

Store Procedures, Functions and Triggers

CE3101 - Bases de Datos



Disclaimer / Descargo de Responsabilidad

Esta presentación corresponde a una guía usada por el profesor durante las clases. La misma ha sido modificada para ser utilizado en el modelo de cursos asistidos por tecnología. No es una versión final, por lo que la misma podría requerir todavía hacer algunos ajustes. Para aspectos de evaluación esta presentación es solo una guía, por lo que el estudiante debe profundizar con el material de lectura asignado y lo discutido en clases para aspectos de evaluación.

This presentation corresponds to a guide material used by the professor during classes. It has been modified to be used in the model of technology-assisted courses. It is not a final version, so it may still require some adjustments. For evaluation aspects, this presentation is only a guide, so the student should delve with the assigned reading material and what has been discussed in class.

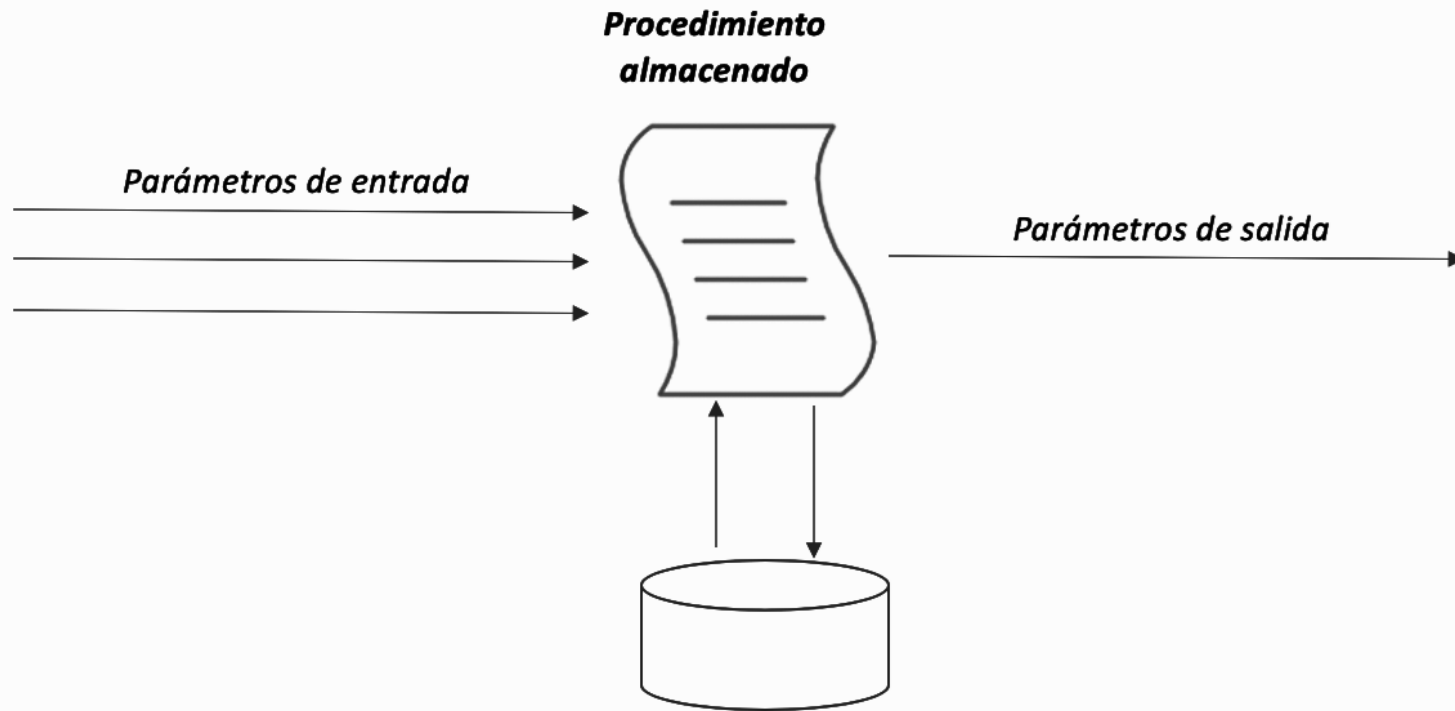
Motivación

**“La disciplina tarde o temprano
vencerá a la inteligencia”**

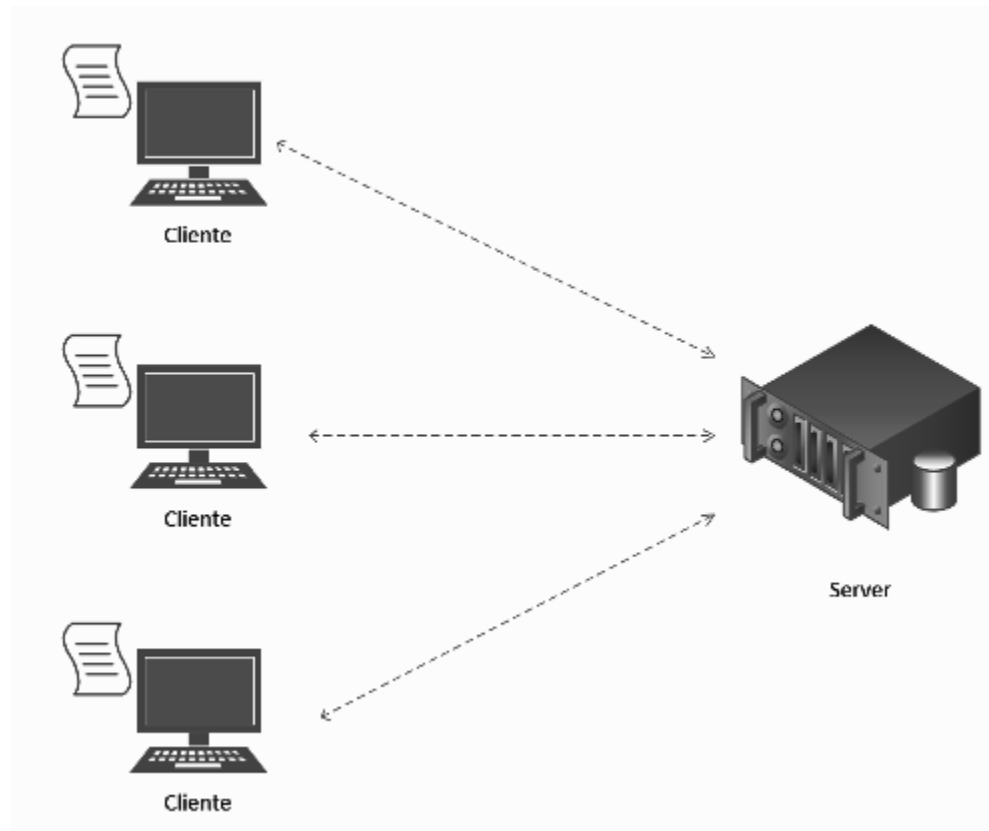
Introducción

- Uno de los objetos programables disponibles en un DBMS son los procedimientos almacenados (Store Procedures).
- Se pueden definir como rutinas guardadas en el servidor que encapsulan código SQL.
- Al igual que un procedimiento o método de un lenguaje de alto nivel como Java o C#, los procedimientos almacenados pueden recibir parámetros y devolver uno o más resultados.

