Complementos de Bases de Dados
– T-SQL –

Engenharia Informática

2º Ano / 1º Semestre

Cláudio Miguel Sapateiro
claudio.sapateiro@estsetubal.ips.pt

DSI :: Escola Superior de Tecnologia de Setúbal :: Instituto Politécnico de Setúbal

Sumário

- T-SQL
  - *Stored procedures*
  - *Functions*
  - *Triggers*
  - *Tratamento de erros*

T-SQL: UDSP

sp's do utilizador (usp)

```
CREATE PROCEDURE <name>
  <parameter list>
AS
  <instructions to execute>
```

- Administração e manutenção dos dados simplificada
- Código armazenado com os dados
- Segurança acrescida
  - por via de diferentes níveis de acesso a diferentes utilizadores
- Ganhos de performance
- Executam em "scope" próprio
  - Não "partilha" variáveis diretamente
  - Aplicam-se as politicas de segurança para os utilizadores
- Utilizadas para:
  - Retornar dados (*dataset*)
  - Inserir/Editar dados
  - Realizar cálculos/processamento

```
EXEC(cute) stored procedure name
```

T-SQL: UDSP

Parâmetros

- Parâmetros
  - de entrada
  - de saída
    - Têm de ter uma variável de "suporte"
- Declaração
  - Nome e tipo
  - Valor *default* é opcional
  - Direção (input/output) é "opcional"
    - input é *default*
    - Output tem de ser explicitamente declarado

```
CREATE PROC uspFindStudentID
@First varchar(25)
@Last varchar(25)
@SID char(9) OUTPUT

AS
SELECT @SID=SchoolID
  FROM Students
  WHERE FirstName=@First AND
    Lastname=@Last
```

T-SQL: UDSP

Valores de Retorno

- Indicam sucesso ou insucesso da execução
- <> 0 indica a ocorrência de um problema
- Tem de ser um inteiro
- É diferente de um parâmetro de output!!
  - que está relacionado com os dados
- RETURN <valor>
  - Termina imediatamente a execução

T-SQL: Vars

Variáveis

Locais

- Ex: Declaração e Atribuição de valor a uma variável
- DECLARE *@nome_variável tipo_dado* [, *n*]
- SET *@nome_variável* = expressão

```
SET @Var=value
```

```
SELECT @Var=Sum(PossiblePoints) FROM Assignments
```

"Globais": são predefinidas e não podem ser alteradas.
(*parameterless sys functions*)

- Os símbolos @@ precedem o nome da variável.
- São de sistema (o utilizador não pode declarar vars globais)
- Ex: @@ERROR, @@TRANCOUNT, @@FETCH_STATUS, ...

T-SQL: “Global Vars”

“Var”	Description
@@ERROR	-Commonly used to check the error status (succeeded or failed) of the most recently executed statement. -It contains 0 if the previous transaction succeeded; otherwise, it contains the last error number generated by the system.
@@IDENTITY	-The last value inserted into an IDENTITY column by an insert, or select into statement. -@@identity is reset each time a row is inserted into a table. -If a statement inserts multiple rows, @@identity reflects the IDENTITY value for the last row inserted. -If the affected table does not contain an IDENTITY column, @@identity is set to 0. -The value of @@identity is not affected by the failure of an insert or select into statement, or the rollback of the transaction that contained it.

T-SQL: “Global Vars”

“Var”	Description
@@ROWCOUNT	- The number of rows affected by the last command. - @@rowcount is set to 0 by any command which does not return rows, such as an if statement. - With cursors, @@rowcount represents the cumulative number of rows returned from the cursor result set to the client, up to the last fetch request.
@@TRANCOUNT	- The nesting level of transactions. - Each begin transaction in a batch increments the transaction count.
…	

T-SQL: “Global Vars”

“Var”	Example
@@ERROR	IF @@ERROR <> 0 PRINT 'Your error message';. **Output**: Your error message
@@IDENTITY	INSERT INTO [someTableWithIdentity] ([Code]) VALUES (5) GO SELECT @@IDENTITY AS 'Identity'; **Output**: 5
@@ROWCOUNT	IF @@ROWCOUNT = 0 PRINT 'Warning: No rows were updated'; **Output**: Warning: No rows were updated

Se “Code” é *Identity então:*

```
SET IDENTITY_INSERT sometableWithIdentity ON
```

T-SQL: “Global Vars”

“Var”	Example

@@TRANCOUNT	PRINT @@TRANCOUNT
	-- The BEGIN TRAN statement will increment the
	-- transaction count by 1.
	BEGIN TRAN
	  PRINT @@TRANCOUNT
	  BEGIN TRAN
	  PRINT @@TRANCOUNT
	-- The COMMIT statement will decrement the transaction count by 1.
	COMMIT
	  PRINT @@TRANCOUNT
	COMMIT
	PRINT @@TRANCOUNT
	Output: 0 1 2 1 0

T-SQL: Operadores

Operadores

- **Aritméticos**: *, /, +, -, % (módulo)
- **Comparação**: <, >, =, >=, <=, <>
- **Strings**: + (concatenação)
- **Lógicos**: AND, OR, NOT

T-SQL: Controlo de Fluxos

Estruturas de controlo

- BEGIN ... END
- IF ... ELSE
- WHILE (BREAK, CONTINUE)
  - Também utilizados com os cursores
- CASE

  WHEN *condição* THEN *comando*

  WHEN *condição* THEN *comando*

  ….

T-SQL: UDSP

Exemplo

criação de um procedimento com parâmetros

```
CREATE PROCEDURE author_info
@lastname varchar(30) = ‘D%',
@firstname varchar(18) = '%'
AS
    SELECT au_lname, au_fname, title, pub_name
    FROM authors a INNER JOIN titleauthor ta
    ON a.au_id = ta.au_id INNER JOIN titles t
    ON t.title_id = ta.title_id INNER JOIN publishers p
    ON t.pub_id = p.pub_id
    WHERE au_fname LIKE @firstname
    AND au_lname LIKE @lastname
GO
```

T-SQL: UDSP

Exemplo

criação e execução de um procedimento

```
CREATE PROCEDURE sales2
@title varchar(80)
AS
    IF NOT EXISTS (SELECT * FROM titles WHERE title = @title)
    RETURN -101
    SELECT sales
    FROM titles
    WHERE title = @title
    RETURN
GO
```

```
-- Execute the procedure
    DECLARE @status int
    EXEC @status = sales2 'Life without Fear'
    IF @status = -101
    PRINT 'No title with that name found.'
```

T-SQL

Cursores

- Apoio ao processamento orientado linha/registo a linha/registo

- Declaração
  - Declare @Cursor Cursor
- Atribuição
  - Set @Cursor = Cursor for (*select statement*)
- Abertura
  - Open @Cursor
- Utilização
  - Fetch Next From @Cursor into (variáveis correspondentes aos campos do select)
- Fecho
  - Close @Cursor

T-SQL

Cursores

```
Declare @First varchar(10)
Declare @Last varchar(10)
Declare @Students Cursor

Set @Students Cursor For (SELECT FirstName, LastName FROM Students)
Open @Students

Fetch Next From @Students Into @First, @Last

While @@Fetch_Status=0
BEGIN
  Print @First + ' ' + @Last
  Fetch Next From @Students Into @First, @Last
END
Close @Students
Deallocate @Students
```

