Table of Contents

1. Introdução

Um banco de dados é como um arquivo eletrônico, ou seja, tem a mesma função que qualquer outro arquivo — armazenar registros. A única diferença é que no banco de dados os registros são armazenados eletronicamente.

De forma mais detalhada, um Banco de Dados Relacional é um conceito abstrato que define maneiras de armazenar, manipular e recuperar dados estruturados unicamente na forma de tabelas, construindo um banco de dados.

Um Banco de Dados Relacional é um banco de dados que segue o Modelo Relacional.

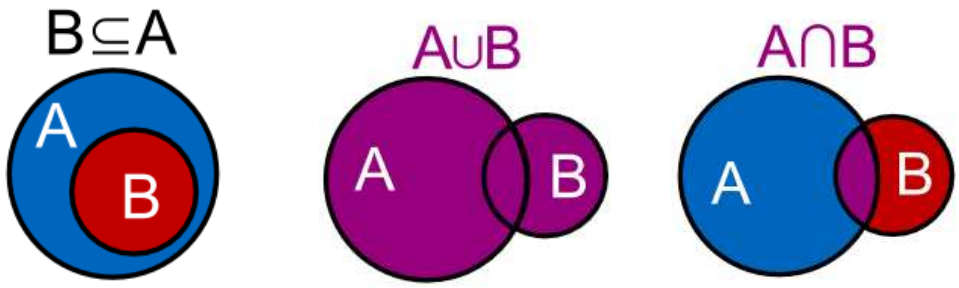
1.1. Origem

O termo também é aplicável aos próprios dados, quando organizados dessa forma, ou a um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) – do inglês Relational database management system (RDBMS) – um programa de computador que implementa a abstração.

O modelo relacional para gerência de bancos de dados (SGBD) é um modelo de dados baseado em lógica e na teoria de conjuntos.

Banco de Dados Relacional é um conceito abstrato que define maneiras de armazenar, manipular e recuperar dados sendo historicamente ele é o sucessor do modelo hierárquico e do modelo em rede. Um Banco de Dados Relacional é um banco de dados que segue o Modelo Relacional.

O modelo relacional para gerência de bancos de dados (SGBD) é um modelo de dados baseado em lógica e na teoria de conjuntos. Lembram-se?



Estas arquiteturas antigas são até hoje utilizadas em alguns data centers com alto volume de dados, onde a migração é inviabilizada pelo custo que ela demandaria

O modelo relacional foi inventado pelo Dr. Codd e subsequentemente mantido e aprimorado por **Chris Date** e Hugh Darwen como um modelo geral de dados. No Terceiro Manifesto (1995) eles mostraram como o modelo relacional pode ser estendido com características de orientação a objeto sem comprometer os seus princípios fundamentais.

O modelo relacional permite ao projetista criar um modelo lógico consistente da informação a ser armazenada. Este modelo lógico pode ser refinado através de um processo de normalização. Um banco de dados construído puramente baseado no modelo relacional estará inteiramente normalizado. O plano de acesso, outras implementações e detalhes de operação são tratados pelo sistema DBMS, e não devem ser refletidos no modelo lógico. Isto se contrapõe à prática comum para DBMSs SQL nos quais o ajuste de desempenho frequentemente requer mudanças no modelo lógico.

2. Sobre SQL

Para termos acesso aos registros armazenados ou mesmo cadastrar novos registros, precisamos de um sistema que gerencie o banco de dados. Este sistema gerenciador de banco de dados é que torna possíveis as operações com o conteúdo do arquivo, como — "Traga-me este arquivo", "Atualize este registro".

Existem vários tipos de sistemas de gerenciamento de banco de dados (SGBD ou DBMS), representando diversas abordagens relativas às tarefas de acesso às informações contidas no banco de dados, preservação da integridade dos dados, acompanhamento dos usuários e manutenção da segurança. Para o nosso estudo, porém, podemos classificar todos os sistemas em dois tipos: relacionais e não relacionais, embora seja visível o predomínio da abordagem relacional nos novos sistemas do mercado**, inclusive o Sistema Protheus utiliza-se de ambos**.

Em um sistema relacional, os dados são armazenados e representados exclusivamente em tabelas. Em nenhum momento faz-se necessário recorrer a outras estruturas, como árvores hierárquicas, para ter acesso aos dados.

A linguagem **SQL** — o nome é a sigla de *Structured Query Language (Linguagem de Query Estruturada)* — é uma linguagem para gerenciar um sistema de banco de dados relacional. Não só é uma linguagem, como também tem sido tão utilizada que pode ser considerada um padrão. Consiste de uma série de declarações, adotadas de comum acordo, que nos permitem realizar diversas operações.

Temos que usar a expressão comum acordo porque, embora uma SQL padrão tenha sido criada pelo Instituto de Padrões Nacionais Americanos (ANSI), todas as implementações particulares da SQL personalizam a linguagem de várias formas. Tais implementações complementam a linguagem padrão com novos tipos de declarações ou expressões e muitas vezes adaptam as declarações padronizadas às necessidades específicas.

3. Visão Geral de um Banco de Dados Relacional

Sistemas relacionais caracterizam um grande avanço no armazenamento e no gerenciamento de grandes quantidades de dados. A principal razão para isso é que, em um sistema relacional, pode-se reduzir bastante o armazenamento de dados redundantes. Na verdade, idealmente falando, em um sistema projetado segundo os princípios teóricos da abordagem relacional, **a redundância não deve existir**. Nenhum relacionamento entre dois itens de dados (uma pessoa possui um endereço, por exemplo) deve aparecer mais de uma vez em cada um banco de dados.

Na prática, os sistemas apenas se aproximam deste ideal, por várias razões, e podemos dizer que normalmente contêm alguns dados repetidos em vários lugares. Mas mesmo em um sistema relacional que apenas se aproxime da situação ideal, minimizar a redundância dos dados acarreta dois benefícios básicos:

- **primeiro lugar**, os dados podem ser reorganizados e combinados de forma mais facilmente em novos relacionamentos; não ficam presos aos relacionamentos em que foram armazenados.

- **segundo lugar**, a atualização torna-se muito mais fácil, pois poucos itens de dados têm que ser atualizados, o que reduz a incidência de erros.

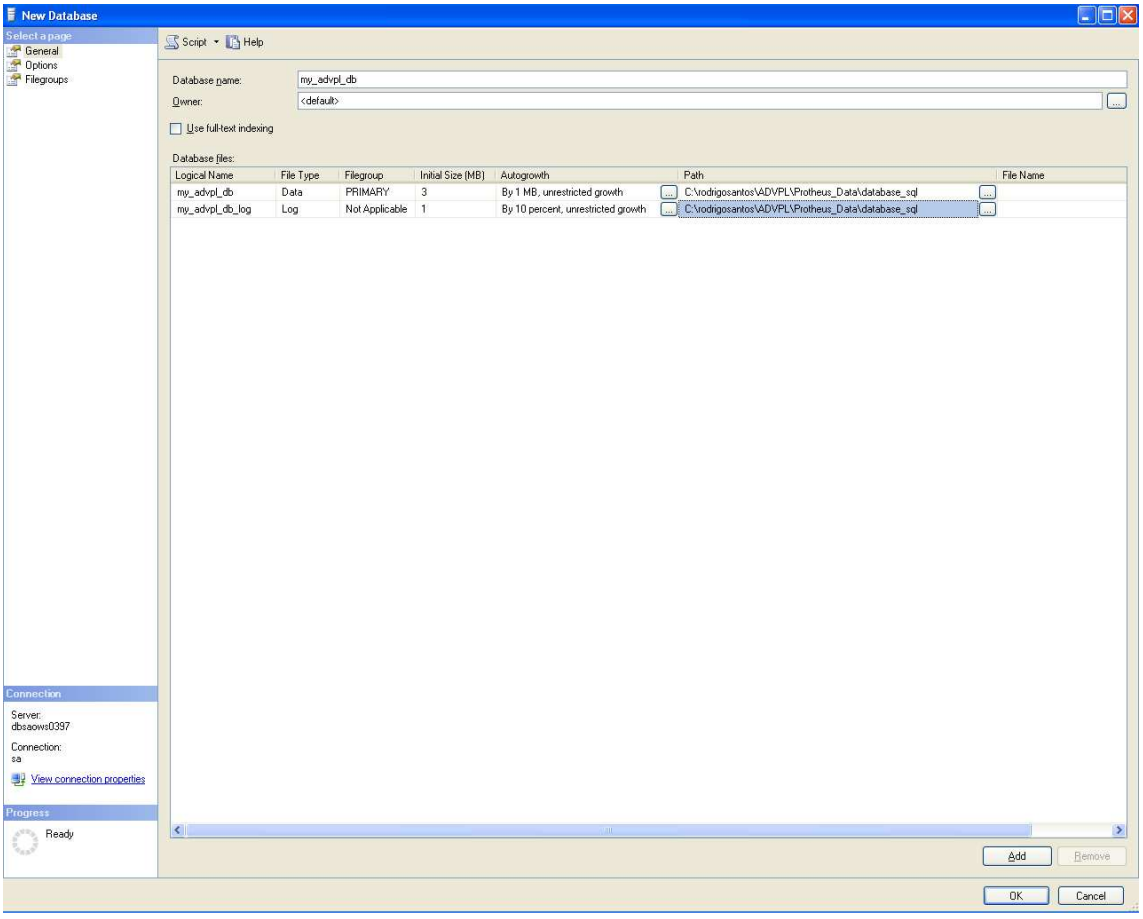
Todos os dados de um sistema relacional são armazenados e exibidos em tabelas. Programas de planilhas e sistemas de banco de dados não relacionais também usam tabelas, portanto não é uma característica exclusiva dos sistemas relacionais. Mas há algo que distingue a forma como os sistemas relacionais usam tabelas. Esta distinção deriva-se da definição e da utilização do banco de dados segundo certos princípios teóricos da abordagem relacional. Mais adiante iremos abordar tais princípios.