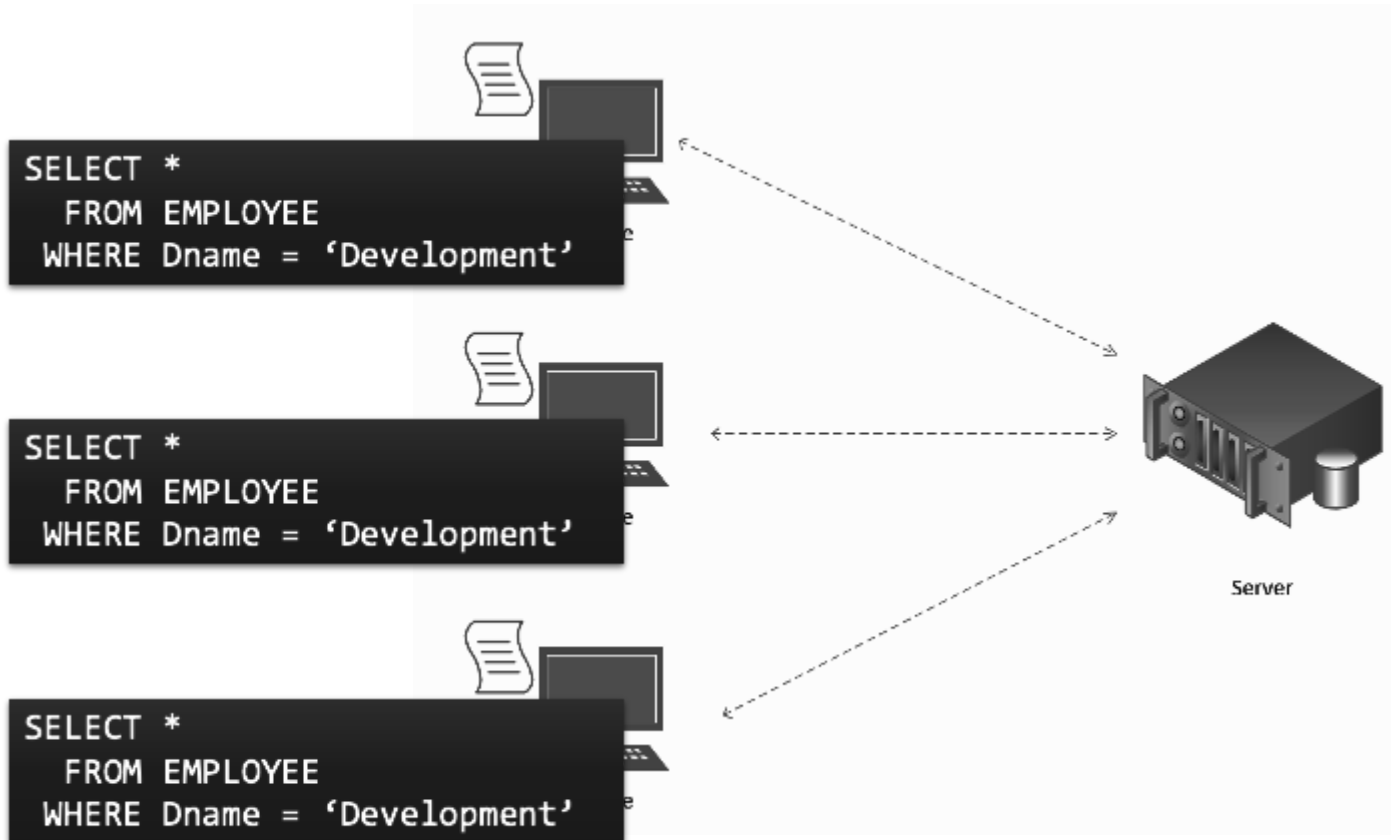
Introducción



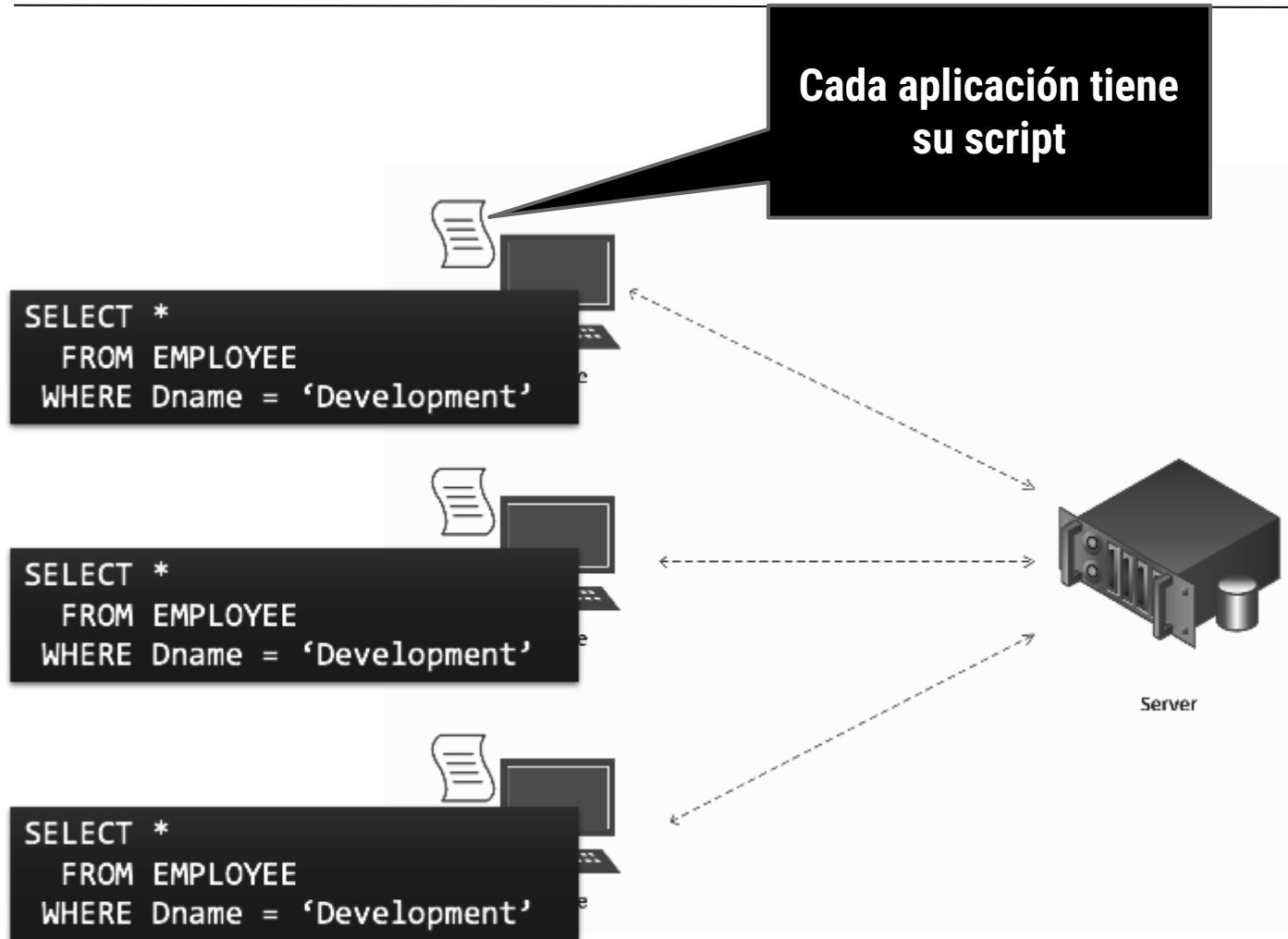
En un esquema tradicional



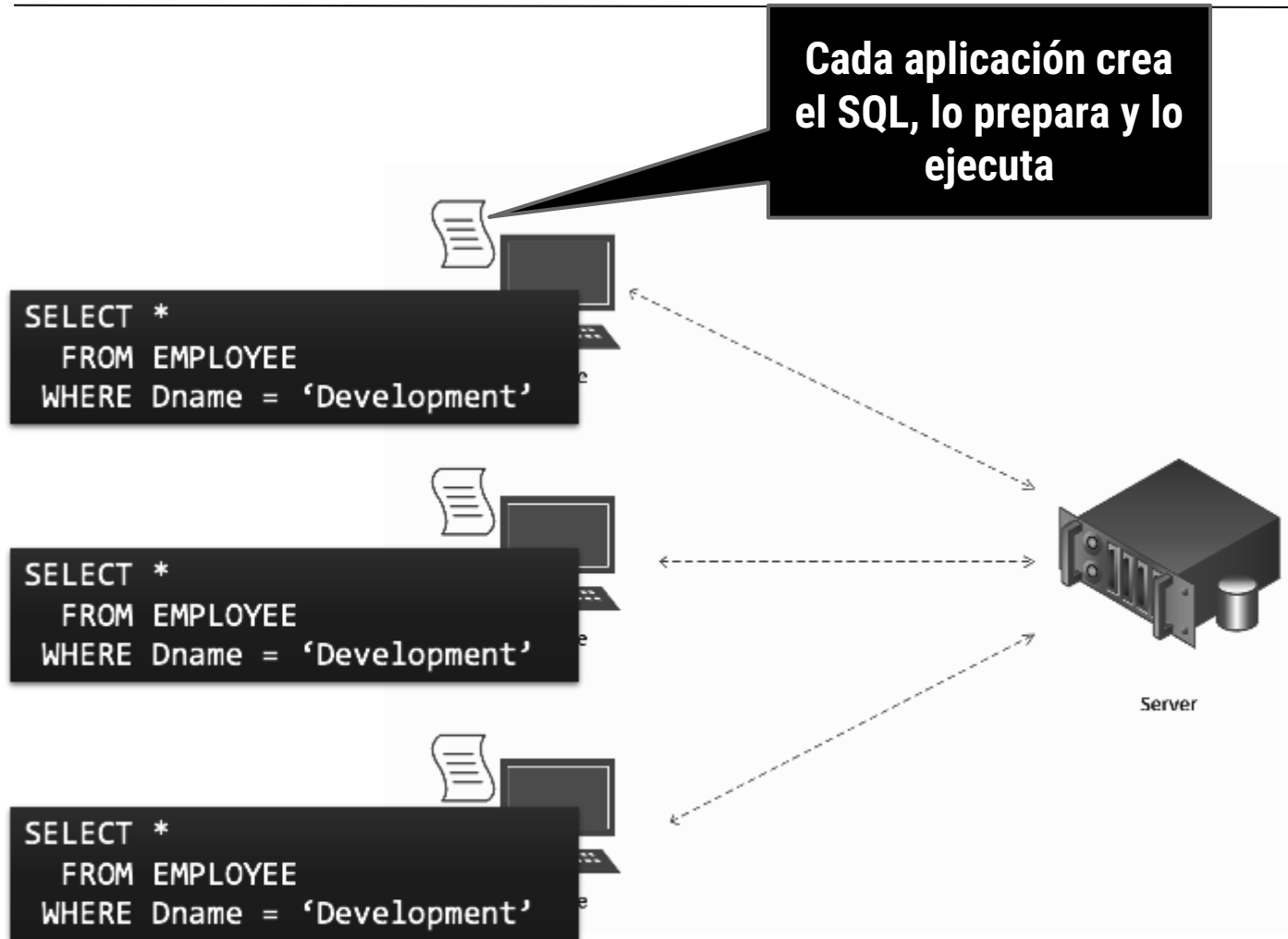
En un esquema tradicional



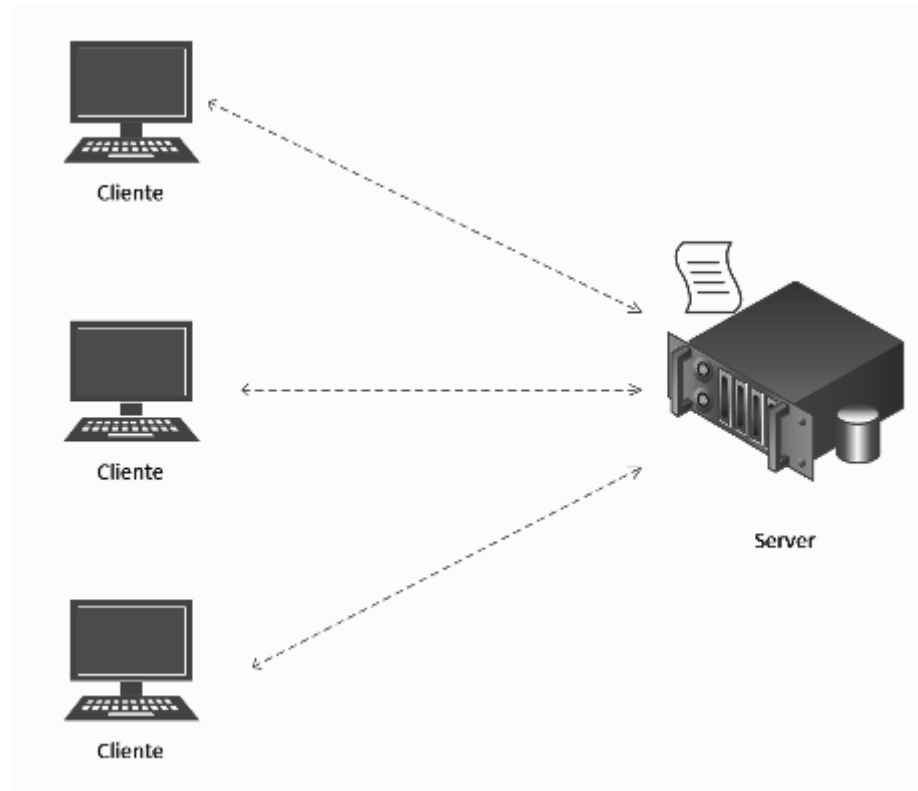
En un esquema tradicional



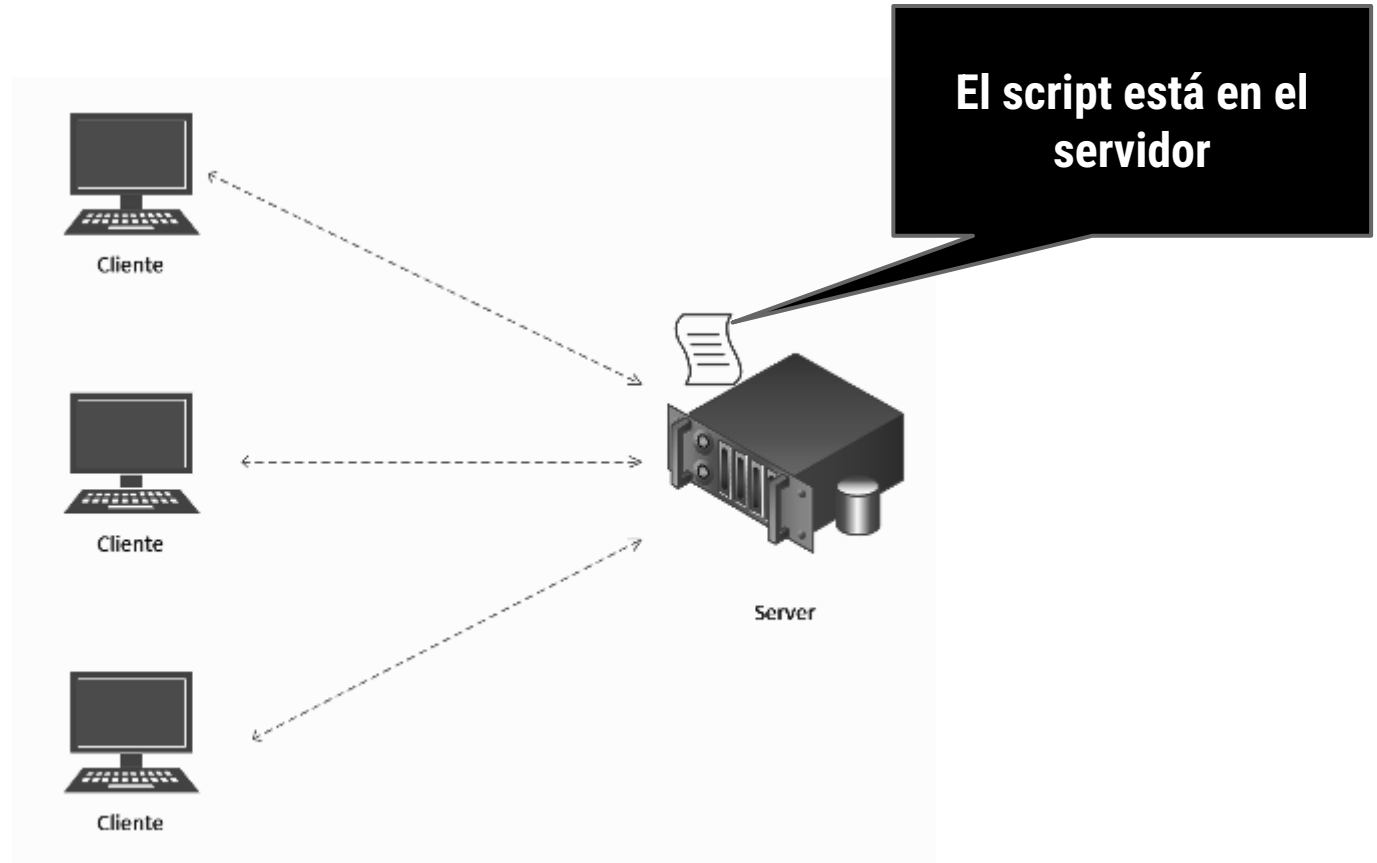
En un esquema tradicional



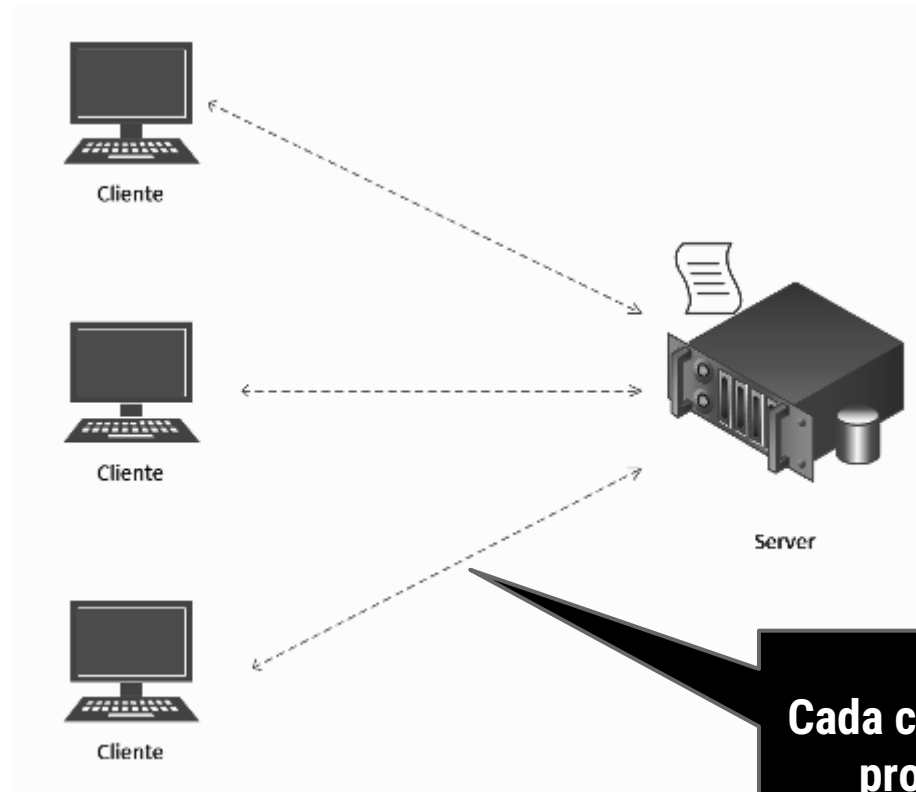
Utilizando Procedimientos Almacenados...



Utilizando Procedimientos Almacenados...



Utilizando Procedimientos Almacenados...



Procedimientos Almacenados (Características)

- No pueden ser utilizados en expresiones.
- No retornan valor.
- Pueden retornar múltiples parámetros de salida.

Por qué utilizar procedimientos almacenados?

→ Encapsulamiento