@@Fetch_Status

- -1 failed
(leitura fora do *resultset*)
- -2 missing record
(e.g. apagado)

T-SQL: UDSP

Exercício

1. Qual a diferença?

```
CREATE procedure PersonByTitle1
@Title  nvarchar(8)
AS
SELECT [BusinessEntityID]
    ,[PersonType]
    ,[NameStyle]
    ,[Title]
    ,[FirstName]
    ,[MiddleName]
    ,[LastName]
    ,[Suffix]
    ,[EmailPromotion]
FROM [Person].[Person]
WHERE [Title] = @Title
```

```
CREATE procedure PersonByTitle2
@Title  nvarchar(8) = 'Mr'
AS
SELECT [BusinessEntityID]
    ,[PersonType]
    ,[NameStyle]
    ,[Title]
    ,[FirstName]
    ,[MiddleName]
    ,[LastName]
    ,[Suffix]
    ,[EmailPromotion]
FROM [Person].[Person]
WHERE [Title] = @Title
```

T-SQL: UDSP

Exercício

2. Como será a chamada para execução?
   e o resultado?

```
CREATE procedure PersonByTitle1
@Title  nvarchar(8)
AS
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
FROM [Person].[Person]
WHERE [Title] = @Title
```

```
CREATE procedure PersonByTitle2
@Title  nvarchar(8) = 'Mr'
AS
SELECT [BusinessEntityID]
      ,[PersonType]
      ,[NameStyle]
      ,[Title]
      ,[FirstName]
      ,[MiddleName]
      ,[LastName]
      ,[Suffix]
      ,[EmailPromotion]
FROM [Person].[Person]
WHERE [Title] = @Title
```

```
EXEC PersonByTitle1/2
EXEC PersonByTitle1/2 @Title='Mr.'
EXEC PersonByTitle1/2 @Title='Mrs.'
```

T-SQL: UDSP

Exercício

3. Como será o procedimento para permitir as seguintes chamadas de execução com sucesso?

```
-- Retorna todas as pessoas
EXEC PersonByTitle3

-- Retorna as senhoras
EXEC PersonByTitle3 @Title='Ms.‘

-- Retorna os senhores
EXEC PersonByTitle3 @Title='Mr.‘
```

```
CREATE procedure PersonByTitle3
@Title  nvarchar(8) = null,
AS
if len(isnull(@Title,'')) = 0
begin
        SELECT [BusinessEntityID]
                ,[PersonType]
                ,[NameStyle]
                ,[Title]
               ….
        FROM [Person].[Person]
end
if len(isnull(@Title,'')) > 0
begin
        SELECT [BusinessEntityID]
                ,[PersonType]
                ,[NameStyle]
                ,[Title]
               …
        FROM [Person].[Person]
        WHERE [Title] = @Title
```

T-SQL: UDSP

Exercício

4. Como fazer para ter as seguintes opções:

-- Retorna todas as pessoas com titulo
(i.e. não retornar registos com titulo a null)

```
EXEC PersonByTitle4
```

-- Retorna as senhoras

```
EXEC PersonByTitle4 @Title='Ms.‘
```

-- Retorna os senhores

```
EXEC PersonByTitle4 @Title='Mr.‘
```

```
CREATE procedure PersonByTitle4
@Title  nvarchar(8) = null,
AS
begin
        SELECT [BusinessEntityID]
                ,[PersonType]
                ,[NameStyle]
                ,[Title]
               …
        FROM [Person].[Person]
        WHERE [Title] = ISNULL(@Title, Title)
```

T-SQL: UDSP

Exercício

Outra forma …

```
CREATE procedure PersonByTitle5
@Title  nvarchar(8) = null,
AS
DECLARE      @sqlstring    nvarchar(4000)

SET @sqlstring = 'SELECT [BusinessEntityID]
                                  ,[PersonType]
                                  ,[NameStyle]
                                  ,[Title]
                                  ,[FirstName]
                                 ....
                FROM Person.Person
                WHERE  1=1 '

 if len(isnull(@Title,'')) > 0
  begin
   set @sqlstring = @sqlstring + ' and Title = '+''''+cast(@Title as varchar)+''''
  end

 -- print @sqlstring
 exec sp_executesql @sqlstring
 GO
```

T-SQL: UDF

User Defined Functions

- Rotinas que retornam valores calculados
  - Escalares (singular)
  - Tabelas
- A sua "chamada" pode ser embebida em *queries*
- Não alteram os dados persistidos nas tabelas
- Normalmente com sufixo "fn_…"

```
--Transact-SQL Scalar/Table Function Syntax
CREATE FUNCTION function_name
(@parameter_name  AS parameter_data_type  [ = default ])
RETURNS return_data_type / TABLE
[ AS ]
BEGIN
   function_body
   RETURN scalar_expression /select_statement
END
```

T-SQL: UDF

Exemplo

```
CREATE FUNCTION fn_CubicVolume (@CubeLength decimal(4,1), @CubeWidth
decimal(4,1),@CubeHeight decimal(4,1) )
-- Input dimensions in centimeters.

RETURNS decimal(12,3) -- Cubic Centimeters.
AS
BEGIN
   RETURN ( @CubeLength * @CubeWidth * @CubeHeight )
END
GO
```

T-SQL: UDF

Exemplo

```
CREATE FUNCTION Continent
(@Country nvarchar(15))
RETURNS varchar(30)
AS
BEGIN
declare @return varchar(30)
select @return = case @Country
                        when 'Argentina' then 'South America'
                        when 'Belgium' then 'Europe'
                        when 'Brazil' then 'South America'
                        when 'Canada' then 'North America'
                        ….
                        else 'Unknown'
                        end

return @return
end
GO
```

T-SQL: UDF

Exemplo (cont.)

```
CREATE FUNCTION SalesByContinent
(@ContinentFromCountry varchar(30))
RETURNS TABLE
AS
RETURN
  SELECT [Sales].[SalesTerritory].*
  FROM [Sales].[SalesTerritory]
  WHERE [Sales].[SalesTerritory].[Group] = dbo.Continent(@ ContinentFromCountry)
GO


--Execução
SELECT * from SalesByContinent('Germany')
```

T-SQL: *Triggers*

Triggers

- Objeto da BD que fica associado a uma tabela (ou BD)
  - Semelhante a uma *sp*, mas cuja a execução é despoletada por operações sobre a tabela
- Utilizados para "auditar" alterações
  - Permitem a execução de uma lógica especifica quando despoletados
  - Auxiliam garantias de integridade de dados e estrutura
- Tipos
  - AFTER/FOR: executam após comandos INSERT, UPDATE ou DELETE
  - INSTEAD OF: substitui os comandos INSERT, UPDATE ou DELETE
  - DATABASE: asseguram a integridade da estrutura de dados

T-SQL: *Triggers*

Exemplos ilustrativos

1. Adicionar um item oferta ao carrinho de compras numa encomenda que ultrapassa um determinado valor total
   - No "fecho" da inserção de produtos (linhas) de encomenda será verificado o valor e se for o caso adicionada uma linha com o item oferta

