**DESEMPENHO**

* **Uso de % no início da pesquisa com LIKE**

**Descrição**

É recomendável utilizar % apenas no final da pesquisa com LIKE.

Há uma alternativa que pode ser usada no lugar do LIKE com “%” no início da pesquisa. Esta nova opção é o FullText Search, ele pode não só substituir o uso do comando LIKE, mas também ter ganho de performance quando implementado.

**Motivação para a regra**

Se um WHERE com LIKE utilizar % no início da pesquisa, irá gerar TABLE SCAN.

Há outro recurso disponível no SQL SERVER que pode substituir o uso do LIKE com grande vantagem em termos de performance. Este recurso é chamado FULLTEXT INDEX SEARCH. É recomendada a utilização do FULLTEXT a partir da versão 2005 do SQL SERVER.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT AU\_LNAME  FROM AUTHORS  WHERE AU\_LNAME LIKE '%SMITH' |  | **Observância da regra**  SELECT AU\_LNAME  FROM AUTHORS  WHERE AU\_LNAME LIKE 'SMITH%'  ou preferencialmente a utilização de FullText Search  SELECT AU\_LNAME  FROM AUTHORS  WHERE CONTAINS(AU\_LNAME, ' "Smith\*" '); |

* **Falta de utilização de nocount**
* **Descrição**  
  É recomendável ao iniciar um SQL BATCH ou compilação de procedures utilizando o comando SET NOCOUNT ON.
* **Motivação para a regra**  
  A observância desta regra minimiza o tráfego de rede com a mensagem 1 row(s) affected .
* **Critério para o apontamento**  
  Serão contadas todas as ocorrências encontradas.
* **Exceção**  
  Quando houver necessidade de retorno de resultset adicional ao aplicativo chamador.
* **Impacto**  
  Minor
* **Falso positivo**
* **Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  USE PUBS  GO  CREATE PROC SP\_IN\_AUTHORS  (  @AU\_LNAME VARCHAR(40) = NULL  , @AU\_FNAME VARCHAR(20) = NULL  , @PHONE CHAR(12)  , @ADDRESS VARCHAR(40) = NULL  , @CITY VARCHAR(20) = NULL  , @STATE CHAR(2) = NULL  , @ZIP CHAR(5) = NULL  , @CONTRACT BIT  )  AS  DECLARE  @ERROR INT,  @ROWCOUNT INT  BEGIN TRAN  INSERT INTO AUTHORS  (  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  )  VALUES  (  @AU\_ID  , @AU\_LNAME  , @AU\_FNAME  , @PHONE  , @ADDRESS  , @CITY  , @STATE  , @ZIP  , @CONTRACT  )  SELECT  @ERROR = @@ERROR,  @ROWCOUNT = @@ROWCOUNT  IF @ERROR > 0  BEGIN     SELECT @ERROR     ROLLBACK TRAN     RETURN @ERROR  END    IF @ROWCOUNT = 0  BEGIN      RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '      ROLLBACK TRAN      RETURN 30002  END  COMMIT TRAN  SELECT 0  RETURN 0  GO |  | **Observância da regra**  USE PUBS  GO  CREATE PROC SP\_IN\_AUTHORS  (  @AU\_LNAME VARCHAR(40) = NULL  , @AU\_FNAME VARCHAR(20) = NULL  , @PHONE CHAR(12)  , @ADDRESS VARCHAR(40) = NULL  , @CITY VARCHAR(20) = NULL  , @STATE CHAR(2) = NULL  , @ZIP CHAR(5) = NULL  , @CONTRACT BIT  )  AS  DECLARE  @ERROR INT,  @ROWCOUNT INT  SET NOCOUNT ON  BEGIN TRAN  INSERT INTO AUTHORS  (  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  )  VALUES  (  @AU\_ID  , @AU\_LNAME  , @AU\_FNAME  , @PHONE  , @ADDRESS  , @CITY  , @STATE  , @ZIP  , @CONTRACT  )  SELECT  @ERROR = @@ERROR,  @ROWCOUNT = @@ROWCOUNT  IF @ERROR > 0  BEGIN      SELECT @ERROR      ROLLBACK TRAN      RETURN @ERROR  END  IF @ROWCOUNT = 0  BEGIN      RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '      ROLLBACK TRAN      RETURN 30002  END  COMMIT TRAN  SELECT 0  RETURN 0  GO |

SELECT sem Where

**Descrição**  
É recomendável que SELECT possua cláusula de seleção de dados.

**Motivação para a regra**  
Evitar a ocorrência de TABLE SCAN e evitar que informações desnecessárias trafeguem pela rede.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**  
Quando feito um SELECT sem WHERE em uma tabela temporária ou tabela do tipo variável para popular uma tabela do Banco de dados Também há outra exceção quando se utiliza dentro de uma instrução SELECT um JOIN com uma tabela temporária ou uma tabela do tipo variável.

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLES.TITLE\_ID,         TITLES.TITLE,         TITLES.TYPE,         TITLES.PUB\_ID,         TITLES.PRICE,         TITLES.ADVANCE,         TITLES.ROYALTY,         TITLES.YTD\_SALES,         TITLES.NOTES,         TITLES.PUBDATE,         PUBLISHERS.PUB\_NAME,         PUBLISHERS.CITY,         PUBLISHERS.STATE,         PUBLISHERS.COUNTRY  FROM TITLES     INNER JOIN PUBLISHERS        ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID |  | **Observância da regra**  SELECT TITLES.TITLE\_ID,         TITLES.TITLE,         TITLES.TYPE,         TITLES.PUB\_ID,         TITLES.PRICE,         TITLES.ADVANCE,         TITLES.ROYALTY,         TITLES.YTD\_SALES,         TITLES.NOTES,         TITLES.PUBDATE,         PUBLISHERS.PUB\_NAME,         PUBLISHERS.CITY,         PUBLISHERS.STATE,         PUBLISHERS.COUNTRY  FROM TITLES     INNER JOIN PUBLISHERS       ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID  WHERE TITLES.PUB\_ID = 0877  ou  SELECT TITLES.TITLE\_ID,         TITLES.TITLE,         TITLES.TYPE,         TITLES.PUB\_ID,         TITLES.PRICE,         TITLES.ADVANCE,         TITLES.ROYALTY,         TITLES.YTD\_SALES,         TITLES.NOTES,         TITLES.PUBDATE,         PUBLISHERS.PUB\_NAME,         PUBLISHERS.CITY,         PUBLISHERS.STATE,         PUBLISHERS.COUNTRY  FROM TITLES     INNER JOIN #PUB\_CONTROL       ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID |

Concatenação de valores na clásula WHERE**Descrição**  
Não é recomendado usar campos concatenados na clausula WHERE para serem usados como critérios de pesquisa

**Motivação para a regra**  
A utilização deste recurso faz com que não seja usado nenhum índice criado na(s) tabela(s) envolvida(s) na query.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID  WHERE TITLE.TITLE + ‘ ‘ + TITLE.CATEGORY =  ‘EMILY ROSS ROMANCE’ |  | **Observância da regra**  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID  WHERE TITLE.TITLE = ‘EMILY ROSS’ AND TITLE.CATEGORY = ‘ROMANCE’ |

DELETE e UPDATE sem Where**Descrição**  
É recomendável que DELETE e UPDATE possuam cláusula de seleção de dados.

**Motivação para a regra**  
Garantir a integridade da base de dados.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**  
Quando feito com a utilização de tabelas temporárias ou variáveis do tipo tabela. Outra exceção a ser levada em consideração é quando for feito um JOIN de uma tabela do banco de dados com uma tabela temporária ou variável do tipo tabela.

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DELETE FROM TITLES  OU  UPDATE TITLES SET TITLE='THE GOURMET MICROWAVE' |  | **Observância da regra**  DELETE FROM TITLES  WHERE PUB\_ID = 1389  OU  UPDATE TITLES   SET TTITLE='THE GOURMET MICROWAVE'  WHERE PUB\_ID = 1389  OU  DELETE FROM #TITLES  OU  UPDATE TITLES   SET TTITLE='THE GOURMET MICROWAVE'  OU  DELETE A       FROM TITLES A           INNER JOIN #TITLE\_AUTHORS B                 A.AUT\_ID = B.AUT\_ID |

Uso de DISTINCT

**Descrição**  
O uso do DISTINCT causa degradação de performance.

**Motivação para a regra**

O uso do Distinct causa degradação de performance e somente deve ser usado quando extremamente necessário. Para esta instrução ser executada, é necessário o consumo de uma grande quantidade de recursos do servidor, além disto o Distinct realiza duas operações de uma só vez:

1.Ordena todos os registros resultantes

2.Elimina todos os registros duplicados

Uma questão a ser levada em consideração é quando a ordenação dos dados retornados por um SELECT com DISTINCT é a mesma já existentes em um índice do tipo CLUSTERED da(s) tabela(s) usada(s). Isto torna o uso do Distinct desnecessário e ele pode ser substituído pelo Group By, que gera o retorno de dados não duplicados com uma quantidade de sobrecarga menor. Obs: o Group BY não realiza operação de ordenação de dados.

Uma alternativa que pode ser feita também utilizar o Group By para eliminar a duplicidade de dados e quando os dados são retornados a aplicação são ordenados na camada de aplicação.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  Select DISTINCT Author.Author\_ID, Author.Name  From AUTHOR        INNER JOIN PUBS            ON PUBS.AUTHOR\_ID = AUTHOR.AUTHOR\_ID  WHERE PUBS.DTPUB >= ‘19800101’ |  | **Observância da regra**  Select Author.Author\_ID, Author.Name  From AUTHOR        INNER JOIN PUBS            ON PUBS.AUTHOR\_ID = AUTHOR.AUTHOR\_ID  WHERE PUBS.DTPUB >= ‘19800101’  GROUP BY Author.Author\_ID, Author.Name |

Uso da cláusula Having

**Descrição**  
O uso da cláusula Having causa degradação de performance.

**Motivação para a regra**

O uso do Having para restringir o resultado retornado pelo GROUP BY causa degradação de performance e somente deve ser usado quando extremamente necessário. Ele é executado logo após a execução do GROUP BY e trabalha eliminando os dados retornados.

