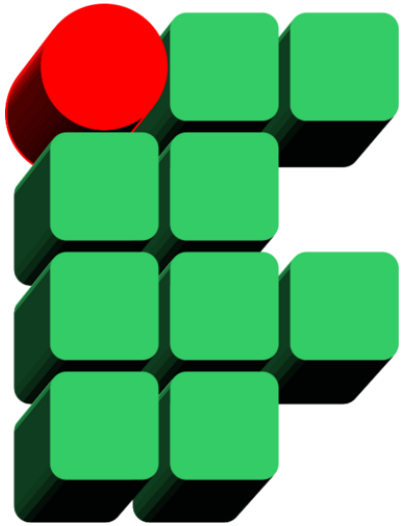
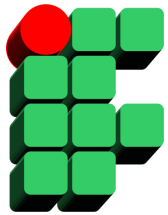


Formação de DBAs SQL Server 2005

Parte 4: Transact-SQL

Material Cedido pelo Professor Leandro Coelho Correia

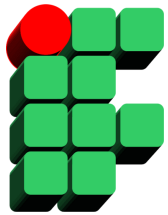
Transact-SQL



- Transact-SQL é uma implementação da linguagem SQL (Structured Query Language), baseada nos padrões da ISO (International Standards Organization) e da ANSI (American National Standards Institute), com algumas extensões;

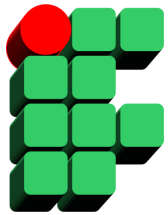
- Operações de administração, criação, alteração e exclusão de objetos, consultas e alteração de dados são realizadas usando Transact-SQL.

Tipos de Dados



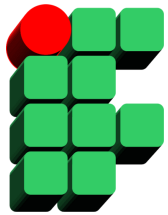
Inteiros	Bigint Int Smallint Tinyint	-2^63 a 2^63 -1 -2^31 a 2^31 -1 -32.768 a 32.767 0 a 255
Bit	Bit	0 ou 1
Numérico	Decimal Numeric	-10^38 +1 a 10^38 –1 Precisão fixa
Monetário	Money Smallmoney	-2^63 a 2^63 –1 -214.748,3648 a 214.748,3647 4 casas decimais
Numérico de ponto flutuante	Float Real	-1,79E + 308 a 1,79E + 308 -3,40E + 38 a 3,40E + 38

Tipos de Dados



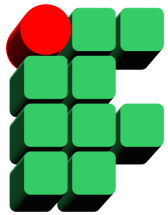
Data e Hora	Datetime Smalldatetime	01/01/1753 a 31/12/9999. 01/01/1900 a 06/06/2079 com precisão de minuto.
String	Char Varchar Text	String tamanho fixo até 8000 caracteres. String tamanho variável até 8000 caracteres. String tamanho variável até 2^31 – 1 caracteres.
String Unicode	nchar nvarchar Ntext	String unicode tamanho fixo até 4000. String unicode tamanho variável até 4000. String unicode tamanho variável até 2^30 – 1 caracteres.
Binários	Binary Varbinary Image	Binário tamanho fixo até 8000 bytes. Binário tamanho variável até 8000 bytes. Binário tamanho variável até 2^31 – 1 bytes.

Novos Tipos de Dados (Somente SQL2005)



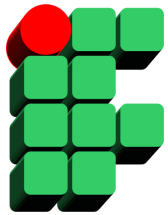
- Large-Value Data Types:
  - Varchar(max), NVarchar(max), Varbinary(max);
  - Permitem armazenar até 2Gb de dados;
  - Substituem os antigos Text, NText e Image, garantindo maior flexibilidade.
- XML
  - Permite o armazenamento de um documento ou fragmento XML em variáveis ou campos de tabelas;
  - Vantagens:
    - Permite a criação de índices específicos para esse tipo de dado;
    - Suporta consultas através da linguagem XQuery;
    - A estrutura do conteúdo XML pode ser validada.
  - Limitação:
    - O conteúdo XML original não é plenamente preservado.

Variáveis



- Variáveis Globais:
  - Representadas por @@;
  - Não precisam ser declaradas;
  - São atualizadas automaticamente pelo SGBD;
  - Escopo:
    - Sessão:
      - @@spid: número da sessão corrente.
    - Servidor:
      - @@version: versão do SGBD.
- Variáveis Locais:
  - Representadas por @;
  - Precisam ser explicitamente declaradas;
  - São atualizadas pelo usuário;
  - Escopo restrito à sessão.

Variáveis Locais



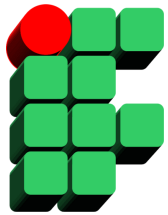
- Comando DECLARE:
  - Utilizado para declaração de variáveis locais.
  - Sintaxe:

    DECLARE @local_variable data_type

  - Exemplo:

    DECLARE @nome varchar(50)

Variáveis Locais



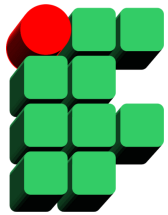
- Atribuição através do comando SET:

  SET @local_variable = expression

- Exemplos:

  - **SET** @nome = ‘Maria’
  - **SET** @lucro = @receita - @despesa
  - **SET** @media = select avg(nota) from aluno

Variáveis Locais



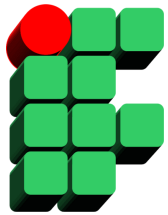
- Atribuição através do comando SELECT:

```
SELECT   @local_variable1 = expression,
                  @local_variable2 = expression,
                  ...
                  @local_variableN = expression
FROM ...
```

  - Exemplo:

```
SELECT        @nome = C.nome_cliente,
                        @idade = C.idade_cliente
FROM cliente C
WHERE C.idade_cliente > 55
```

Comandos de Controle de Fluxo



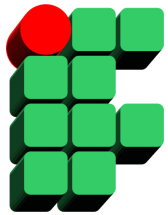
BEGIN ... END Estabelecem um bloco de código/comandos SQL

```
BEGIN
        { sql_statement | statement_block }
END
```

GOTO Desvio incondicional no fluxo de execução.

```
GOTO  label
        . . .
label:
```

Comandos de Controle de Fluxo



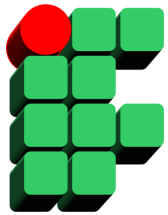
IF ... ELSE Impõe condições de execução para blocos de código SQL.

```
IF      boolean_expression
        { sql_statement | statement_block }
[ELSE   [IF boolean_expression]
        { sql_statement | statement_block } ]
```