2. Num levantamento numa ATM, pode ser utilizado para:
   - (Ao registar o movimento na tabela de movimentos)
     Validar se existe saldo que o permite
   - atualizar o saldo do cliente
     (Numa tabela/coluna que o agrega)

T-SQL: *Triggers*

Desvantagens

- aumentam o *overhead* de processamento das operações sobre a BD
- A sua execução é transparente para os utilizadores, pelo que em caso de erros poderá ser difícil ou tardia a deteção

- Uma das razões do *overhead*:
  - Recurso a duas tabelas especializadas: ***deleted*** e ***inserted***
  - Tipicamente na sua execução estas são referenciadas o que motiva a degradação de desempenho das operações sobre a BD
  - ➢ Alem disso, a lógica prevista para execução deve ser "breve" e de complexidade reduzida
    (quanto mais tempo demorar, mais tempo demorará a libertar *locks associados*)

❖ Devem ser usados ponderada e cautelosamente

T-SQL: *Triggers*

Tabelas *inserted* e *deleted*

- ***inserted*** *table*
  - Contem copias dos dados referenciados no comando INSERT
- ***deleted*** *table*
  - Contem copias dos dados da tabela referenciados pelo comando DELETE

❖ No caso do Update

❑ É feito uso das duas tabelas

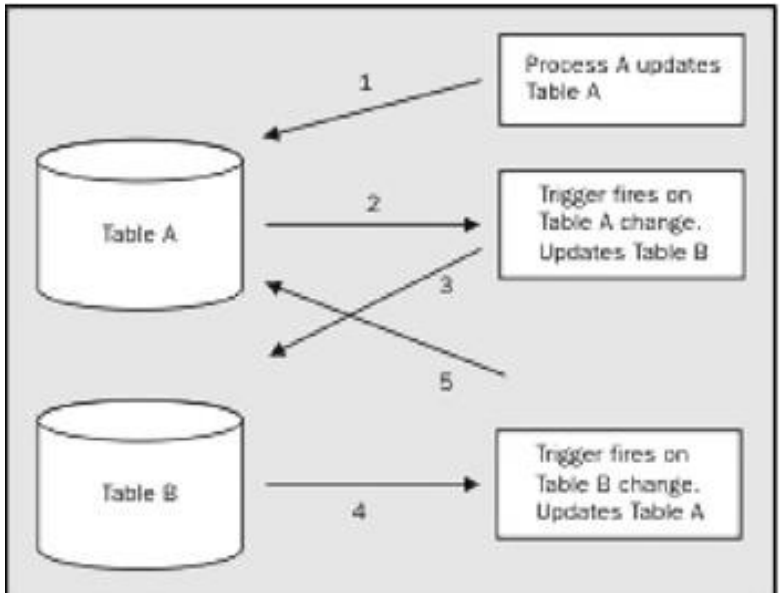
✓ Os dados novos estarão presentes na tabela ***inserted***

✓ Os dados a atualizar/substituir estarão presentes na tabela ***deleted***

T-SQL: *Triggers*

Cuidados !!

***Endless loop*:**
`Update trigger` on TableA, which inserts into TableB,
and
`trigger` for TableB, which updates TableA.



T-SQL: *Triggers*

Exemplo

Mostra o *timestamp* corrente quando é inserida uma linha na tabela "Source"

```
CREATE TABLE Source (Sou_ID int IDENTITY, Sou_Desc varchar(10))
go

CREATE TRIGGER tr_Source_INSERT
ON Source
FOR INSERT
AS
 PRINT GETDATE()
go

-- Operação que o despoletará
INSERT Source (Sou_Desc) VALUES ('Test 1')

-- Resultado (na consola)
Apr 28 2001 9:56AM
```

T-SQL: *Triggers*

Exemplo

O que faz?

```
CREATE TABLE Orders (Ord_ID int IDENTITY, Ord_Priority varchar(10), …)
go

CREATE TRIGGER tr_Orders_INSERT
ON Orders
FOR INSERT
AS
IF (SELECT COUNT(*) FROM inserted WHERE Ord_Priority = 'High') >= 1
BEGIN
  PRINT 'Email Code Goes Here' --e.g. notify for high priority orders placed
END
go

-- Operação que o despoletará
INSERT Orders (Ord_Priority,…) VALUES ('High‘,…)

-- Resultado (na consola) --
Email Code Goes Here
```

T-SQL: *Triggers*

Exemplo (cont.)

O que acontece se executar:

I.

INSERT Orders (Ord_Priority,…) VALUES ('High',…)
INSERT Orders (Ord_Priority,…) VALUES ('High',…)

-- Resultado (na consola)
Email Code Goes Here
Email Code Goes Here

II.

INSERT Orders SELECT * FROM OrdersTemp Where Ord_Priority = 'High'
[e.g. 3 registos resultantes]

-- Resultado (na consola)
Email Code Goes Here

O trigger só é despoletado e executado 1 vez por operação, independentemente do numero de linhas afetadas

T-SQL: *Triggers*

Exemplo (cont.)

Obtenção do numero de linhas que foram afetadas pela operação

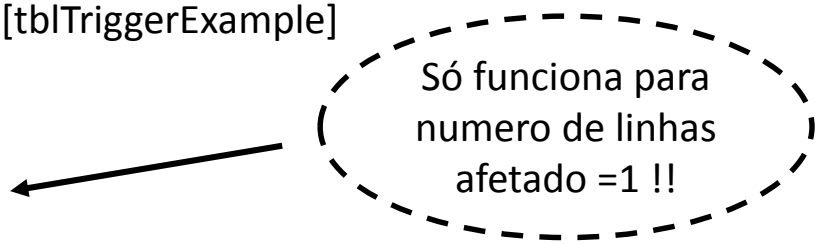
```
ALTER TRIGGER tr_Orders_INSERT ON Orders
FOR INSERT
AS
IF EXISTS (SELECT * FROM inserted WHERE Ord_Priority = 'High')
BEGIN
  DECLARE @Count tinyint
  SET @Count = (SELECT COUNT(*) FROM inserted WHERE Ord_Priority = 'High')
  PRINT CAST(@Count as varchar(3)) + ' row(s) with a priority of High were entered'
END
Go

-- Execução
INSERT Orders SELECT * FROM OrdersTemp Where Ord_Priority = ‘High’

-- Resultado (e.g.)
3 row(s) with a priority of High were entered
```

T-SQL: *Triggers*

Exercício

O que faz?

```
CREATE TRIGGER [TRIGGER_ALTER_COUNT] ON [dbo].[tblTriggerExample]
FOR INSERT, UPDATE
AS
BEGIN
  DECLARE @TransID  VARCHAR(36)
  SELECT @TransID = TransactionID FROM INSERTED
  UPDATE [dbo].[tblTriggerExample] SET AlterCount = AlterCount + 1, LastUpdate = GETDATE()
    WHERE TransactionID = @TransID
END
```



[tblTriggerExample]