Ao invés de ser usado a clausula Having em conjunto com o GROUP BY, é recomendada a utilização da clausula WHERE mais restritiva como substituto.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  Select Author.Author\_ID, Author.Name, Author.DtPub  From AUTHOR        INNER JOIN PUBS            ON PUBS.AUTHOR\_ID = AUTHOR.AUTHOR\_ID  GROUP BY Author.Author\_ID, Author.Name, Author.DtPub  HAVING Author.DtPub >= ‘19800101’ |  | **Observância da regra**  Select Author.Author\_ID, Author.Name, Author.DtPub  From AUTHOR        INNER JOIN PUBS            ON PUBS.AUTHOR\_ID = AUTHOR.AUTHOR\_ID  WHERE Author.DtPub >= ‘19800101’  GROUP BY Author.Author\_ID, Author.Name, Author.DtPub |

Chamada de Procedure sem Owner

**Chamada de Procedure sem Owner**

**Descrição**  
Quando forem feitas chamadas de Stored Procedure é recomendado incluir o OWNER da procedure.

**Motivação para a regra**  
Há ganho de performance quando são feitas chamadas de Stored Procedures informando o OWNER. Isto porque, o SQL SERVER não precisa realizar um pesquisa no catalogo de objetos para associar a qual owner pertence determinado objeto.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  Exec STP\_CalculaEstornoCompra 123456 |  | **Observância da regra**  Exec dbo.STP\_CalculaEstornoCompra 123456 |

**Uso de HINTS**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não é recomendável usar HINTS, mas caso se faça necessário, HINTS sejam identificados com a cláusula WITH.

**Motivação para a regra**  
Melhorar a legibilidade do código.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**  
O SQL Server 2005 não aceita a utilização de HINTS sem a cláusula WITH.

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT AU\_ID FROM DBO.AUTHORS (UPDLOCK, PAGLOCK) |  | **Observância da regra**  SELECT AU\_ID FROM DBO.AUTHORS WITH (UPDLOCK, PAGLOCK) |

**Uso de cursores**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Se houver necessidade de criar cursores, verificar com o AD e DBA se a modelagem atende a real necessidade de negócio.

**Motivação para a regra**

Cursor força a engine do banco de dados a gerenciar locks e tratar geração de resultados, gerando consumo de banda de rede, consumo de memória, espaço em disco e refresh de tela. Se mesmo assim, a utilização de cursor se fizer necessário, siga as seguintes regras:

•Procure usar apenas cursores readonly;

•Procure usar Forward Only Cursor;

•Não esquecer de usar CLOSE e DEALLOCATE após usar o cursor;

•Usar o mínimo de colunas possíveis nos cursores.

Uma alternativa que pode ser usada no lugar de cursores é a utilização de uma tabela temporária ou variável do tipo tabela para armazenar o resultset gerado pela query. Cada linha desta tabela temporária deve ser processada por um laço While.

Para versões do SQL SERVER (2005, 2008 e 2008 R2), o uso de cursor não será permitido.

Nestas versões do SQL SERVER, há um novo tipo de join chamado CROSS APPLY que pode ser usado em algumas situações para substituir o uso do Cursor. Neste caso, o CROSS APPLY faz uma chamada de função para cada linha resultante da execução de uma instrução (SELECT, UPDATE ou DELETE).

Mais uma outra alternativa que pode ser usada, caso o recurso CROSS APPLY não consiga atender é usar CLR Procedures ou CLR Functions. O desenvolvimento de CLR Procedures e CLR Functions é feito nas linguagens VB.NET ou C#.NET, nelas não há problema em se utilizar cursores nativos da plataforma .NET. Obs: Apesar deste recurso estar disponível, há um problema ser utilizado que é o mapeamento das DLLs chamadas pelo SQL SERVER. Este recurso precisa estar muito bem documentado quando a localização das DLLs, pois caso haja necessidade de mudança de servidor, estas DLLs precisam ser movidas para a nova máquina.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**  
N/A

**Impacto**  
Blocker

**Falso positivo**  
N/A

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE @AU\_ID VARCHAR(11),   @MESSAGE VARCHAR(80),   @TITLE\_ID INT  DECLARE TITLES\_CURSOR CURSOR   FOR SELECT T.TITLE\_ID          FROM TITLEAUTHOR TA,        INNER JOIN TITLES T       ON TA.TITLE\_ID = T.TITLE\_ID       AND TA.AU\_ID = @AU\_ID  Declare @Price MONEY,          @UpdDate DateTime  OPEN TITLES\_CURSOR FETCH NEXT FROM TITLES\_CURSOR INTO @TITLE\_ID  WHILE @@FETCH\_STATUS = 0  BEGIN      SELECT @Price = Price,             @UpdDate = UpdDate           FROM TITLEPRICE      WHERE TITLE\_ID = @TITLE\_ID      IF @UpdDate > ‘20100101’         SET @Price = @Price \* 1.05      ELSE         SET @Price = @Price \* 1.15      SELECT @TITLE\_ID, @Price, @UpdDate   FETCH NEXT FROM TITLES\_CURSOR INTO @TITLE  END  CLOSE TITLES\_CURSOR  DEALLOCATE TITLES\_CURSOR |  | **Observância da regra**  DECLARE @AU\_ID VARCHAR(11),   @MESSAGE VARCHAR(80),   @TITLE\_ID INT  DECLARE TITLES\_CURSOR CURSOR   FAST\_FORWARD FOR SELECT T.TITLE          FROM TITLEAUTHOR TA,        INNER JOIN TITLES T       ON TA.TITLE\_ID = T.TITLE\_ID       AND TA.AU\_ID = @AU\_ID  Declare @Price MONEY,          @UpdDate DateTime  OPEN TITLES\_CURSOR FETCH NEXT FROM TITLES\_CURSOR INTO @TITLE\_ID  WHILE @@FETCH\_STATUS = 0  BEGIN      SELECT @Price = Price,             @UpdDate = UpdDate           FROM TITLEPRICE      WHERE TITLE\_ID = @TITLE\_ID        IF @UpdDate > ‘20100101’         SET @Price = @Price \* 1.05      ELSE         SET @Price = @Price \* 1.15      SELECT @TITLE\_ID, @Price, @UpdDate     FETCH NEXT FROM TITLES\_CURSOR INTO @TITLE\_ID  END  CLOSE TITLES\_CURSOR  DEALLOCATE TITLES\_CURSOR  OU  DECLARE @AU\_ID VARCHAR(11),          @MESSAGE VARCHAR(80),          @TITLE\_ID INT,          @Ct\_ID INT,          @Ct\_Tot INT  Declare @Price MONEY,          @UpdDate DateTime  CREATE TABLE #Proc\_TB  (ID INT IDENTITY(1,1),   TITLE\_ID INT)  INSERT INTO #Proc\_TB (Title\_ID)  SELECT T.TITLE\_ID      FROM TITLEAUTHOR TA,           INNER JOIN TITLES T           ON TA.TITLE\_ID = T.TITLE\_ID           AND TA.AU\_ID = @AU\_ID  SET @Ct\_Tot = @@ROWCOUNT()  SET @Ct\_ID = 1  WHILE (@Ct\_ID <= @CT\_Tot)  BEGIN  SELECT @Price = Price,         @UpdDate = UpdDate  FROM #Proc\_tb Proc\_TB       INNER JOIN TitlePrice TitlePrice           ON Proc\_TB.Title\_ID = TitlePrice.Title\_ID  WHERE ID = @CT\_Id  IF @UpdDate > ‘20100101’             SET @Price = @Price \* 1.05          ELSE             SET @Price = @Price \* 1.15          SELECT @TITLE\_ID, @Price, @UpdDate          SET @Ct\_ID = @Ct\_Id + 1  END  Para SQL SERVER 2005, 2008 e 2008 R2:  CREATE FUNCTION dbo.fn\_GetTitlePrice (@Title\_ID AS INT  RETURNS TABLE  AS  RETURN     (        Declare @Price MONEY,                 @UpdDate DateTime        SELECT @Price = Price              ,@UpdDate = UpdDate        FROM TitlePrice        WHERE Title\_ID = @Title\_ID  IF @UpdDate > ‘20100101’             SET @Price = @Price \* 1.05         ELSE             SET @Price = @Price \* 1.15         SELECT @TITLE\_ID as Title\_ID,                @Price as Price,                @UpdDate as UpdDate           )  GO  GRANT SELECT ON PUBLIC TO dbo.fn\_GetTitlePrice  GO  SELECT T.TITLE\_ID, FUN.PRICE, FUN.UpdDate      FROM TITLEAUTHOR TA,          INNER JOIN TITLES T             ON TA.TITLE\_ID = T.TITLE\_ID             AND TA.AU\_ID = @AU\_ID          CROSS APLLY dbo.fn\_GetTitlePrice(T.TITLE\_ID) AS FUN |

**Uso de Linked Server em query**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não é recomendado a utilização de Linked Server em queries.

**Motivação para a regra**

A utilização de Linked Server em queries pode impactar na performance da query com o linked Server, além de causar impactos em demais rotinas em execução.

Para substituir o uso de Linked Server deve ser consultado o DBA para desenhar uma solução que possa atender às necessidades de negócio.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN SERVER1.PUBS.DBO.TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID       INNER JOIN @PROC\_TB PROC\_TB           ON TITLE.TITLE\_ID = PROC\_TB.TITLE\_ID  WHERE TITLE.DTPUB >= ‘20100101’ |  | **Observância da regra**  Consulte DBA para apontar a melhor solução. |

**Uso de funções aritiméticas na cláusula WHERE**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não é recomendado usar funções aritméticas em campos na clausula WHERE para serem usados como critérios de pesquisa

**Motivação para a regra**  
A utilização deste recurso faz com que não seja usado nenhum índice criado na(s) tabela(s) envolvida(s) na query.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID  WHERE (TITLE.PRICE \* 0.10) <= TITLE.DESCOUNTMAX |  | **Observância da regra**  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID  WHERE TITLE.DESCOUNTMAX > (TITLE.PRICE \* 0.10) |

**Existência de non-sargs (!=, , !, !, IS NULL, LIKE)**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Deve-se evitar a utilização de non-SARGs, pois, estes suprimem a utilização de índices existentes nas colunas, e normalmente precisam varrer toda a tabela para satisfazer a condição.