3.1.Criando DATABASE (SQL Server Management Studio)

Aqui descreve-se como criar um banco de dados usando o SQL Server Management Studio.

Para criar um banco de dados

1. No **Pesquisador de Objetos**, conecte-se a uma instância do Mecanismo de Banco de Dados do SQL Server e expanda essa instância.
2. Clique com o botão direito do mouse em **Bancos de Dados**, depois clique em **Novo Banco de Dados**.
3. Em **Novo Banco de Dados**, digite um nome de banco de dados.
4. Para criar o banco de dados aceitando todos os valores padrão, clique em **OK**; do contrário, passe para as etapas opcionais a seguir.



5. Para alterar o nome do proprietário, clique em (**...**) para selecionar outro proprietário.

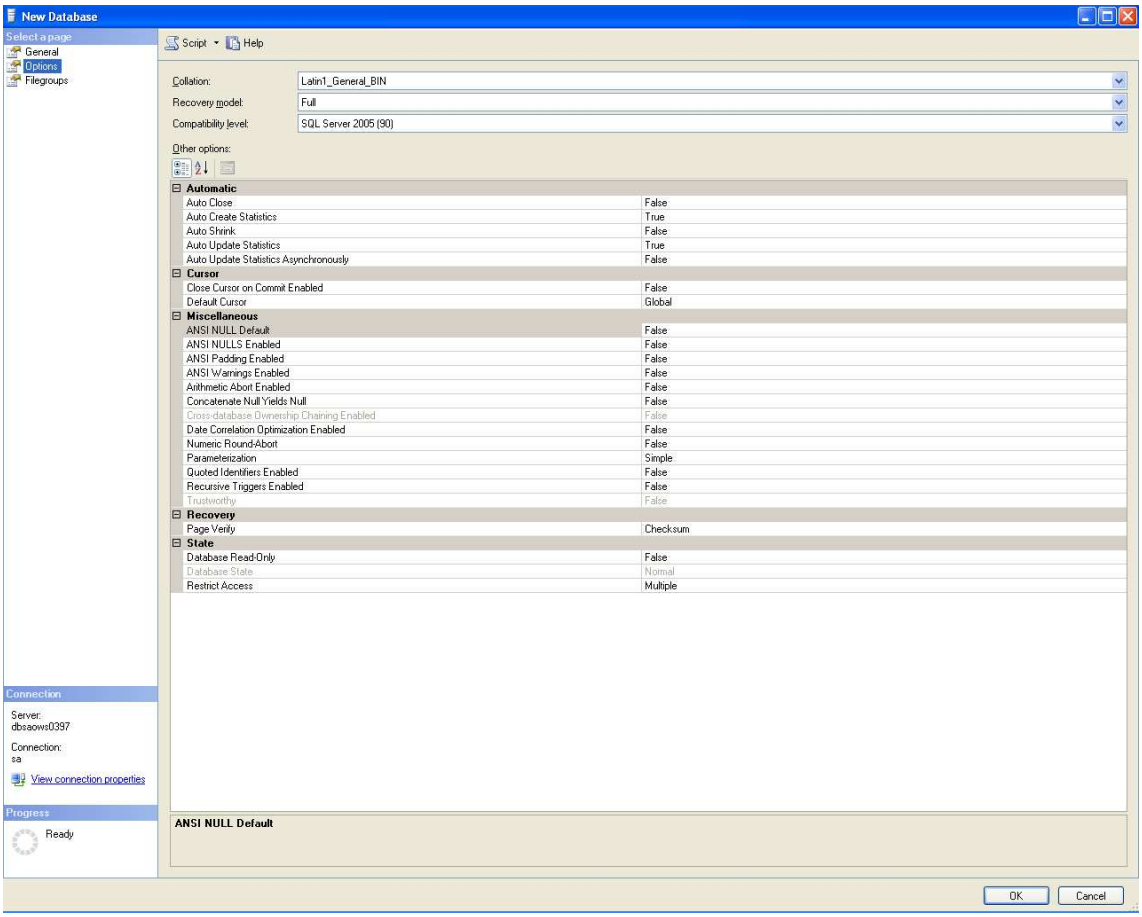
6. Para alterar os valores padrão dos arquivos de log de dados primários e de transação, na grade **Arquivos de Banco de Dados** clique na célula apropriada e digite o novo valor.

> **Dica:**
>
> Para organizar de uma maneira lógica, coloque os seus arquivos de dados dentro da pasta do protheus_data. Se preferir crie uma pasta, por exemplo: database_sql ou somente dentro da pasta **data.** (Vide figura acima)

7. Para alterar o agrupamento do banco de dados, selecione a página **Opções** e depois marque um agrupamento na lista.

8. Para alterar o modelo de recuperação, selecione a página **Opções** e marque um modelo de recuperação na lista.

9. Para alterar opções de banco de dados, selecione a página **Opções** e depois modifique as opções de banco de dados.



10. Para adicionar um novo grupo de arquivos, clique na página **Grupos de Arquivos**. Clique em **Adicionar** e, em seguida, digite os valores para o grupo de arquivos.

11. Para adicionar uma propriedade estendida ao banco de dados, selecione a página **Propriedades Estendidas**.

    a. Na coluna **Nome**, digite um nome para a propriedade estendida.

    b. Na coluna **Valor**, digite o texto da propriedade estendida. Por exemplo, uma ou mais instruções que descrevem o banco de dados.

12. Para criar o banco de dados, clique em **OK**.

Dica:

```
select * from sys.databases  -- database existentes
select * from sys.database_files -- arquivos do database(mdf e log)
select * from sys.data_spaces -- espaços de dados
select * from sys.filegroups -- exibe o grupo de arquivos
```

3.2.Attach Database

Supondo que deseje acoplar um determinado database externo, utilize-se do seguinte comando.

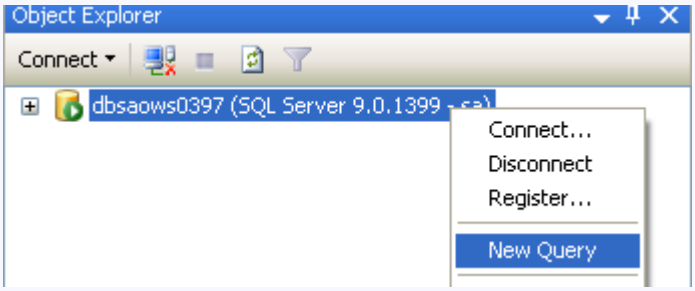
Exemplo:

```
sp_attach_db aula,"c:\rodrigosantos\ADVPL\attach\aula.mdf"
go
```

Dica:

Geral: Esta dica vale tanto para o attach quanto para deattach

1 - Abra uma conexão distinta no servidor



2 - Após o comando acima, dê um refresh no seu database. 

3.3.Deattach Database

Supondo que deseje acoplar um determinado database externo, utilize-se do seguinte comando.

Exemplo:

```
sp_detach_db aula
go
```

Dica:

Geral: Esta dica vale tanto para o attach quanto para deattach

Não poderá estar com conexão aberta e tampouco estar dentro do database que se deseja dar um deattach.

3.4.Apagando um DATABASE

> **Dica:**
>
> TENHA SEMPRE UM BACKUP ATUALIZADO DO(S) SEU(S) DATABASE !!!

Para excluir um database você não poderá estar dentro dele ou possuir qualquer ligação a ele no momento.

Sintaxe: DROP DATABASE <database_name>

> **Cuidado:**
>
> TENHA CERTEZA que deseja REALMENTE exclui-lo.
>
> ```
> drop database my_advpl_db
> ```

4. Modelo Relacional

A arquitetura de um banco de dados relacional pode ser descrita de maneira informal ou formal. Na descrição informal estamos preocupados com aspectos práticos da utilização e usamos os termos tabela, linha e coluna. Na descrição formal estamos preocupados com a semântica formal do modelo e usamos termos como relação(tabela), tupla(linhas) e atributo(coluna).

