T-SQL

Flow Control:

BEGIN...END

also known as *statement_block*

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

BREAK

Stop a WHILE loop.

CONTINUE

Restart a WHILE loop.

GOTO ***label***

Alter the flow of execution to a label.

```
GOTO label
```

Note:

definig the label:

```
label :
```

IF...ELSE

```
IF boolean expression
{sql_statement | statement_block}
[
ELSE
{sql_statement | statement_block}
]
```

RETURN

Exits uncondicionally from a query or procedure

```
RETURN [ integer expression ]
```

TRY...CATCH

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

WAITFOR

Block the execution of a batch, stored procedure or transaction until:

- specified time or a time interval is reached

- specified statement modifies or returns at least one row

```
WAITFOR
{
DELAY 'time_to_pass'
| TIME 'time_to_execute'
| [ ( receive_statement )
| ( get_conversation_group_statement ) ]
[ , TIMEOUT timeout ]
}
```

WHILE

```
WHILE Boolean_expression
{ sql_statement | statement_block | BREAK | CONTINUE }
```

BATCHES

The following rules apply to using batches:

- CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE SCHEMA, CREATE TRIGGER, and CREATE VIEW statements cannot be combined with other statements in a batch. The CREATE statement must start the batch. All other statements that follow in that batch will be interpreted as part of the definition of the first CREATE statement.

- A table cannot be changed and then the new columns referenced in the same batch.

- If an EXECUTE statement is the first statement in a batch, the EXECUTE keyword is not required. The EXECUTE keyword is required if the EXECUTE statement is not the first statement in the batch.

Reference

```
/*...*/      (Comment)
--           (Comment)
```

CASE

Evaluates a list of conditions and returns one of multiple possible result expressions.

The CASE expression has two formats:

- The simple CASE expression compares an expression to a set of simple expressions to determine the result.

- The searched CASE expression evaluates a set of Boolean expressions to determine the result.

Both formats support an optional ELSE argument.

Simple CASE expression:

```
CASE input_expression
 WHEN when_expression THEN result_expression [ ...n ]
 [ ELSE else_result_expression ]
END
```

Searched CASE expression:

```
CASE
 WHEN Boolean_expression THEN result_expression [ ...n ]
 [ ELSE else_result_expression ]
END
```

DECLARE @local_variable

Variables are declared in the body of a batch or procedure with the DECLARE statement and are assigned values by using either a SET or SELECT statement. Cursor variables can be declared with this statement and used with other cursor-related statements. After declaration, all variables are initialized as NULL, unless a value is provided as part of the declaration.

```
DECLARE
{
{{ @local_variable [AS] data_type } | [ = value ] }
| { @cursor_variable_name CURSOR }
} [,...n]
| { @table_variable_name [AS] <table_type_definition>
 | <user-defined table type>
}

<table_type_definition> ::=
 TABLE ( { <column_definition> | <table_constraint> } [ ,... ] )
<column_constraint> ::=
 {
 [ NULL | NOT NULL ]
 | [ PRIMARY KEY | UNIQUE ]
 | CHECK ( logical_expression )
 }

<table_constraint> ::=
 {
 { PRIMARY KEY | UNIQUE }
 ( column_name [ ,... ] )
 | CHECK ( search_condition )
 }

<column_definition> ::=
 column_name {scalar_data_type | AS computed_column_expression}
 [ COLLATE collation_name ]
 [ [ DEFAULT constant_expression ]
 | IDENTITY [ (seed ,increment ) ] ]
 [ ROWGUIDCOL ]
 [ <column_constraint> ]
```

EXECUTE (Transact-SQL)

Executes a command string or character string within a Transact-SQL batch, or one of the following modules: system stored procedure, user-defined stored procedure, scalar-valued user-defined function, or extended stored procedure.