**Motivação para a regra**  
Índices são utilizados com operadores de igualdade. Operadores de diferença suprimem a utilização de índices causando full table scan.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TOP 10        TITLE\_ID,        TITLE,        TYPE,        PUB\_ID,        PRICE,        ADVANCE,        ROYALTY,        YTD\_SALES,        NOTES,        PUBDATE  FROM TITLES  WHERE TITLE != 'NET ETIQUETTE'  AND PUBDATE <> '2000-08-06'  AND PRICE!>10 |  | **Observância da regra**  SELECT TOP 10        TITLE\_ID,        TITLE,        TYPE,        PUB\_ID,        PRICE,        ADVANCE,        ROYALTY,        YTD\_SALES,        NOTES,        PUBDATE  FROM TITLES  WHERE TITLE in ('PRIVATE ETIQUETTE', 'PUBLIC ETIQUETTE')  AND (PUBDATE < '2000-08-06' or PUBDATE > '2000-08-06')  AND PRICE < 10 |

**Where com função**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não é recomendável utilizar funções em colunas na cláusula WHERE, ou seja, as funções devem estar sempre sobre os valores. Da mesma forma, não é recomendado utilizar operadores aritméticos sobre as colunas na cláusula WHERE.

**Motivação para a regra**  
O uso de função na cláusula WHERE (convert, isnull, round, ltrim, rtrim), impede o engine da base de dados a utilizar índices de maneira eficiente.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TOP 10       TITLE\_ID,       TITLE,       TYPE,       PUB\_ID,       PRICE,       ADVANCE,       ROYALTY,       YTD\_SALES,       NOTES,       PUBDATE  FROM TITLES  WHERE LOWER(TITLE) ='NET ETIQUETTE' |  | **Observância da regra**  SELECT TOP 10       TITLE\_ID,       TITLE,       TYPE,       PUB\_ID,       PRICE,       ADVANCE,       ROYALTY,       YTD\_SALES,       NOTES,       PUBDATE  FROM TITLES  WHERE TITLE ='NET ETIQUETTE' |

**Uso de Parâmetros com lista de valores de Filtro em Stored Procedure**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não é recomendado usar parâmetros VARCHAR(Max) contendo uma lista de valores a serem usados como critério de seleção em queries.

**Motivação para a regra**

A utilização deste recurso faz com que sejam consumidos muitos recursos para a realização de parse desta lista de valores para serem utilizada como critério de seleção de valores.

Para solucionar este problema, pode ser usado o parâmetro como tipo TABLE e imediatamente usado em um JOIN para filtrar os dados.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROCEDURE stp\_GetTitle  (@TITLE\_ID VARCHAR(MAX))  AS  BEGIN      DECLARE @PROC\_TB TABLE(ID INT)      INSERT INTO @PROC\_TB      SELECT DBO.fn\_ParseTitle\_ID(@TITLE\_ID)  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID       INNER JOIN @PROC\_TB PROC\_TB           ON TITLE.TITLE\_ID = PROC\_TB.TITLE\_ID  WHERE TITLE.DTPUB >= ‘20100101’  END  GO |  | **Observância da regra**  CREATE PROCEDURE stp\_GetTitle  (@PROC\_TB TABLE(ID INT))  AS  BEGIN    SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID       INNER JOIN @PROC\_TB PROC\_TB           ON TITLE.TITLE\_ID = PROC\_TB.TITLE\_ID  WHERE TITLE.DTPUB >= ‘20100101’  END  GO |

**Uso de SQL dinâmico**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável que todos os statements SQL sejam devidamente declarados dentro de SQL batchs.

**Motivação para a regra**

A utilização de SQL dinâmico faz com que o otimizador do banco de dados não reaproveite o plano de execução.

Outro fator que inviabiliza a utilização desse recurso do banco de dados é a necessidade de dar grant a todos os objetos da tabela ao SQL batch.

A utilização de SQL dinâmico, gera sérios problemas de segurança, pois uma stored procedure pode receber uma query mal intencionada via parâmetro. Caso exista a real necessidade prefira utilizar a store procedure sp\_executesql .

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE @STR CHAR(100),   @ID INT  SET @ID=35  SET @STR = 'SELECT TITLE FROM TITLES WHERE TITLE\_ID = '+@ID  EXEC (@STR) |  | **Observância da regra**  EXECUTE SP\_EXECUTESQL 'SELECT TITLE FROM TITLES WHERE TITLE\_ID = @ID', '@ID INT', @ID = 35 |

**Uso de tabelas temporárias**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

Procure usar soluções como variáveis do tipo TABLE, o qual provê solução para pequenas tabelas em memória. A utilização dessa solução é melhor pelos seguintes motivos:

•Possui escopo definido, podendo assim ser usada como tabela regular;

•Variáveis do tipo TABLE, são automaticamente zeradas ao final da execução do bloco;

•Menor utilização de recursos de log.

**Motivação para a regra**

O uso de tabelas temporárias gera um maior consumo de recursos do servidor em termos de I/O principalmente sobre o bando de dados TEMPDB. A utilização de tabelas temporárias dentro de Stored Procedure, por exemplo, gera recompilação do plano de execução, por este motivo é aconselhável, sempre que possível, a utilização de variáveis do tipo tabela.

Quando for utilizada uma variável do tipo tabela, é aconselhável a utilização de uma PK na declaração da variável.

Caso haja necessidade de utilização de tabelas temporárias, de sempre preferência para utilização as temporárias locais definidas por (#). As tabelas temporárias definidas por (##), conhecidas também como tabelas temporárias globais, não são recomendadas, pois o escopo destas tabelas é de visão global para todo o servidor.

Para as versões do SQL SERVER 2005, 2008 e R2, há um recurso que pode ser usado para substituir o uso de tabelas temporárias. O CTE (common table expressions) permite a criação de uma estrutura de dados baseada em uma query e esta estrutura de dados tem funcionamento igual à de uma tabela, com exceção de que os dados são somente leitura.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_SE\_AUTHORS  (  @AU\_ID ID = NULL  )  AS  SELECT AU\_ID, AU\_LNAME, AU\_FNAME, PHONE, ADDRESS, CITY, STATE, ZIP, CONTRACT  INTO #TMP  FROM AUTHORS  WHERE (AU\_ID = @AU\_ID OR @AU\_ID IS NULL)  SELECT  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  FROM #TMP  ORDER BY AU\_LNAME  GO  AU\_FNAME VARCHAR(20), PHONE |  | **Observância da regra**  CREATE PROC SP\_SE\_AUTHORS  (  @AU\_ID ID = NULL  )  AS  DECLARE @TMP TABLE         (AU\_ID INT PRIMARY KEY,          AU\_LNAME VARCHAR(40),          AU\_FNAME VARCHAR(20),          PHONE CHAR(12),          ADDRESS VARCHAR(40) ,          CITY VARCHAR(20),          STATE CHAR(2),          ZIP CHAR(5),          CONTRACT BIT)  INSERT @TMP  SELECT  AU\_ID, AU\_LNAME, AU\_FNAME, PHONE, ADDRESS, CITY, STATE, ZIP, CONTRACT  FROM AUTHORS  WHERE (AU\_ID = @AU\_ID OR @AU\_ID IS NULL)  SELECT  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  FROM @TMP  ORDER BY AU\_LNAME  GO  Para SQL SERVER 2005, 2008 e R2:  CREATE PROC SP\_SE\_AUTHORS  (  @AU\_ID ID = NULL  )  AS  WITH CTE\_Author AS (  INSERT @TMP  SELECT  AU\_ID, AU\_LNAME, AU\_FNAME, PHONE, ADDRESS, CITY, STATE, ZIP, CONTRACT  FROM AUTHORS  WHERE (AU\_ID = @AU\_ID OR @AU\_ID IS NULL))  SELECT  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  FROM @TMP  ORDER BY AU\_LNAME  GO |

**Formatação de Valores na Cláusula SELECT**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
A utilização de formatação de campos na instrução SELECT causa uma demora no retorno de todas as informações, pois é necessário que mais recursos de processamento sejam alocados.

**Motivação para a regra**

Não é recomendável a utilização de formatação de campos em instrução SELECT, isto porquê a instrução pode levar mais tempo e consumir mais recursos do servidor.

Uma solução para resolver este tipo de problema é a utilização do recurso de formatação de campos do lado da aplicação (client-side).

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT CAST(TITLE.TITLE\_ID AS VARCHAR(05)) + ‘-’ +          TITLE.TITLE as TITLE,          CONVERT(VARCHAR(06), TITLE.DTPUB, 110) AS DATE,          CAST(TITLEAUTHOR.AUT\_ID AS VARCHAR(05)) + ‘-’ + TITLEAUTHOR.AUT\_NAME AS AUTHOR    FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID  WHERE TITLE.DTPUB > ‘20100520’ |  | **Observância da regra**  SELECT TITLE.TITLE\_ID,         TITLE.TITLE,         TITLE.DTPUB,         TITLEAUTHOR.AUT\_ID,         TITLEAUTHOR.AUT\_NAME  FROM TITLE       INNER JOIN TITLEAUTHOR           ON TITLE.AUT\_ID = TITLEAUHTOR.AUT\_ID  WHERE TITLE.DTPUB > ‘20100520’ |

**Falta de Tratamento de Erro em Instruções SQL que envolvam somente comandos do tipo SELECT.[close](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)**

**Descrição**

Após a utilização de comandos SELECT, deve-se verificar se a transação ocorreu com sucesso através do comando @@ERROR.

Quando uma única instrução envolver somente tabela(s) temporária(s) não é preciso tratar erro, porém se tivemos uma ou mais tabela(s) física(s) envolvida(s) na instrução SQL, este quesito deve ser tratado.