4.1.Tabelas (ou relações, ou entidades)

Todos os dados de um banco de dados relacional (BDR) são armazenados em tabelas. Uma tabela é uma simples estrutura de linhas e colunas. Em uma tabela, cada linha contém um mesmo conjunto de colunas. Em um banco de dados podem existir uma ou centenas de tabelas, sendo que o limite pode ser imposto tanto pela ferramenta de software utilizada, quanto pelos recursos de hardware disponíveis no equipamento.

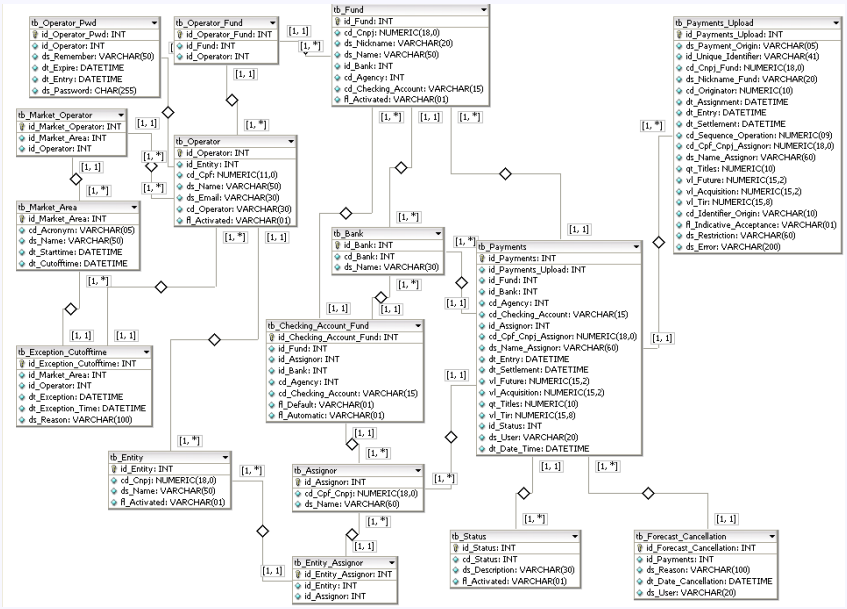
Exemplo de Tabela:

Results | Messages

	A1_COD	A1_NOME	A1_END	A1_EST	A1_MUN	R_E_C_N_O_	R_E_C_D_E_L_	D_E_L_E_T_
1	000001	BIANCA	RUA DO MERCADO, 630	SP	SAO PAULO	1	0	
2	000002	VITOR	RUA DAS COUVES, 100	PR	LONDRINA	2	0	
3	000003	PATRICK SOUZA NETO	RUA NEVES DE AZEVEDO, 455	SP	SAO PAULO	3	0	
4	000004	CRISTIANE	RUA LEONARDO DA VINCI, 608	SP	SAO PAULO	4	0	
5	000005	CARMEN LUCIA	RUA PENTEADO SILVA, 1562	SP	SAO PAULO	5	0	
6	000006	MARCOS	AV DA GLORIA, 1560	SP	SAO PAULO	6	0	
7	000007	HELIA	RUA DA ABOBORA, 10	SP	SAO PAULO	7	0	
8	000008	CATIA	RUA MACA, 1090	SP	SAO PAULO	8	0	
9	C00001	CLIENTE COMPRA BEM S.A.	RUA DO ROCIO 123	SP	SAO PAULO	9	0	
10	000009	VANESSA	RUA DA FLOR,100	SP	SAO PAULO	10	0	

As tabelas associam-se entre si através de regras de relacionamentos, estas regras consistem em associar um ou vários atributo de uma tabela com um ou vários atributos de outra tabela.

Exemplo de MER (Modelo de Entidade e Relacionamento):



Exemplo:

A tabela de cliente SA1990 relaciona-se com a Pedido de Venda SC5990. Através deste relacionamento esta última tabela fornece a lista de pedidos efetuados pelo cliente.

Modelo teórico usado para representar conceitualmente um BD, Idealizado por Codd (1970). Baseado numa estrutura de dados simples chamada relação. É o modelo mais amplamente usado, principalmente em apliações convencionais de BD.

4.2.Registros (ou tuplas)

Cada linha formada por uma lista ordenada de colunas representa um **registro**, ou **tupla**. Os registros não precisam conter informações em todas as colunas, podendo assumir valores nulos quando assim se fizer necessário.

Resumidamente, um registro é uma instância de uma tabela, ou entidade.

Exemplo:

O cliente CRISTIANE é uma instância (registro) da tabela de Cliente(SA1990).

4.3.Colunas (ou atributos)

As colunas de uma tabela são também chamadas de Atributos. Ao conjunto de valores que um atributo pode assumir chama-se domínio. Por exemplo: em um campo do tipo numérico, serão somente armazenados números.

Exemplo:

Na tabela exemplo SA1990 temos as Colunas ou Atributos A1_COD, A1_NOME, A1_END, assim por diante. A função das colunas verticais de uma tabela: conter informações sobre os atributos das entidades a que se refere a tabela.

Cada linha horizontal da tabela SA1990 contém as informações sobre todos os atributos referentes a um determinado Cliente. Portanto, enquanto a coluna A1_NOME exibe os nomes de todos os clientes da tabela e a coluna A1_END o sexo de todos os clientes e assim por diante em relação às outras colunas, a linha em que aparece o nome CRISTIANE contém informações apenas referentes aos atributos do cliente chamado CRISTIANE.

Mais adiante introduziremos um novo conceito relacional: o de chave primária.

4.4.Entidades e Chave Primária

No mundo real, você teria inúmeras razões para querer tratar cada CLIENTE individualmente: se você não considerá-los separadamente, não poderá designar os pedidos adequadamente, elaborar as faturas, notas, pedidos corretamente, e assim por diante. Pelas mesmas razões, você terá que ter os clientes também individualizados no banco de dados. Em termos práticos, isto significa que as linhas da tabela devem ser diferenciadas. Se você não puder diferenciar a linha de CRISTIANE de CARMEN LUCIA, o banco de dados não lhe dará condições para designar os pedidos, compras, etc adequadamente.

Para que uma linha possa se distinguir das outras, tem que ser de alguma forma diferente, ou seja, tem que ter uma característica que a identifique. Em um sistema relacional, esta característica identificadora não pode ser identificador externo, como a posição que a linha ocupa em relação às outras; tem que ser um dos próprios componentes da linha. Como a linha só consiste de itens de dados, temos que tentar localizar dentre os dados da linha aquele que poderá identificá-la univocamente.

Em outras palavras, temos que procurar por uma coluna (ou grupo de colunas) que apresente um conteúdo diferente em cada linha — dados que são duplicados em duas linhas. Esta característica então servirá para identificar a linha da mesma forma que usamos um nome para identificar uma pessoa.

Volte à tabela SA1990. Observe que nem todos os atributos, ou colunas, são igualmente suficientes para identificar as linhas. Por exemplo, não basta saber apenas o municipio de um cliente que você queira identificar se este atributo é compartilhado por metade dos seus clientes. Da mesma forma, se soubermos o estado(UF) de um cliente teremos um grupo mais reduzido mas não conseguiremos localizar um cliente em particular. O problema é que as colunas municipio e estado(UF) contêm valores duplicados. Quando duas linhas contêm o mesmo valor no atributo, então este atributo não pode ser usado para distinguir as linhas entre si.

Concluindo, a única coluna que pode servir de atributo identificador é a coluna A1_NOME como **teoricamente** não contém valores duplicados, as informações nela contidas são por si só suficientes para distinguir a linha de um hóspede na tabela.

Descrevemos, na realidade, uma diferença entre os dois tipos de colunas. O primeiro tipo está baseado em um atributo que identifique univocamente ou defina uma linha. O segundo tipo baseia-se em atributos descritivos que fornecem informações, mas não são suficientes para identificar uma linha ou entidade.

A coluna (ou grupo de colunas) baseada em um atributo identificador de uma linha é chamada de Chave ou Chave Primária. A chave de uma tabela lhe permite

identificar as linhas individualmente, definindo também as entidades às quais a tabela se refere.

Um banco de dados relacional todas as tabelas tem que ter uma chave primária que identifique cada linha.

As tabelas relacionam-se umas as outras através de chaves. Uma chave é um conjunto de um ou mais atributos que determinam a unicidade de cada registro.

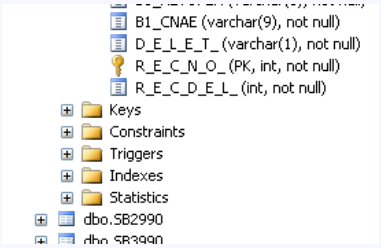
A unicidade dos registros, determinada por sua chave, também é fundamental para a criação dos índices.

Temos dois tipos de chaves:

4.4.1. Chave primária(PK - Primary Key):

É a chave que identifica cada registro dando-lhe unicidade. A chave primária nunca se repetirá. No Protheus a chave única é sempre o R_E_C_N_O_, no qual falaremos sobre este atributo adiante.

