Resumo dos Comandos da Linguagem SQL

Comandos para *Triggers* e *Stored Procedures*

Grupo de Base de Dados - SCE

DDL - Comandos para a definição de *Stored Procedures*

Comando CREATE PROCEDURE

Cria uma "Stored Procedure"

Sintaxe:

```
CREATE PROCEDURE name
            [( param <datatype> [, param <datatype> …])]
            [ RETURNS ( param <datatype> [, param <datatype> …])]
      AS <procedure_body> [terminator]

      <procedure_body> =
            [ <variable_declaration_list>]
            <block>

      <variable_declaration_list> =
            DECLARE VARIABLE var <datatype>;
            [DECLARE VARIABLE var <datatype>; …]

      < block> =
            BEGIN
                  <compound_statement>
                  [ <compound_statement> …]
            END

      <compound_statement> = {< block> | statement;}

      < datatype> =
              SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION
            | {DECIMAL | NUMERIC} [( precision [, scale])]
            | DATE
            | { CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [( int)]
            [CHARACTER SET charname]
            | NCHAR | NATIONAL CHARACTER | NATIONAL CHAR
            [VARYING] [( int)]
```

Stored Procedures são procedimentos executados no servidor. Existem dois tipos de *Stored Procedures*:

Select Procedure são *procedures* usadas como se fossem uma relação em um comando `SELECT`. Uma *Select Procedure* deve retornar um ou mais valores como parâmetros de saída, ou um erro.

Executable procedures são *procedures* chamadas explicitamente por uma aplicação, através de um comando `EXECUTE PROCEDURE`. Uma *Executable procedure* pode opcionalmente retornar valores e/ou erros.

A sintaxe de ambos os tipos de *Stored Procedures* é a mesma, e a diferença é apenas conceitual, mudando a forma como são definidos e utilizados. No entanto, existem alguns comandos disponíveis que foram concebidos especificamente para um dos tipos.

Note-se que *Stored Procedures* são armazenadas na base de dados, portanto somente se tornam disponíveis para outras transações depois que aquela que o criou é finalizada por um comando `COMMIT`.

Comando DROP PROCEDURE

Elimina uma *Stored Procedure* da base de dados

Sintaxe:

Comandos para *Triggers* e *Stored Procedures*

```
DROP PROCEDURE name
```

Comando ALTER PROCEDURE

Modifica uma *Stored Procedure* da base de dados

Sintaxe:

```
ALTER PROCEDURE name
            [( param <datatype> [, param <datatype> …])]
            [RETURNS ( param <datatype> [, param <datatype> …])]
      AS <procedure_body> [ terminator]
```

O comando `ALTER PROCEDURE` é idêntico ao comando `CREATE PROCEDURE`, exceto que um procedimento com o mesmo nome tem que estar armazenado na base de dados. Note-se que como o comando `CREATE PROCEDURE`, este comando somente ficará disponível para outras transações após o `COMMIT` da transação atual. Além disso, a alteração da *procedure* não requer a recompilação dos programas que a utilizam. Mas devido a isso, deve-se tomar cuidado para não alterar a interface da *procedure* (como por exemplo o tipo ou número de parâmetros de entrada e de saída), ou os programas que a chamam terão que ser modificados também.

Comando EXECUTE PROCEDURE

Executa uma *Stored Procedure* armazenada na base de dados

Sintaxe:

```
EXECUTE PROCEDURE [TRANSACTION transaction] name
      [:param [, :param]]
      [RETURNING_VALUES : param [, :param ]];
```

Se a *procedure* requer parâmetros de entrada, eles devem ser passados como variáveis da linguagem hospedeira, ou como constantes. Se uma *procedure* retorna parâmetros de saída, as variáveis da linguagem hospedeira que recebem esses valores devem estar indicados na cláusula `RETURNING_VALUES`. Em **isql** não indique a cláusula `RETURNING_VALUES`, pois **isql** mostra os valores retornados automaticamente.