**Motivação para a regra**  
Garantir a integridade da informação, testando a correta execução das transações.

**Critério para o apontamento**  
Todas as ocorrências encontradas.

**Exceção**  
Instruções de SELECT que envolvam somente tabelas temporárias.

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  Select     <Coluna 1>,                  <Coluna 2>,                  <Coluna 3>  From      <Tabela>  Where <Condição de Restrição> |  | **Observância da regra**  Select     <Coluna 1>,                  <Coluna 2>,                  <Coluna 3>  From      <Tabela>  Where <Condição de Restrição>  SELECT   @ERROR               = @@ERROR,  @ROWCOUNT     = @@ROWCOUNT  --  IF             @ERROR               !=            0  BEGIN  SELECT @ERROR  RETURN @ERROR  END  OU  Select     <Coluna 1>,                  <Coluna 2>,                  <Coluna 3>  From      <Tabela>  Where <Condição de Restrição>  IF             @@ERROR           !=            0  BEGIN  SELECT @@ERROR  RETURN @@ERROR  END |

**Produto cartesiano**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Havendo joins em queries, deve haver relacionamento entre as colunas identificadoras das tabelas.

**Motivação para a regra**  
A má codificação de joins, causa uso desnecessário de recurso de máquina, além de comprometer o resultado final da query.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT AUTHORS.AU\_ID ,   AUTHORS.AU\_LNAME ,   AUTHORS.AU\_FNAME ,   AUTHORS.PHONE ,   AUTHORS.ADDRESS ,   AUTHORS.CITY ,   AUTHORS.STATE ,   AUTHORS.ZIP ,   AUTHORS.CONTRACT ,   TITLEAUTHOR.TITLE\_ID ,   TITLEAUTHOR.AU\_ORD ,   TITLEAUTHOR.ROYALTYPER  FROM AUTHORS, TITLEAUTHOR |  | **Observância da regra**  SELECT AUTHORS.AU\_ID ,   AUTHORS.AU\_LNAME ,   AUTHORS.AU\_FNAME ,   AUTHORS.PHONE ,   AUTHORS.ADDRESS ,   AUTHORS.CITY ,   AUTHORS.STATE ,   AUTHORS.ZIP ,   AUTHORS.CONTRACT ,   TITLEAUTHOR.TITLE\_ID ,   TITLEAUTHOR.AU\_ORD ,   TITLEAUTHOR.ROYALTYPER  FROM AUTHORS   INNER JOIN TITLEAUTHOR    ON AUTHORS.AU\_ID = TITLEAUTHOR.AU\_ID |

**Cursor for UPDATE sem as colunas explícitas**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Se houver necessidade de criar cursor com a cláusula FOR UPDATE, é recomendável explicitar as colunas, pois o SQL otimiza a operação. É recomendado não se utilizar cursores para este tipo de processamento, sempre que possível substitua o uso dos cursores por tabelas temporárias ou variáveis do tipo tabela para armazenar o resultset gerado pela query. Cada linha desta tabela temporária deve ser processada por um laço While.

**Motivação para a regra**  
Garantir a integridade da informação e melhorar a performance da operação.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE EMPLOYEE\_CURSOR  CURSOR FOR SELECT FIRSTNAME    FROM NORTHWIND.DBO.EMPLOYEES  FOR UPDATE  OPEN EMPLOYEE\_CURSOR FETCH NEXT FROM EMPLOYEE\_CURSOR  WHILE @@FETCH\_STATUS = 0  BEGIN   FETCH NEXT FROM EMPLOYEE\_CURSOR  END  CLOSE EMPLOYEE\_CURSOR  DEALLOCATE EMPLOYEE\_CURSOR |  | **Observância da regra**  DECLARE @VFIRSTNAME VARCHAR(20)  DECLARE EMPLOYEE\_CURSOR   CURSOR FOR SELECT FIRSTNAME     FROM NORTHWIND.DBO.EMPLOYEES   FOR UPDATE OF FIRSTNAME  OPEN EMPLOYEE\_CURSOR  FETCH NEXT FROM EMPLOYEE\_CURSOR INTO @VFIRSTNAME  WHILE @@FETCH\_STATUS = 0  BEGIN   PRINT @VFIRSTNAME   FETCH NEXT FROM EMPLOYEE\_CURSOR INTO @VFIRSTNAME  END  CLOSE EMPLOYEE\_CURSOR  DEALLOCATE EMPLOYEE\_CURSOR |

**Uso de (IN) EXISTS ao invés de (EXISTS) IN**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável usar EXISTS ao invés de IN, pois o engine trabalha de forma mais eficiente. O processamento é interrompido quando um registro é encontrado, o IN continua o processamento até o final, consumindo mais recurso.

**Motivação para a regra**  
Diminuir o tempo de processamento, assim como também, reduzir o consumo de recursos do servidor.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  USE PUBS  GO  SELECT DISTINCT PUB\_NAME  FROM PUBLISHERS  WHERE PUB\_ID IN             (SELECT PUB\_ID                  FROM TITLES              WHERE TYPE = 'BUSINESS')  GO |  | **Observância da regra**  USE PUBS  GO  SELECT DISTINCT PUB\_NAME  FROM PUBLISHERS  WHERE EXISTS (SELECT PUB\_ID                   FROM TITLES                WHERE PUB\_ID = PUBLISHERS.PUB\_ID                AND TYPE = 'BUSINESS')  GO |

**Transação aberta**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Cada transação aberta deve possuir COMMIT TRAN e ROLLBACK TRAN explícito.

**Motivação para a regra**  
Evitar que transações fiquem abertas, bloqueando acesso aos dados.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  USE PUBS  GO  CREATE PROC SP\_IN\_AUTHORS   ( @AU\_LNAME VARCHAR(40) = NULL ,   @AU\_FNAME VARCHAR(20) = NULL ,   @PHONE CHAR(12) ,   @ADDRESS VARCHAR(40) = NULL ,   @CITY VARCHAR(20) = NULL ,   @STATE CHAR(2) = NULL ,   @ZIP CHAR(5) = NULL ,   @CONTRACT BIT )  AS  DECLARE @ERROR INT,   @ROWCOUNT INT  SET NOCOUNT ON  BEGIN TRAN   INSERT INTO AUTHORS ( AU\_ID ,      AU\_LNAME ,      AU\_FNAME ,      PHONE ,      ADDRESS ,      CITY ,      STATE ,      ZIP ,      CONTRACT )   VALUES ( @AU\_ID ,    @AU\_LNAME ,    @AU\_FNAME ,    @PHONE ,    @ADDRESS ,    @CITY ,    @STATE ,    @ZIP ,    @CONTRACT )  SELECT @ERROR = @@ERROR,   @ROWCOUNT = @@ROWCOUNT  IF @ERROR > 0   BEGIN    SELECT @ERROR    ROLLBACK TRAN    RETURN @ERROR   END  IF @ROWCOUNT = 0   BEGIN    RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '    ROLLBACK TRAN    RETURN 30002   END  SELECT 0  RETURN 0  GO  GRANT EXECUTE ON SP\_IN\_AUTHORS TO APPPUBSAUT  GO |  | **Observância da regra**  USE PUBS  GO  CREATE PROC SP\_IN\_AUTHORS   ( @AU\_LNAME VARCHAR(40) = NULL ,   @AU\_FNAME VARCHAR(20) = NULL ,   @PHONE CHAR(12) ,   @ADDRESS VARCHAR(40) = NULL ,   @CITY VARCHAR(20) = NULL ,   @STATE CHAR(2) = NULL ,   @ZIP CHAR(5) = NULL ,   @CONTRACT BIT )  AS  DECLARE @ERROR INT,   @ROWCOUNT INT  SET NOCOUNT ON  BEGIN TRAN   INSERT INTO AUTHORS    ( AU\_ID ,    AU\_LNAME ,    AU\_FNAME ,    PHONE ,    ADDRESS ,    CITY ,    STATE ,    ZIP ,    CONTRACT )   VALUES ( @AU\_ID ,    @AU\_LNAME ,    @AU\_FNAME ,    @PHONE ,    @ADDRESS ,    @CITY ,    @STATE ,    @ZIP ,    @CONTRACT )  SELECT @ERROR = @@ERROR,   @ROWCOUNT = @@ROWCOUNT  IF @ERROR > 0   BEGIN    SELECT @ERROR    ROLLBACK TRAN    RETURN @ERROR   END  IF @ROWCOUNT = 0   BEGIN RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '    ROLLBACK TRAN    RETURN 30002   END  COMMIT TRAN  SELECT 0  RETURN 0  GO  GRANT EXECUTE ON SP\_IN\_AUTHORS TO APPPUBSAUT  GO |

**Uso de DISTINCT com UNION**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Quando se estiver utilizando UNION, não se deve utilizar DISTINCT, pois, ambos eliminam registros duplicados.

**Motivação para a regra**  
O operador UNION e a cláusula DISTINCT realizam ordenação do resultset a qual estão submetidos. A utilização dos dois ao mesmo tempo é desnecessária e consume mais recursos do que o necessário.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT DISTINCT COL1,         COL2,         COL3  FROM TAB1  UNION  SELECT COL1,         COL2,         COL3  FROM TAB2 |  | **Observância da regra**  SELECT COL1,         COL2,         COL3  FROM TAB1  UNION  SELECT COL1,         COL2,         COL3  FROM TAB2 |

**Uso de INLINE/DERIVED TABLE view**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

Caso se faça necessário utilizar uma subquery na cláusula FROM, deve-se estudar as seguintes possibilidades:

•Criação de view com a subquery;

•Utilização de join;

•Revisão da modelagem da base de dados (normalização).

**Motivação para a regra**

Melhorar a legibilidade e diminuir complexidade e possibilidade de erros em querys.

- Para as versões de SQL SERVER 2005, 2008 e R2

Como uma forma de melhora de performance pode-se usar um recurso chamado Common Table Expressions. O CTE além melhorar a performance, também torna o código mais simples de ser entendido.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT A.AU\_LNAME,                  A.AU\_FNAME,                  B.PHONE  FROM AUTHORS A,             (SELECT AU\_ID,                             PHONE              FROM AUTHORS) B  WHERE A.AU\_ID = B.AU\_ID  Para SQL SERVER 2005, 2008 e R2:  SELECT A.AU\_LNAME,                  A.AU\_FNAME,                  B.PHONE  FROM AUTHORS A,             (SELECT AU\_ID,                             PHONE              FROM AUTHORS                    INNER JOIN PUBS                       ON AUTHORS.AU\_ID = PUBS.AU\_ID where PUBS.DtPub >= ‘20101010’) B  WHERE A.AU\_ID = B.AU\_ID |  | **Observância da regra**  SELECT       A.AU\_LNAME,       A.AU\_FNAME,       B.PHONE  FROM AUTHORS A       INNER JOIN AUTHORS B           ON A.AU\_ID = B.AU\_ID  Para SQL SERVER 2005, 2008 e R2:  WITH Auhtors\_Pubs(AU\_ID,PHONE)  As(SELECT AU\_ID,                             PHONE              FROM AUTHORS                    INNER JOIN PUBS                       ON AUTHORS.AU\_ID = PUBS.AU\_ID where PUBS.DtPub >= ‘20101010’)  SELECT A.AU\_LNAME,                  A.AU\_FNAME,                  B.PHONE  FROM AUTHORS A         INNER JOIN Auhtors\_Pubs B              ON A.AU\_ID = B.AU\_ID |

**Uso de SELECT ... INTO para criação de Tabelas**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
O uso de SELECT... INTO para criar tabelas a partir de um RESULTSET não é recomendado.