TransactionID	ItemCode	Price	Comments	IsExpired	LastUpdate	AlterCount	Created
2A590601-5179-4E	1	30	NA	N	8/7/2009 8:35:10 F	3	8/7/2009 8:33:58 F
073E7D5E-942C-4E	2	10	NA	N	8/7/2009 8:35:26 F	1	8/7/2009 8:35:26 F
A90A47DC-E3AA-4	3	25	NA	N	8/7/2009 8:35:31 F	1	8/7/2009 8:35:31 F
36F63DAB-F310-4E	4	35	NA	N	8/7/2009 8:35:35 F	1	8/7/2009 8:35:35 F
F4D925B2-D716-4(	5	40	NA	N	8/7/2009 8:35:38 F	1	8/7/2009 8:35:38 F
88DE4A58-315F-4(	6	50	NA	N	8/7/2009 8:35:45 F	1	8/7/2009 8:35:45 F
FB16825E-246A-4(	7	90	NA	N	8/7/2009 8:36:10 F	2	8/7/2009 8:35:47 F
F79838F4-AA70-4(	8	80	NA	N	8/7/2009 8:35:49 F	1	8/7/2009 8:35:49 F
B65CDAC5-60CD-4	9	58	NA	N	8/7/2009 8:35:54 F	1	8/7/2009 8:35:54 F
4BCFD963-8893-4(	10	100	NA	N	8/7/2009 8:36:00 F	1	8/7/2009 8:36:00 F

T-SQL: *Triggers*

Boas Práticas !

- Se no caso de não haver alterações decorrentes/linhas afetadas = 0
  e.g. de um comando à tabela com erro
  (e.g. Insert ... where 1=0)
  - Retornar antes de entrar pela lógica do trigger!
- Tambem ter presente a politica de row count,
  - uma vez que as operações sobre tabelas já tipicamente retornam este resultado

- **Exemplo:**

```
alter trigger trgData_AI on dbo.Data
after insert
as
begin
  if @@ROWCOUNT = 0
   return
 set nocount on
 /* Some Code Here */
end
```

- ***Ainda***, activar (coerentemente) a clausula relativa à replicação
  - [***Not for replication***] (not to be executed when a *replication agent* modifies a table)

T-SQL: *Triggers*

INSTEAD OF

- Ao contrário dos FOR/AFTER *triggers* que só são executados depois de realizada a operação sobre a tabela a que estão associados
  - Assim sendo com limitações ao nível de validações
  - Que em caso de ***Erro*** na operação provocará a não execução do *trigger*
- Também ao contrário dos FOR/AFTER estes estão disponíveis também para *views*
- Neste caso o *trigger* é executado antes da operação na tabela que o despoleta
  - Permite uma margem maior no que a validações diz respeito

T-SQL: *Triggers*

INSTEAD OF

Exemplo

```
CREATE TRIGGER deletePerson
ON Person.Person
INSTEAD OF DELETE
AS
  BEGIN
    Update Person.Person
      Set [Active] = 0
      where [BusinessEntityID] in
                (select [BusinessEntityID]
                from deleted)
  END

  go
```

T-SQL: *Triggers*

Database Triggers

Exemplo

```
-- Criação de database trigger
Create trigger tableIntegrity
on Database
FOR DROP_TABLE, ALTER_TABLE
AS
BEGIN
        PRINT 'Operação não autorizada!'
        ROLLBACK;
END

go
```

T-SQL

Síntese

Funcionalidade/objetos	triggers	functions	Stored Procedures
alteração de dados	sim	não	sim
retorno de valor	não	sim (unitário ou tabela)	depende do utilizador
chamada	disparam em reação a evento	em comando SQL	execute

T-SQL: Error Handling

Mensagens de Erro

```
--------------Try To Access Non-Existing Table ---------------
SELECT * FROM dbo.NonExistingTable
GO
```

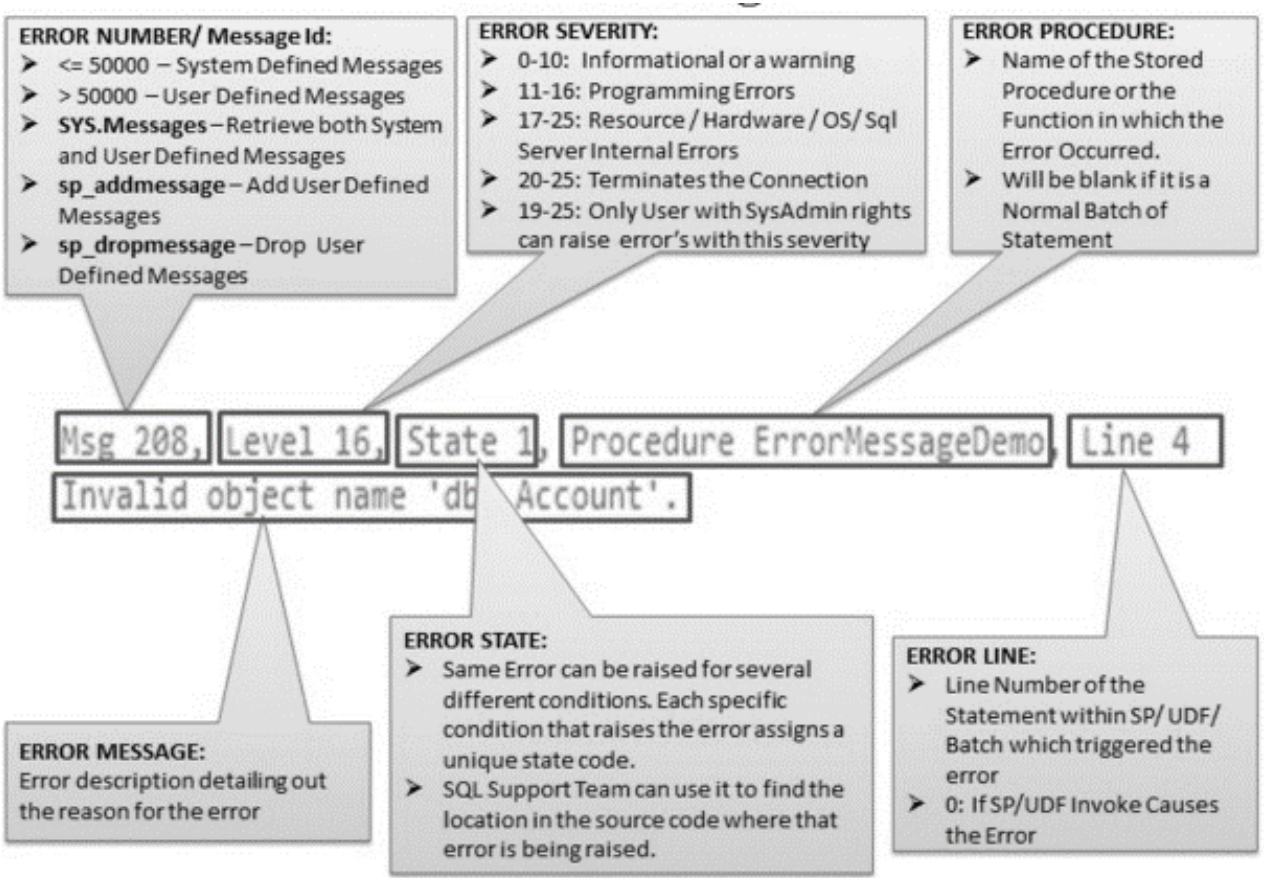
```
Msg 208, Level 16, State 1, Line 2
Invalid object name 'dbo.NonExistingTable'.
```