WAITFOR Especifica um horário ou intervalo de espera.

```
WAITFOR { DELAY “time” |  TIME “time” }
```

Comandos de Controle de Fluxo

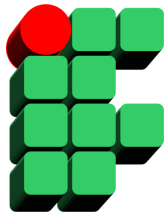


WHILE Cria uma estrutura de repetição para a execução de código SQL.

```
WHILE boolean_expression
        { sql_statement | statement_block }
        [BREAK]
        {sql_statement | statement_block }
        [CONTINUE]
```

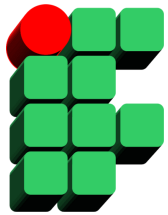
BREAK	Causa uma saída do loop;
CONTINUE	Reinicia o loop, ignorando comandos que estiverem após o Continue.

Tabelas Temporárias



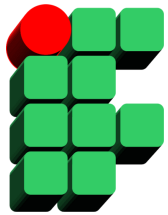
- Tabelas similares às tabelas permanentes;
- Armazenadas no TempDB;
- Removidas automaticamente quando não são mais utilizadas;
- Tabelas Temporárias Locais:
  - Nome começa com o caracter (#);
  - Visível apenas para a conexão corrente do usuário;
  - Excluída após desconexão do usuário.
- Tabelas Temporárias Globais:
  - Nome começa com os caracteres (##);
  - Visível a todos os usuários;
  - Excluídas após a desconexão de todos os usuários que fazem referência a ela.

Tratamento de Erros



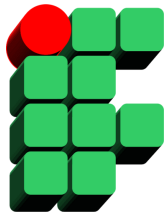
- Severidade
  - Indica a natureza e a gravidade do erro;
  - Níveis de severidade não críticos:
    - 0 a 10 Mensagens informativas;
    - 11 a 16 Erros que podem ser corrigidos pelo usuário;
    - 17 Indica falta de recursos para a execução de uma rotina;
    - 18 Erro interno não crítico. O comando executado é finalizado, mas a sessão com o SGBD é mantida.
  - Níveis de severidade críticos:
    - 19 Indica que limite não configurável do SGBD foi excedido. A rotina causadora é encerrada, mas a conexão com o banco é mantida;
    - 20 a 25 Erros fatais que indicam falhas no SGBD. A conexão com o servidor é fechada.

Tratamento de Erros



- Visão sys.messages
  - Permite visualizar mensagens de erro predefinidas;
  - A cada mensagem é atribuída uma severidade (de 1 a 25);
  - Erros com severidade entre 19 e 25 são gravados automaticamente no log de erros do SGBD;
  - Usuários podem cadastrar e disparar erros com severidade de 1 a 18;
  - Somente administradores podem cadastrar e disparar erros com severidade de 19 a 25.

Tratamento de Erros



SP_ADDMESSAGE [@msgnum =] msg_id ,
[@severity =] severity ,
[@msgtext =] 'msg'
[, [@with_log =] { true | false }]

Exemplo:

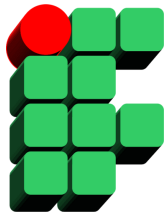
EXEC **SP_ADDMESSAGE** 50001, 10, ‘Parâmetro inválido’, false

SP_DROPMESSAGE [@msgnum =] *message_number*

Exemplo:

EXEC **SP_DROPMESSAGE** *50001*

Tratamento de Erros



- Comando RAISERROR
  - Retorna uma mensagem de erro predefinida (sysmessages) ou uma mensagem ad hoc:

RAISERROR **(** { msg_id | msg_str } { **,** severity **,** state }
[**,** argument [**,**...n]] **)**
[WITH option [**,**...n]]

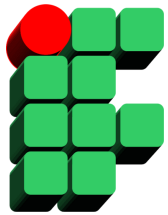
Exemplo:

RAISERROR (‘Informe o nome do usuário’, 10, 1)

RAISERROR (50012,10,1)

RAISERROR (50013, 15, 1, @@spid)

Tratamento de Erros

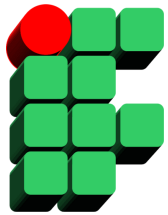


- TRY...CATCH:
  - Utilizado para controlar o tratamento de erros dentro de um bloco de comandos de forma similar às linguagens C# e Java;
  - Os comandos são executados dentro do bloco TRY e quando algum erro ocorre, o controle é transferido ao bloco CATCH.

  - Sintaxe:

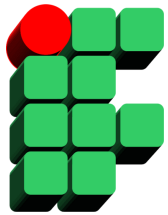
```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    { sql_statement | statement_block }
END CATCH
```

Tratamento de Erros



- TRY...CATCH:
  - Dentro do bloco CATCH as seguintes funções podem ser utilizadas para consultar informações sobre o erro gerado:
    - ERROR_NUMBER()
      - Retorna o número do erro.
    - ERROR_SEVERITY():
      - Retorna a severidade do erro.
    - ERROR_STATE():
      - returns the error state number.
    - ERROR_PROCEDURE():
      - Retorna o nome da procedure ou trigger que contém o comando gerador do erro.
    - ERROR_LINE():
      - Retorna a linha onde o erro ocorreu.
    - ERROR_MESSAGE():
      - Retorna o texto da mensagem de erro.

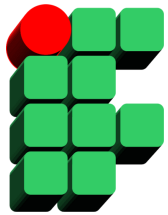
Tratamento de Erros



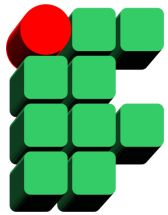
- TRY...CATCH:
  - Exemplo:

```
USE AdventureWorks;
GO

BEGIN TRY
    -- Força um erro de divisão por 0
    SELECT 1/0;
END TRY
BEGIN CATCH
    SELECT ERROR_LINE() AS ErrorLine;
END CATCH;
GO
```

Tratamento de Erros

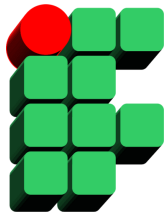


- Variável @@error:
  - Captura o erro gerado pelo último comando SQL executado em uma determinada sessão;
  - @@error = 0 → Comando executado com sucesso.