**Motivação para a regra**

O uso deste tipo de recurso acaba causando bloqueios de alguns dos recursos do SQL SERVER fazendo com que o mesmo fique ocupado até que a instrução seja completamente executada.

Uma alternativa para solucionar este problema seria separa o processo de criação e alimentação da tabela em duas etapas. Se possível for, deixar a tabela já criada antes do inicio do processo, ou seja, se é um processo que for muito usado, considerar a criação permanente desta tabela.

Outra alternativa, se o volume de dados retornados for pequeno, considere o uso de variável do tipo tabela.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT Author.Author\_Id, Author.firstname, Author.lastname  INTO Athor\_Orlando  FROM Author  WHERE Author.city     = 'Orlando' |  | **Observância da regra**  CREATE TABLE Author\_Orlando  (Author\_id   INT,  FirstName    VARCHAR(50),  LastName     VARCHAR(50))  INSERT INTO Author\_Orlando  (Author\_ID, FirstName, LastName)  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE city     = 'Orlando'  Ou  DECLARE @Tbl\_Orlando TABLE(Author\_Id INT, firstname VARCHAR(50), lastname VARCHAR(50))  INSERT INTO @Tbl\_Orlando  (Author\_Id, firstname, lastname)  SELECT Author\_Id, firstname, lastname  FROM Author  WHERE city     = 'Orlando'  SELECT Author\_Id, firstname, lastname  FROM @Tbl\_Orlando |

**Uso de IF exists com select count(\*)**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
O uso de IF EXISTS com um SELECT COUNT(\*) gera uma grande quantidade de sobrecarga e não é recomendado.

**Motivação para a regra**

O uso deste recurso do SQL SERVER causa uma grande quantidade de processamento, pois primeiro devem ser contadas todas as linhas do SELECT COUNT(\*), para depois ser verificada existência de dados na query pelo IF EXISTS(). Se o resultado de uma query retornar milhões de registros a serem contados, isto causará um demora grande para a instrução ser concluída.

Como o intuito é apenas verificar se existem dados ou não, pode ser usada a seguinte instrução no lugar IF EXISTS(SELECT TOP 1 ....).

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROCEDURE STP\_RETORNAMENSAGEM  @DtPub  DATETIME  AS  BEGIN     IF EXISTS(SELECT COUNT(\*) FROM PUBS  WHERE   DTPUB >= @DTPUB)     BEGIN         SELECT ‘Há publicações’     END     ELSE     BEGIN         SELECT ‘Não há publicações’     END  END |  | **Observância da regra**  CREATE PROCEDURE STP\_RETORNAMENSAGEM  @DtPub  DATETIME  AS  BEGIN     IF EXISTS(SELECT TOP 1 Pub\_ID FROM PUBS  WHERE   DTPUB >= @DTPUB)     BEGIN         SELECT ‘Há publicações’     END     ELSE     BEGIN         SELECT ‘Não há publicações’     END  END |

**SEGURANÇA**

**Falta de uso de transação**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Sempre que escrever um script que execute os comandos INSERT, UPDATE e DELETE, deve-se usar transações.

**Motivação para a regra**  
A utilização de transações garante integridade dos dados, quando algum problema ocorrer durante uma manipulação de dados. A má utilização de transações, pode gerar contenção desnecessária no banco de dados, por isso, inicie e finalize uma transação o quanto antes possível.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**  
N/A

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_IN\_AUTHORS  (  @AU\_LNAME VARCHAR(40) = NULL  , @AU\_FNAME VARCHAR(20) = NULL  , @PHONE CHAR(12)  , @ADDRESS VARCHAR(40) = NULL  , @CITY VARCHAR(20) = NULL  , @STATE CHAR(2) = NULL  , @ZIP CHAR(5) = NULL  , @CONTRACT BIT  )  AS  INSERT INTO AUTHORS  (  AU\_LNAME , AU\_FNAME , PHONE , ADDRESS , CITY , STATE , ZIP , CONTRACT  )  VALUES  (  @AU\_LNAME , @AU\_FNAME , @PHONE , @ADDRESS , @CITY , @STATE , @ZIP , @CONTRACT  )  Go |  | **Observância da regra**  CREATE PROC SP\_IN\_AUTHORS  (  @AU\_LNAME VARCHAR(40) = NULL  , @AU\_FNAME VARCHAR(20) = NULL  , @PHONE CHAR(12)  , @ADDRESS VARCHAR(40) = NULL  , @CITY VARCHAR(20) = NULL  , @STATE CHAR(2) = NULL  , @ZIP CHAR(5) = NULL  , @CONTRACT BIT  )  AS  DECLARE @ERROR INT,          @ROWCOUNT INT  BEGIN TRAN        INSERT INTO AUTHORS        ( AU\_LNAME ,          AU\_FNAME ,          PHONE ,          ADDRESS ,          CITY ,          STATE ,          ZIP ,          CONTRACT)        VALUES (          @AU\_LNAME ,          @AU\_FNAME ,          @PHONE ,          @ADDRESS ,          @CITY ,          @STATE ,          @ZIP ,          @CONTRACT)        SELECT @ERROR = @@ERROR,               @ROWCOUNT = @@ROWCOUNT        IF @ERROR > 0        BEGIN            SELECT @ERROR            ROLLBACK TRAN            RETURN @ERROR        END        IF @ROWCOUNT = 0        BEGIN            RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '            ROLLBACK TRAN            RETURN 30002        END        COMMIT TRAN        SELECT 0        RETURN 0  GO |

**Uso de Impersonate em Stored Procedures e Functions**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
A utilização do recurso de Impersonate do SQL SERVER 2005, 2008 e R2 é muito útil, mas deve ser usado com cautela.

**Motivação para a regra**  
Uso do Impersonate em STORED PROCEDURE e FUNCTIONS é muito útil, porém ele é um recurso difícil de ser rastreado após a sua compilação com o objeto. Isto posto, ele pode camuflar execuções de processos não autorizados para determinados usuários, portanto, deve ser usado com muita cautela.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Info

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROCEDURE DBO.STP\_SHOW\_SALES\_TOTAL       WITH EXECUTE AS OWNER  AS  BEGIN      SELECT COUNT(\*) AS QTDSALES             SUM(PRICE) AS TOTAL      FROM SALES\_BOOK      WHERE STATUS = ‘N’  END |  | **Observância da regra**  N/A |

**Uso de identificador non-autoidentity**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Quando houver a necessidade de auto-incrementar o identificador de uma tabela, procurar usar o SCOPE\_IDENTITY ou IDENT\_CURRENT na criação da tabela.

**Motivação para a regra**  
Quando há necessidade de se inserir em uma tabela detalhe, e se faz necessário SELECT MAX(...) na tabela master, existe alta probabilidade de ocorrer falhas quando usuários concorrentes inserem na mesma instância.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_IN\_AUTHORS  (  @AU\_LNAME VARCHAR(40) = NULL  , @AU\_FNAME VARCHAR(20) = NULL  , @PHONE CHAR(12)  , @ADDRESS VARCHAR(40) = NULL  , @CITY VARCHAR(20) = NULL  , @STATE CHAR(2) = NULL  , @ZIP CHAR(5) = NULL  , @CONTRACT BIT  )  AS  DECLARE @AU\_ID INT  SET @AU\_ID = SELECT MAX(AU\_ID)+1 FROM  AUTHORS  INSERT INTO AUTHORS  (  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  )  VALUES  (  @AU\_ID  , @AU\_LNAME  , @AU\_FNAME  , @PHONE  , @ADDRESS  , @CITY  , @STATE  , @ZIP  , @CONTRACT  )  GO |  | **Observância da regra**  CREATE PROC SP\_IN\_AUTHORS  (  @AU\_LNAME VARCHAR(40) = NULL  , @AU\_FNAME VARCHAR(20) = NULL  , @PHONE CHAR(12)  , @ADDRESS VARCHAR(40) = NULL  , @CITY VARCHAR(20) = NULL  , @STATE CHAR(2) = NULL  , @ZIP CHAR(5) = NULL  , @CONTRACT BIT  )  AS  DECLARE @AU\_ID INT  SET @AU\_ID = SELECT IDENT\_CURRENT('AUTHORS') + 1  INSERT INTO AUTHORS  (  AU\_ID  , AU\_LNAME  , AU\_FNAME  , PHONE  , ADDRESS  , CITY  , STATE  , ZIP  , CONTRACT  )  VALUES  (  @AU\_ID  , @AU\_LNAME  , @AU\_FNAME  , @PHONE  , @ADDRESS  , @CITY  , @STATE  , @ZIP  , @CONTRACT  )  GO |

**Possibilidade de DEBUG em stored procedures**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não se deve utilizar PRINT em stored procedures, pois, desta forma evidência possibilidade de exibição de dados preliminares.

**Motivação para a regra**  
Detectar procedure complexa, ou testada insuficientemente.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_SE\_AUTHORS  ( @AU\_LNAME VARCHAR(40) = NULL )  AS  PRINT 'PASSEI AQUI'  SELECT AU\_ID       , AU\_LNAME       , AU\_FNAME       , PHONE       , ADDRESS       , CITY       , STATE       , ZIP       , CONTRACT  FROM AUTHORS  WHERE AU\_LNAME = @AU\_LNAME  PRINT 'PASSEI AQUI TB'  GO |  | **Observância da regra**  CREATE PROC SP\_SE\_AUTHORS  ( @AU\_LNAME VARCHAR(40) = NULL )  AS  SELECT AU\_ID       , AU\_LNAME       , AU\_FNAME       , PHONE       , ADDRESS       , CITY       , STATE       , ZIP       , CONTRACT  FROM AUTHORS  WHERE AU\_LNAME = @AU\_LNAME  GO |

**Possibilidade de falha de segurança**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
A conexão com o banco de dados deve ser realizada com um usuário com os devidos privilégios de execução e manutenção dos dados, por isso, não se deve utilizar SETUSER em stored procedures.