Composição da Mensagem de Erro:

Msg **208** – Error Number
Level **16** – Severity of the Error
State **1** – State of the Error
Line **2** – Line Number of the statement which generated the Error
Invalid object name 'dbo.NonExistingTable'. – Actual Error Message

T-SQL: Error Handling

Mensagens de Erro



T-SQL: Error Handling

Mensagens de Erro

```
select * from sys.messages
```

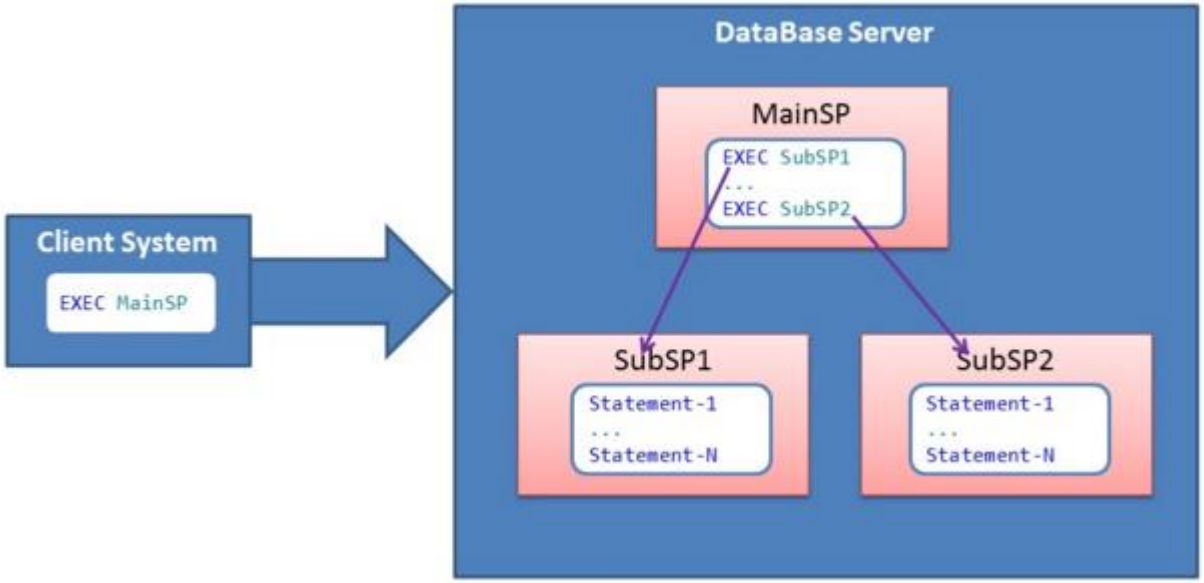
Results | Messages

	message_id	language_id	severity	is_event_logged	text
1	21	1033	20	0	Warning: Fatal error %d occurred at %S_DATE. Not...
2	101	1033	15	0	Query not allowed in Waitfor.
3	102	1033	15	0	Incorrect syntax near '%.*ls'.
4	103	1033	15	0	The %S_MSG that starts with '%.*ls' is too long. Max...
5	104	1033	15	0	ORDER BY items must appear in the select list if the...
6	105	1033	15	0	Unclosed quotation mark after the character string '...
7	106	1033	16	0	Too many table names in the query. The maximum a...
8	107	1033	15	0	The column prefix '%.*ls' does not match with a tabl...
9	108	1033	15	0	The ORDER BY position number %ld is out of range...
10	109	1033	15	0	There are more columns in the INSERT statement t...
11	110	1033	15	0	There are fewer columns in the INSERT statement t...
12	111	1033	15	0	'%ls' must be the first statement in a query batch.
13	112	1033	15	0	Variables are not allowed in the %ls statement.
14	113	1033	15	0	Missing end comment mark '*/'.
15	114	1033	15	0	Browse mode is invalid for a statement that assigns

T-SQL: Error Handling

Ações consequentes de Erro

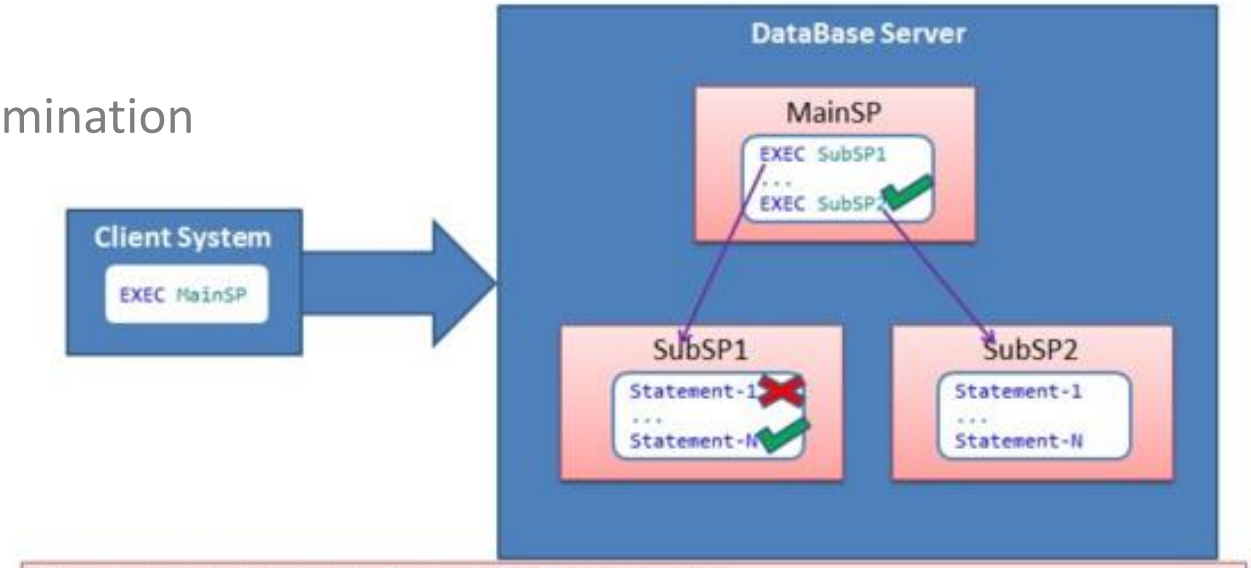
- Statement Termination
- Scope Abortion
- Batch Abortion
- Connection Termination



T-SQL: Error Handling

Ações consequentes de Erro

- **Statement Termination**
- Scope Abortion
- Batch Abortion
- Connection Termination