Exemplo:



Por exemplo, se um banco de dados tem como chave R_E_C_N_O_, sempre que acontecer uma inserção de dados o sistema de gerenciamento de banco de dados irá fazer uma consulta para identificar se o registro já não se encontra gravado na tabela. Neste caso, um novo registro não será criado, resultando esta operação apenas da alteração do registro existente.

4.4.2. Chave Estrangeira: (FK - Foreign Key)

É a chave formada através de um relacionamento com a chave primária de outra tabela. Define um relacionamento entre as tabelas e pode ocorrer repetidas vezes. Caso a chave primária seja composta na origem, a chave estrangeira também o será.

4.4.3. Chave Primária Composta

A chave primária pode consistir de mais de uma coluna, as vezes não podemos distinguir as linhas uma das outras usando apenas uma coluna, mais sim mais de uma. O Protheus trabalha com o R_E_C_N_O_ como chave única e primária, no entanto, não se aplica este conceito.

5. A Linguagem SQL (Structure Query Language)

A SQL (Structured Query Language) era originalmente chamada SEQUEL (Structured English QUEry Language), quando foi implementada no protótipo de pesquisa do SGBD (Sistema de Gerencia de Banco de Dados) relacional "System R", da IBM, em meados dos anos 70. Na época, como o nome fazia supor, a SQL foi projetada como uma linguagem de alto nível para banco de dados, próximo da linguagem natural, para ser usada em programas de aplicação e, também, diretamente por usuários finais em consulta ocasionais. Num esforço conjunto do ANSI (American Natural Standard Institute) e da ISO (International Standards Organization), a SQL foi adotada como padrão para bancos de dados relacionais. Em 1992, foi desenvolvido o padrão hoje em vigor, chamado de SQL-2 ou SQL-92, incorporando novas características presentes em SGBDs comerciais.

Embora se fale que a linguagem SQL é uma linguagem de consulta, essa linguagem possui outras capacidades além de realizar consultas em um banco de dados. A linguagem SQL possui recursos para definição da estrutura de dados, para modificar dados no banco de dados e recursos para especificar restrições de segurança e integridade.

SQL é uma linguagem própria para a realização de operações relacionais. Em linhas gerais, é uma linguagem para gerenciar um sistema relacional. Através das declarações SQL, dados são recuperados, atualizados ou eliminados, colunas são alteradas, tabelas são criadas e eliminadas, e qualquer outras modificações são efetuadas na estrutura de um banco de dados. As declarações em SQL podem ser subdivididas em quatro categorias: definição de dados, manipulação de dados, controle de dados e controle de transação.

5.1.DDL (Data Definition Language)

A SQL DDL(**Linguagem de Definições de Dados)** fornece comandos para definição e modificação de esquemas de relação, remoção de relações e criação de índices. Os principais comandos que fazem parte da DDL são: CREATE, ALTER, DROP.

5.2.DML (Data Manipulation Language)

A SQL DML(Linguagem de Manipulação de Dados) inclui uma linguagem de consulta baseada em álgebra relacional e também comandos para inserir, remover e modificar informações em um banco de dados. Os comandos básicos da DML são: SELECT, INSERT, UPDATE, DELETE.

5.3.DCL (Data Control Language)

DCL(Linguagem de Controle de Dados) É o conjunto de comandos que fazem o cadastramento de usuários e determina seu nível de privilégio para os objetos do banco de dados. Os principais comandos são: GRANT, REVOKE.

5.4.Transactions Control

A SQL inclui comandos para especificação do início e fim das transações. Diversas implementações permitem o trancamento explícito de dados para o controle de concorrência. (COMMIT, ROLLBACK, SAVEPOINT).

5.5.Tipos de Campos

Abaixo segue os tipos de campos disponíveis no SQL e suas variações e tamanho:

5.5.1. Numérico Exato

Inteiros

bigint

Inteiros entre -2^63 (-9223372036854775808) e 2^63-1 (9223372036854775807).

Utiliza 8 bytes.

int

Inteiros entre -2^31 (-2.147.483.648) e 2^31 - 1 (2.147.483.647).

Utiliza 4 bytes.

smallint

Inteiros entre -2^15 (-32.768) e 2^15 - 1 (32.767).

Utiliza 2 bytes.

tinyint

Inteiros entre 0 e 255.

Utiliza 1 bytes.

5.5.2. Bit

bit

Inteiro com valores 1 ou 0.

Se a tabela tiver de 1 até 8 colunas do tipo bit serão guardas em 1 byte, de 9 até 16 em 2 byte e assim por diante.

5.5.3. Decimais

decimal(p,e) / numeric(p,e)

Numérico com precisão fixa entre -10^38 +1 e 10^38 –1. Onde "p" é a precisão e "e" é a escala, ou seja, quantos dígitos da direita para esquerda serão decimais.

A quantidade de bytes depende da Precisão.

Precisão	Tamanho (bytes)
1 – 9	5
10 – 19	9
20 – 28	13
29 – 38	17

5.5.4. Monetário

money

Valor monetário entre -2^63 (-922.337.203.685.477,5808) e 2^63 - 1 (+922.337.203.685.477,5807), com 4 casas decimais.

Utiliza 8 bytes.

smallmoney

Valor monetário entre –214.748,3648 e +214.748,3647, com 4 casas decimais.

Utiliza 4 bytes.

5.5.5. Numérico Aproximado

float(n)

Número com precisão flutuante entre –1,79E + 308 e 1,79E + 308.

Onde "n" é numero de bits que será utilizado para guardar a mantissa do número flutuante na notificação cientifica e assim indica a precisão e o tamanho.

n	Precisão	Tamanho (bytes)
1 – 24	7 dígitos	4
24 – 53	15 dígitos	8

real

Número com precisão flutuante entre –3,40E + 38 e 3,40E + 38. É sinônimo do float(24)

5.5.6. Data e Hora

datetime

Data e Hora entre 1 de Janeiro de 1753 e 31 de Dezembro de 9999, com precisão de 3.33 milissegundos.

Utiliza 8 bytes, os primeiros 4 bytes para guardar um numero inteiro de dias antes ou depois da data base 1 de Janeiro 1900 e os outros 4 bytes representa o tempo no dia indicando quantos milissegundos depois da meia-noite.

smalldatetime

Data e Hora entre 1 de Janeiro de 1900 e 6 de Junho de 2079, com precisão de 1 minuto.

Utiliza 4 bytes, os primeiros 2 bytes para guardar um numero inteiro de dias depois da data base 1 de Janeiro 1900 e os outros 2 bytes representa o tempo no dia indicando quantos segundos depois da meia-noite.

5.5.7. Seqüência de caracteres

char(n) / nchar(n)

Tamanho fixo de caracteres com o máximo de 8000 caracteres ou 4000 para Unicode (nchar).

O tamanho do campo char é de 1 byte para cada n e nchar 2 bytes para cada n.

varchar(n) / nvarchar(n)

Tamanho variável de caracteres com o máximo de 8000 caracteres ou 4000 para Unicode (nvarchar).

O tamanho do campo varchar é de 1 byte para cada n e nvarchar 2 bytes para cada n. Porém o tamanho varia, conforme o tamanho do dado armazenado.

text / ntext

Tamanho variável de caracteres com o máximo de 2^31 - 1 (2.147.483.647) caracteres ou 2^30 - 1 (1,073,741,823) para Unicode(ntext).

O tamanho pode ser de até 2.147.483.647 bytes depende dos dados armazenado.

5.5.8. Seqüência Binária

binary

Tamanho fixo com o máximo de 8000 bytes.

varbinary

Tamanho variável com o máximo de 8000 bytes.

image

Tamanho variável de dados binários de até 2^31 - 1 (2.147.483.647) bytes.

5.5.9. Outros tipos de dados

cursor
Referencia um cursor.

sql_variant
Um tipo de dados que armazena valores de vários tipos suportados pelo Servidor SQL, menos text, ntext, timestamp, e sql_variant. Tamanho de até 8016 bytes.

table
É um tipo de dado especial que guarda uma tabela em memória para um processo posterior.

timestamp
É um tipo de dados automaticamente gerados que mostra números binários que são garantidos não terem iguais dentro de um banco de dados. É tipicamente usado como um mecanismo para estampar versão em linhas de uma tabela.

Utiliza 8 bytes.

uniqueidentifier

É um tipo de dado que armazena valores binários de 16 byte que operam como identificadores únicos globais (GUIDs). UM GUID é um número binário sem igual nenhum outro computador no mundo gerará uma duplicata daquele valor de GUID. O uso principal para um GUID é para nomear um identificador único dentro de uma rede onde existem vários computadores.