**Motivação para a regra**  
Detectar falhas de segurança.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_SE\_AUTHORS  ( @AU\_LNAME VARCHAR(40) = NULL )  AS  SETUSER 'MARY'  SELECT AU\_ID       , AU\_LNAME       , AU\_FNAME       , PHONE       , ADDRESS       , CITY       , STATE       , ZIP       , CONTRACT  FROM AUTHORS  WHERE AU\_LNAME = @AU\_LNAME  GO |  | **Observância da regra**  CREATE PROC SP\_SE\_AUTHORS  ( @AU\_LNAME VARCHAR(40) = NULL )  AS  SELECT AU\_ID       , AU\_LNAME       , AU\_FNAME       , PHONE       , ADDRESS       , CITY       , STATE       , ZIP       , CONTRACT  FROM AUTHORS  WHERE AU\_LNAME = @AU\_LNAME  GO |

**Falta de tratamento de erro**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

Após a utilização de comandos INSERT, UPDATE e DELETE, deve-se verificar se a transação ocorreu com sucesso através do comando @@ERROR. Detectando erro, deve-se utilizar o comando ROLLBACK para voltar à transação no estágio inicial.

Para as versões do SQL SERVER: 2005, 2008 e 2008 R2, há um recurso para tratamento de erros. O bloco de tratamento de erros TRY...CATCH deve ser usado no lugar do @@ERROR, nestas versões do SQL SERVER.

**Motivação para a regra**  
Garantir a integridade dos dados, testando a correta execução das transações.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_IN\_AUTHORS ( @AU\_LNAME VARCHAR(40) = NULL ,         @AU\_FNAME VARCHAR(20) = NULL ,         @PHONE CHAR(12) ,         @ADDRESS VARCHAR(40) = NULL ,         @CITY VARCHAR(20) = NULL ,         @STATE CHAR(2) = NULL ,         @ZIP CHAR(5) = NULL ,         @CONTRACT BIT )  AS  BEGIN TRAN  INSERT INTO AUTHORS   ( AU\_LNAME,     AU\_FNAME,     PHONE,     ADDRESS,     CITY,     STATE,     ZIP,    CONTRACT )  VALUES ( @AU\_LNAME,    @AU\_FNAME,    @PHONE,    @ADDRESS,    @CITY,    @STATE,    @ZIP ,    @CONTRACT )  COMMIT TRAN  SELECT 0  RETURN 0  GO |  | **Observância da regra**  CREATE PROC SP\_IN\_AUTHORS ( @AU\_LNAME VARCHAR(40) = NULL ,         @AU\_FNAME VARCHAR(20) = NULL ,         @PHONE CHAR(12) ,         @ADDRESS VARCHAR(40) = NULL ,         @CITY VARCHAR(20) = NULL ,         @STATE CHAR(2) = NULL ,         @ZIP CHAR(5) = NULL ,         @CONTRACT BIT )  AS  DECLARE @ERROR INT,   @ROWCOUNT INT  BEGIN TRAN   INSERT INTO AUTHORS ( AU\_LNAME,           AU\_FNAME,           PHONE,           ADDRESS,           CITY,           STATE,           ZIP,         CONTRACT )   VALUES ( @AU\_LNAME,     @AU\_FNAME,     @PHONE,     @ADDRESS,     @CITY,     @STATE,     @ZIP ,     @CONTRACT )  SELECT @ERROR = @@ERROR,   @ROWCOUNT = @@ROWCOUNT  IF @ERROR > 0  BEGIN   SELECT @ERROR   ROLLBACK TRAN   RETURN @ERROR  END  IF @ROWCOUNT = 0  BEGIN   RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '   ROLLBACK TRAN   RETURN 30002  END  COMMIT TRAN  SELECT 0  RETURN 0  GO  Para SQL SERVER 2005, 2008 e 2008 R2:  CREATE PROC SP\_IN\_AUTHORS ( @AU\_LNAME VARCHAR(40) = NULL ,         @AU\_FNAME VARCHAR(20) = NULL ,         @PHONE CHAR(12) ,         @ADDRESS VARCHAR(40) = NULL ,         @CITY VARCHAR(20) = NULL ,         @STATE CHAR(2) = NULL ,         @ZIP CHAR(5) = NULL ,         @CONTRACT BIT )  AS  DECLARE @ERROR INT,   @ROWCOUNT INT  BEGIN TRAN        BEGIN TRY             INSERT INTO AUTHORS                ( AU\_LNAME,                  AU\_FNAME,                  PHONE,                  ADDRESS,                  CITY,                  STATE,                  ZIP,                  CONTRACT )             VALUES                ( @AU\_LNAME,                  @AU\_FNAME,                  @PHONE,                  @ADDRESS,                  @CITY,                  @STATE,                  @ZIP ,                  @CONTRACT )               SELECT  @ROWCOUNT = @@ROWCOUNT               IF @ROWCOUNT = 0               BEGIN                   RAISERROR 30002 '\\OPERACAO NAO EFETUADA EM AUTHORS // '                   ROLLBACK TRAN                   RETURN 30002               END               COMMIT TRAN               SELECT 0               RETURN 0        END TRY        BEGIN CATCH           RETURN ERROR\_NUMBER()           ROLLBACK TRAN        END CATCH  GO |

**COMPLEXIDADE**

**Uso de Cláusula OR no WHERE**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
O uso da clausula OR em uma instrução WHERE como um recurso retornar o máximo de valores dentro da mesma query.

**Motivação para a regra**

O uso de OR em clausula Where não é recomendado, pois mesmo que as colunas possuam índice o SQL SERVER acaba realizando SCAN.

Há duas alternativas que podem ser usadas no lugar do OR:

•Quebra a query conforme o número de ORs usados e usar a instrução UNION ALL para unificar todas as queries.

•Se a quantidade de registros for pequena, usar uma variável do tipo tabela e quebrar a query conforme o número de ORs usados, sendo que cada query deve realizar um INSERT nesta variável do tipo tabela. Obs: apesar desta alternativa realizar mais operações E/S em uma quantidade maior do que o uso de UNION ALL, ela pode ser usada por ser mais rápida do que a  forma original usando ORs.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE City     = 'NY'     or city     = 'Orlando'     or division = 'food' |  | **Observância da regra**  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE City     = 'NY'  UNION ALL  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE city     = 'Orlando'  UNION ALL  SELECT Author\_Id, firstname, lastname  FROM Author  WHERE division = 'food'  Ou  DECLARE @Tbl\_OR TABLE(Author\_Id INT, firstname VARCHAR(50), lastname VARCHAR(50))  INSERT INTO @Tbl\_OR  (Author\_Id, firstname, lastname)  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE City     = 'NY'  INSERT INTO @Tbl\_OR  (Author\_Id, firstname, lastname)  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE city     = 'Orlando'  INSERT INTO @Tbl\_OR  (Author\_Id, firstname, lastname)  SELECT Author.Author\_Id, Author.firstname, Author.lastname  FROM Author  WHERE division = 'food'  SELECT Author\_Id, firstname, lastname  FROM @Tbl\_OR |

**Declaração de Variáveis**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Utilizar um único bloco DECLARE para instanciar variáveis em memória.

**Motivação para a regra**  
Quando se utiliza várias chamadas do comando DECLARE para instanciar variáveis, ocasiona em demora de execução, pois o SQL SERVER executa para chamada um processo de alocação de memória e isto resulta em demora para a alocação.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE @AUTHOR\_NAME AS VARCHAR(50)  DECLARE @AUTHOR\_ID AS INT  DECLARE @AUTHOR\_DTCREATION AS DATE  DECLARE @AUTHOR\_ADDRESS AS VARCHAR(50)  DECLARE @AUTHOR\_CITY VARCHAR(50)  DECLARE @AUTHOR\_CITY VARCHAR(50)  DECLARE @PUBS\_ID AS INT  DECLARE @PUBS\_NAME AS VARCHAR(50)  DECLARE @PUBS\_DTPUB AS DATE |  | **Observância da regra**  DECLARE @AUTHOR\_NAME AS VARCHAR(50),  @AUTHOR\_ID AS INT,  @AUTHOR\_DTCREATION AS DATE,  @AUTHOR\_ADDRESS AS VARCHAR(50),  @AUTHOR\_CITY VARCHAR(50),  @AUTHOR\_CITY VARCHAR(50),  @PUBS\_ID AS INT,  @PUBS\_NAME AS VARCHAR(50),  @ECLARE @PUBS\_DTPUB AS DATE |

**Uso de GOTO**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Havendo necessidade de testar algum valor e criar um desvio em um programa, dê preferência ao uso de IF.