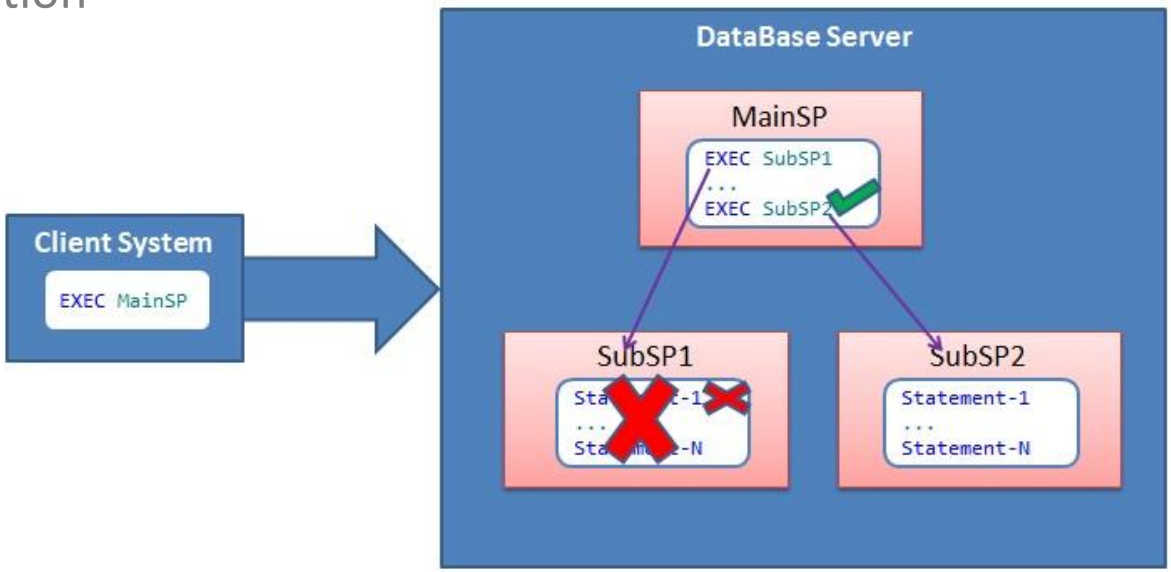
Errors which results in Statement Termination:

- NOT NULL, PRIMARY KEY, CHECK CONSTRAINT, FOREIGN KEY Violation
- Invoking Non-Existing SP. Invoking Parameterized SP with less or More Parameters
- Permission issue on Table, SP etc.
- ROLLBACK/COMMIT without an Active Transaction

T-SQL: Error Handling

Ações consequentes de Erro

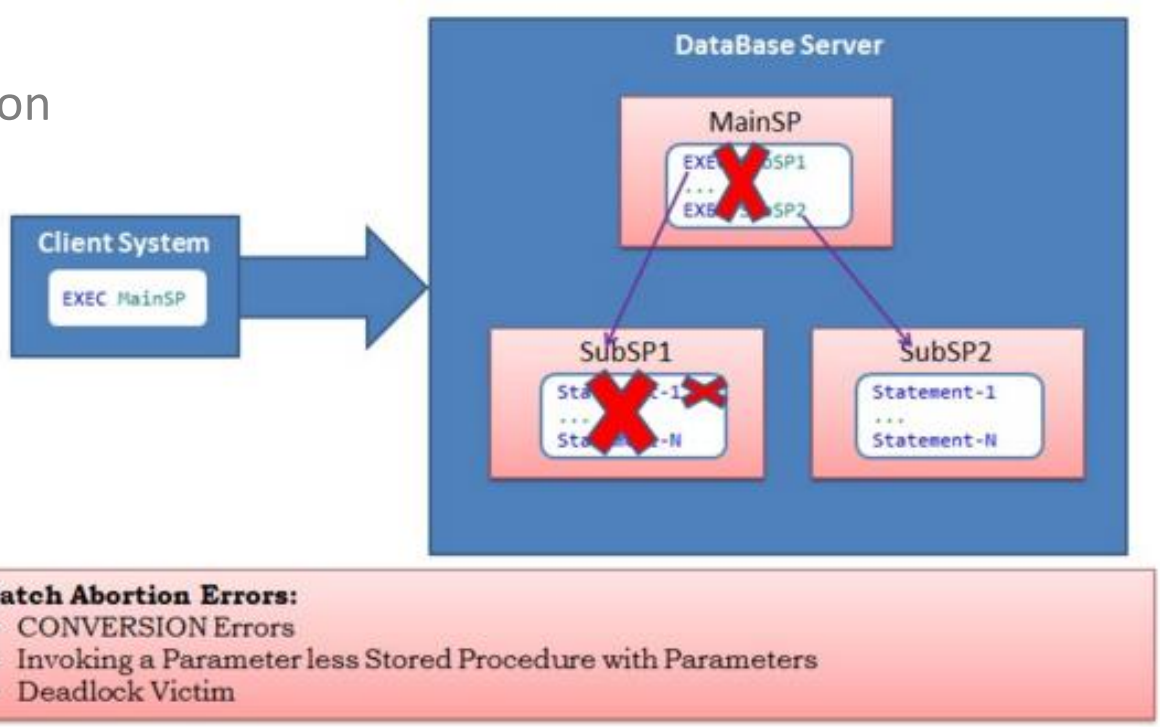
- Statement Termination
- **Scope Abortion**
- Batch Abortion
- Connection Termination



T-SQL: Error Handling

Ações consequentes de Erro

- Statement Termination
- Scope Abortion
- **Batch Abortion**
- Connection Termination

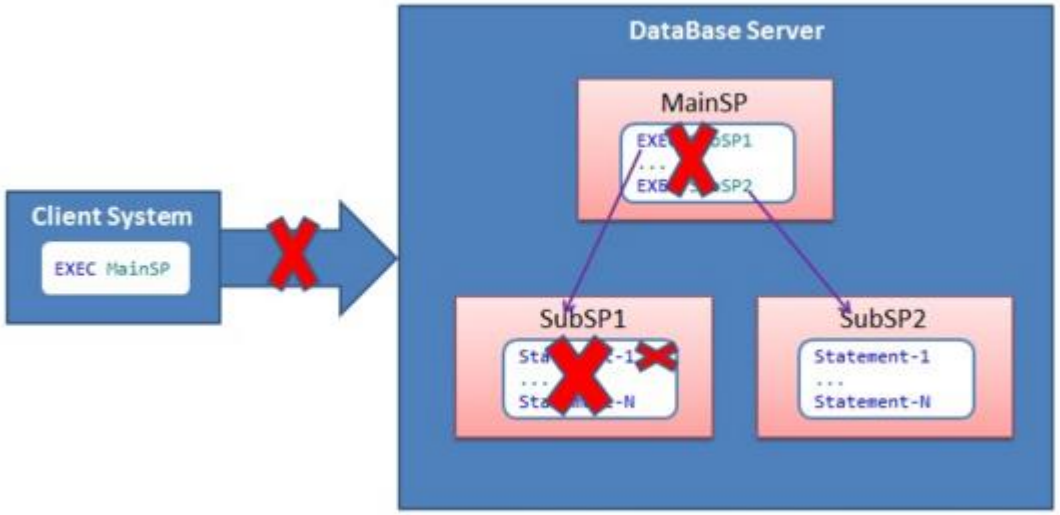


** Independente de
SET XACT_ABORT ON|OFF

T-SQL: Error Handling

Ações consequentes de Erro

- Statement Termination
- Scope Abortion
- Batch Abortion
- **Connection Termination**



T-SQL: Error Handling

TRY…CATCH
& ERROR Functions

```
BEGIN TRY
 -- T-Sql Statements
END TRY
BEGIN CATCH
 -- T-Sql Statements
 /*Control is passed to CATCH block only if
   there are any exceptions in the TRY block*/
END CATCH
```