- ◆ Si se necesita realizar un cambio, **únicamente se debe hacer en un solo lugar** y el procedimiento se actualiza para todos los usuarios del mismo.
- ◆ Las aplicaciones y programadores no necesariamente tienen acceso al código del procedimiento.

→ Reutilización de código

- ◆ Las aplicaciones se limitan a invocar un procedimiento **sin tener que duplicar el código** una y otra vez, reduciendo las inconsistencias y la probabilidad de errores.

Por qué utilizar procedimientos almacenados?

→ Mayor seguridad

- ◆ Es posible darle autorización a los usuarios para ejecutar el procedimiento **sin tener que darles autorización sobre las tablas** que el procedimiento utiliza.

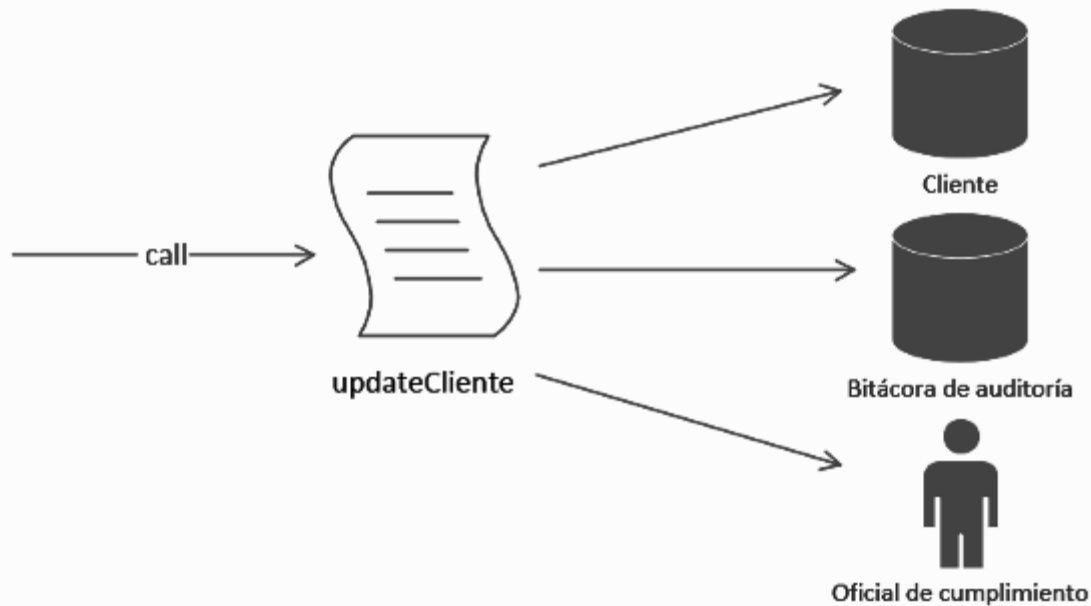
Por qué utilizar procedimientos almacenados?