5.6.Criando uma Tabela

Usamos a declaração **CREATE TABLE** para criar uma tabela.

```
CREATE TABLE
   [ database_name.[owner].|owner.] table_name
    ( { [    column_definition
          | column_name AS computed_column_expression ]
        [ table_constraint ] }
)

column_definition

column_name data_type  [[DEFAULT Constant_expression]
                    |[IDENTITY[(seed,increment)]]
                   ][ROWGUIDCOL][column_constraint][...n]

column_constraint

CONSTRAINT constraint_name
   {   [NULL|NOT NULL]
     | [{PRIMARY KEY|UNIQUE}[CLUSTERED|NONCLUSTERED]]
     | [[FOREIGN KEY] REFERENCES ref_table[(ref_column)]
        [ON DELETE{CASCADE|NO ACTION}]
        [ON UPDATE{CASCADE|NO ACTION}]
     | CHECK( logical_expression )
   }

table_constraint

CONSTRAINT constraint_name
   { [{PRIMARY KEY|UNIQUE}[CLUSTERED|NONCLUSTERED]
      {(column[ASC|DESC][ ,...n ] )}
     ]
     |FOREIGN KEY[(column[,...n])]
      REFERENCES ref_table [(ref_column[,...n])]
      [ON DELETE {CASCADE|NO ACTION}]
      [ON UPDATE {CASCADE|NO ACTION}]
     |CHECK ( search_conditions )
   }
```

Argumentos

database_name

Nome do Database que será criada a tabela, se for omitido esse parâmetro será criado no Database corrente.

Owner

É o nome do usuário que será dono da tabela, se omitido o próprio dbo será o dono.

table_name

É o nome da nova tabela. O nome da tabela poderá ter 128 caracteres, exceto tabelas temporárias, aquelas que tem o sinal # no prefixo do nome, que aceitam no máximo 116 caracteres.

column_name

É o nome da coluna na tabela. Devem ser únicos por tabela.

computed_column_expression

É a expressão que define o valor de uma coluna calculada. A coluna calculada é virtual, não é gravada na base de dados. É calculada usando outras colunas da mesma tabela, não são permitidas subquerys.

data_type

Especifica o tipo de dado da coluna.

DEFAULT

Especifica o valor padrão da coluna quando a mesma não estiver sendo referenciada em um INSERT.

constant_expression

E uma constante, null ou função de sistema que será usado como valor DEFAULT.

IDENTITY

Indica que a nova coluna é uma coluna identity. Quando uma nova linha é adicionada à tabela, o SGBD provê um único e incremental valor para a coluna. Somente uma coluna identity pode ser criada por tabela.

seed

É o primeiro valor usado ao carregar a tabela.

increment

É o valor que será usado para incrementar a coluna identity.

ROWGUIDCOL

Indica que a coluna será um identificador único global. Somente uma coluna ROWGUIDCOL poderá ser criada por tabela. A propriedade ROWGUIDCOL não

obriga que os valores sejam únicos na coluna. Também não gera automaticamente valores nas novas linhas inseridas na tabela. Para gerar valores únicos utilize a função NEWID no INSERT ou coloque a função no DEFAULT da coluna.

collation_name

Especifica a collation da coluna. O collation_name é aplicável somente em colunas tipo char, varchar, text, nchar, nvarchar e ntext. Se não for especificado será usado a collation padrão do database.

CONSTRAINT

É uma palavra chave utilizada para indicar o começo de uma definição de PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY ou CHECK. Constraints são propriedades especiais para garantir a integridade dos dados e elas podem criar índices nas tabelas.

constraint_name

É o nome da constraint. Esse nome são únicos dentro do database.

NULL | NOT NULL

Esta palavra chave indica se serão aceitos valore nulos ou não nas colunas. NULL não é estritamente uma constraint, porém pode ser especificado da mesma maneira que NOT NULL.

PRIMARY KEY

É uma constraint que indica que a coluna ou colunas são identificadores únicos das linhas na tabela. Somente uma PRIMARY KEY pode ser criada por tabela. Ela cria automaticamente um índice com os campos da PRIMARY KEY

UNIQUE

É uma constraint que garante que a coluna ou colunas terá um valor único na tabela impedindo duplicidade. A tabela poderá ter várias constraints UNIQUE. Ela cria automaticamente um índice com os campos da UNIQUE

CLUSTERED | NONCLUSTERED

São palavras chaves que indica se os índices da PRIMARY KEY ou UNIQUE serão criados com clustered ou nonclustered. O padrão da PRIMARY KEY é CLUSTERED e da UNIQUE é NONCLUSTERED. Só pode ser criado um único índice CLUSTERED na tabela.

FOREIGN KEY...REFERENCES

É a constraint que garante a integridade dos dados de uma coluna ou colunas referenciando uma outra tabela. A constraints FOREIGN KEY requer que os valores existam nas colunas das tabela referenciada. A constraints FOREIGN KEY somente pode referenciar colunas que são constraints PRIMARY KEY e UNIQUE ou colunas referenciadas em um Índice Único.

ref_table

É o nome da tabela referenciada na constraint FOREIGN KEY.

(ref_column[,...n])

É a coluna ou lista de colunas da tabela referenciada pela constraint FOREIGN KEY.

ON DELETE {CASCADE | NO ACTION}

Especifica a ação que será tomada para a linha da tabela criada, se esta linha é referencia de um relacionamento onde a linha referenciada na tabela pai for apagada. O padrão é NO ACTION.

Se for especificado CASCADE, quando for excluída uma linha na tabela pai, as linhas das tabelas que fizerem referencia a está serão excluídas também. Se for especificado NO ACTION então o SGBD devolverá um erro para a aplicação que enviou o comando e desfazerá a operação.

ON UPDATE {CASCADE | NO ACTION}

Especifica a ação que será tomada para a linha da tabela criada, se esta linha é referencia de um relacionamento onde a linha referenciada na tabela pai for alterada. O padrão é NO ACTION.

Se for especificado CASCADE, quando for alterada uma linha na tabela pai, as linhas das tabelas que fizerem referencia a está serão alteradas também. Se for especificado NO ACTION então o SGBD devolverá um erro para a aplicação que enviou o comando e desfazerá a operação.

CHECK

É a constraint que garante a integridade de domínio, limitando os valores possíveis que podem ser incluídos ou alterados nas colunas.

logical_expression

É uma expressão lógica que retorna Verdadeiro ou Falso (TRUE / FALSE).

column

É a coluna ou lista de colunas, em parênteses, usadas na definição da constraint.

[ASC | DESC]

Especifica a ordem que as colunas da constraint serão ordenadas, o padrão é ASC.

Exemplo:

```
-- Setando database correto
use aula
go

-- Criando tabela empregados
create table tb_empregados(
 id empregado int IDENTITY(1,1) NOT NULL,
 nome         varchar(50)  not null
CONSTRAINT [PK  tb empregados] PRIMARY KEY CLUSTERED
(
        [id_empregado] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
go

-- Criando tabela de pagamentos
create table tb_pagamentos(
 id_pagamento int not null,
 id empregado int not null,
 valor decimal(10,2) default 10
)
go

-- Criando tabela de Descontos
create table tb descontos(
 id desconto int not null,
 id_empregado int not null,
 valor decimal(10,2)  not null
)
```

5.7.Clausula INTO

Existe uma outra forma de criar tabelas, através da clausula INTO no SELECT. Com esse comando podem ser criadas tabelas permanentes ou temporárias "#" com o resultado de uma query. A Sintaxe é a seguinte:

INTO table_name

Nome da nova tabela. Se colocar o sinal "#", antes do nome indica que a tabela é temporária. Tabelas temporárias são como as tabelas permanentes e tem as mesmas características, o que as difere é o fato de que a tabela temporária ela só é visível dentro da mesma instancia e quando a instancia é fechada, ela é excluída.

Exemplo:

```
-- Setando database correto
use aula
go

-- Inserindo na Temporária
select * INTO #minha_tmp from SA1990
```

```
go

-- Verificando a inserção
select * from #minha_tmp
go
```

5.8.Exclusão de Tabelas

Para excluir uma tabela é utilizado o comando DROP TABLE. Abaixo a sintaxe do comando

DROP TABLE <table_name>

> ⚠ **CUIDADO !!!**
>
> TENHA CERTEZA que deseja REALMENTE exclui-la.
>
> ```
> drop table #minha_tmp
> ```

5.9.ALTER TABLE

Modifica uma definição de tabela para alterar, adicionar ou excluir colunas e constraints ou habilitar e desabilitar constraints e triggers.

Sintaxe

```
ALTER TABLE table
  { [ ALTER COLUMN column_name {new_data_type[(precision[,scale])]
     [NULL|NOT NULL]
    |{ADD|DROP} ROWGUIDCOL }]

    | ADD {[column_definition]
          |column_name AS computed_column_expression
          }[ ,...n ]

    | [WITH CHECK|WITH NOCHECK] ADD {table_constraint}
      [ ,...n ]
    | DROP {[CONSTRAINT] constraint_name | COLUMN column}
      [ ,...n ]
    | {CHECK|NOCHECK} CONSTRAINT {ALL|constraint_name [ ,...n ]}
    | {ENABLE|DISABLE} TRIGGER {ALL|trigger_name [ ,...n ]}
  }
```

Argumentos

Os argumento de definições de campos de constraints do comando ALTER TABLE são semelhantes aos do CREATE TABLE.

Abaixo seguem os argumentos específicos do ALTER TABLE.