Stored Procedures

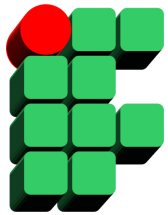
Stored Procedures



- Definição:
  - Trecho de código, normalmente escrito em linguagem SQL, que é compilado, armazenado e executado no servidor de banco de dados.

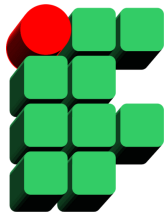
- Benefícios:
  - Performance:
    - Sintaxe é verificada no momento de criação;
    - Plano de execução é criado (compilação) na 1a. execução;
    - Nas execuções sub-seqüentes, não é necessário verificar sintaxe e o plano de execução em cache é utilizado.

Stored Procedures



- Benefícios:
  - Reutilização:
    - Uma vez criada, a procedure pode ser executada sempre que necessário;
    - Facilita alterações/manutenções, sem gerar impacto nas aplicações.
  - Segurança:
    - Pode ser concedido acesso ao usuário para executar uma procedure sem que ele tenha acesso direto aos objetos manipulados por ela;
    - Código da procedure pode ser armazenado criptografado.

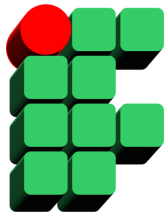
Criando Stored Procedures



```
CREATE PROCEDURE procedure_name
            [ { @parameter data_type } [=DEFAULT] [OUTPUT]
[ WITH ENCRYPTION | EXECUTE AS ... ]
AS
    sql_statements...
```

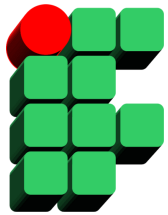
- **@parameter**
  - Seguem os padrões estabelecidos para variáveis locais;
  - Cada SP suporta até 2.100 parâmetros.
- **DEFAULT**
  - Valor default para um parâmetro não informado.
- **OUTPUT**
  - Indica um parâmetro de saída da procedure.
- **ENCRYPTION**
  - Armazena o código da procedure criptografada.
- **EXECUTE AS**
  - Define o contexto de segurança para execução da procedure.

Retorno de uma SP



- RETURN
  - Encerra incondicionalmente um procedimento;
  - Utilizado para gerar um código de retorno (valor inteiro) para a aplicação/usuário que executou o procedimento.

Exemplos de SPs



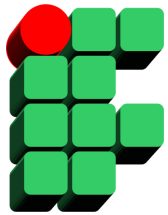
Procedure que retorna o resultado de SELECT:

```
CREATE PROCEDURE HumanResources.uspGetAllEmployees
AS
    SELECT    LastName, FirstName, JobTitle, Department
    FROM      HumanResources.vEmployeeDepartment;
GO
```

Executando:

```
EXECUTE HumanResources.uspGetAllEmployees;
```

Exemplos de SPs



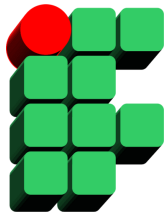
Procedure com passagem de parâmetros:

```
CREATE PROCEDURE HumanResources.uspGetEmployees
                                    @LastName nvarchar(50),
                                    @FirstName nvarchar(50)
AS
   SELECT FirstName, LastName, JobTitle, Department
   FROM HumanResources.vEmployeeDepartment
   WHERE FirstName = @FirstName AND LastName = @LastName;
GO
```

Executando:

```
EXECUTE HumanResources.uspGetEmployees N'Ackerman', N'Pilar';
-- ou
EXEC HumanResources.uspGetEmployees   @LastName = N'Ackerman',
                                      @FirstName = N'Pilar';
```

Exemplos de SPs

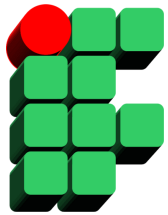


Procedure com parâmetros de OUTPUT:

```
CREATE PROCEDURE Production.uspGetList @Product varchar(40)
                                       , @MaxPrice money
                                       , @ListPrice money OUTPUT
AS
SET @ListPrice = (SELECT MAX(p.ListPrice)
      FROM Production.Product AS p
      JOIN  Production.ProductSubcategory AS s
        ON p.ProductSubcategoryID = s.ProductSubcategoryID
      WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice);
GO
```

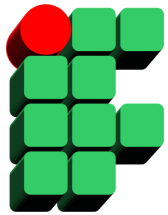
Executando:

```
DECLARE @Cost money
EXECUTE Production.uspGetList '%Bikes%', 700, @Cost OUTPUT
```



Transacões e Níveis de Isolamento

Definição de Transação



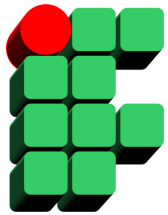
- Seqüência de operações realizadas como uma unidade lógica de trabalho.

Transação

```
Comando 1
Comando 2
Comando 3
. . .
```

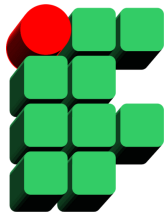
- Propriedades:
  - Atomicidade;
  - Consistência;
  - Isolamento;
  - Durabilidade.

Bloqueios



- Definição:
  - Mecanismo que permite a uma transação impedir que outras acessem ou atualizem registros de forma a evitar os problemas de concorrência.
- XLOCK:
  - Bloqueio exclusivo sobre os recursos;
  - Utilizado durante operações de atualização.
- SLOCK:
  - Bloqueio compartilhado sobre os recursos;
  - Utilizado durante operações de consulta.