→ Mayor seguridad

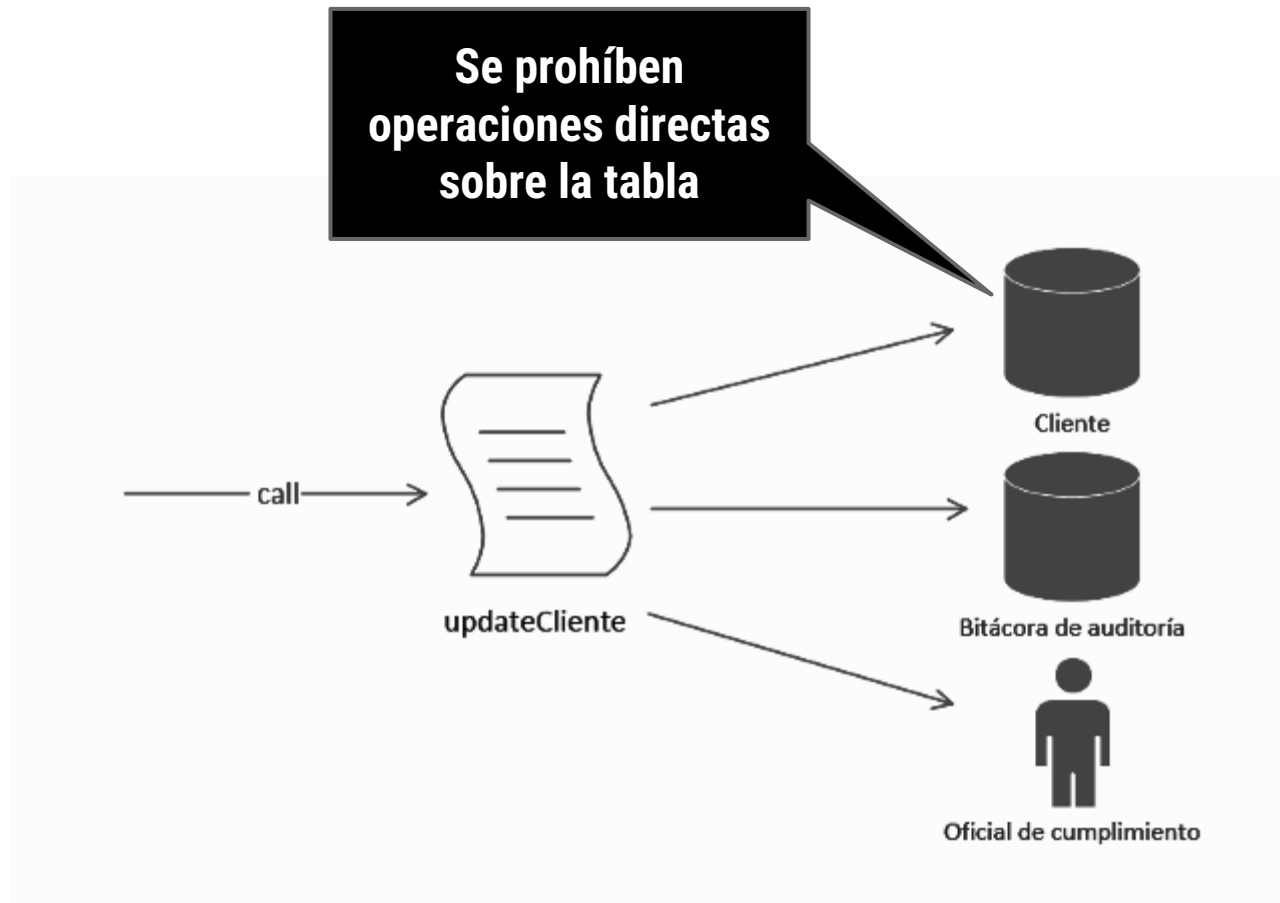
- ◆ Es posible darle autorización a los usuarios para ejecutar el procedimiento **sin tener que darles autorización sobre las tablas** que el procedimiento utiliza.

En qué escenario práctico aplica esto?

Por qué utilizar procedimientos almacenados?



Por qué utilizar procedimientos almacenados?



Por qué utilizar procedimientos almacenados?



Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ El procedimiento almacenado puede **incluir código necesario para manejar cualquier error** que pueda ocurrir durante la ejecución del script.
- ◆ De lo contrario, la aplicación tendría que manejar los errores y dependería de la pericia del programador, lo bien o mal, que estos se manejen.
- ◆ Cuando se produce un error, el DBMS envía un mensaje de error que contiene información específica que incluye:

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ El procedimiento almacenado puede **incluir código necesario para manejar cualquier error** que pueda ocurrir durante la ejecución del script.
- ◆ De lo contrario, la aplicación tendría que manejar los errores y dependería de la pericia del programador, lo bien o mal, que estos se manejen.
- ◆ Cuando se produce un error, el DBMS envía un mensaje de error que contiene información específica que incluye:

Número de error:

Identifica inequívocamente el error generado

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ El procedimiento almacenado puede **incluir código necesario para manejar cualquier error** que pueda ocurrir durante la ejecución del script.
- ◆ De lo contrario, la aplicación tendría que manejar los errores y dependería de la pericia del programador, lo bien o mal, que estos se manejen.
- ◆ Cuando se produce un error, el DBMS envía un mensaje de error que contiene información específica que incluye:

Nivel de la severidad:

Es un valor entre 0 y 25 que indica el grado de severidad del error generado

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ El procedimiento almacenado puede **incluir código necesario para manejar cualquier error** que pueda ocurrir durante la ejecución del script.
- ◆ De lo contrario, la aplicación tendría que manejar los errores y dependería de la pericia del programador, lo bien o mal, que estos se manejen.
- ◆ Cuando se produce un error, el DBMS envía un mensaje de error que contiene información específica que incluye:

Número de línea:

Identifica la línea en el procedimiento, función, trigger o batch en el que el error ocurrió

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ El procedimiento almacenado puede **incluir código necesario para manejar cualquier error** que pueda ocurrir durante la ejecución del script.
- ◆ De lo contrario, la aplicación tendría que manejar los errores y dependería de la pericia del programador, lo bien o mal, que estos se manejen.
- ◆ Cuando se produce un error, el DBMS envía un mensaje de error que contiene información específica que incluye:

Nombre del procedimiento:

Indica el nombre del procedimiento o trigger en el que ocurrió el error

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ El procedimiento almacenado puede **incluir código necesario para manejar cualquier error** que pueda ocurrir durante la ejecución del script.
- ◆ De lo contrario, la aplicación tendría que manejar los errores y dependería de la pericia del programador, lo bien o mal, que estos se manejen.
- ◆ Cuando se produce un error, el DBMS envía un mensaje de error que contiene información específica que incluye:

Mensaje del error:
Descripción textual del error ocurrido

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ Es posible consultar la lista de errores que puede enviar el DBMS. Por ejemplo, en SQL Server, los errores posibles se pueden consultar:

```
SELECT message_id, language_id, severity,  
       is_event_logged, text  
FROM sys.messages  
WHERE language_id = 1033
```

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ Se pueden agregar mensajes personalizados a dicha tabla a través del procedimiento almacenado **sp_addmessage** (SQL Server):

```
sp_addmessage [ @msgnum= ] msg_id , [ @severity=
] severity , [ @msgtext= ] 'msg'
    [ , [ @lang= ] 'Language' ]
    [ , [ @with_log= ] { 'TRUE' | 'FALSE' } ]
    [ , [ @replace= ] 'replac' ]
```

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ Se pueden agregar mensajes personalizados a dicha tabla a través del procedimiento almacenado **sp_addmessage** (SQL Server):

```
sp_addmessage [ @msgnum= ] msg_id , [ @severity=  
] severity , [ @msgtext= ] 'msg'  
    [ , [ @lang= ] 'Language' ]  
    [ , [ @with_log= ] { 'TRUE' | 'FALSE' } ]  
    [ , [ @replace= ] 'replacE' ]
```

Para lanzar errores personalizados, se usa **THROW** o **RAISERROR**

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ Se pueden agregar mensajes personalizados a dicha tabla a través del procedimiento almacenado **sp_addmessage** (SQL Server):

```
EXECUTE sp_addmessage 53431, 19,  
    'You have specified an invalid product code';
```

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

```
BEGIN TRY
    SELECT 5/0
END TRY
BEGIN CATCH
    BEGIN
        SELECT ERROR_NUMBER() AS ErrorNum,
               ERROR_SEVERITY() AS Severity,
               ERROR_STATE() AS State,
               ERROR_LINE() AS LineNum,
               ERROR_PROCEDURE() AS ProcName,
               ERROR_MESSAGE() AS MessageText

    END
END CATCH
```

```
BEGIN TRY
    RAISERROR('This error was raised in the TRY
block.', 12, 1)
END TRY
BEGIN CATCH
    BEGIN
        SELECT ERROR_NUMBER() AS ErrorNum,
               ERROR_SEVERITY() AS Severity,
               ERROR_STATE() AS State,
               ERROR_LINE() AS LineNum,
               ERROR_PROCEDURE() AS ProcName,
               ERROR_MESSAGE() AS MessageText

    END
END CATCH
```

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

```
BEGIN TRY
    SELECT 5/0
END TRY
BEGIN CATCH
    BEGIN
        SELECT ERROR_NUMBER() AS ErrorNum,
               ERROR_SEVERITY() AS Severity,
               ERROR_STATE() AS State,
               ERROR_LINE() AS LineNum,
               ERROR_PROCEDURE() AS ProcName,
               ERROR_MESSAGE() AS MessageText
    END
END CATCH
```

```
BEGIN TRY
    RAISERROR('This error was raised in the TRY
block.', 12, 1)
END TRY
BEGIN CATCH
    BEGIN
        SELECT ERROR_NUMBER() AS ErrorNum,
               ERROR_SEVERITY() AS Severity,
               ERROR_STATE() AS State,
               ERROR_LINE() AS LineNum,
               ERROR_PROCEDURE() AS ProcName,
               ERROR_MESSAGE() AS MessageText
    END
END CATCH
```

**Estas funciones
retornan NULL si se
les llama fuera de un
CATCH**

Por qué utilizar procedimientos almacenados?