Execute a stored procedure or function

```
[ { EXEC | EXECUTE } ]
{
[ @return_status = ]
{ module_name [ ;number ] | @module_name_var }
[ [ @parameter = ] { value
 | @variable [ OUTPUT ]
 | [ DEFAULT ]
 }
]
[ ,...n ]
[ WITH RECOMPILE ]
}
;
```

Execute a character string

```
{ EXEC | EXECUTE }
( { @string_variable | [ N ]'tsql_string' } [ + ...n ] )
[ AS { LOGIN | USER } = ' name ' ]
;
```

Execute a pass-through command against a linked server

```
{ EXEC | EXECUTE }
(
{ @string_variable | [ N ] 'command_string [ ? ]' } [ + ...n ]
[ { , { value | @variable [ OUTPUT ] } } [ ...n ] ]
)
[ AS { LOGIN | USER } = ' name ' ]
[ AT linked_server_name ]
;
```

PRINT (Transact-SQL)

```
PRINT msg_str | @local_variable | string_expr
```

RAISERROR (Transact-SQL)

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

CURSORES:

ISO Syntax

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
    FOR select_statement
    [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
;
```

Transact-SQL Extended Syntax

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
    [ FORWARD_ONLY | SCROLL ]
    [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
    [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
    [ TYPE_WARNING ]
    FOR select_statement
    [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
;
```

STORED PROCEDURES

CREATE PROCEDURE (Transact-SQL)

Creates a Transact-SQL or common language runtime (CLR) stored procedure in SQL Server 2008 R2. Stored procedures are similar to procedures in other programming languages in that they can:

- Accept input parameters and return multiple values in the form of output parameters to the calling procedure or batch.

- Contain programming statements that perform operations in the database, including calling other procedures.

- Return a status value to a calling procedure or batch to indicate success or failure (and the reason for failure).

Use this statement to create a permanent procedure in the current database or a temporary procedure in the tempdb database.

--Transact-SQL Stored Procedure Syntax

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

--CLR Stored Procedure Syntax

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name
[ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ = default ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH EXECUTE AS Clause ]
AS { EXTERNAL NAME assembly_name.class_name.method_name }
;
```

CREATE TRIGGER (Transact-SQL)

Creates a DML, DDL, or logon trigger. A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. DML triggers execute when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view. DDL triggers execute in response to a variety of data definition language (DDL) events. These events primarily correspond to Transact-SQL CREATE, ALTER, and DROP statements, and certain system stored procedures that perform DDL-like operations. Logon triggers fire in response to the LOGON event that is raised when a user sessions is being established. Triggers can be created in the SQL Server 2005 Database Engine directly from Transact-SQL statements or from methods of assemblies that are created in the Microsoft .NET Framework common language runtime (CLR) and uploaded to an instance of SQL Server. SQL Server allows for creating multiple triggers for any specific statement.

Trigger on an INSERT, UPDATE, or DELETE statement to a table or view (DML Trigger)

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier
[ ; ] > }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE, or UPDATE STATISTICS statement (DDL Trigger)

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier
>  [ ; ] }

<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Trigger on a LOGON event (Logon Trigger)

```
CREATE TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR | AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier>
[ ; ] }
<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]

<method_specifier> ::=
    assembly_name.class_name.method_name
```

Transacções

Objectivos

* Fornecer mecanismos de recuperação em caso de falhas do sistema
* Facilitar o tratamento de erros ao nivel das aplicações
* Fornecer mecanismos que permitam controlar de forma eficiente as interferências entre aplicações que concorrem no acesso aos mesmos recursos

Propriedades(ACID)

* **Atomicidade (Atomicity) :** Uma transacção é indivisível no seu processamento
* **Consistência (Consistency preservation) :** Uma transacção conduz a BD de um estado consistente para outro estado consistente
* **Isolamento (Isolation) :** As transacções concorrentes não devem interferir umas com as outras durante a sua execução
* **Durabilidade (Durability) :** O resultado de uma transacção válida deve ser tornado persistente (mesmo na presença de falhas, após commit)

Estados

Activa: é o estado após inicio da transação e mantém-se enquanto se forem realizando operações de leitura e escrita sobre os dados.

Parcialmente commited: quando se indica que a transacção deve terminar com sucesso, entra-se neste estado. Nele é garantido que todos os dados são transferidos para disco (force-writing) e só se isso acontecer é que a transacção atinge o commit point

Commited: a transaccão entra neste estado quando atinge o commit point

Falhada: a transacção vem para este estado se for abortada no seu estado activa ou se os testes realizados no estado parcialmente commited falharem

Terminada: a transação deixa de existir no sistema.

SET TRANSACTION ISOLATION LEVEL (Transact-SQL)