ALTER COLUMN

Especifica que uma determinada coluna será alterada.
As seguintes colunas não podem ser alteradas.
Coluna do tipo text, image, ntext ou timestamp.
A coluna ROWGUIDCOL da tabela.
Uma coluna calculada ou usada em uma calculada.
Usada em um índice, a menos que seja uma coluna varchar, nvarchar ou varbinary onde o tipo não será alterado, e o novo tamanho seja igual ou maior que o antigo.
Usada em uma constraint PRIMARY KEY ou [FOREIGN KEY] REFERENCES.
Usada em uma constraint CHECK ou UNIQUE, exceto quando é alterado somente o tamanho do campo que tenha tamanho variável.
Associado a um DEFAULT, exceto quando alterado tamanho, precisão e escada da coluna se o tipo da coluna não for alterado.

Algumas alterações de tipo de dados podem causar alteração no próprio dado armazenado nessa coluna

column_name

Nome da coluna que será adicionada, alterada ou excluída da tabela.

new_data_type

É o novo tipo de dado da tabela.
O critério para o novo tipo de dado são:

O tipo anterior de dado deve ser conversível para o novo tipo implicitamente.
O novo tipo não pode ser timestamp.
E a coluna alterada for do tipo identity, o novo tipo deve suportar a propriedade identity.

The current setting for SET ARITHABORT is ignored. ALTER TABLE operates as if the ARITHABORT option is ON.

[{ADD|DROP} ROWGUIDCOL]

Especifica que a propriedade ROWGUIDCOL será adicionada ou excluida da coluna especificada.

ADD
Especifica que será adicionada uma ou mais colunas na tabela

WITH CHECK | WITH NOCHECK

Especifica se os dados serão validados com a nova ou re-habilitada constraint CHECK ou FOREIGN KEY.

WITH CHECK e WITH NOCHECK não pode ser usado para constraints PRIMARY KEY e UNIQUE.

DROP { [CONSTRAINT] constraint_name | COLUMN column_name }

Especifica que uma constraint ou coluna será removida da tabela. Podem ser listadas multiplas colunas e constraints. Colunas utilizadas em Indices, constraints do tipo CHECK, FOREIGN KEY, UNIQUE ou PRIMARY KEY e assiociadas a uma definição de DEFAULT não poderão ser removidas.

{ CHECK | NOCHECK} CONSTRAINT

Especifica que uma constraint será habilitada ou desabilitada. Esta opção só pode ser usada com constraints tipo FOREIGN KEY e CHECK.

ALL
Especifica que todas as constraints serão habilitadas ou desabitadas.

{ENABLE | DISABLE} TRIGGER
Especifica que a TRIGGER referenciada será habilitada ou desabilitada.

ALL
Especifica que todas as Triggers serão habilitadas ou desabitadas.

trigger_name
Especifica o nome da Trigger que será habilitada ou desabilitada.

Cuidado:

TENHA CERTEZA que deseja REALMENTE Alterar.

Exemplo:

```
-- Setando database correto
use aula
go

-- verifica se tabela existe no database
IF OBJECT_ID ( 'dbo.tb_teste', 'U' ) IS NOT NULL
    DROP TABLE dbo.tb_teste;
GO

-- Cria uma tabela com duas colunas e um indice unico.
CREATE TABLE dbo.tb_teste ( col_a varchar(5) UNIQUE NOT NULL, col_b decimal
```

```
(4,2));
GO

INSERT INTO dbo.tb_teste VALUES ('Test', 99.99);
GO

--Verifica os dados correntes
SELECT * FROM dbo.tb teste

-- Verifica o tamanho da coluna atual
SELECT name, TYPE_NAME(system_type_id), max_length, precision, scale
FROM sys.columns WHERE object_id = OBJECT_ID(N'dbo.tb_teste');
GO

-- Aumenta o tamanho da coluna a
ALTER TABLE dbo.tb_teste ALTER COLUMN col_a varchar(25);
GO

-- Aumenta o scale and precision of the decimal column.
ALTER TABLE dbo.tb teste ALTER COLUMN col b decimal (10,4);
GO

-- Insert a new row.
INSERT INTO dbo.tb teste VALUES('AfterReSize', 99999.9999) ;
GO

-- Verifica os dados atualizados
SELECT * FROM dbo.tb_teste

ALTER TABLE dbo.tb teste ADD col c varchar(20)

-- Verifica o tamanho das colunas atuais
SELECT name, TYPE_NAME(system_type_id), max_length, precision, scale
FROM sys.columns WHERE object_id = OBJECT_ID(N'dbo.tb_teste');

-- Insert a new row.
INSERT INTO dbo.tb teste VALUES('AfterReSize1', 99999.9999, 'profrodrigo') ;
GO

-- Verifica os dados atualizados
SELECT * FROM dbo.tb teste

-- Limpando
IF OBJECT_ID ( 'dbo.tb_teste', 'U' ) IS NOT NULL
    DROP TABLE dbo.tb_teste;
GO
```

5.10. Restrição de Integridade

As restrições de integridade servem para garantir as regras inerentes ao sistema que está sendo implementado, prevenindo a entrada de informações inválidas pelos usuários desse sistema. Para isso, o Sistema de Banco de Dados deve possibilitar a definição de regras de integridade a fim de evitar a inconsistência dos dados que nele serão armazenados. Essas regras são chamadas de CONSTRAINT.

NOT NULL

Essa constraint evita que valores nulos sejam inseridos nas tabelas.

CHECK

Com esta clausula é possível limitar a entrada de dados a um domínio como exemplo podemos citar o campo "sexo" de uma tabela, esta campo só aceita M (Masculino) ou F (Feminino), portanto criando uma constraint CHECK é possível limitar os valores para esses dois.

UNIQUE

A constrait UNIQUE obrigam a singularidade dos valores em uma lista de colunas. Não é permitida duas linhas de uma tabela terem os mesmo valores não nulos com uma constraint UNIQUE.

PRIMARY KEY

Uma PRIMARY KEY identifica unicamente uma linhas na tabela atraves de uma ou mais colunas. Colunas utilizadas na PRIMARY KEY não aceita valores nulos.

FOREIGN KEY

Esta constraint é utilizada para criar relacionamento entre tabelas, como por exemplo Tabela de Cabeçalho de Pedidos e Itens de pedidos, com essa constraint não é possível incluir um item de pedido se ter o cabeçalho de pedido.

DEFAULT

Default não é uma constraint, porém também é utilizado para garantir Integridade. Valor Default é utilizado quando é inserido uma linha na tabela e o campo não é mensionado na tabela e também como inicializador padrão de um campo.

6. INDICES

Índices são utilizados pelo SGBD para otimizar a busca de dados na tabela. Quando um campo é indexado e é utilizado em uma clausula WHERE, o SGBD consegue buscar mais rápido os dados, pois não precisa varrer a tabela e sim o índice e o índice alem de estar ordenado, ele é menor em volume de dados.

Criação de Índices

CREATE INDEX

A sintaxe é a seguinte:

CREATE [UNIQUE] [CLUSTERED|NONCLUSTERED] INDEX *index_name*
ON {*table*|*view*} (*column*[ASC|DESC][,...*n*])

Argumentos

UNIQUE

Criar um índice do tipo único, ou seja, não permite que os dados das colunas do indice sejam iguais em mais de uma linha da tabela.

Se tentar criar um Índice único em uma tabela que tenha dados duplicados nas colunas, o SGBD irá gerar um erro na aplicação e será cancelado. Dois valores nulos são considerados duplicidade para Índice único.

Quando a clausula IGNORE_DUP_KEY é utilizada somente as linhas que violam o Índice Único serão desprezada nos casos de inclusão de dados na tabela.

CLUSTERED

Cria um objeto onde a ordem física das linhas é a mesma da ordem indexada das linhas, e o ultimo nível do índice contem os dados atuais das linhas.

Uma view com índice clustered é chamada de Indexed view. Antes de criar qualquer índice em uma view devesse criar um Índice Único Clustered.

NONCLUSTERED

Cria um objeto que especifica a ordem lógica de uma tabela. Com um Índice nonclustered, a ordem física das linhas é independente da ordem indexada. O ultimo nível de um Índice nonclustered contem as linhas do índice. Cada linha de índice contém um valor de chave e um ou mais endereço de linhas apontando para um valor. Se a tabela não contém um índice clustered, o

endereço da linha é o endereço da linha no disco. Se a tabela tem um índice clustered, o endereço da linha é a chave do índice clustered para a linha.

Cada tabela pode ter muitos até 249 índice nonclustered (independente da maneira que são criados: implicitamente com a constraint PRIMARY KEY e UNIQUE ou explicitamente com CREATE INDEX). Cada índice pode prover acesso a um dado em diferentes ordens.

index_name

É o nome do índice, o nome deve ser único para cada tabela, porém dentro do database pode ter índices com o mesmo nome para tabelas diferentes.

table
É a tabela onde estão as colunas que deseja indexar.

view
É o nome da view que se deseja indexar

column
É a coluna ou colunas para as quais o índice se aplica. Especifique duas ou mais colunas para criar um índice composto com valores combinados das colunas. Liste as colunas entre parêntese na ordem de prioridade de ordenação.