→ Manejo de errores

- ◆ La variable @@ERROR se puede consultar para determinar si ocurrió un error después de ejecutar una sentencia SQL:

```
GO
INSERT Sales.Client(ClientID) VALUES(500);
INSERT Sales.Client(ClientID) VALUES(500);
SELECT @@ERROR AS ErrorNum
```


Por qué utilizar procedimientos almacenados?

→ Manejo de errores

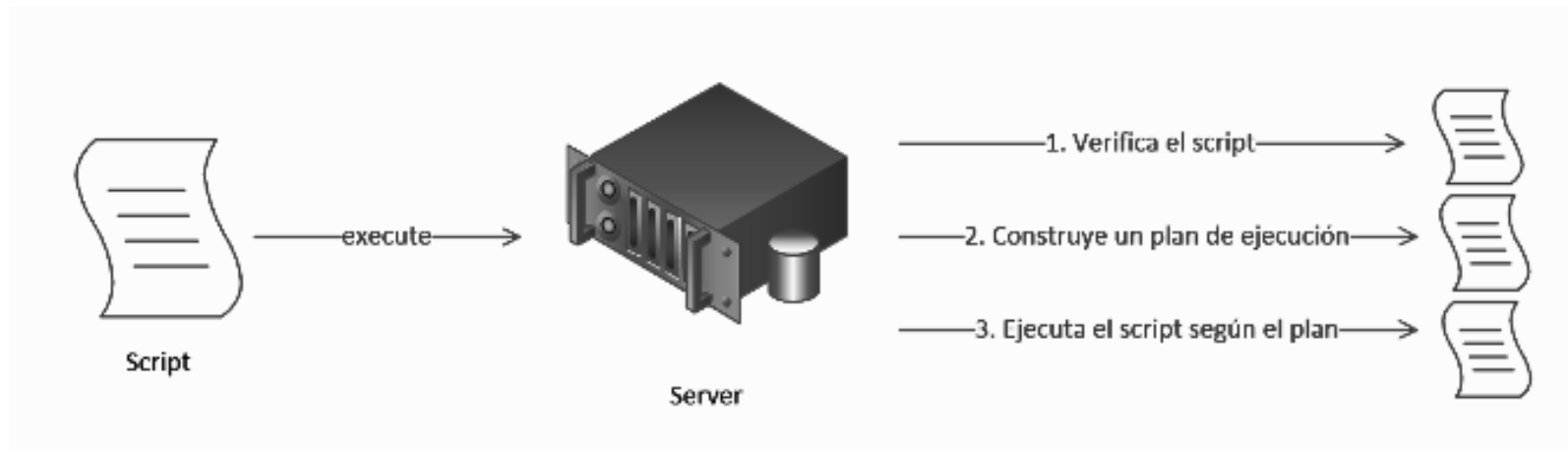
- ◆ La variable @@ERROR se puede consultar para determinar si ocurrió un error después de ejecutar una sentencia SQL:

```
GO
INSERT Sales.Client(ClientID) VALUES(500);
INSERT Sales.Client(ClientID) VALUES(500);
SELECT @@ERROR AS ErrorNum
```

**Se inicializa después
de cada sentencia
ejecutada**

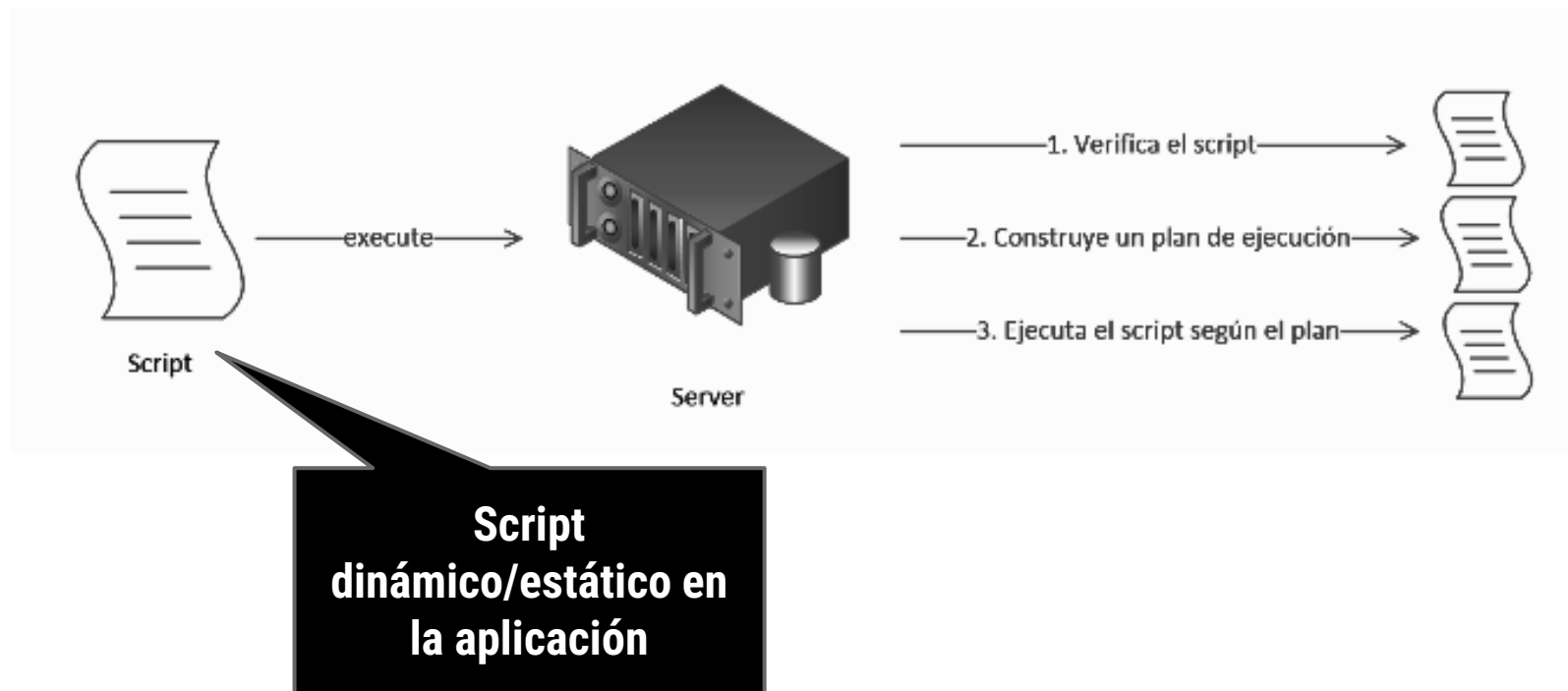
Por qué utilizar procedimientos almacenados?