```
SET TRANSACTION ISOLATION LEVEL
{    READ COMMITTED     (por omissão)
| READ UNCOMMITTED
| REPEATABLE READ
| SERIALIZABLE
| SNAPSHOT
}
```

BEGIN TRANSACTION (Transact-SQL)

```
BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable }
      [ WITH MARK [ 'description' ] ]
    ];
```

COMMIT TRANSACTION (Transact-SQL)

```
COMMIT { TRAN | TRANSACTION } [ transaction_name |
@tran_name_variable ] ];
```

COMMIT WORK (Transact-SQL)

```
COMMIT [ WORK ] ;
```

ROLLBACK WORK (Transact-SQL)

```
ROLLBACK [ WORK ] ;
```

ROLLBACK TRANSACTION (Transact-SQL)

```
ROLLBACK { TRAN | TRANSACTION }
    [ transaction_name | @tran_name_variable
    | savepoint_name | @savepoint_variable ] ;
```

SAVE TRANSACTION (Transact-SQL)

```
SAVE { TRAN | TRANSACTION } { savepoint_name |
@savepoint_variable } ;
```

XML

Constituido

- **Elementos**
  - Pode conter elementos
  - Pode conter atributos
  - Não pode:
    - ter sinais de pontuação
    - ter a palavra XML (seja como estiver escrita)
    - ter espaços
    - no conteudo o caracter menor (<)
  - Evitar o uso de :
    - sinal de subtracção (-) e ponto (.)
    - dois pontos(:)
- **Atributos**
  - Pertence sempre a um e só um elemento
  - Não pode conter outros elementos ou atributos
- **Regras**
- As Tags
  - São case Sensitive
  - Não aceitam espaços
  - Andam aos pares. Tem tag de inicio e de fim
  - Valor dos atributos são escritos entre plicas(') ou aspas (")
  - Adequadamente aninhados
- **Declaração**
- Deve iniciar com uma declaração XML (é opcional)
  - Tem que ser a primeira linha, não pode conter qualquer caracter antes
  - Declaração tem o formato <?xml atributos>
    - Atributos:
      - version (obrigatório)
      - encoding(opcional)
        - ISO-8859-1
        - UTF8
      - standalone(opcional)
        - Não precisa de analisar informação externa = yes
        - É analisada informação externa (DTD)= no
  - Comentários:
    - <!-- Comentário →
  - O operador &, designa-se por referência a entidade.
    - o parser substitui o &tag, pelo texto representado por tag
  - 5 referências a entidades
    - &lt (obrigatório no conteudo de um elemento) (<)
    - &amp(obrigatório no conteudo de um elemento)(&)
    - &gt(>)
    - &quot(")
    - &apos(')

Espaço de Nomes (Nome qualificado)

- Distingue elementos e atributos de diferentes vocabulários
- Agrupar elementos e atributos que se relacionam
- Todos os elementos e atributos com o mesmo URI tem o mesmo namespace
  - URI
    - <schemeName>:<hierarchicalPart>[?<query>][#<fragment>]
    - URN = urn:namespaceID:namespaceSpecificString
- Atributo XNLS
  - Evita-se conflitos entre nomes
  - Usa-se para associar um previxo a um URI
    - formato: xmlns:prefixo="URI"
      - nome ficam na forma prefixo:nome
- Quando o namespace é definido num elemento, contextualiza todos os seus descendentes
- Processamento sobre um Namespace
  - URI fornece nome único ao namespace
  - não é utilizado pelos parsers
  - valida se os nomes foram associados a um URI

Tratamento de Texto

- Dentro da secção CDATA tudo é ignorado
  - <![CDATA[...]]>

DTD

Linguagem para especificar a gramática

Especifica

- elementos, atributos e entidades
- contexto em cada componente
- atributos com valor unico e referencias para esses atributos

Convenção

- ? = zero ou uma ocorrência
- * = zero ou mais ocorrências
- + = uma ou mais ocorrências

Declaração

- Inclusão
  - Inline:<!DOCTYPE elemento-raiz [declarações-de-elementos]>
  - externo:<!DOCTYPE elemento-raiz SYSTEM "nome-ficheiro">
- Elementos
  - <!ELEMENT nome categoria>
  - <!ELEMENT nome (tipo-de-conteudo-do-elemento)>
  - Diferentes declarações