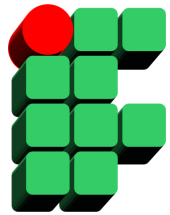
Bloqueios



- Relação entre os Bloqueios:

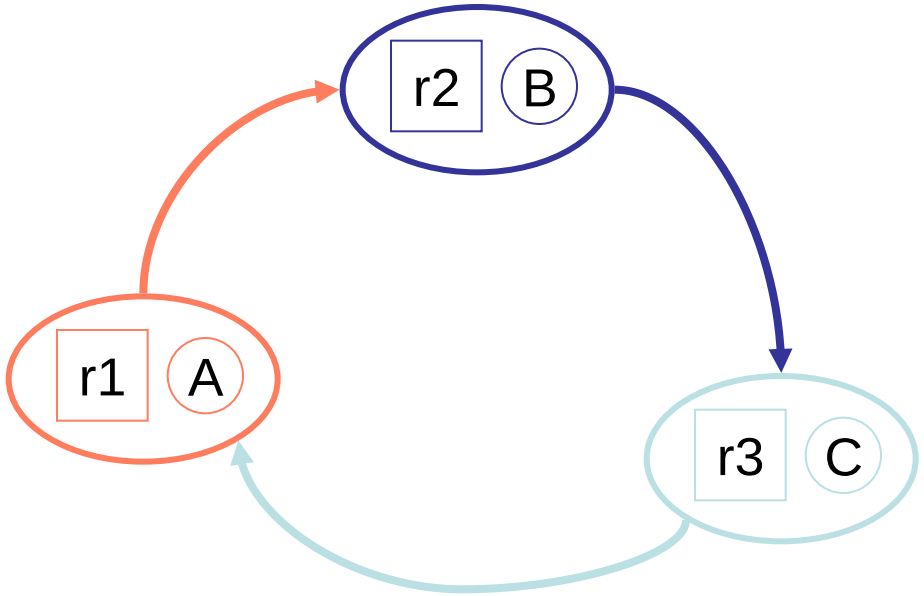
	XLOCK	SLOCK
XLOCK	Não	Não
SLOCK	Não	Sim

Bloqueios

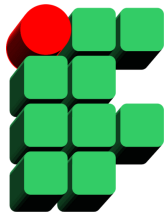


- Problemas gerados pela utilização de bloqueios:
  - Deadlock
    - Dependência cíclica entre dois ou mais processos em relação a um conjunto de recursos.

Transações A, B, e C
Recursos r1, r2 e r3

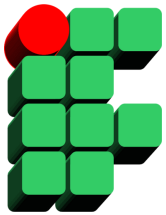


Bloqueios

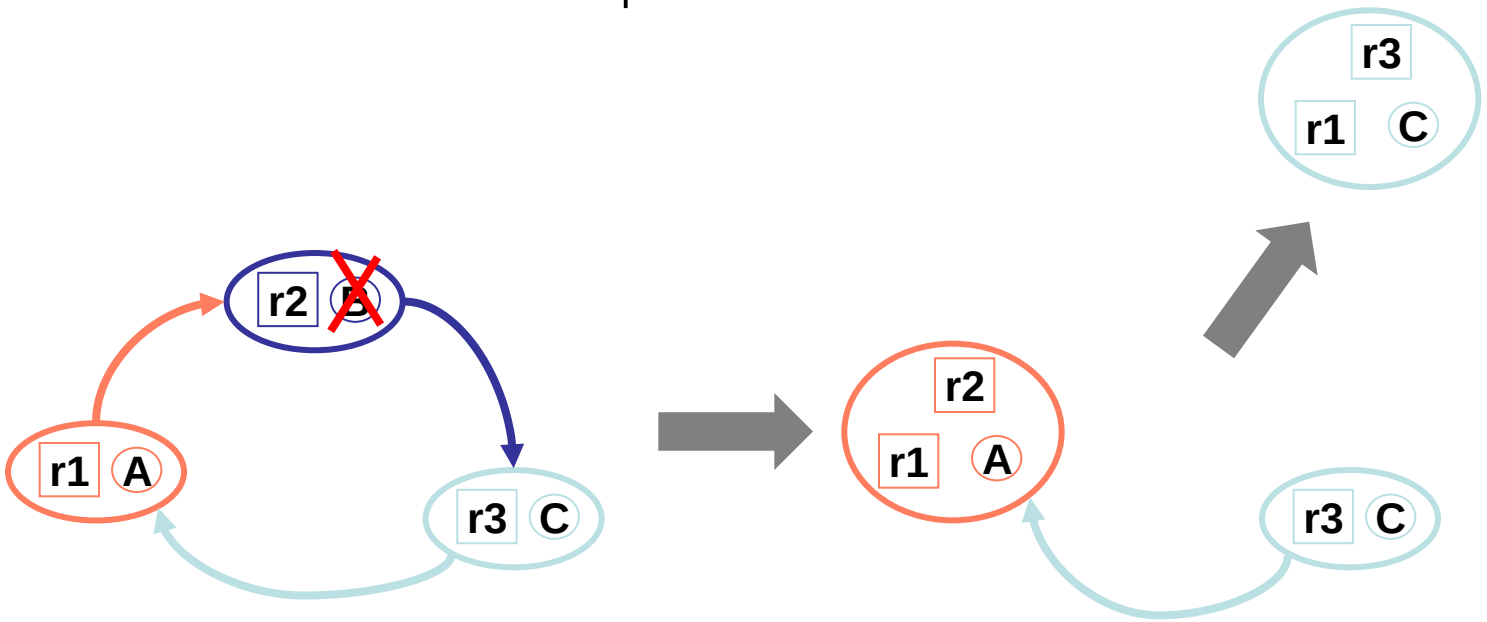


- Problemas gerados pela utilização de bloqueios:
  - Deadlock
    - Tratamento automático pelo SGBD:
      - Lock Monitor Thread: responsável pela detecção automática de deadlocks;
      - A transação que tiver o menor custo de rollback será sacrificada;
      - O comando SET DEADLOCK_PRIORITY LOW pode ser usado para aumentar a possibilidade da sessão corrente ser sacrificada quando ocorrer um deadlock.

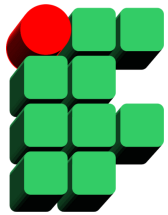
Bloqueios



- Problemas gerados pela utilização de bloqueios:
  - Deadlock
    - Tratamento automático pelo SGBD:

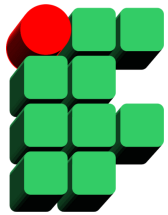


Bloqueios



- Problemas gerados pela utilização de bloqueios:
  - Monopólio de recursos
    - Espera indefinida por um recurso que está bloqueado;
    - O comando SET LOCK_TIMEOUT permite configurar qual o tempo máximo (em milisegundos) pelo qual uma sessão irá aguardar por um recurso que está bloqueado.
      - O valor -1 indica espera indefinida (sem timeout);
      - O valor 0 indica timeout imediato (sem espera);
      - Ao final do período configurado, o SGBD retorna o erro 1222;
      - O erro 1222 não interrompe a transação e precisa ser tratado;
      - A variável @@LOCK_TIMEOUT contém o valor atual de timeout configurado.