→ Mejoría en rendimiento



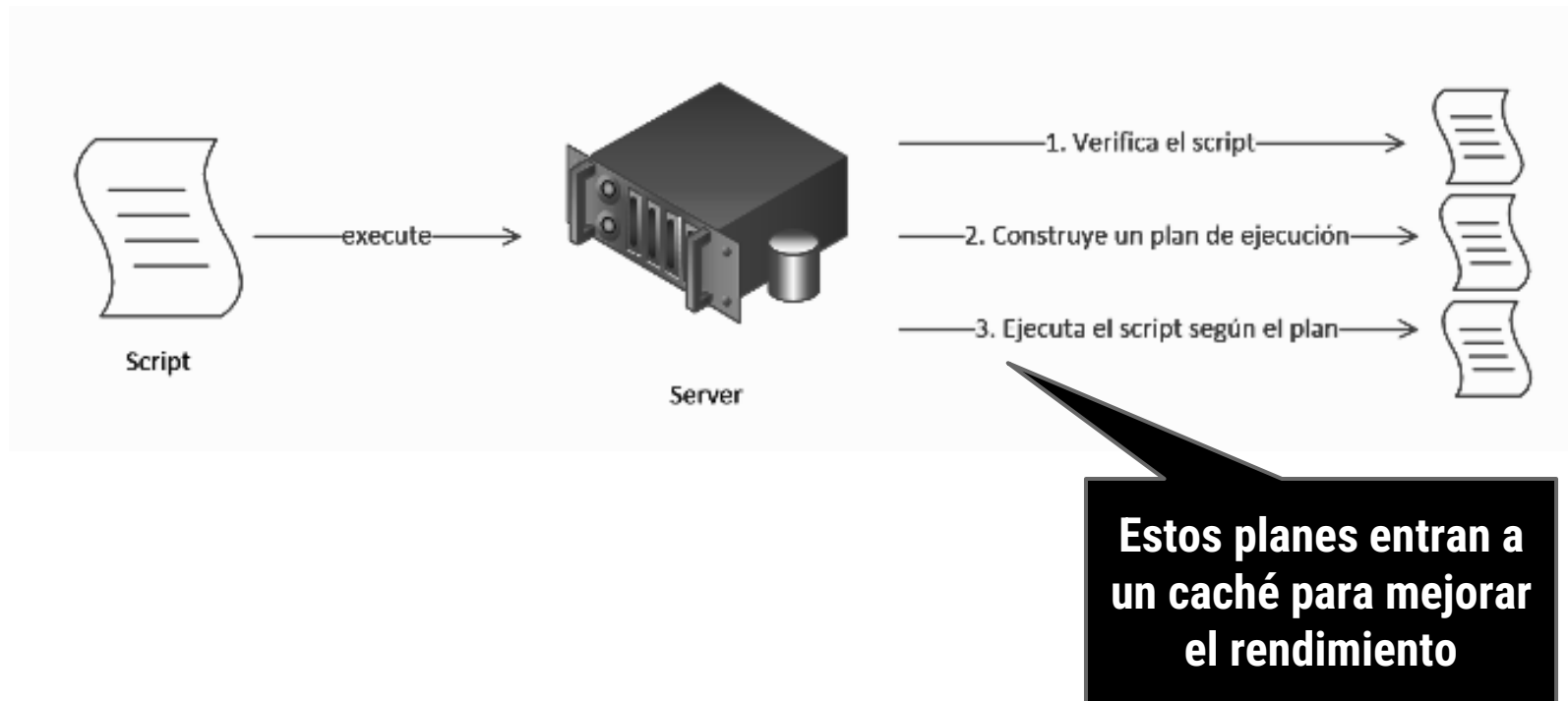
Por qué utilizar procedimientos almacenados?

→ Mejoría en rendimiento



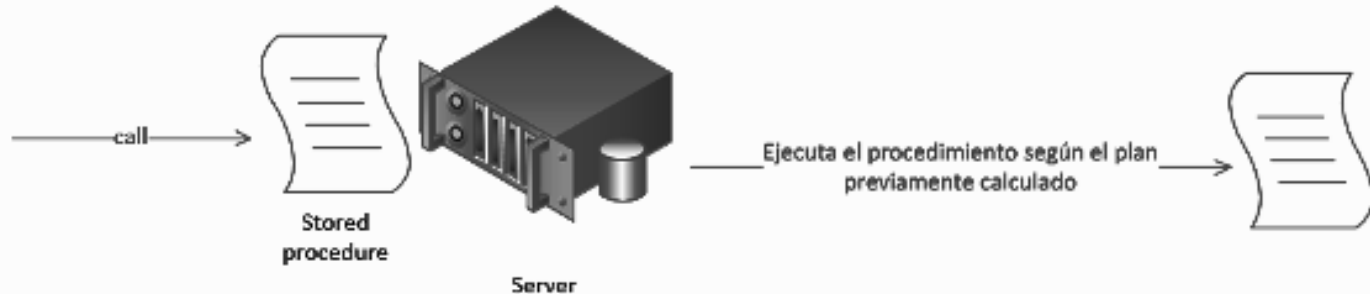
Por qué utilizar procedimientos almacenados?

→ Mejoría en rendimiento



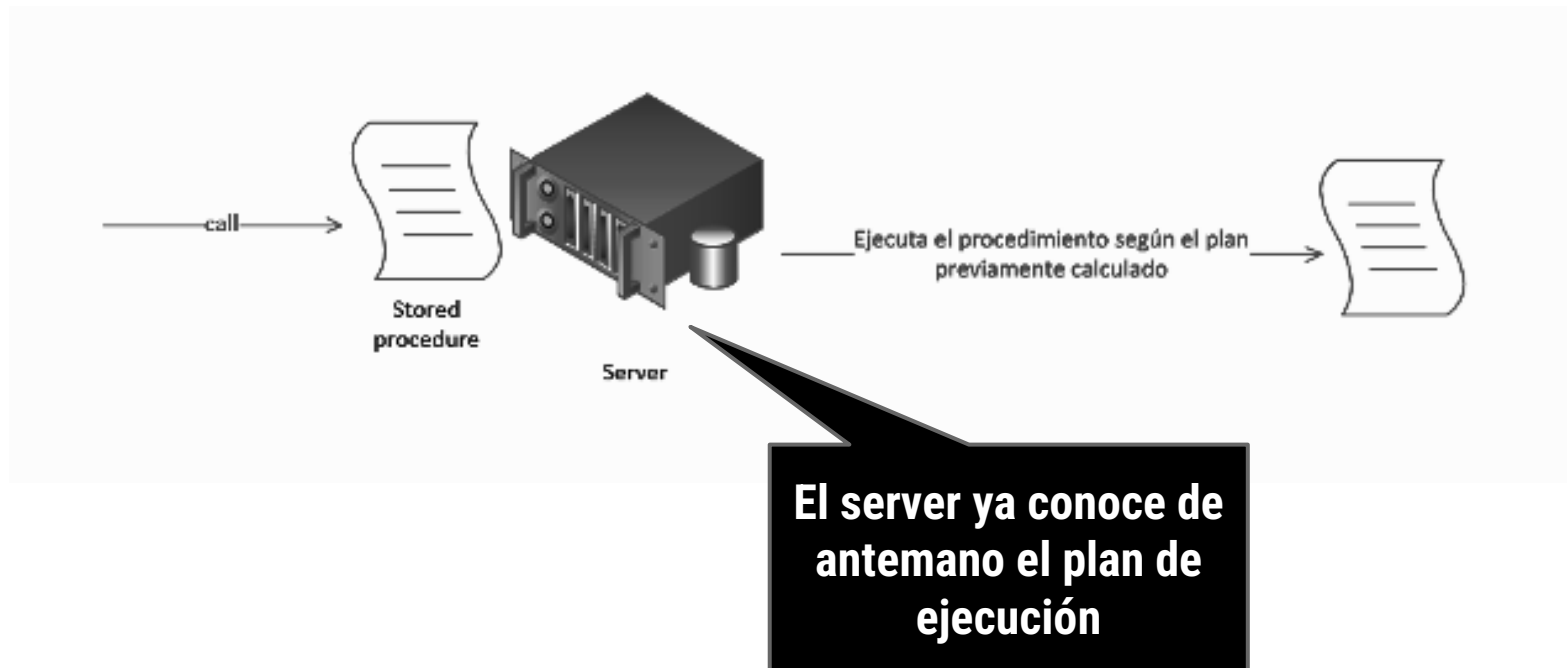
Por qué utilizar procedimientos almacenados?

→ Mejoría en rendimiento



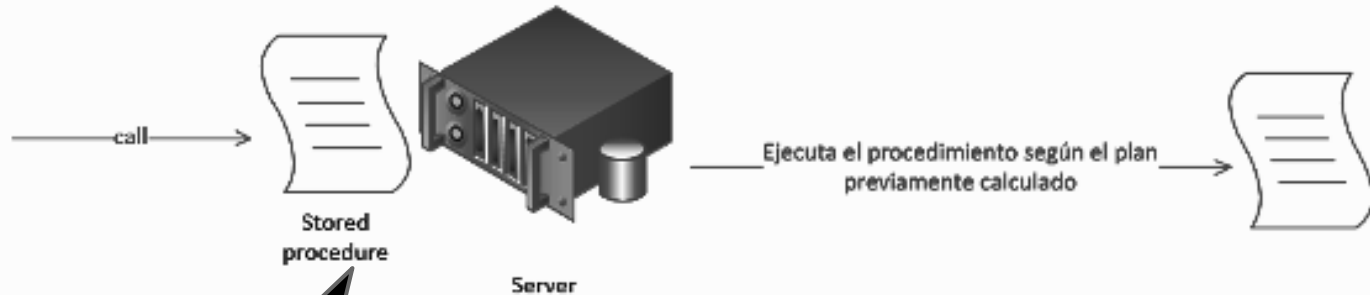
Por qué utilizar procedimientos almacenados?

→ Mejoría en rendimiento



Por qué utilizar procedimientos almacenados?

→ Mejoría en rendimiento



**Previamente
verificado y compilado**

Por qué utilizar procedimientos almacenados?

→ Reducción del tráfico de red

- ◆ En vez de tener que enviar múltiples scripts de múltiples clientes a través de la red, simplemente se envía una petición de múltiples clientes

Tipos de Store Procedures

USER DEFINED

- Son procedimientos definidos por el usuario utilizando el comando `CREATE PROCEDURE`.
- Pueden ser definidos con SQL o en un lenguaje de alto nivel (no todos los motores lo soportan).

Tipos de Store Procedures

USER DEFINED

- Son procedimientos definidos por el usuario utilizando el comando CREATE PROCEDURE.
- Pueden ser definidos con SQL o en un lenguaje de alto nivel (no todos los motores lo soportan).

SYSTEM

- Procedimientos almacenados provistos por el motor y que cumplen una gran variedad de funciones. En SQL Server se nombran con *sp_* al inicio.

Tipos de Store Procedures

USER DEFINED

- Son procedimientos definidos por el usuario utilizando el comando CREATE PROCEDURE.
- Pueden ser definidos con SQL o en un lenguaje de alto nivel (no todos los motores lo soportan).

SYSTEM

- Procedimientos almacenados provistos por el motor y que cumplen una gran variedad de funciones. En SQL Server se nombran con *sp_* al inicio.

**Procedimientos
definidos por el
usuario no deben
empezar con sp_**

Consideraciones al crear un procedimiento

- Deben especificarse los parámetros de entrada y parámetros de salida. Los parámetros de salida corresponden a el o los resultados del procedimiento.
- Solo se pueden especificar 2100 parámetros.