Colunas do tipo ntext, text ou image não podem ser especificadas em um índice.

Índices compostos são usados quando duas ou mais colunas são sempre procuradas em conjunto ou se muitas querys referenciam somente as colunas do índice. Até 16 colunas podem ser combinadas em um único índice composto. O tamanho máximo dos dados combinados das colunas que compõe o índices é de 900 bytes.

[ASC | DESC]

Determina se a ordem da coluna em particular será ascendente ou descendente. O Padrão é ASC.

6.1.Index_option

Especifica um conjunto de opções que podem ser aplicadas a um índice que faz parte de uma definição de restrição criada com ALTER TABLE.

Sintaxe

```
    {

   PAD_INDEX = { ON | OFF }
  | FILLFACTOR = fillfactor
  | IGNORE_DUP_KEY = { ON | OFF }
  | STATISTICS_NORECOMPUTE = { ON | OFF }
  | ALLOW_ROW_LOCKS = { ON | OFF }
  | ALLOW_PAGE_LOCKS = { ON | OFF }
  | SORT_IN_TEMPDB = { ON | OFF }
  | ONLINE = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
  | DATA_COMPRESSION = { NONE |ROW | PAGE}
    [ ON PARTITIONS ( { <partition_number_expression> | <range> }
        [ , ...n ] ) ]
}

<range> ::=
<partition_number_expression> TO <partition_number_expression>

<single_partition_rebuild__option> ::=
{
   SORT_IN_TEMPDB = { ON | OFF }
  | MAXDOP = max_degree_of_parallelism
  | DATA_COMPRESSION = {NONE | ROW | PAGE } }
}
```

Argumentos

PAD_INDEX = { ON | OFF }

Especifica o preenchimento do índice. O padrão é OFF.

ON

A porcentagem de espaço livre especificada por FILLFACTOR é aplicada às páginas de nível intermediário do índice.

OFF ou *fillfactor* não está especificado

As páginas de nível de intermediário são preenchidas até próximo de sua capacidade, deixando espaço suficiente para pelo menos uma linha do

tamanho máximo que o índice pode ter, dado o conjunto de chaves em páginas intermediárias.

FILLFACTOR = ***fillfactor***

Especifica uma porcentagem que indica quanto Mecanismo de Banco de Dados deve preencher o nível folha de cada página de índice durante a criação ou alteração do índice. O valor especificado deve ser um valor inteiro entre 1 e 100. O padrão é 0.

IGNORE_DUP_KEY = **{ ON | OFF }**

Especifica a resposta de erro quando uma operação de inserção tentar inserir valores de chave duplicados em um índice exclusivo. A opção IGNORE_DUP_KEY aplica-se apenas a operações de inserção depois que o índice é criado ou recriado. A opção não tem nenhum efeito ao executar CREATE INDEX, ALTER INDEX ou UPDATE. O padrão é OFF.

ON

Uma mensagem de aviso ocorrerá quando valores de chave duplicada forem inseridos em um índice exclusivo. Haverá falha somente nas linhas que violarem a restrição de exclusividade.

OFF

Ocorrerá uma mensagem de erro quando os valores da chave duplicada forem inseridos em um índice exclusivo. Toda a operação INSERT será revertida.

IGNORE_DUP_KEY não pode ser definido como ON para índices criados em uma exibição, índices não exclusivos, índices XML, índices espaciais e índices filtrados.

Para exibir IGNORE_DUP_KEY, use sys.indexes.

Na sintaxe compatível com versões anteriores, WITH IGNORE_DUP_KEY é equivalente a WITH IGNORE_DUP_KEY = ON.

STATISTICS_NORECOMPUTE = **{ ON | OFF }**

Especifica se as estatísticas são recalculadas. O padrão é OFF.

ON

Estatísticas desatualizadas não são recalculadas automaticamente.

OFF

A atualização automática de estatísticas está habilitada.

ALLOW_ROW_LOCKS = { ON | OFF }

Especifica se bloqueios de linha são permitidos. O padrão é ON.

ON

Bloqueios de linha são permitidos ao acessar o índice. O Mecanismo de Banco de Dados determina quando os bloqueios de linha são usados.

OFF

Bloqueios de linha não são usados.

ALLOW_PAGE_LOCKS = { ON | OFF }

Especifica se bloqueios de página são permitidos. O padrão é ON.

ON

Bloqueios de página são permitidos ao acessar o índice. O Mecanismo de Banco de Dados determina quando os bloqueios de página são usados.

OFF

Bloqueios de página não são usados.

SORT_IN_TEMPDB = { ON | OFF }

Especifica se os resultados de classificação devem ser armazenados em **tempdb**. O padrão é OFF.

ON

Os resultados de classificação intermediários usados para criar o índice são armazenados em **tempdb**. Isso pode reduzir o tempo necessário para criar um índice se **tempdb** estiver em um conjunto de discos diferente do banco de dados de usuário. Entretanto, isso aumenta a quantidade de espaço em disco usado durante a criação do índice.

OFF

Os resultados de classificação intermediários são armazenados no mesmo banco de dados que o índice.

ONLINE = { ON | OFF }

Especifica se as tabelas subjacentes e índices associados estão disponíveis para consultas e modificação de dados durante a operação de índice. O padrão é OFF.

ON

Bloqueios de tabela de longa duração não são mantidos pelo tempo de operação do índice. Durante a fase principal da operação de índice, apenas um bloqueio IS (Tentativa Compartilhada) é mantido na tabela de origem. Ele permite o prosseguimento de consultas ou atualizações feitas na tabela e nos índices subjacentes. No início da operação, um bloqueio Compartilhado (S) é mantido no objeto de origem por um período muito curto. Ao término da operação, por um curto período de tempo, um bloqueio compartilhado (S) será adquirido na origem se um índice não clusterizado estiver sendo criado; ou um bloqueio de modificação de esquema (SCH-M) será adquirido quando um índice clusterizado for criado ou cancelado online e quando um índice clusterizado ou não clusterizado estiver sendo recriado. Não será possível definir ONLINE como ON quando um índice estiver sendo criado em uma tabela temporária local.

OFF

Os bloqueios de tabela são aplicados enquanto durar a operação de índice. Uma operação de índice offline que cria, recria ou cancela um índice clusterizado ou recria ou cancela um índice não clusterizado, adquire um bloqueio de Modificação de esquema (Sch-M) na tabela. Isso evita o acesso de todos os usuários à tabela subjacente enquanto durar a operação. Uma operação de índice offline que cria um índice não clusterizado adquire um bloqueio compartilhado (S) na tabela. Ele impede a realização de atualizações na tabela subjacente, mas permite operações de leitura, como instruções SELECT.

6.2.Exclusão de Índices

Para excluir um índice utilize o comando abaixo:

DROP INDEX 'table.index | view.index' [,...n]

⚠ Cuidado !!!

```
USE [aula]
GO
IF  EXISTS (SELECT * FROM sys.indexes WHERE object_id =
OBJECT_ID(N'[dbo].[tb_teste]') AND name = N'UQ__tb_teste__4D0E7069')
ALTER TABLE [dbo].[tb_teste] DROP CONSTRAINT [UQ__tb_teste__4D0E7069]
```

table | view

É o nome da tabela ou da Indexed View que contem o índice que se quer excluir.

index
É o nome do indice que deseja excluir.

7. CONSULTA EM SQL

Para recuperar dados armazenados em um banco de dados, são utilizadas querys. Uma query é um comando, especificamente SELECT, que contem os campos, as tabelas e as condições de busca onde, por meio dessa, o SGBD devolva as linhas. É possível agrupar dados, sumarizar os dados retornados em uma query.

7.1.SELECT

O SELECT é o comando utilizado para recuperar os dados armazenados no banco de dados. Sua sintaxe é extensa e por isso é dividida em blocos.

7.1.1. Clausula SELECT

Especifica as colunas a ser devolvidas pela query.

Sintaxe

```
SELECT [ ALL | DISTINCT ]
      [ TOP n [ PERCENT ] [ WITH TIES ] ]
      < select_list >

select_list

{ * | { table_name | view_name | table_alias }.*
   | { column_name | expression | IDENTITYCOL | ROWGUIDCOL }
      [ [ AS ] column_alias ]
   | column_alias = expression }
[ ,...n ]
```

Argumentos

ALL

Especifica que as linhas duplicadas podem aparecer no resultado da query.

Dica:

ALL é um argumento default, portanto não precisa ser declarado.

DISTINCT

Especifica que linhas iguais só aparecem uma vez. Valores nulos são considerados iguais.

Exemplo:

```
Select Distinct(A1_NOME) From SA1990
```

TOP *n* [PERCENT]

Especifica que só as n primeiras linhas serão retornadas no resultado da query. n é um inteiro entre 0 e 4294967295. Se PERCENT for declarado, só n por cento das linhas resultantes da query serão retornadas, para isso n deverá ser um inteiro entre 0 e 100.