Transações no SQL Server 2005



- Variável @@TRANCOUNT:
  - Registra o nível de “aninhamento” das transações na sessão corrente.

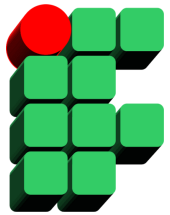
@@TRANCOUNT > 0

→A sessão está em estado de transação

@@TRANCOUNT = 0

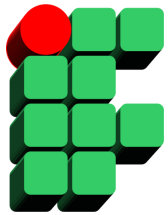
→A sessão não está em estado de transação

Transações no SQL Server 2005



- Variável @@TRANCOUNT:
  - A cada BEGIN TRANSACTION, o valor de @@TRANCOUNT é incrementado;
  - A cada COMMIT TRANSACTION, o valor de @@TRANCOUNT é decrementado;
  - Quando um ROLLBACK TRANSACTION é executado, o valor de @@TRANCOUNT é atualizado para zero.

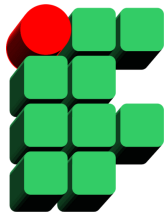
Considerações sobre Transações



- O SQL Server não permite o aninhamento de transações e a variável @@TRANCOUNT é somente um mecanismo para informar se uma transação está ou não em execução na sessão corrente;

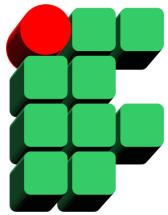
- Transações exigem recursos adicionais para serem executadas, aumentam a contenção dos recursos e consequentemente as chances de deadlock, logo, devem ser utilizadas de forma criteriosa e no menor escopo possível.

Nível de Isolamento



- Propriedade que define o grau em que uma transação deve ser isolada de outras transações:
  - Um nível de isolamento menor permite uma maior concorrência, mas está sujeito a mais problemas de consistência;
  - Um nível de isolamento maior permite uma menor concorrência, mas está sujeito a menos problemas de consistência.

Problemas de Concorrência

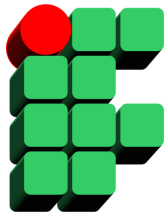


- Leitura Suja (Dirty Read):

1. Uma transação (T1) executa um update em um registro (R);
2. Uma outra transação (T2) recupera o registro R, com valor alterado;
3. A transação T1 termina através de um rollback.

– T2 acessou o registro R com um valor que nunca foi persistido (registro “sujo”).

Problemas de Concorrência

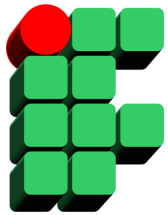


- Leitura não Repetível (Nonrepeatable Read)

1. Uma transação (T1) recupera um registro (R);
2. Uma outra transação (T2) atualiza o registro R;
3. A transação T1 recupera novamente o registro R, já alterado.

– T1 recupera o mesmo registro, em dois momentos distintos, com diferentes valores.

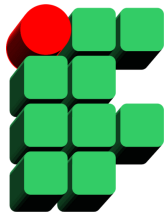
Problemas de Concorrência



- Registro Fantasma (Phantom):

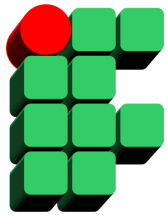
1. Uma transação (T1) recupera um conjunto de registros que satisfazem uma determinada condição;
2. Uma outra transação (T2) insere um registro que satisfaz a mesma condição da consulta realizada por T1;
3. Se a transação T1 realizar novamente a consulta, haverá um novo registro em relação ao conjunto recuperado anteriormente, um “registro fantasma” (phantom).

Níveis de Isolamento (SQL-92)



Nível de Isolamento	Leitura Suja (Dirty Read)	Leitura não Repetível (Nonrepeatable Read)	Registro Fantasma (Phantom)
Read Uncommitted	Ocorre	Ocorre	Ocorre
Read Committed	Não ocorre	Ocorre	Ocorre
Repeatable Read	Não ocorre	Não ocorre	Ocorre
Serializable	Não ocorre	Não ocorre	Não ocorre

Níveis de Isolamento

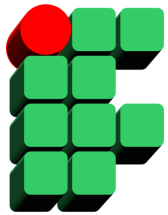


SET TRANSACTION ISOLATION LEVEL { *READ COMMITTED*
| *READ UNCOMMITTED*
| *REPEATABLE READ*
| *SERIALIZABLE* }

Exemplo:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
GO
BEGIN TRANSACTION
    SELECT ...
    UPDATE ....
    ...
COMMIT TRANSACTION
```

Versionamento (SQL Server 2005)

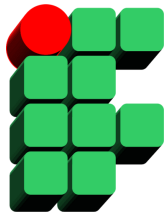


- O SQL Server 2005 permite a utilização de 2 configurações que habilitam o isolamento baseado em versionamento: READ_COMMITTED_SNAPSHOT e ALLOW_SNAPSHOT_ISOLATION;

- Com versionamento, o banco de dados mantém cópias dos dados anteriores ao início de uma atualização, permitindo que outras transações possam consultar os valores antigos até que a atualização termine;

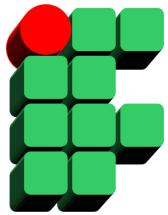
- A utilização de versionamento permite uma concorrência maior em ambientes onde frequentemente operações de consulta precisam ser executadas em paralelo com atualizações;

- O versionamento acarreta um custo adicional pela forte utilização do TempDB e pode gerar conflitos de atualização.



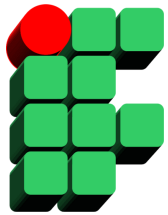
Triggers

Triggers



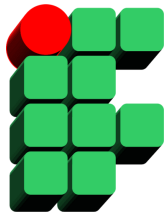
- Procedimentos executados automaticamente, associados a uma operação de atualização (insert, update ou delete), comandos DDL (create table, alter procedure, drop index, etc) ou logon do usuário;
- O comando que dispara o trigger e o código do próprio trigger são executados em estado de transação.

Tipos de Trigger



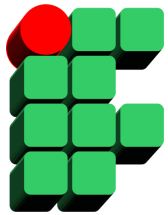
- DML Trigger:
  - After
    - Trigger é executado após o comando de atualização;
    - Caso o comando de atualização cause um erro, o trigger não é disparado;
    - Podem ser utilizados apenas sobre tabelas;
    - Podem existir múltiplos triggers para cada comando de atualização (Insert, Update, Delete).

Tipos de Trigger



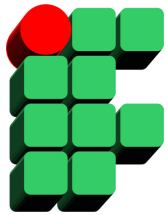
- DML Trigger:
  - Instead Of:
    - Trigger é executado no lugar do comando de atualização;
    - Permitida a criação de apenas um trigger para cada comando de atualização (Insert, Update, Delete);
    - Podem ser utilizados sobre tabelas e visões;
    - Utilizado para permitir ou estender as possibilidades de update sobre visões.

Tabelas Inserted e Deleted



- Tabelas temporárias automaticamente criadas pelo SQL Server durante a execução de um trigger;
- Possuem a mesma estrutura da tabela base (ou visão);
- Utilizadas para identificar os registros atualizados pelos comandos que dispararam o trigger;
- A tabela INSERTED possui os registros que foram afetados pelos comandos INSERT e UPDATE;
- A tabela DELETED possui os registros que foram afetados pelos comandos UPDATE e DELETE.

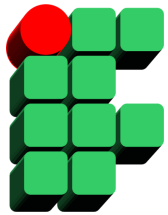
Tabelas Inserted e Deleted



- Relação entre as operações e as tabelas:

	Inserted	**Deleted**
Insert	Cópia dos registros que foram incluídos.	X
Update	Cópia dos registros alterados, contendo os novos valores.	Cópia dos registros alterados, contendo os valores antigos.
Delete	X	Cópia dos registros que foram excluídos.

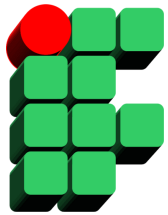
Criando Triggers DML