Consideraciones al crear un procedimiento

- Deben especificarse los parámetros de entrada y parámetros de salida. Los parámetros de salida corresponden a el o los resultados del procedimiento.
- Solo se pueden especificar 2100 parámetros.



**Qué hago si necesito
2101 parámetros?**

Consideraciones al crear un procedimiento

- Deben especificarse los parámetros de entrada y parámetros de salida. Los parámetros de salida corresponden a el o los resultados del procedimiento.
- Solo se pueden especificar 2100 parámetros.

**Qué hago si necesito
2101 parámetros?**

**Indicador de que algo
anda mal con el
procedimiento!**

Consideraciones al crear un procedimiento

- Deben especificarse los parámetros de entrada y parámetros de salida. Los parámetros de salida corresponden a el o los resultados del procedimiento.
- Solo se pueden especificar 2100 parámetros.
- Los parámetros son únicos para cada procedimiento, entonces se pueden repetir nombres de parámetros en diferentes procedimientos.
- Los nombres de parámetros empiezan con @

Consideraciones al crear un procedimiento

- Deben ser pequeños, entre más largos pueden ocurrir deadlocks.
- No se pueden usar los siguientes comandos:
 - ◆ CREATE AGGREGATE
 - ◆ CREATE SCHEMA
 - ◆ SET SHOWPLAN_TEXT
 - ◆ CREATE DEFAULT
 - ◆ ALTER TRIGGER
 - ◆ SET SHOWPLAN_XML

Consideraciones al crear un procedimiento

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name
    [ { @parameter [ type_schema_name. ] data_type }
      [ VARYING ] [ = default ] [ OUT | OUTPUT
    ] [ READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
AS { [BEGIN ] <sql_statement> [;][ ...n ] [END] }

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

Consideraciones al crear un procedimiento

```
USE AdventureWorks2012
GO
CREATE PROCEDURE Sales.GetEmployeeSalesYTD
    @SalesPerson nvarchar(50) = NULL,
    @SalesYTD money OUTPUT
AS
    IF @SalesPerson IS NULL
        BEGIN
            PRINT 'You must give a name'
            RETURN(1)
        END
    ELSE
        BEGIN
            SELECT @SalesYTD = SalesYTD
            FROM Sales.SalesPerson AS sp
            JOIN Person.Person AS p
            ON p.BusinessEntityID = sp.BusinessEntityID
            WHERE LastName = @SalesPerson;
        END
GO
```

Consideraciones al crear un procedimiento

- Cuando se construye un índice para una tabla utilizada por un procedimiento, dicho procedimiento no va a aprovechar el índice a menos que se recompile.
- Para recompilar un procedimiento:

```
USE AdventureWorks2012
GO
DECLARE @diff smallint
EXECUTE Sales.SubtractNumbers 10, 5, @diff OUTPUT
WITH RECOMPILE
SELECT @diff
GO
```

Consideraciones al crear un procedimiento

- Cuando se construye un índice para una tabla utilizada por un procedimiento, dicho procedimiento no va a aprovechar el índice a menos que se recompile.
- Para recompilar un procedimiento:

```
USE AdventureWorks2012
GO
DECLARE @diff smallint
EXECUTE Sales.SubtractNnumbers 10, 5, @diff OUTPUT
WITH RECOMPILE
SELECT @diff
GO
```

Funciones

Funciones en el contexto de SQL

- Es un objeto programable.
- Tiene el propósito de encapsular lógica que calcula algo útil, basado posiblemente en parámetros de entrada y que retorna un valor.
- Se conocen como UDF (User-Defined Functions).
- Hay dos tipos de funciones:
 - ◆ **Scalar UDF**: retornan un valor único.
 - ◆ **Table-valued UDF**: retornan una tabla (UDTF User Defined Table Functions)

Funciones

- UDFs no pueden tener efectos secundarios
 - ◆ No se permite utilizar funciones que tengan efectos secundarios.
 - ◆ No pueden modificar el estado de la base de datos (insert, update, delete).

- Otras restricciones:
 - ◆ No pueden retornar múltiples result set.
 - ◆ No soporta TRY...CATCH
 - ◆ No pueden llamar procedimientos almacenados
 - ◆ No pueden utilizar SQL Dinámico o tablas temporales. Si pueden usar Common Table Expressions (CTE)

Funciones (Ejemplos)

```
IF OBJECT_ID (N'dbo.ufnGetInventoryStock', N'FN') IS NOT NULL
    DROP FUNCTION ufnGetInventoryStock;
GO
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
-- Returns the stock level for the product.
BEGIN
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID
        AND p.LocationID = '6';
    IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
GO
```


Funciones (Ejemplos)

Es una función escalar

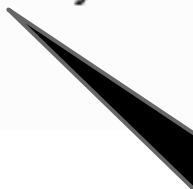
```
IF OBJECT_ID('dbo.ufnGetInventoryStock') IS NOT NULL
    DROP FUNCTION dbo.ufnGetInventoryStock;
GO
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
-- Returns the stock level for the product.
BEGIN
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID
        AND p.LocationID = '6';
    IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
GO
```

Funciones (Ejemplos)

```
SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS CurrentSupply  
FROM Production.Product  
WHERE ProductModelID BETWEEN 75 and 80;
```

Funciones (Ejemplos)

```
SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS CurrentSupply  
FROM Production.Product  
WHERE ProductModelID BETWEEN 75 and 80;
```



**Se puede utilizar en la
lista de columnas del
SELECT**

Funciones (Ejemplos)

```
IF OBJECT_ID (N'Sales.ufn_SalesByStore', N'IF') IS NOT NULL
    DROP FUNCTION Sales.ufn_SalesByStore;
GO
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
(
    SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'
    FROM Production.Product AS P
    JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
    JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID
    JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
    WHERE C.StoreID = @storeid
    GROUP BY P.ProductID, P.Name
);
```

Funciones (Ejemplos)

```
IF OBJECT_ID (N'Sales.ufn_SalesByStore', N'IF') IS NOT NULL
    DROP FUNCTION Sales.ufn_SalesByStore;
GO
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
(
    SELECT P.ProductID, P.Name, SUM(SD.Quantity * SD.UnitPrice) AS 'Total'
    FROM Production.Product AS P
    JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
    JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID = SD.SalesOrderID
    JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
    WHERE C.StoreID = @storeid
    GROUP BY P.ProductID, P.Name
);
```





Table-valued UDF

Funciones (Ejemplos)

```
SELECT * FROM Sales.ufn_SalesByStore (602);
```

Funciones (Ejemplos)

```
SELECT * FROM Sales.ufn_SalesByStore (602);
```



**La tabla es retornada
por la función**

Triggers

Qué es un Trigger?

- Es un tipo especial de procedimiento almacenado. No se puede ejecutar directamente.
- Está asociado a un evento (event-driven) que lo ejecuta.
- Cuando el evento asociado se materializa, el trigger se dispara y el código asociado se ejecuta.
- El código asociado al trigger se puede escribir en SQL o en un lenguaje de alto nivel.
- Pueden cancelar el evento, evitando que la sentencia se ejecute.

Con cuáles eventos se puede asociar un Trigger?

- Depende del DBMS.
- Por ejemplo, SQL Server permite asociar Triggers con eventos DDL y DML.
 - ◆ INSERT, DELETE, UPDATE (DML)
 - ◆ CREATE TABLE, DROP TABLE, entre otros (DDL)

Triggers DDL

- Se disparan por eventos DDL.
- Estos eventos corresponden a sentencias SQL que comienza con las palabras reservadas CREATE, ALTER, DROP, GRANT, DENY, REVOKE, o UPDATE STATISTICS.
- Se pueden utilizar para los siguientes propósitos:
 - ◆ Evitar cambios en el esquema.
 - ◆ Reaccionar en respuesta a un cambio en el esquema.
 - ◆ Registrar cambios o eventos en el esquema.
- Los Triggers DDL siempre se disparan después del evento.

Triggers DDL (Ejemplo)

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
    PRINT 'You must disable Trigger "safety" to drop or alter tables!'
    ROLLBACK;
```