Ou:
RaiseError ou *Throw* !!

```
BEGIN TRY
 SELECT 5/0
END TRY
BEGIN CATCH
    PRINT '*************Error Detail****************'
    PRINT 'Error Number  :' + CAST(ERROR_NUMBER() AS VARCHAR)
    PRINT 'Error Severity:' + CAST(ERROR_SEVERITY() AS VARCHAR)
    PRINT 'Error State   :' + CAST(ERROR_STATE() AS VARCHAR)
    PRINT 'Error Line    :' + CAST(ERROR_LINE() AS VARCHAR)
    PRINT 'Error Message :' + ERROR_MESSAGE()
END CATCH
```

```
(0 row(s) affected)

*************Error Detail****************
Error Number  :8134
Error Severity:16
Error State   :1
Error Line    :2
Error Message :Divide by zero error encountered.
```

T-SQL: Error Handling

Raiserror *vs* Throw

```
-- Using RAISERROR()
DECLARE
 @ERR_MSG AS NVARCHAR(4000)
 ,@ERR_SEV AS SMALLINT
 ,@ERR_STA AS SMALLINT

BEGIN TRY
 SELECT 1/0 as DivideByZero
END TRY
BEGIN CATCH
 SELECT @ERR_MSG = ERROR_MESSAGE(),
 @ERR_SEV =ERROR_SEVERITY(),
 @ERR_STA = ERROR_STATE()

 SET @ERR_MSG= 'Error occurred while retrieving the data from database: ' + @ERR_MSG

 RAISERROR (@ERR_MSG, @ERR_SEV, @ERR_STA) WITH NOWAIT
END CATCH
GO
```

Pode receber *message_id* se registado em *sys.messages*

Output:

```
(0 row(s) affected)
Msg 50000, Level 16, State 1, Line 15
Error occurred while retrieving the data from database: Divide by zero error encountered.
```

T-SQL: Error Handling

Raiserror *vs* Throw

```
-- Using THROW - 1
BEGIN TRY
 SELECT 1/0 as DivideByZero
END TRY
BEGIN CATCH
 THROW;
END CATCH
GO
```

Output:

(0 row(s) affected)

Msg 8134, Level 16, State 1, Line 2

Divide by zero error encountered.

Parametros não obrigatórios mas pode usar

```
-- Using THROW - 2
DECLARE
 @ERR_MSG AS NVARCHAR(4000)
 ,@ERR_STA AS SMALLINT

BEGIN TRY
 SELECT 1/0 as DivideByZero
END TRY
BEGIN CATCH
 SELECT @ERR_MSG = ERROR_MESSAGE(),
 @ERR_STA = ERROR_STATE()

 SET @ERR_MSG= 'Error occurred while retrieving the data from database: ' + @ERR_MSG;

 THROW 50001, @ERR_MSG, @ERR_STA;
END CATCH
GO
```

Pode usar ids, que não em sys.messages

severity defaults to 16

Output:

(0 row(s) affected)

Msg 50001, Level 16, State 1, Line 14

Error occurred while retrieving the data from database: Divide by zero error encountered.

Exemplos e Exercícios

```
CREATE PROCEDURE [dbo].[uspLogError]
    @ErrorLogID [int] = 0 OUTPUT -- contains the ErrorLogID of the row inserted
AS                               -- by uspLogError in the ErrorLog table
BEGIN
    SET NOCOUNT ON;

    SET @ErrorLogID = 0;

    BEGIN TRY
        IF ERROR_NUMBER() IS NULL
            RETURN;

        INSERT [dbo].[ErrorLog]
            (
            [UserName], [ErrorNumber], [ErrorSeverity], [ErrorState], [ErrorProcedure], [ErrorLine], [ErrorMessage]
            )
        VALUES
            (
            CONVERT(sysname, CURRENT_USER), ERROR_NUMBER(),ERROR_SEVERITY(),ERROR_STATE(),ERROR_PROCEDURE(),
            ERROR_LINE(),
            ERROR_MESSAGE()
            );

        SET @ErrorLogID = @@IDENTITY;
    END TRY
    BEGIN CATCH
        EXECUTE [dbo].[uspPrintError];
        RETURN -1;
    END CATCH
END;
```

Exemplos e Exercícios

```
CREATE PROCEDURE [HumanResources].[uspUpdateEmployeeLogin]
    @BusinessEntityID [int],
    @OrganizationNode [hierarchyid],
    @LoginID [nvarchar](256),
    @JobTitle [nvarchar](50),
    @HireDate [datetime],
    @CurrentFlag [dbo].[Flag]

AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        UPDATE [HumanResources].[Employee]
        SET [OrganizationNode] = @OrganizationNode
            ,[LoginID] = @LoginID
            ,[JobTitle] = @JobTitle
            ,[HireDate] = @HireDate
            ,[CurrentFlag] = @CurrentFlag
        WHERE [BusinessEntityID] = @BusinessEntityID;
    END TRY
    BEGIN CATCH
        EXECUTE [dbo].[uspLogError];
    END CATCH;
END;
```