```
CREATE TRIGGER trigger_name ON { table | view }
[ WITH ENCRYPTION | EXECUTE AS ...]
        { FOR | AFTER | INSTEAD OF }
                { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
    AS
        sql_statement...
```

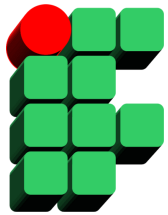
- **AFTER**
  - O código do trigger será executado após a operação que o disparou.
- **INSTEAD OF**
  - O código do trigger será executado no lugar da operação que o disparou.
- **ENCRYPTION**
  - Armazena o código do trigger criptografado.
- **EXECUTE AS**
  - Define o contexto de segurança para execução do trigger.

Exemplo de Trigger DML



```
CREATE TRIGGER Purchasing.iPurchaseOrderDetail ON Purchasing.PurchaseOrderDetail  AFTER INSERT AS
BEGIN
   BEGIN TRY
      INSERT INTO Production.TransactionHistory  (      ProductID, ReferenceOrderID, ReferenceOrderLineID,
                                                        TransactionType, TransactionDate, Quantity, ActualCost)

      SELECT inserted.ProductID, inserted.PurchaseOrderID, inserted.PurchaseOrderDetailID,
                                    'P', GETDATE(), inserted.OrderQty, inserted.UnitPrice
      FROM inserted INNER JOIN Purchasing.PurchaseOrderHeader
         ON inserted.PurchaseOrderID = Purchasing.PurchaseOrderHeader.PurchaseOrderID;
   END TRY
   BEGIN CATCH
      IF @@TRANCOUNT > 0
      BEGIN
                                    RAISERROR(...)
                           ROLLBACK TRANSACTION;
      END
   END CATCH;
END;
```

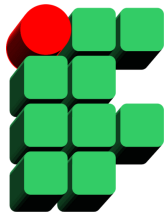
Tipos de Trigger (SQL Server 2005)



- DDL Triggers:
  - Utilizados para controlar ou auditar a execução de comandos DDL (create, alter, drop, etc);
  - Assim como o trigger AFTER, o trigger DDL é disparado após a execução, com sucesso, do comando DDL.

- Logon Triggers:
  - Utilizados para controlar ou auditar o acesso dos usuários ao SGBD;
  - Assim como o trigger AFTER, o trigger de logon é disparado após a execução, com sucesso, do logon.

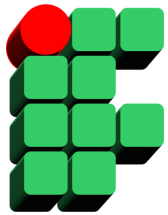
Criando Triggers DDL



```
CREATE TRIGGER trigger_name
ON              { ALL SERVER | DATABASE }
[ WITH          ENCRYPTION | EXECUTE AS ...]
                                { FOR | AFTER } { DDL_event } [ ,...n ]
    AS { sql_statement }
```

- **ALL SERVER**
  - Indica que o escopo de atuação do trigger é a instância como um todo.
- **DATABASE**
  - Indica que o escopo de atuação do trigger está restrito ao banco de dados onde o trigger foi criado.
- **DDL_Event**
  - Comando(s) DDL que dispara(m) o trigger.

Exemplo de Trigger DDL

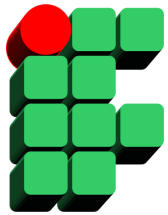


```
CREATE TRIGGER trigger_name
ON              { ALL SERVER | DATABASE }
[ WITH          ENCRYPTION | EXECUTE AS ...]
                                { FOR | AFTER } { DDL_event } [ ,...n ]
      AS { sql_statement }
```

Exemplo:

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE
AS
  PRINT 'You must disable Trigger ‘’safety’’ to drop tables!‘
  ROLLBACK
GO
```

Criando Triggers de Logon

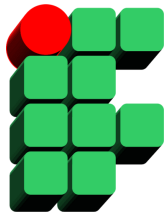


```
CREATE TRIGGER trigger_name
ON              ALL SERVER
[ WITH          ENCRYPTION | EXECUTE AS ...]
                                { FOR | AFTER } LOGON
      AS { sql_statement }
```

Exemplo:

```
CREATE TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'login_test' FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'login_test' AND
   (SELECT COUNT(*) FROM sys.dm_exec_sessions
         WHERE is_user_process = 1 AND original_login_name = 'login_test') > 3
   ROLLBACK;
END;
```

Considerações

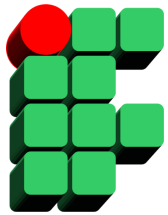


- A execução de triggers pode ser recursiva a depender da configuração ***recursive triggers*** de cada banco de dados;

- A execução de triggers pode ser aninhada (um trigger disparar outro trigger) a depender da configuração ***nested triggers*** do servidor;

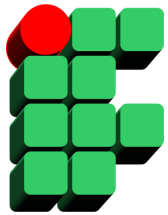
- Triggers podem ser habilitados e desabilitados através do comando ALTER TABLE...ENABLE / DISABLE TRIGGER;

- DML Triggers e a tabela associada precisam pertencer ao mesmo schema.



