre fechas. Para calcular dias entre dos fechas se usa DATEDIFF(DAY, fechaInicio, fechaFin).

Error: Faltó la consulta final por tercera vez
Este error se repitio en los tres parciales. Es el mas facil de evitar: siempre leer el enunciado hasta el ultimo punto.

Reglas clave para funciones

Tip para el parcial: Antes de escribir el tipo de retorno, preguntate: ¿el resultado tiene decimales? ¿es dinero? → money. ¿es un numero entero? → int. ¿es texto? → varchar(N).

Tip para el parcial: Funciones de fecha que hay que dominar: DATEDIFF(DAY/MONTH/YEAR, fecha1, fecha2) calcula diferencia. GETDATE() devuelve hoy. MAX(fecha) devuelve la mas reciente.

Tip para el parcial: Siempre leer el enunciado completo. Si dice 'luego escribir una consulta que...', esa parte tambien vale puntos.

```
CREATE FUNCTION fn_Ejemplo(@ID int)
```

```
RETURNS money -- tipo segun lo que retorna

AS

BEGIN

DECLARE @retorno money


SELECT @retorno = SUM(od.Quantity * od.UnitPrice * (1 - od.Discount))

FROM OrderDetails od

WHERE od.ProductID = @ID


RETURN @retorno

END


-- Consulta que usa la funcion (NO olvidar esto)

SELECT ProductID, ProductName, dbo.fn_Ejemplo(ProductID) as Total

FROM Products
```

Ejercicio 3 — PROCEDIMIENTO

Errores por parcial

Parcial 1 — Primer intento

✓ El procedimiento RebajarInvendibles estuvo bien. Sin errores significativos.

Parcial 2 — Segundo intento

Error 1: Faltó el WHERE SupplierID = @SupplierID en el UPDATE
Sin ese filtro, el UPDATE actualizaria los precios de TODOS los productos de todos los proveedores. Siempre verificar que el UPDATE tenga el filtro correcto del parametro recibido.

Error 2: El SET @@ROWCOUNT estaba fuera del bloque ELSE
Si @@ROWCOUNT queda despues del END del bloque, se ejecuta siempre, incluso cuando no se hizo el UPDATE. Debe ir INMEDIATAMENTE despues del UPDATE, adentro del mismo bloque.

Parcial 3 — Tercer intento

✓ No intento escribirlo pero entendio la logica al ver la solucion explicada.

Reglas clave para procedimientos

Tip para el parcial: @@ROWCOUNT siempre va INMEDIATAMENTE despues del INSERT/UPDATE/DELETE que queres medir. Si pones cualquier otra instruccion en el medio, @@ROWCOUNT ya no tiene el valor correcto.

Tip para el parcial: Cuando hay multiples validaciones, usar RETURN despues de cada mensaje de error es mas limpio que anidar IF/ELSE. Cada validacion que falla corta la ejecucion con RETURN.

Tip para el parcial: Siempre verificar que el UPDATE/DELETE tenga el WHERE correcto apuntando al parametro recibido.

```
CREATE PROCEDURE sp_Ejemplo

@ID int,

@Porcentaje real,

@Afectados int OUTPUT

AS

BEGIN

-- Validacion 1

IF NOT EXISTS (SELECT * FROM Tabla WHERE ID = @ID)
```

```
BEGIN

SELECT 'No existe' as Mensaje

RETURN -- corta aqui, no sigue

END


-- Validacion 2

IF @Porcentaje NOT BETWEEN 1 AND 100

BEGIN

SELECT 'Fuera de rango' as Mensaje

RETURN -- corta aqui, no sigue

END


-- Operacion

UPDATE Tabla

SET Precio = Precio * (1 + @Porcentaje/100)

WHERE ID = @ID -- siempre filtrar por el parametro


SET @Afectados = @@ROWCOUNT -- siempre justo despues del UPDATE

END
```

Ejercicio 4 — TRIGGER AFTER

Errores por parcial

Parcial 1 — Primer intento

✓ El trigger de auditoria de Suppliers estuvo practicamente perfecto. Solo un detalle menor: el enunciado pedia SUSER_SNAME() pero se uso HOST_NAME().

Parcial 2 — Segundo intento

✓ El trigger LogPedidosBorrados fue perfecto. Sin errores. El mejor ejercicio del parcial.

Parcial 3 — Tercer intento

✓ El trigger fue resuelto correctamente por el profesor y entendido sin problemas.

El trigger AFTER, bien entendido

El AFTER es el tipo mas intuitivo de trigger. Se usa cuando el objetivo es REACCIONAR a algo que ya paso (auditar, registrar, actualizar otra tabla). No bloquea nada, solo agrega acciones adicionales despues del evento.

Tip para el parcial: AFTER siempre cuando el enunciado dice: registrar, auditar, guardar historial, actualizar otra tabla cuando pase algo.

Tip para el parcial: En un AFTER UPDATE: deleted tiene los valores VIEJOS, inserted tiene los valores NUEVOS. Siempre cruzarlos por la PK (WHERE d.ProductID = i.ProductID).

Tip para el parcial: IF UPDATE(columna) detecta si esa columna especifica fue parte del SET del UPDATE. Usarlo cuando solo hay que auditar cambios en ciertas columnas, no en todas.

```
CREATE TRIGGER trg_Auditoria

ON Tabla

AFTER UPDATE

AS

BEGIN

IF UPDATE(Columna) -- solo si se modifico esa columna

INSERT INTO TablaAuditoria

SELECT GETDATE(), i.ID, i.Nombre,

d.Columna, -- valor VIEJO (de deleted)

i.Columna, -- valor NUEVO (de inserted)
```

```
SUSER_SNAME() -- usuario de SQL Server

FROM inserted i, deleted d

WHERE i.ID = d.ID -- emparejar viejo con nuevo

END
```

Ejercicio 5 — TRIGGER INSTEAD OF

Errores por parcial

Parcial 1 — Primer intento

Error 1: AFTER en vez de INSTEAD OF
El enunciado pedia 'no permitir' el ingreso. Para bloquear una operacion, siempre es INSTEAD OF, no AFTER. Con AFTER el INSERT ya ocurrio y habria que revertirlo con ROLLBACK.