**Motivação para a regra**  
O comando GOTO gera desordem no fluxo normal de um script. Esse salto de código, pode gerar problemas e indica fortemente problema de modelagem ou desenho do script.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE @TABLENAME SYSNAME  DECLARE TNAMES\_CURSOR  CURSOR FOR SELECT NAME FROM SYSOBJECTS WHERE XTYPE='U'  OPEN TNAMES\_CURSOR FETCH NEXT FROM TNAMES\_CURSOR INTO @TABLENAME  SET @TABLENAME = N'AUTHORS'  TABLE\_LOOP: IF (@@FETCH\_STATUS <> -2)  BEGIN SELECT @TABLENAME = RTRIM(UPPER(@TABLENAME))   EXEC ('SELECT ''' + @TABLENAME + ''' = COUNT(\*) FROM ' + @TABLENAME )  END  FETCH NEXT FROM TNAMES\_CURSOR INTO @TABLENAME  IF (@@FETCH\_STATUS <> -1)      GOTO TABLE\_LOOP  GO  CLOSE TNAMES\_CURSOR  DEALLOCATE TNAMES\_CURSOR |  | **Observância da regra**  DECLARE @TABLENAME SYSNAME  DECLARE @Ct INT  DECLARE @CtTotal INT  DECLARE @tbl\_Name TABLE  (ID INT IDENTITY(1,1),Name VARCHAR(100))  INSERT INTO #tbl\_Name  SELECT NAME FROM SYSOBJECTS  WHERE XTYPE='U'  SET @CTTOTAL = @@ROWCOUNT()  SET @CT = 1  WHILE(@CT <= @CTTOTAL)  BEGIN   SELECT @TABLENAME = NAME FROM @TBL\_NAME WHERE ID = @CT   SELECT @TABLENAME = RTRIM(UPPER(@TABLENAME))   EXEC ('SELECT ''' + @TABLENAME + ''' = COUNT(\*) FROM ' + @TABLENAME )   SET @CT = @CT + 1  END  GO |

**Query complexa**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

É recomendável que a modelagem da base de dados responda de forma fácil a qualquer “pergunta” que o sistema necessite efetuar. Por isso, considera-se complexa queries com as seguintes ocorrências:

•Mais de 5 funções;

•Mais de 5 joins e

•Mais de 2 subqueries.

**Motivação para a regra**  
Evidenciar problemas de codificação do SQL ou de problemas de modelagem da base de dados. Caso essa regra seja violada, considere a inspeção da estrutura da base dados.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLES.TITLE\_ID,   TITLES.TITLE,   TITLES.TYPE,   TITLES.PUB\_ID,   TITLES.PRICE,   TITLES.ADVANCE,   TITLES.ROYALTY,   TITLES.YTD\_SALES,   TITLES.NOTES,   TITLES.PUBDATE,   PUBLISHERS.PUB\_NAME,   PUBLISHERS.CITY,   UPPER(PUBLISHERS.STATE),   PUBLISHERS.COUNTRY,   SALES.ORD\_NUM,   SALES.ORD\_DATE,   SALES.QTY,   ROYSCHED.ROYALTY  FROM TITLES   INNER JOIN PUBLISHERS    ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID   INNER JOIN SALES    ON SALES.TITLE\_ID = TITLES.TITLE\_ID   INNER JOIN ROYSCHED    ON ROYSCHED.TITLE\_ID = TITLES.TITLE\_ID   INNER JOIN EMPLOYEE    ON EMPLOYEE.PUB\_ID = TITLES.PUB\_ID  WHERE EMPLOYEE.JOB\_ID IN (SELECT MAX(JOB\_ID) FROM JOBS)  AND EMPLOYEE.JOB\_ID IN (SELECT MIN(JOB\_ID) FROM JOBS2)  AND EMPLOYEE.JOB\_ID IN (SELECT LOWER(JOB\_ID) FROM JOBS3)  AND EMPLOYEE.JOB\_ID IN (SELECT UPPER(JOB\_ID) FROM JOBS4)  AND EMPLOYEE.JOB\_ID IN (SELECT MAX(JOB\_ID) FROM JOBS5)  AND EMPLOYEE.JOB\_ID IN (SELECT JOB\_ID FROM JOBS6) |  | **Observância da regra**  DECLARE @MAX\_JOB\_ID1 INT  DECLARE @MIN\_JOB\_ID2 INT  DECLARE @LOWER\_JOB\_ID1 INT  DECLARE @UPPER\_JOB\_ID4 INT  DECLARE @MAX\_JOB\_ID5 INT  SELECT @MAX\_JOB\_ID1 = MAX(JOB\_ID) FROM JOBS  SELECT @MIN\_JOB\_ID2 = MIN(JOB\_ID) FROM JOBS2  SELECT @LOWER\_JOB\_ID1 = LOWER(JOB\_ID) FROM JOBS3  SELECT @UPPER\_JOB\_ID4 = UPPER(JOB\_ID) FROM JOBS4  SELECT @MAX\_JOB\_ID5 = MAX(JOB\_ID) FROM JOBS5  SELECT TITLES.TITLE\_ID,   TITLES.TITLE,   TITLES.TYPE,   TITLES.PUB\_ID,   TITLES.PRICE,   TITLES.ADVANCE,   TITLES.ROYALTY,   TITLES.YTD\_SALES,   TITLES.NOTES,   TITLES.PUBDATE,   PUBLISHERS.PUB\_NAME,   PUBLISHERS.CITY,   UPPER(PUBLISHERS.STATE),   PUBLISHERS.COUNTRY,   SALES.ORD\_NUM,   SALES.ORD\_DATE,   SALES.QTY,   ROYSCHED.ROYALTY  FROM TITLES   INNER JOIN PUBLISHERS    ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID   INNER JOIN SALES    ON SALES.TITLE\_ID = TITLES.TITLE\_ID   INNER JOIN ROYSCHED    ON ROYSCHED.TITLE\_ID = TITLES.TITLE\_ID   INNER JOIN EMPLOYEE    ON EMPLOYEE.PUB\_ID = TITLES.PUB\_ID   INNER JOIN JOBS6    ON JOBS6.JOB\_ID = EMPLOYEE.JOB\_ID  WHERE EMPLOYEE.JOB\_ID IN (@MAX\_JOB\_ID1, @MIN\_JOB\_ID2, @LOWER\_JOB\_ID1, @UPPER\_JOB\_ID4, @MAX\_JOB\_ID5) |

**Mudando Status de Ansi\_Defaults, Ansi\_Nulls , Ansi\_Padding, Ansi\_Warning e Concat\_Null\_Yields\_Null em procedures**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Utilizar comandos para alterar o comportamento da query.

**Motivação para a regra**

Ao se utilizar qualquer alteração nestas configurações de ambiente, faz com que o SQL SERVER recompile a procedure cada vez que ela é executada.

Segundo uma recomendação de boas práticas, este recurso deve ser utilizado somente em casos de extrema necessidade.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROCEDURE STP\_SHOW\_MESSAGE  @MESSAGE VARCHAR(50)  AS  BEGIN       SET ANSI\_DEFAULTS OFF       SET ANSI\_NULLS OFF       SET ANSI\_PADDING OFF       SET ANSI\_WARNINGS OFF       SET CONCAT\_NULL\_YIELDS\_NULL OFF       PRINT @MESSAGE       SET ANSI\_DEFAULTS ON       SET ANSI\_NULLS ON       SET ANSI\_PADDING ON       SET ANSI\_WARNINGS ON       SET CONCAT\_NULL\_YIELDS\_NULL ON  END |  | **Observância da regra**  N/A |

**INTEGRIDADE**

**SELECT com TOP sem ORDER BY**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável que ao utilizar TOP em SELECTs, utilizar ORDER BY.

**Motivação para a regra**  
Essa regra tem por objetivo apontar possíveis distorções nos resultados de uma query que utilize TOP.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TOP 10         TITLE\_ID,         TITLE,         TYPE,         PUB\_ID,         PRICE,         ADVANCE,         ROYALTY,         YTD\_SALES,         NOTES,         PUBDATE  FROM TITLES |  | **Observância da regra**  SELECT TOP 10         TITLE\_ID,         TITLE,         TYPE,         PUB\_ID,         PRICE,         ADVANCE,         ROYALTY,         YTD\_SALES,         NOTES,         PUBDATE  FROM TITLES  ORDER BY TITLE |

**Uso de comandos DDL em SQL Batches**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

É recomendável que comandos que afetem estruturas ou dados não estejam em SQL batch. Exemplo de comandos ofensivos :

•TRUNCATE TABLE;

•DROP (TABLE, PROC, USER e ETC);

•CREATE (TABLE, USER e ETC);

**Motivação para a regra**

Garantir a integridade da estrutura da base de dados.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**  
Quando forem usadas tabelas temporárias ou tabelas do tipo variável.

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  ALTER TABLE TITLES  ADD COLTESTE INT  CREATE TABLE JOBS  (JOB\_ID SMALLINT IDENTITY(1,1)  PRIMARY KEY CLUSTERED,  JOB\_DESC VARCHAR(50) NOT NULL  DEFAULT ,  MIN\_LVL TINYINT NOT NULL  CHECK (MIN\_LVL >= 10),  MAX\_LVL TINYINT NOT NULL  CHECK (MAX\_LVL <= 250 |  | **Observância da regra**  N/A |

**CONTADORES**

**Operações por stored procedure**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Contar total de:

- INSERT

- DELETE

- UPDATE

- SELECT

**Motivação para a regra**  
Verificar se a operação básica da procedure está sendo seguida, além de medir complexidade por procedure. Essa regra será utilizada para indicação de stored procedure muito complexa, o que a indicará para revisão manual.

**Critério para o apontamento**  
N/A

**Exceção**  
N/A

**Impacto**  
Info

**Falso positivo**  
N/A

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras** |  | **Observância da regra**  N/A |

**PADRONIZAÇÃO**

**Uso de Select \***[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável que todas as colunas de uma seleção sejam explicitamente especificadas.

**Motivação para a regra**  
Garantir que apenas as informações necessárias serão trafegadas pela rede, além de garantir a integridade dos dados retornados na ocorrência de modificação na estrutura de uma tabela.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT \* FROM TITLES  WHERE TITLE\_ID = 1023 |  | **Observância da regra**  SELECT TITLE\_ID,   TITLE,   TYPE,   PUB\_ID,   PRICE,   ADVANCE,   ROYALTY,   YTD\_SALES,   NOTES,   PUBDATE  FROM TITLES  WHERE TITLE\_ID = 1023 |

**CASE sem ELSE**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Todo valor deve preferencialmente retornar resultado na forma de result set.

**Motivação para a regra**  
Garantir que o resultado do processamento de uma procedure seja utilizado. A utilização de parâmetros que retornam valores, pode gerar problemas de entendimento na utilização da mesma.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT CATEGORY =         CASE TYPE              WHEN 'POPULAR\_COMP'                   THEN 'POPULAR COMPUTING'              WHEN 'MOD\_COOK'                   THEN 'MODERN COOKING'              WHEN 'BUSINESS'                   THEN 'BUSINESS'              WHEN 'PSYCHOLOGY'                   THEN 'PSYCHOLOGY'         END,  PRICE AS PRICE  FROM TITLES |  | **Observância da regra**  SELECT CATEGORY =         CASE TYPE              WHEN 'POPULAR\_COMP'                   THEN 'POPULAR COMPUTING'              WHEN 'MOD\_COOK'                   THEN 'MODERN COOKING'              WHEN 'BUSINESS'                   THEN 'BUSINESS'              WHEN 'PSYCHOLOGY'                   THEN 'PSYCHOLOGY'              ELSE 'NOT YET CATEGORIZED'         END,  PRICE AS PRICE  FROM TITLES |

**Parâmetro do tipo OUT em procedure**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Todo valor deve preferencialmente retornar resultado na forma de result set.