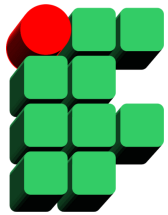
Cursores

Definição de Cursor



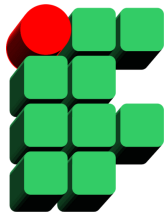
- Mecanismo que permite a manipulação de um registro a cada vez, a partir de um comando SELECT;
- Tipos de Cursor:
  - Montagem:
    - Static;
    - Keyset-driven;
    - Dynamic.
  - Navegação:
    - Forward-only;
    - Scroll.

Tipos de Cursor (montagem)



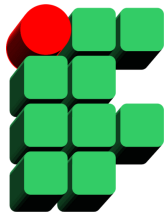
- Static Cursor:
  - Conjunto de registros do cursor é montado no momento em que o cursor é aberto;
  - O resultado é totalmente armazenado no TempDB;
  - Atualizações sobre os dados não são visíveis através do cursor;
  - Requer menos recursos para a navegação.

Tipos de Cursor (montagem)



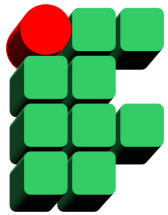
- Dynamic Cursor:
  - Somente uma pequena parte do cursor (conjunto de apontadores) é armazenado no TempDB;
  - As atualizações sobre os dados são visíveis através do cursor;
  - Os dados são consultados na tabela original a cada navegação;
  - Requer um conjunto de recursos maior para navegação;
  - Não permite a opção ABSOLUTE no comando FETCH.

Tipos de Cursor (montagem)



- Keyset-driven Cursor:
  - Conjunto de registros do cursor é montado no momento em que o cursor é aberto;
  - Um conjunto de identificadores de registros (keyset) é armazenado no TempDB;
  - Atualizações sobre os atributos não pertencentes ao keyset são visíveis através do cursor;
  - Comandos INSERT de outros usuários não são visíveis através do cursor;
  - Comandos DELETE são detectados através da variável @@FECTHSTATUS com valor -2, durante a navegação.
  - Os atributos não pertencentes ao keyset são consultados na tabela original a cada navegação.

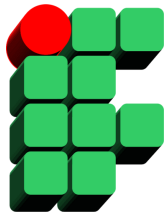
Tipos de Cursor (navegação)



- Forward-only Cursor:
  - Suporta apenas fetch de linhas serializado (FECTH NEXT).

- Scroll Cursor
  - Suporta fetch de linhas não serializado (FETCH PRIOR, FIRST, LAST, ABSOLUTE, RELATIVE).

Trabalhando com Cursores



DECLARE *cursor_name* **CURSOR**

[FORWARD_ONLY | SCROLL]

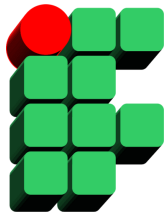
[STATIC |KEYSET | DYNAMIC]

FOR *select_statement*

Exemplo:

```
DECLARE contact_cursor CURSOR FOR
   SELECT LastName, FirstName FROM Person.Contact
   WHERE LastName LIKE 'B%'
   ORDER BY LastName
```

Trabalhando com Cursores

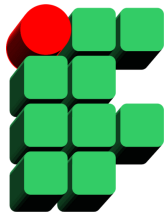


OPEN *cursor_name*

- Comando que abre o cursor;
- Momento em que os dados do cursor são populados em tabelas temporárias (TempDB) de acordo com o seu tipo;
- Variável **@@CURSOR_ROWS**: retorna o número de linhas do cursor, após a sua abertura.

Exemplo: OPEN contact_cursor

Trabalhando com Cursores



FETCH

[[NEXT | PRIOR | FIRST | LAST | ABSOLUTE { *n* | **@***nvar* }

|RELATIVE { *n* | **@***nvar* }]

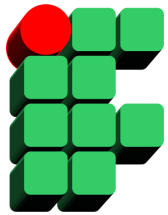
FROM *cursor_name* [**INTO @***variable_name* [**,**...n]]

- Recupera uma linha do cursor;
- Armazena o registro recuperado em variáveis locais.

Exemplo:

```
FETCH NEXT FROM contact_cursor
INTO @LastName, @FirstName
```

Trabalhando com Cursores



@@FETCH_STATUS

- Retorna o status do último comando FETCH executado;

  0 → Fetch executado com sucesso

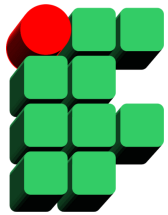
  -1 → Fetch executado com erro

  -2 → Registro do cursor ausente

Exemplo:

```
WHILE @@FETCH_STATUS = 0
BEGIN
   PRINT 'Contact Name: ' + @FirstName + ' ' +  @LastName
   FETCH NEXT FROM contact_cursor INTO @LastName, @FirstName
END
```

Trabalhando com Cursores

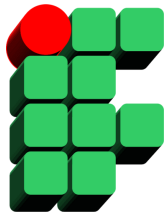


CLOSE *cursor_name*

- Comando que fecha o cursor;
- Comandos de recuperação de registros não serão permitidos até que o cursor seja novamente aberto.

Exemplo: CLOSE contact_cursor

Trabalhando com Cursores

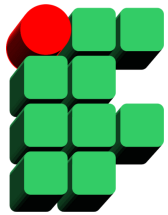


DEALLOCATE *cursor_name*

• Comando que remove a declaração do cursor.