    - Vazio: <!ELEMENT nome EMPTY>
    - Qualquer conteudo: <!ELEMENT nome ANY>
    - Não contem elementos , só texto: <!ELEMENT nome (#PCDATA)>
    - Contem elementos descendentes: <!ELEMENT nome (nome-desc-1, nome-desc-2, ...)>
    - Contem conteudo misto: <!ELEMENT nome (nome-elem-a | nome-elem-b | ...)>
- Atributos
  - <!ATTLIST nome-elem nome-atr tipo-atr valor-omissao>
  - tipo-atr
    - texto que não será analisado : CDATA
    - e.g. nome XML válido : NMTOKEN
    - lista de NMTOKEN : NMTOKENS
    - uma entidade : ENTITY
    - lista de entidades : ENTITIES
    - valor (deste atributo) é único : ID
    - valor de outro ID : IDREF
    - lista de outros IDs : IDREFS
    - um de vários TOKENs (ei) : (e1 | e2 | …)
    - nome de uma anotação : NOTATION
    - valor XML predefinido : xml:
  - valor-omissao
    - valor omissão do atributo : "valor"
    - atributo obrigatório : #REQUIRED
    - atributo opcional : #IMPLIED
    - atributo de valor fixo : #FIXED "valor"
- Entidade
  - Usada para conter texto
  - Refencia a Entidade, refere uma Entidade (&ENTIDADE)
  - Declara-se:
    - Interna: <!ENTITY nome-entidade "valor-entidade">
    - Externa: <!ENTITY nome-entidade SYSTEM "URI">

DTD: subconjunto interno

- Definições que estão no ficheiro xml
- Boa abordagem para construir um bom DTD
- Subconjunto
  - Interno
    - está contido, entre os parêntesis rectos:
      - <!DOCTYPE r [subconjunto interno]>
  - Externo
    - está num ficheiro
      - <!DOCTYPE r SYSTEM "ficheiro.dtd">
    - na declaração XML tem-se: standalone="no"
  - Ambos os subconjuntos têm que ser compativeis
    - Não pode redefinir elementos e atributos
    - As ENTITY podem ser redefinidas

Entidade Parametro

- Entidade Geral
- Define Elementos ou Atributos
- Declaração
  - <!ENTITY % nome-entidade "valor-entidade">
- Diferenças entre Entidade
  - entidade (geral): referida com & ;
  - entidade parâmetro: referida com %
  - Só pode ser utilizada em subconjunto externo
- Redefinição
  - Analisa primeiro o subconjunto interno e depois o externo
  - Quando uma entidade têm mais de uma definição, é considerada a primeira

XPATH

7 tipos de nós

- raiz : /
- elemento : nome
- atributo : @oNome
- texto : text()
- comentario : comment()
- instrução de processamento : processing-instruction()
- espaço de nomes : namespace-uri(), local-name(), name()

Localização

- procurar a partir deste ponto : // expressão
- selecciona o nó corrente : .
- selecciona o nó ascentente : ..
- selecciona todos os nós do tipo elemento: *
- selecciona todos os nós do tipo atributo: @*
- selecciona todos os nós : node()
- selecciona os nós que verificam as diversas expressões: expr1 | expr2 | expr3

Predicado

- É um passo de localização sobre o qual se define uma condição
- A condição é escrita entre parêntesis rectos
- Os operadores disponiveis são: =, !=, < , <=, >, >= , and , or
- Os valores podem ser escritos entre plicas ou com aspas
- Os parênteses curvos explicitam as precedencias

Eixo

- self: o nó corrente
- child: nós filho (eixo de omissão)
- descendant: toda a descendência; nós filho, nós filho do filho, etc
- descendant-or-self: o nó corrente e toda a sua descendência
- parent: o nó que contém o corrente; o nó raiz não tem parent
- ancestor: todos os antepassados; o nó raiz não tem ancestor
- ancestor-or-self: o nó corrente e todos os seus antepassados
- following: nós irmãos escritos depois do nó corrente
- preceding: nós irmãos escritos antes do nó corrente
- following-sibling: todos os descendentes dos nós irmãos escritos depois do nó corrente
- preceding-sibling: todos os antepassados dos nós irmãos escritos antes do nó corrente