Exemplo:

```
EXECUTE PROCEDURE TotalTurmasMinistra /* TotalTurmasMinistra e' o nome */
    "Amauri", :CodigoDisciplina   /* Amauri 'e uma constante, CodigoDisciplina uma variável*/
    RETURNING_VALUES :TotTurmas;
```

Comando CREATE TRIGGER

Cria um *Trigger* na base de dados, incluindo quando ele é acionado, e que ações tomar.

Sintaxe:

```
CREATE TRIGGER name FOR table
            [ACTIVE | INACTIVE]
            {BEFORE | AFTER}
            {DELETE | INSERT | UPDATE}
            [POSITION number]
      AS <trigger_body> terminator

      <trigger_body> =
            [ <variable_declaration_list>]
            <block>

      <variable_declaration_list> =
            DECLARE VARIABLE variable <datatype>;
            [DECLARE VARIABLE variable <datatype>; …]
```

Comandos para *Triggers* e *Stored Procedures*

```
< block> =
      BEGIN
            <compound_statement>
            [ <compound_statement> …]
      END

<compound_statement> = { <block> | statement;}

< datatype> =
        SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION
      | {DECIMAL | NUMERIC} [( precision [, scale])]
      | DATE
      | { CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [( int)]
      [CHARACTER SET charname]
      | NCHAR | NATIONAL CHARACTER | NATIONAL CHAR
      [VARYING] [( int)]
```

Triggers são procedimentos executados no servidor, e sempre são associados a uma tabela ou *view*, e a um comando `INSERT`, `UPDATE`, ou `DELETE` . Sempre que o comando associado é executado na tabela associada, o procedimento *trigger* é automaticamente executado. A execução é repetida para cada tupla afetada. Ele não pode ser chamado explicitamente. Assim, a definição de um *trigger* inclui além do seu nome, também a especificação da tabela e operação associadas, e a indicação de quando o *trigger* dispara.

Exemplo:

```
SET TERM !! ;

CREATE TRIGGER NovaTurma FOR Turma BEFORE INSERT AS
  BEGIN
    NEW.codigo = GEN_ID(ContaTurma, 1);
  END !!

SET TERM ; !!
```

Comando DROP TRIGGER

Elimina um *trigger* da base de dados

Sintaxe:

```
DROP TRIGGER name
```

Comando ALTER TRIGGER

Modifica um *trigger* da base de dados.

Sintaxe:

```
ALTER TRIGGER name FOR table
            [ACTIVE | INACTIVE]
            {BEFORE | AFTER}
            {DELETE | INSERT | UPDATE}
            [POSITION number]
      AS <trigger_body> terminator
```

Extensões ao SQL para suporte a *Stored Procedures*

Comando FOR SELECT ... DO

Recupera múltiplas linhas em uma *procedure*.

Comandos para *Triggers* e *Stored Procedures*

Sintaxe:
```
FOR <select_expr>
    DO <compound_statement>;
```

O comando `FOR SELECT` difere de um comando SQL `SELECT` comum porque o `FOR SELECT` é na realidade um comando de *loop*, que para cada tupla recuperada pelo `SELECT` executa o `<compound_statement>`. A `<select_expr>` é um comando SQL `SELECT` normal, com a cláusula `INTO` obrigatória.

No exemplo seguinte, esse comando é usado na *procedure* `TotalizaMatricula` para contar quantas matrículas existem na tabela `Matricula` para cada disciplina existente na tabela `Disciplina`, e então atualizar o atributo `NNalunos` dessa tabela.