```
IF EXISTS (SELECT * FROM sys.server_triggers
    WHERE name = 'ddl_trig_database')
DROP TRIGGER ddl_trig_database
ON ALL SERVER;
GO
CREATE TRIGGER ddl_trig_database
ON ALL SERVER
FOR CREATE_DATABASE
AS
    PRINT 'Database Created.'
    SELECT EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(max)')
GO
DROP TRIGGER ddl_trig_database
ON ALL SERVER;
GO
```

Triggers DML

- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.
- Un trigger puede ejecutarse en uno de los dos tiempo:
 - ◆ AFTER
 - ◆ BEFORE
 - ◆ INSTEAD OF

Triggers DML

- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.
- Un trigger puede ejecutarse en uno de los dos tiempo:
 - ◆ AFTER
 - ◆ BEFORE
 - ◆ INSTEAD OF



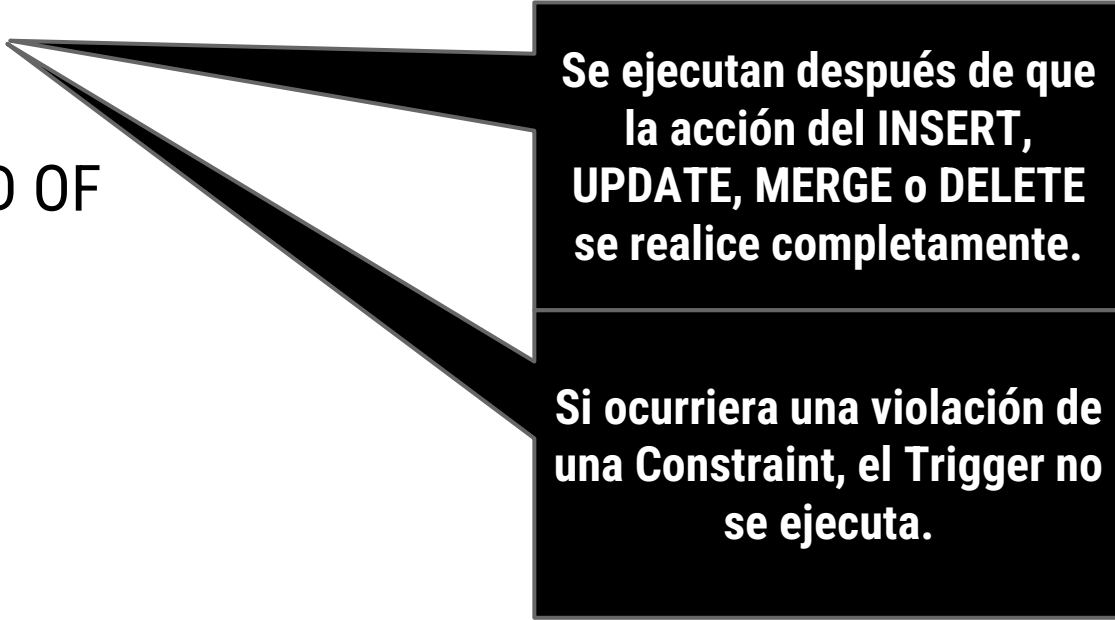
Se ejecutan después de que la acción del INSERT, UPDATE, MERGE o DELETE se realice completamente.

Triggers DML

- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.

→ Un trigger puede ejecutarse en uno de los dos tiempo:

- ◆ AFTER
- ◆ BEFORE
- ◆ INSTEAD OF



Se ejecutan después de que la acción del INSERT, UPDATE, MERGE o DELETE se realice completamente.

Si ocurriera una violación de una Constraint, el Trigger no se ejecuta.

Triggers DML


- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.
- Un trigger puede ejecutarse en uno de los dos tiempo:
 - ◆ AFTER
 - ◆ BEFORE
 - ◆ INSTEAD OF



Se ejecutan antes de que la acción del INSERT, UPDATE, MERGE o DELETE se realice completamente.

Triggers DML

- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.
- Un trigger puede ejecutarse en uno de los dos tiempo:
 - ◆ AFTER
 - ◆ BEFORE
 - ◆ INSTEAD OF



Se ejecutan antes de que la acción del INSERT, UPDATE, MERGE o DELETE se realice completamente.

Permiten validar los datos de entrada antes de que se realice la acción.

Triggers DML

→ Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.

→ Eventos DML incluyen INSERT, DE

→ Un trigger puede ejecutarse antes o después de la acción

- ◆ AFTER
- ◆ BEFORE
- ◆ INSTEAD OF

SQL Server no los soporta

Se ejecutan antes de que la acción del INSERT, UPDATE, MERGE o DELETE se realice completamente.

Permiten validar los datos de entrada antes de que se realice la acción.

Triggers DML

- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.
- Un trigger puede ejecutarse en uno de los dos tiempo:
 - ◆ AFTER
 - ◆ BEFORE
 - ◆ INSTEAD OF



**Se ejecutan en vez de la
acción INSERT, DELETE o
UPDATE**

Triggers DML

- Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.
- Eventos DML incluyen INSERT, DELETE o UPDATE.
- Un trigger puede ejecutarse en uno de los dos tiempo:
 - ◆ AFTER
 - ◆ BEFORE
 - ◆ INSTEAD OF



Se ejecutan en vez de la acción INSERT, DELETE o UPDATE

No se ejecuta la acción, se ejecuta el Trigger

Triggers DML

→ Triggers que se ejecutan al ocurrir un evento DML que afecta una tabla o vista definida en el Trigger.

→ Eventos DML incluyen INSERT, DELETE o UPDATE

→ Un trigger puede ejecutarse en tres tipos de eventos:

- ◆ AFTER
- ◆ BEFORE
- ◆ INSTEAD OF

Similar al BEFORE

Se ejecutan en vez de la acción INSERT, DELETE o UPDATE

No se ejecuta la acción, se ejecuta el Trigger

Trigger DML (Ejemplo)

```
CREATE TRIGGER reminder2
ON Sales.Customer
AFTER INSERT, UPDATE, DELETE
AS
    EXEC msdb.dbo.sp_send_dbmail
        @profile_name = 'AdventureWorks2012 Administrator',
        @recipients = 'danw@Adventure-Works.com',
        @body = 'Don''t forget to print a report for the sales force.',
        @subject = 'Reminder';

GO
```

Trigger DML (Ejemplo)

```
CREATE TRIGGER TransactionBeforeTrigger BEFORE INSERT ON TransactionTable
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
BEGIN
    DECLARE newmonth SMALLINT;
    SET newmonth = MONTH(new_row.DateOfTransaction);
    IF newmonth < 4 THEN
        SET new_row.FiscalQuarter=3;
    ELSEIF newmonth < 7 THEN
        SET new_row.FiscalQuarter=4;
    ELSEIF newmonth < 10 THEN
        SET new_row.FiscalQuarter=1;
    ELSE
        SET new_row.FiscalQuarter=2;
    END IF;
END
```

Trigger DML (Ejemplo)

```
CREATE TRIGGER IO_Trig_INS_Employee ON Employee
INSTEAD OF INSERT
AS
BEGIN
SET NOCOUNT ON
-- Check for duplicate Person. If there is no duplicate, do an insert.
IF (NOT EXISTS (SELECT P.SSN
                FROM Person P, inserted I
                WHERE P.SSN = I.SSN))
    INSERT INTO Person
        SELECT SSN,Name,Address,Birthdate
        FROM inserted
ELSE
-- Log an attempt to insert duplicate Person row in PersonDuplicates table.
    INSERT INTO PersonDuplicates
        SELECT SSN,Name,Address,Birthdate,SUSER_SNAME(),GETDATE()
        FROM inserted
-- Check for duplicate Employee. If no there is duplicate, do an INSERT.
IF (NOT EXISTS (SELECT E.SSN
                FROM EmployeeTable E, inserted
                WHERE E.SSN = inserted.SSN))
    INSERT INTO EmployeeTable
        SELECT EmployeeID,SSN, Department, Salary
        FROM inserted
ELSE
--If there is a duplicate, change to UPDATE so that there will not
--be a duplicate key violation error.
    UPDATE EmployeeTable
        SET EmployeeID = I.EmployeeID,
            Department = I.Department,
            Salary = I.Salary
        FROM EmployeeTable E, inserted I
        WHERE E.SSN = I.SSN
END
```


Cuándo utilizar Triggers?

- Debe preferirse utilizar CHECK constraints en vez de Triggers hasta donde sea posible.

- Los Triggers pueden utilizarse en escenarios como los siguientes:
 - ◆ La validación es más compleja de lo que permite un CHECK, UNIQUE o FOREIGN constraints.
 - ◆ Se necesita considerar valores de columnas de otras tablas en la validación.
 - ◆ Se necesita considerar el estado de la tabla antes y después de la sentencia.
 - ◆ Se necesita cancelar el evento aplicado sobre la tabla.
 - ◆ Se necesita reaccionar ante eventos.
 - ◆ Se necesita forzar autorizaciones de seguridad complejas.
 - ◆ Se necesita replicar datos.
 - ◆ Se necesita auditar modificaciones de datos.

Cuándo utilizar Triggers?

- Debe preferirse utilizar CHECK en vez de Triggers hasta donde sea posible.

Regla general
Utilice Triggers únicamente si es necesario

- Los Triggers pueden utilizarse en escenarios como los siguientes:
 - ◆ La validación es más compleja de lo que permite un CHECK, UNIQUE o FOREIGN constraints.
 - ◆ Se necesita considerar valores de columnas de otras tablas en la validación.
 - ◆ Se necesita considerar el estado de la tabla antes y después de la sentencia.
 - ◆ Se necesita cancelar el evento aplicado sobre la tabla.
 - ◆ Se necesita reaccionar ante eventos.
 - ◆ Se necesita forzar autorizaciones de seguridad complejas.
 - ◆ Se necesita replicar datos.
 - ◆ Se necesita auditar modificaciones de datos.

Triggers y Performance

- Los triggers no tienen un mal desempeño per se.
- El mal desempeño puede ser ocasionados por el código que se ejecuta.
- Consideraciones:
 - ◆ Deben ser escritos cuidadosamente y mantenerlos sencillos sin hacer tareas complejas (envío de emails).
 - ◆ Evitar recorrer cursores.
 - ◆ Evitar ejecutar muchas sentencias que involucren otra tablas.
 - ◆ No retornar result sets desde un Trigger.

Triggers y Performance

Aparte del performance, los Triggers puede ser un reto a la hora de hacer mantenimiento o determinar la causa de un error

- Los triggers no tienen un mal desempeño per se.
- El mal desempeño puede ser ocasionados por el código que se ejecuta.
- Consideraciones:
 - ◆ Deben ser escritos cuidadosamente y mantenerlos sencillos sin hacer tareas complejas (envío de emails).
 - ◆ Evitar recorrer cursores.
 - ◆ Evitar ejecutar muchas sentencias que involucren otra tablas.
 - ◆ No retornar result sets desde un Trigger.

Triggers y Performance

Aparte del performance, los Triggers puede ser un reto a la hora de hacer mantenimiento o determinar la causa de un error

→ Los triggers no tienen un mal desempeño per se.

→ El mal desempeño puede ser ocasionado por el trigger que se ejecuta.

Nadie se acuerda de aquel Trigger que hacía A y B cosas

→ Consideraciones:

- ◆ Deben ser escritos cuidadosamente y mantenerlos sencillos sin hacer tareas complejas (envío de emails).
- ◆ Evitar recorrer cursores.
- ◆ Evitar ejecutar muchas sentencias que involucren otras tablas.
- ◆ No retornar result sets desde un Trigger.

Store Procedures, Functions and Triggers

CE3101 - Bases de Datos