- attribute: todos os atributos do nó corrente
- namespace: todos os "namespace" no âmbito do nó corrente

xmlDoc.setProperty "SelectionLanguage", "XPath"

- indexação começa em 1

Tipos

- node-set
- string
- boolean
- number

Funções (Algumas)

- position()
- last()
- count(expr)
- string(expr)
- starts-with(s1,s2)
- contains(s1, s2)
- translate(s1, s2 , s3)
- boolean(expr)
- not(expr)
- number(expr)
- sum(expr)
- round(n)

XSLT

Cada Regra especifica a transformação de um molde ("template") num padrão ("match")

Elemento raiz: stylesheet ou transform

- Estão no namespace: http://www.w3.org/1999/XSL/Transform

Regra de transformação

- <xsl:template match="padrão">molde</xsl:template>
  - padrão: expressão Xpath
  - molde: Transformação a aplicar
- Valores dos Nós
  - <xsl:value-of select="padrão"/>

Percorrer todos os nós

- <xsl:for-each select="padrão"/>

Testar alternativas

- + <xsl:if test="padrão">molde</xsl:if>

Definir relação de ordem

- <xsl:sort
  select = "expressãoXPath"
  data-type = "text" | "number" | "PrefixedName"
  lang = "langcode"
  order = "ascending" | "descending"
  case-order = "upper-first" | "lower-first"
  />

Avaliação especifica

- <xsl:apply-templates select="padrão"/>

Atributo modo

- Quando um nós tem que ser transportado multiplas vezes
  - No <xsl:template> o valor de mode indica o modo em que a regra deve ser activada
  - No <xsl:apply-template> o valor de mode indica em que modo serão activadas as correspondentes regras
- Atributo mode

Padrão do valor Atributo

- construir um atributo com base no valor de um outro qualquer elemento
- escreve-se como uma expressão XPath entre chavetas
- colocada como valor de atributo de um elemento (no documento XSLT)

Indicar o documento resultante

- <xsl:output
  method="xml | html | text | name"
  version="string" encoding="string"
  omit-xml-declaration="yes | no"
  standalone="yes | no"
  doctype-public="string"
  doctype-system="string"
  indent="yes | no"
  />

Noção de Variavel

- <xsl:variable>
  - representar uma variável global ou local
    - global se for declarado como elemento de topo
    - local se for declarado no contexto de um elemento <xsl:template>
    - apenas permite que lhe seja atribuído valor uma única vez
    - está mais próximo de uma constante do que de uma variável
- pagina(36)

XSD

Formato geral de um esquema

- tem um elemento raiz e pode ter uma declaração XML (<schema>)
- entre outros, tem o atributo targetNamespace
- Diz-se que um elemento é global quando
  - é descendente imediato do elemento schema
- Diz-se que um elemento é local quando
  - é descendente imediato de um elemento diferente de schema

Estruturas

- Tipo Simples
  - elemento, atributo ou restrição
  - um elemento só pode conter texto
  - <xs:element name="xxx" type="yyy" />
  - Um atributo é sempre do tipo simples
  - <xs:atribute
    name = "XMLname"
    type = "QName">
    default = "string"
    fixed = "string"
    use = "(optional | prohibited | required)"
    ref = "QName"
    </xs:attribute>
- Tipo Complexo
  - elementos que contem
    - outros elementos
    - contendo atributos
  - indicadores
    - indicador de relação de ordem (all, choice, sequence)
    - indicador de ocorrência (maxOccurs, minOccurs)
    - indicador de agrupamento (group name, attributeGroup name)
  - dar nome ao tipo complexo
  - <xs:element name="receita" type="tReceita"/>
  - <xs:complexType name="tReceita">
- Restrição
  - É sempre do tipo simples
  - <xs:simpleType>
  - <xs:restriction base="xs:integer">

XQuery

Estrutura FLWOR (pronuncia-se "flower"), cujo acrónimo significa

- for, let, where, order by, return

Expressões