**Motivação para a regra**  
Garantir que o resultado do processamento de uma procedure seja utilizado. A utilização de parâmetros que retornam valores, pode gerar problemas de entendimento na utilização da mesma.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE @RESULT MONEY  EXEC TITLES\_SUM 'THE GOURMET MICROWAVE', @RESULT  SELECT @RESULT  CREATE PROCEDURE TITLES\_SUM  (      @TITLE VARCHAR(40) = '%',      @SUMTOT MONEY OUTPUT  )  AS  SELECT 'TITLE NAME' = TITLE  FROM TITLES  WHERE TITLE = @TITLE  SELECT @SUMTOT = SUM(PRICE)  FROM TITLES  WHERE TITLE = @TITLE |  | **Observância da regra**  DECLARE @RESULT MONEY  EXEC TITLES\_SUM 'THE GOURMET MICROWAVE'  GO    CREATE PROCEDURE TITLES\_SUM  (     @TITLE VARCHAR(40) = '%'  )  AS  DECLARE @SUMTOT MONEY  SELECT 'TITLE NAME' = TITLE ,         SUM(PRICE) AS SUMTOT  FROM TITLES  WHERE TITLE = @TITLE  GROUP BY TITLE  GO |

**Uso de JOIN ANSI-STANDARD**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável utilizar ANSI JOIN, o que facilita a legibilidade e entendimento de uma query e permite que a cláusula WHERE trate apenas do filtro da query.

**Motivação para a regra**  
Melhorar a legibilidade de query com JOIN.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT A.AU\_ID, T.TITLE  FROM TITLES T,               AUTHORS A,              TITLEAUTHOR TA  WHERE A.AU\_ID = TA.AU\_ID  AND TA.TITLE\_ID = T.TITLE\_ID  AND T.TITLE LIKE 'COMPUTER%' |  | **Observância da regra**  SELECT A.AU\_ID, T.TITLE      FROM AUTHORS A         INNER JOIN TITLEAUTHOR TA            ON A.AU\_ID = TA.AU\_ID         INNER JOIN TITLES T            ON TA.TITLE\_ID = T.TITLE\_ID  WHERE T.TITLE LIKE 'COMPUTER%' |

**INSERT sem as colunas declaradas**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

É recomendável que todas as colunas de um insert sejam declaradas explicitamente no statement.

Quando os campos da tabela são informados, isto evita que após uma mudança na tabela, a instrução insert não gere um erro de número de campos não correspondente ou mesmo valores informados sejam de tipo errado.

**Motivação para a regra**  
Evitar problemas com mudança de estrutura em tabelas.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  INSERT INTO JOBS VALUES('MANAGER',50,70) |  | **Observância da regra**  INSERT INTO JOBS(JOB\_DESC,MIN\_LVL,MAX\_LVL) VALUES('MANAGER',50,70) |

**Nome de Stored Procedure**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Não utilizar o prefixo sp\_ para nome de stored procedure. Isto porque o SQL asssume que qualquer procedure com este prefixo seja de sistema e encontra-se no Master.

**Motivação para a regra**

Quando o Sql inicia a pesquisa no banco Master, não encontrando o plano de execução o objeto sofre um COMPILE lock, que, em determinadas situação pode gerar contenção onerando o SQL.

Outro ponto que requer atenção é a execução de procedures com o mesmo nome que foram compiladas, mesmo em bancos case-insensitive. O SQL precisa corretamente identificar e executar a stored procedure.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROCEDURE sp\_ShowMessage  @Message VARCHAR(40)  AS  BEGIN          PRINT @Message  END |  | **Observância da regra**  CREATE PROCEDURE stp\_ShowMessage  @Message VARCHAR(40)  AS  BEGIN          PRINT @Message  END |

**Uso de Non-ANSI Outer Joins**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável utilizar ANSI Outer Join, o que facilita a legibilidade e entendimento de uma query, permite que a cláusula WHERE trate apenas do filtro da query e evitará problemas em migração para versões futuras do SQL Server.

**Motivação para a regra**  
A Utilização de NON-ANSI Outer Join não é reconhecida pelo SQL Server 2005, 2008 e 2008 R2.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT employee.emp\_id,         employee.job\_id  FROM employee,jobs  WHERE employee.job\_id \*= jobs.job\_id  And (jobs.job\_id IS NULL) |  | **Observância da regra**  SELECT employee.emp\_id,         employee.job\_id  FROM employee           LEFT OUTER JOIN jobs                ON employee.job\_id = jobs.job\_id  WHERE (jobs.job\_id IS NULL) |

**Uso de número para identificar colunas no ORDER BY**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
É recomendável que as colunas que necessitam serem ordenadas sejam devidamente declaradas na cláusula ORDER BY.

**Motivação para a regra**  
Evitar problemas com mudança (alteração ou exclusão) das colunas selecionadas.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLE\_ID,         TITLE,         TYPE,         PUB\_ID,         PRICE,         ADVANCE,         ROYALTY,         YTD\_SALES,         NOTES,         PUBDATE  FROM TITLES  ORDER BY 2 |  | **Observância da regra**  SELECT TITLE\_ID,         TITLE,         TYPE,         PUB\_ID,         PRICE,         ADVANCE,         ROYALTY,         YTD\_SALES,         NOTES,         PUBDATE  FROM TITLES  ORDER BY TITLE |

**Comparação com valores NULL**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**

É recomendável que todas as comparações com valores NULL utilizem operador IS ao invés de “=” e utilizar SET ANSI\_NULLS ON.

**Motivação para a regra**  
Evitar erros em comparações e garantir o correto funcionamento do script SQL.

**Critério para o apontamento**

Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  DECLARE @VARNAME INT  SELECT @VARNAME = NULL  SELECT COL\_A  FROM T1  WHERE A = @VARNAME |  | **Observância da regra**  SET ANSI\_NULLS ON  GO  SELECT COL\_A  FROM T1  WHERE A IS NULL |

**Elementos na lista do SELECT (colunas/expressões) não estão qualificados com o nome de tabela/view**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Comando SELECT que possua JOIN deve possuir o nome das tabelas/views identificadas na frente da coluna.

**Motivação para a regra**  
Melhorar a legibilidade e entendimento de um comando SQL.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Blocker

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  SELECT TITLE\_ID,         TITLE,         PUB\_NAME,         CITY,         STATE,         COUNTRY  FROM TITLES       INNER JOIN PUBLISHERS           ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID  WHERE TITLE = 'THE GOURMET MICROWAVE' |  | **Observância da regra**  SELECT TITLES.TITLE\_ID,         TITLES.TITLE,         PUBLISHERS.PUB\_NAME,         PUBLISHERS.CITY,         PUBLISHERS.STATE,         PUBLISHERS.COUNTRY  FROM TITLES       INNER JOIN PUBLISHERS           ON PUBLISHERS.PUB\_ID = TITLES.PUB\_ID  WHERE TITLES.TITLE = 'THE GOURMET MICROWAVE' |

**WHILE com mais de um ponto de saída**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Ao montar um WHILE, o desenvolvedor não deve utilizar BREAK, pois, desta forma, evidência problemas na lógica do programa.

**Motivação para a regra**  
Garantir a melhor lógica na construção de um script SQL-BATCH.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  USE PUBS  GO  WHILE (SELECT AVG(PRICE) FROM TITLES) < $30  BEGIN       UPDATE TITLES              SET PRICE = PRICE \* 2       SELECT MAX(PRICE) FROM TITLES       IF (SELECT MAX(PRICE) FROM TITLES) = $28           BREAK       ELSE           CONTINUE  END  PRINT 'TOO MUCH FOR THE MARKET TO BEAR' |  | **Observância da regra**  USE PUBS  GO  WHILE (SELECT AVG(PRICE) FROM TITLES) < $28  BEGIN      UPDATE TITLES             SET PRICE = PRICE \* 2      SELECT MAX(PRICE) FROM TITLES  END  PRINT 'TOO MUCH FOR THE MARKET TO BEAR' |

**Uso de CONVERT ou CAST em variável parâmetro de entrada**[**close**](https://ssl.tech4b.com.br/Homolog/CodeScan/Telas/regras.aspx?pid=26&codProjeto=105&codFabrica=60&codTecnologia=6&tecnologia=TSQL)

**Descrição**  
Todo parâmetro de entrada, deve estar declarado com o mesmo tipo da coluna na tabela em que o dado será utilizado, garantindo inclusive a melhor utilização dos índices.

**Motivação para a regra**  
O objetivo desta regra é detectar procedures mal desenvolvidas ou evidenciar mudanças emergências do tipo WORKAROUND.

**Critério para o apontamento**  
Serão contadas todas as ocorrências encontradas.

**Exceção**

**Impacto**  
Minor

**Falso positivo**

**Exemplos**

|  |  |  |
| --- | --- | --- |
| **Classificação das regras**  CREATE PROC SP\_SE\_AUTHORS  ( @AU\_LNAME CHAR(40) = NULL ) AS  SELECT AU\_ID ,         AU\_LNAME ,         AU\_FNAME ,         PHONE ,         ADDRESS ,         CITY ,         STATE ,         ZIP ,         CONTRACT  FROM AUTHORS  WHERE AU\_LNAME = ONVERT(VARCHAR(40),@AU\_LNAME)  GO |  | **Observância da regra**  CREATE PROC SP\_SE\_AUTHORS  ( @AU\_LNAME VARCHAR(40) = NULL ) AS  SELECT AU\_ID       , AU\_LNAME       , AU\_FNAME       , PHONE       , ADDRESS       , CITY       , STATE       , ZIP       , CONTRACT  FROM AUTHORS  WHERE AU\_LNAME = @AU\_LNAME  GO |