Complementos de Bases de Dados
– T-SQL –

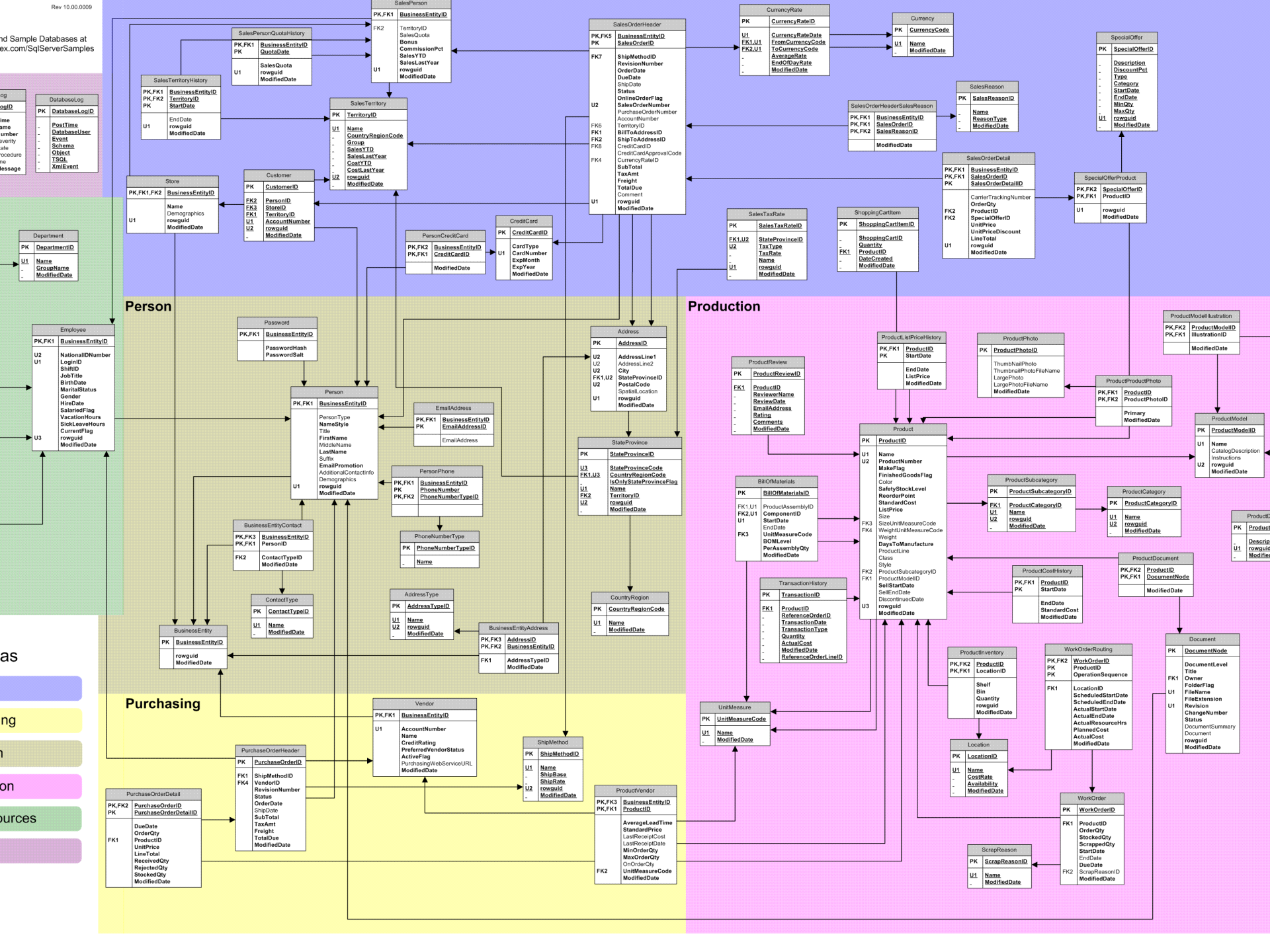
Engenharia Informática

2º Ano / 1º Semestre

2015/16

Cláudio Miguel Sapateiro
claudio.sapateiro@estsetubal.ips.pt

DSI :: Escola Superior de Tecnologia de Setúbal :: Instituto Politécnico de Setúbal


Rev 10.00.0009
nd Sample Databases at
ex.com/SqlServerSamples
DatabaseLog
DatabaseLogID
PostTime
DatabaseUser
Event
Schema
Object
TSQL
XmlEvent
Person
Purchasing
Production
SalesPerson
BusinessEntityID
TerritoryID
SalesQuota
Bonus
CommissionPct
SalesYTD
SalesLastYear
rowguid
ModifiedDate
SalesPersonQuotaHistory
QuotaDate
SalesQuota
SalesTerritoryHistory
StartDate
EndDate
SalesTerritory
TerritoryID
Name
CountryRegionCode
Group
SalesYTD
SalesLastYear
CostYTD
CostLastYear
SalesOrderHeader
SalesOrderID
ShipMethodID
RevisionNumber
OrderDate
DueDate
ShipDate
Status
OnlineOrderFlag
SalesOrderNumber
PurchaseOrderNumber
AccountNumber
CreditCardID
CreditCardApprovalCode
CurrencyRateID
SubTotal
TaxAmt
Freight
TotalDue
Comment
CurrencyRate
CurrencyRateID
FromCurrencyCode
ToCurrencyCode
AverageRate
EndOfDayRate
Currency
CurrencyCode
SalesReason
SalesReasonID
ReasonType
SalesOrderHeaderSalesReason
SpecialOffer
SpecialOfferID
Description
DiscountPct
Type
Category
MinQty
MaxQty
SalesOrderDetail
SalesOrderDetailID
CarrierTrackingNumber
OrderQty
ProductID
SpecialOfferID
UnitPrice
UnitPriceDiscount
LineTotal
SpecialOfferProduct
Store
CustomerID
Demographics
Customer
PersonID
StoreID
TerritoryID
AccountNumber
CreditCard
CardType
CardNumber
ExpMonth
ExpYear
PersonCreditCard
SalesTaxRate
StateProvinceID
TaxType
TaxRate
ShoppingCartItem
ShoppingCartItemID
ShoppingCartID
Quantity
ProductID
DateCreated
Department
DepartmentID
GroupName
Employee
NationalIDNumber
LoginID
ShiftID
JobTitle
BirthDate
MaritalStatus
Gender
HireDate
SalariedFlag
VacationHours
SickLeaveHours
CurrentFlag
Password
PasswordHash
PasswordSalt
Address
AddressID
AddressLine1
AddressLine2
City
StateProvinceID
PostalCode
SpatialLocation
EmailAddress
EmailAddressID
PersonPhone
PhoneNumber
PhoneNumberTypeID
PhoneNumberType
BusinessEntityContact
ContactTypeID
ContactType
AddressType
AddressTypeID
CountryRegion
CountryRegionCode
BusinessEntity
BusinessEntityAddress
Vendor
AccountNumber
CreditRating
PreferredVendorStatus
ActiveFlag
PurchasingWebServiceURL
ShipMethod
ShipBase
ShipRate
PurchaseOrderHeader
PurchaseOrderID
PurchaseOrderDetail
PurchaseOrderDetailID
DueDate
OrderQty
ProductID
UnitPrice
LineTotal
ReceivedQty
RejectedQty
StockedQty
ProductVendor
AverageLeadTime
StandardPrice
LastReceiptCost
LastReceiptDate
MinOrderQty
MaxOrderQty
OnOrderQty
UnitMeasureCode
ProductReview
ProductReviewID
ReviewerName
ReviewDate
EmailAddress
Rating
Comments
ProductListPriceHistory
StartDate
EndDate
ListPrice
ProductPhoto
ProductPhotoID
ThumbNailPhoto
ThumbnailPhotoFileName
LargePhoto
LargePhotoFileName
ProductProductPhoto
Primary
ProductModelIllustration
ProductModelID
IllustrationID
ProductModel
CatalogDescription
Instructions
Product
Name
ProductNumber
MakeFlag
FinishedGoodsFlag
Color
SafetyStockLevel
ReorderPoint
StandardCost
ListPrice
Size
SizeUnitMeasureCode
WeightUnitMeasureCode
Weight
DaysToManufacture
ProductLine
Class
Style
ProductSubcategoryID
ProductModelID
SellStartDate
SellEndDate
DiscontinuedDate
BillOfMaterials
BillOfMaterialsID
ProductAssemblyID
ComponentID
StartDate
EndDate
UnitMeasureCode
BOMLevel
PerAssemblyQty
ProductSubcategory
ProductSubcategoryID
ProductCategory
ProductCategoryID
ProductCostHistory
StandardCost
ProductDocument
DocumentNode
TransactionHistory
TransactionID
ReferenceOrderID
TransactionDate
TransactionType
Quantity
ActualCost
ReferenceOrderLineID
ProductInventory
LocationID
Shelf
Bin
Quantity
Document
DocumentNode
DocumentLevel
Title
Owner
FolderFlag
FileName
FileExtension
Revision
ChangeNumber
Status
DocumentSummary
Document
UnitMeasure
Location
CostRate
Availability
WorkOrderRouting
WorkOrderID
OperationSequence
PlannedCost
WorkOrder
ScrapReasonID