Exemplo:
```
SET TERM !! ;

CREATE PROCEDURE TotalizaMatricula AS
  DECLARE VARIABLE TotTurma INTEGER;
  DECLARE VARIABLE Codigo DECIMAL(4);
    BEGIN
    FOR  SELECT codigoturma, COUNT(NUSP)
           FROM Matricula
           GROUP BY codigoturma
           INTO :Codigo, :TotTurma
       DO BEGIN
         UPDATE Turma
           SET NNalunos=:TotTurma
           WHERE codigo=:Codigo;
         END
    EXIT;
    END !!

SET TERM ; !!
```

***Comandos* SUSPEND e EXIT**

Sintaxe:
```
SUSPEND;
EXIT;
```

O Comando `SUSPEND` suspende a execução do *procedure*, retornando os valores adequados dos parâmetros de saída. Se o comando `SUSPEND` ocorre em um *executable procedure*, ele tem o mesmo efeito de um comando `EXIT`, pois o *procedure* não é retomado. Porém, em um *select procedure*, a execução é retomada no comando seguinte ao `SUSPEND` quando o próximo `FETCH` do `SELECT` é executado. Atenção para que um parâmetro de saída que não tenha tido atribuição pode voltar qualquer valor.

O Comando `EXIT` desvia o controle para o último `END` do *procedure*. O que acontece quando esse `END` é atingido depende do tipo do *procedure*. Em um *select procedure*, o `END` final retorna o controle para a módulo que chamou esse *procedure*, e coloca o valor de `SQLCODE` como sendo `100`, o que indica não haver mais tuplas a serem recuperadas. Em um *executable procedure*, o `END` final apenas retorna os valores dos parâmetros de saída para o módulo que o chamou.

Como definir *Stored Procedures*

Caracter terminador de comandos

Comandos para *Triggers* e *Stored Procedures*

O comando `CREATE PROCEDURE` é um comando como qualquer outro, e portanto termina com o caracter '`;`'. No entanto, o comando `CREATE PROCEDURE` contém outros comandos internamente, que usam o mesmo terminador. Assim, se este comando estiver sendo utilizado pelo para definir (e não executar imediatamente) uma *Stored Procedure*, então um *script file* deve modificar o caracter utilizado para terminar um comando, e restaurá-lo depois de encerrada a definição da *Stored Procedure*. Isso é feito pelo comando `SET TERM`, como no exemplo a seguir. O primeiro `SET TERM define "##"` como o caracter de finalização de comandos, para o interpretador **isql** . O segundo `SET TERM` restaura o "`;`" como o caracter de finalização .

```
SET TERM ## ;
CREATE PROCEDURE proc (var1 SMALLINT, var2 CHAR(5)) AS
      BEGIN
       . . .
      END
RETURN;
END ##   /* o ## encerra a definição do Create procedure */
SET TERM ; ##
```

Variáveis

Existem três tipos de variáveis que podem ser utilizadas no corpo de uma *procedure*:

Parâmetros de entrada - utilizadas para passar variáveis da aplicação para a *procedure*. Estas variáveis não podem ser utilizadas em *triggers*. Variáveis de entrada são passadas por valor;

Parâmetros de saída - utilizadas para passar variáveis da *procedure* para a aplicação;

Variáveis locais - utilizadas para armazenar valores internamente à *procedure*. São visíveis apenas dentro do corpo da *procedure*, e devem ser sempre inicializadas cada vez que a *procedure* for chamada, antes de poderem ser usadas. Todas as variáveis locais devem ser definidas no início do corpo do procedimento, pelo comando

```
DECLARE VARIABLE var datatype;
```

Note-se que cada variável local requer uma comando `DECLARE VARIABLE`, finalizado por ponto-e-vírgula (`;`).

As variáveis são utilizadas da mesma maneira onde uma expressão puder ser utilizada. Quando as variáveis forem utilizadas em comandos SQL, elas devem ser precedidas por dois-pontos (`:`), para indicar que são variáveis e não nomes de atributos. Nos demais comandos (como `while` e `if`) não devem ser utilizado o caracter dois-pontos.

Quanto as variáveis de saída de uma *select procedure* , os nomes das colunas do comando `SELECT` devem coincidir em tipo e em nome com os que aparecem na definição da *procedure*. As variáveis de saída de uma *executable procedure* apenas o tipo de dados devem ser coincidentes

---