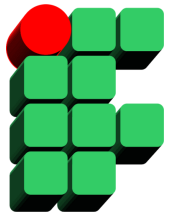
Exemplo: DEALLOCATE contact_cursor

Trabalhando com Cursores



```
DECLARE @LastName varchar(50), @FirstName varchar(50)

DECLARE contact_cursor CURSOR FOR
        SELECT LastName, FirstName FROM Person.Contact
        WHERE LastName LIKE 'B%'
        ORDER BY LastName, FirstName

OPEN contact_cursor

FETCH NEXT FROM contact_cursor INTO @LastName, @FirstName

WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT 'Contact Name: ' + @FirstName + ' ' +  @LastName
  FETCH NEXT FROM contact_cursor INTO @LastName, @FirstName
END

CLOSE contact_cursor
DEALLOCATE contact_cursor
```

Considerações sobre Cursores



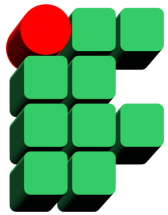
- Quando não utilizar cursores:
  - Quando não houver necessidade de tratamento registro a registro e todo conjunto de registros precisar ser enviado ao cliente (ex.: relatórios);

- Cursores dynamic são mais rápidos durante a abertura (OPEN) e minimizam a utilização do TempDB, mas são mais sujeitos a problemas de concorrência;

- Cursores baseados em joins tendem a ser mais rápidos quando implementados como static or keyset-driven;

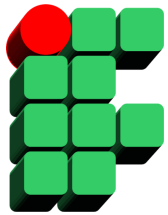
- A opção ABSOLUTE do comando FECTH só pode ser utilizada em cursores static or keyset-driven.

Considerações sobre Cursores



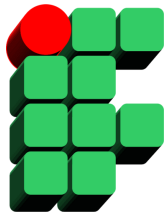
- Declarações:
  - Somente FORWARD_ONLY → DYNAMIC é o default;
  - Somente STATIC, KEYSET ou DYNAMIC → SCROLL é o default.

- Cursores FAST_FORWARD:
  - Forward-Only + Static;
  - Melhor desempenho para consultas seriais quando não existe necessidade de atualização.



Funções Definidas pelo Usuário

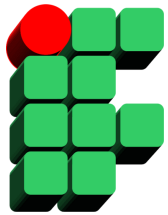
Definição de UDF



- Built-in Functions:
  - Estão prontas para uso após a instalação do SQL Server 2005;
  - Não podem ser alteradas pelo usuário;
  - Exemplos:
    - Funções de data: dateadd(...), datediff(...), datepart(...);
    - Funções de manipulação de strings: left(...), right(...), substring(...);
    - Funções matemáticas: sin(...), cos(...), round(...), rand(...);
    - ...

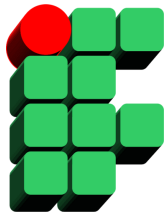
- User-Defined Functions (UDFs):
  - São criadas, alteradas e excluídas pelo usuário;
  - Algumas funções no SQL Server 2005 são implementadas como UDFs (fn_helpcollations(...), fn_trace_gettable(...), etc).

Tipos de UDF



- Scalar Function;
  - Função que retorna um único valor, do tipo de dado definido como retorno da função;
  - Os tipos de dados text, ntext, image, cursor, timestamp e table não são aceitos como tipos de dados de retorno;
  - Corpo da função definido dentro de um bloco BEGIN... END.

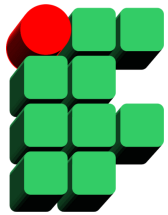
Tipos de UDF



- Exemplo:

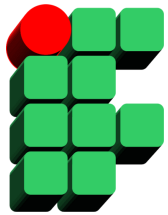
```
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
BEGIN
    DECLARE @ret int;

    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;
```

Tipos de UDF



- Inline Table Function;
  - Função que retorna uma tabela;
  - Retorno da função determinado por um comando SELECT;
  - Usada para implementação de visões parametrizáveis;
  - Podem ser utilizadas na cláusula FROM de comandos DML.

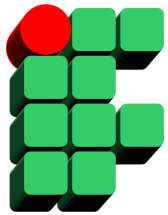
Tipos de UDF



- Exemplo:

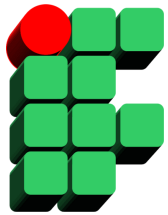
```
CREATE FUNCTION Sales.ufn_CustomerNamesInRegion
                                        ( @Region nvarchar(50) )
RETURNS table
AS
RETURN (
      SELECT DISTINCT S.Name AS Store, A.City
      FROM Sales.Store AS S
  JOIN Sales.CustomerAddress AS CA
            ON CA.CustomerID = S.CustomerID
            JOIN Person.Address AS A
            ON A.AddressID = CA.AddressID
            JOIN Person.StateProvince SP
            ON SP.StateProvinceID = A.StateProvinceID
       WHERE SP.Name = @Region
      );
```

Tipos de UDF



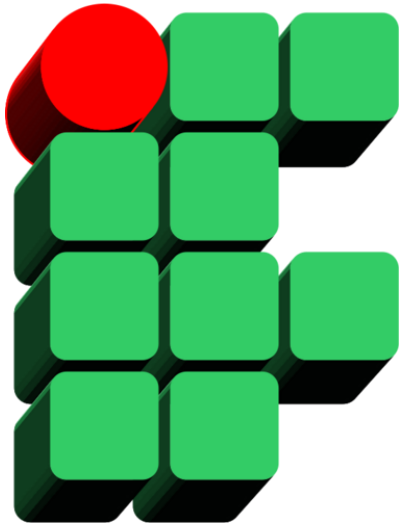
- Multistatement Table Function:
  - Função que retorna uma tabela;
  - Corpo da função definido dentro de um bloco BEGIN..END;
  - Contém comandos SQL que manipulam a tabela a ser retornada pela função;
  - Usada para implementação de visões parametrizáveis;
  - Podem ser utilizadas na cláusula FROM de comandos DML.

Tipos de UDF



- Exemplo:

```
CREATE FUNCTION LargeOrderShippers ( @FreightParm money )
RETURNS @OrderShipperTab TABLE
        (ShipperID int, ShipperName nvarchar(80), OrderID int,
        ShippedDate datetime, Freight money )
AS
BEGIN
        INSERT @OrderShipperTab
        SELECT S.ShipperID, S.CompanyName, O.OrderID, O.ShippedDate, O.Freight
         FROM Shippers AS S INNER JOIN Orders AS O
                ON S.ShipperID = O.ShipVia
        WHERE O.Freight > @FreightParm
        RETURN
END
```



Formação de DBAs SQL Server 2005
Parte 4: Transact-SQL