Error 2: La condicion del IF al reves (IF NOT EXISTS en vez de IF EXISTS)
Se buscaban filas validas en vez de filas invalidas. La condicion correcta es: IF EXISTS (busco algo malo) → si encuentro algo malo → RAISERROR.

Parcial 2 — Segundo intento

Error 1: AFTER UPDATE en vez de INSTEAD OF UPDATE (mismo error repetido)
El enunciado pedia validar antes de permitir el UPDATE. Para eso siempre es INSTEAD OF.

Error 2: Condicion del IF al reves (mismo error repetido)
Buscaba UnitPrice BETWEEN 1 AND 500 (precios validos) cuando tenia que buscar precios invalidos: UnitPrice < 1 OR UnitPrice > 500.

Parcial 3 — Tercer intento

Error: Condicion del IF al reves (tercer parcial consecutivo con el mismo error)
Escribio IF NOT EXISTS buscando filas validas. Tiene que ser IF EXISTS buscando filas invalidas (CustomerID o EmployeeID que NO existen).

✓ Uso de INSTEAD OF INSERT correcto esta vez. Se corrigio el primer error.

La regla definitiva para INSTEAD OF

Tip para el parcial: INSTEAD OF cuando el enunciado dice: no permitir, bloquear, validar antes, no se puede insertar si...

Tip para el parcial: La pregunta magica antes de escribir el IF: ¿Que es lo MALO que quiero detectar? Eso va en el IF EXISTS. Si encuentro algo malo → RAISERROR. Si todo esta bien → hago la operacion en el ELSE.

El error mas repetido de todos los parciales (3 veces)
La condicion del IF SIEMPRE busca lo MALO (lo que quiere bloquear), no lo bueno. IF EXISTS (condicion del problema) → RAISERROR. ELSE → hacer la operacion.

```
CREATE TRIGGER tr_Validar
```

```
ON Tabla

INSTEAD OF INSERT -- siempre INSTEAD OF para bloquear

AS

BEGIN


-- Busco lo MALO (no lo bueno)

IF EXISTS (SELECT *

FROM inserted i

WHERE i.Campo NOT IN (SELECT Campo FROM OtraTabla)

OR i.OtroCampo < 0)


RAISERROR('Mensaje de error claro', 16, 1)


ELSE

INSERT INTO Tabla -- si todo esta bien, insertar manualmente

SELECT * FROM inserted


END
```

Resumen General — Evolucion por Parcial

Tabla de errores por ejercicio

Ejercicio	Parcial 1	Parcial 2	Parcial 3
Vista	3 errores	1 error	Casi perfecto
Funcion	2 errores	2 errores	2 errores
Procedimiento	Sin errores	2 errores	No intentado
Trigger AFTER	Casi perfecto	Perfecto ✓	Perfecto ✓
Trigger INSTEAD OF	2 errores	2 errores	1 error

Los 4 errores que se repitieron en todos los parciales

Error 1 (3 veces): Condicion del IF al reves en INSTEAD OF
IF NOT EXISTS buscando casos validos, cuando hay que usar IF EXISTS buscando casos invalidos. Regla: siempre buscar lo MALO en el IF EXISTS.

Error 2 (2 veces): AFTER en vez de INSTEAD OF
Usar AFTER UPDATE/INSERT cuando el enunciado pide bloquear la operacion. Regla: 'no permitir' = INSTEAD OF. 'registrar/auditar' = AFTER.

Error 3 (3 veces): Faltó la consulta final de la funcion
El enunciado de la funcion siempre incluye 'luego escribir una consulta que...'. Esa parte tambien vale puntos. Leer el enunciado completo.

Error 4 (2 veces): BEGIN/END en vistas y tablas repetidas en el FROM
Las vistas no llevan BEGIN/END. Ademas, si una tabla ya esta resuelta en la subconsulta, no hace falta repetirla en el FROM externo.

Los 3 puntos fuertes que hay que mantener

✓ Trigger AFTER: dominado completamente. Estructura, deleted/inserted, GETDATE(), SUSER_SNAME(), todo correcto.

✓ Estructura general de funciones y procedimientos: DECLARE, SELECT @var =, RETURN, OUTPUT, @@ROWCOUNT, todo bien entendido.

✓ El patron de subconsulta en el FROM para vistas ya esta entendido y aplicado correctamente.

Checklist para el dia del parcial

Ejercicio	Que verificar antes de entregar

Vista	Sin BEGIN/END. Subconsulta agrupada en el FROM. CASE afuera sobre el resultado calculado. No repeti
Funcion	RETURNS del tipo correcto (money/int/varchar). DECLARE @retorno. RETURN al final. Escribir la consult
Procedimiento	Validar parametros primero con RETURN. WHERE con el parametro en el UPDATE/DELETE. @@ROWC
Trigger AFTER	Para auditar/registrar. IF UPDATE(col) si aplica. deleted=viejo, inserted=nuevo. WHERE para emparejar p
Trigger INSTEAD OF	Para bloquear. IF EXISTS buscando lo MALO. RAISERROR si hay error. INSERT/UPDATE manual en el

---

Mucho exito en el parcial.