Exemplo:

```
Select TOP 5 A1_NOME, A1_LOJA, A1_PESSOA, A1_END, A1_MUN, A1_EST From SA1990
```

Se ORDER BY estiver especificado, a clausula TOP trará as n linhas respeitando a ordenação completa da query.

WITH TIES

Especifica que as linhas adicionais, além de n, serão retornadas se tiverem os mesmos valores da ultima linha. WITH TIES só pode ser especificado com a clausula ORDER BY.

select_list

É uma lista de colunas e expressões separada por vírgulas.

*** (ASTERISCO)**

Especifica que todas as colunas de todas as tabelas e views especificadas na clausula FROM devera ser devolvidas. As colunas são devolvidas na ordem em que elas existem nas tabelas e views.

{ table_name | view_name | table_alias }.*

Limita a abrangência do * para a tabela ou view especificada.

Exemplo:

```
Select * From SA1990
```

column_name

É o nome da coluna que deseja retornar. As colunas devem ser especificadas com as tabela ou views que as contém, porque se existir em duas tabelas colunas com o mesmo nome causará erro na query, para isso devera ser colocada o nome da tabela ou seu alias na antes do nome da coluna (tabela.coluna).

expression
É um nome de coluna, constante, função, qualquer combinação de coluna, constantes, e funções conectadas por um operador(s), ou uma subquery.

IDENTITYCOL

Retorna a coluna identity da tabela especificada na clausula FROM.

ROWGUIDCOL
Retorna a coluna com identificador global único da tabela especificada na clausula FROM.

column_alias

É um nome alternativo para substituir o nome da coluna no resultado da query.

Podem ser usados column_alias na cláusula ORDER BY. Porém, não pode ser usado em um WHERE, GROUP BY, ou HAVING.

7.1.2. Cláusula INTO

Cria uma nova tabela e inclui as linhas resultantes da query.

Sintaxe

[INTO new_table]

Exemplo:

```
use aula
go
Select * INTO #my_table_tmp FROM SA1990 --Inserindo na Tabela #my_table_tmp
go
Select * from #my table tmp
go
```

Argumentos

new_table

Especifica o nome de uma nova tabela, baseado nas colunas informadas na query. O formato da nova tabela é determinado avaliando as expressões na lista do SELECT.

As colunas são criadas na ordem especificada no SELECT. Cada coluna na nova tabela tem mesmo nome, e conteúdo conforme o resultado da query, as colunas resultantes de uma expressão computada não serão campos calculados e sim conterão o resultado da formula.

7.1.3. Cláusula FROM

Especifica a(s) tabela(s) de qual serão retornadas as linhas. A cláusula FROM é obrigatória a menos que o SELECT contenha somente constantes, variáveis, e expressões aritméticas. (nenhuma coluna).

Sintaxe

```
[ FROM { < table_source > } [ ,...n ] ]

table_source

  table_name [[AS] table_alias] [WITH(<table_hint>[,...n ])]
  | view_name [[AS] table_alias]
  | rowset_function [[AS] table_alias]
  | OPENXML
  | derived_table [AS] table_alias [(column_alias[,...n ])]
  | <joined_table>

joined_table

  <table_source> <join_type> <table_source> ON <search_condition>
  | <table_source> CROSS JOIN <table_source>
  | <joined_table>

join_type

[INNER|{{LEFT|RIGHT|FULL}[OUTER]}] JOIN
```

Argumentos

<table_source>
Especifica tabelas, views, tabelas derivadas(subqueries), e unir tabelas para a declaração SELECT.

table_name [[AS] table_alias]
Especifica o nome de uma tabela e um pseudônimo (alias) opcional.

view_name [[AS] table_alias]

Especifica o nome de uma view, e um pseudônimo opcional.

rowset_function [[AS] table_alias]

É o nome de uma função que retorna tabela e um pseudônimo opcional.

WITH (< table_hint > [,...n])
Especifica um ou mais argumentos de tabela.

derived_table [[AS] table_alias]

É uma subquery que retorna suas linhas como se fosse um tabela através de outro SELECT.

column_alias
É um pseudônimo opcional para substituir o nome das colunas da subquery.

<joined_table>
É o resultado de da junção de duas ou mais tabelas

<join_type>
Especifica o tipo de operação de união.

INNER

Especifica que serão retornadas todas as linhas das tabelas utilizadas na junção, descartando as linhas que não correspondem a condição da clausula ON.

Essa clausula é padrão se não for especificada nenhuma.

LEFT [OUTER]

Especifica que todas as linhas da tabela da esquerda serão retornadas e somente as linhas da tabela da direita que satisfazem a condição da clausula ON serão retornadas, as linhas que não existirem na tabela da direita serão retornada com o valor NULL em seus campos.

Exemplo:

```
use aula
go
Select A1.A1 COD, A1.A1 LOJA, A1.A1 NREDUZ, C5.C5 FILIAL, C5.C5 NUM
from SC5990 C5
Left Join SA1990 A1 ON A1.A1 COD = C5.C5 CLIENTE
-- Retornara somente os itens que possuirem Pedidos de Venda(SC5)—
Retorno:
```

Results | Messages

	A1_COD	A1_LOJA	A1_NRED...	C5_FILIAL	C5_NUM
1	000001	01	BIANCA	01	000005
2	000002	01	VITOR	01	000006
3	000003	01	PATRICK	01	000003
4	000004	01	CRIS	01	000019
5	000006	01	MARCOS	01	000004

RIGHT

Especifica que todas as linhas da tabela da direita serão retornadas e somente as linhas da tabela da esquerda que satisfazem a condição da clausula ON serão retornadas, as linhas que não existirem na tabela da esquerda serão retornada com o valor NULL em seus campos.

Exemplo:

```
use aula
go
Select A1.A1_COD, A1.A1_LOJA, A1.A1_NREDUZ, C5.C5_FILIAL, C5.C5_NUM
from SC5990 C5
Right Join SA1990 A1 ON A1.A1 COD = C5.C5 CLIENTE
-- Neste caso, retornara TODOS os CLIENTES indepente de possuirem Pedidos de
Venda(SC5)
```

Retorno:

Results | Messages

	A1_COD	A1_LOJA	A1_NREDUZ	C5_FILIAL	C5_NUM
1	000001	01	BIANCA	01	000005
2	000002	01	VITOR	01	000006
3	000003	01	PATRICK	01	000003
4	000004	01	CRIS	01	000019
5	000005	01	CARMEN	NULL	NULL
6	000006	01	MARCOS	01	000004
7	000007	01	HELIA	NULL	NULL
8	000008	01	CATIA	NULL	NULL
9	C00001	01	CLIENTE COMPRA	NULL	NULL
10	000009	01	VANESSA	NULL	NULL

FULL [OUTER]

Especifica que todas as linhas de ambas tabelas serão retornadas emparelhadas conforme a condição do ON e as que não satisfazem essa condição serão retornadas emparelhadas a valores NULL

JOIN (INNER JOIN)

Indica que devera ser se juntadas às tabelas ou views especificadas.

Exemplo:

```
use aula
go
Select A1.A1_COD, A1.A1_LOJA, A1.A1_NREDUZ, C5.C5_FILIAL, C5.C5_NUM
from SC5990 C5
Join SA1990 A1 ON A1.A1 COD = C5.C5 CLIENTE
-- Retornara somente os itens que possuirem Pedidos de Venda(SC5)
```

Retorno:

Results | Messages

	A1_COD	A1_LOJA	A1_NREDUZ	C5_FILIAL	C5_NUM
1	000001	01	BIANCA	01	000005
2	000002	01	VITOR	01	000006
3	000003	01	PATRICK	01	000003
4	000004	01	CRIS	01	000019
5	000005	01	CARMEN	NULL	NULL
6	000006	01	MARCOS	01	000004
7	000007	01	HELIA	NULL	NULL
8	000008	01	CATIA	NULL	NULL
9	C00001	01	CLIENTE COMPRA	NULL	NULL
10	000009	01	VANESSA	NULL	NULL

ON < search_condition >

Especifica a condição no qual a união é baseada. A condição pode ser qualquer expressão que retorne um valor lógico. Porém na maioria das vezes é utilizado expressões que comparem conteúdos de campos de ambas tabelas, podendo utilizar clausula AND caso hajam mais de um campo em comum entre as tabelas.

Exemplo:

```
Select C5.C5_NUM, A1.A1_NOME, A1.A1_COD, B1.B1_DESC, C5.C5_EMISSAO
From SC6990 C6
Join SC5990 C5 ON C5.C5_NUM     = C6.C6_NUM AND C5.C5_FILIAL = C6.C6_FILIAL
Join SA1990 A1 ON C5.C5_CLIENTE = A1.A1_COD
Join SB1990 B1 ON C6.C6_PRODUTO = B1.B1_COD AND B1.B1_FILIAL = C6.C6_FILIAL
```

CROSS JOIN

Especifica o cruzamento de linhas de duas tabelas. O resultado da Query será todas as linhas da primeira tabela emparelhada com cada linha da segunda tabela a quantidade de linhas retornada será o produto da quantidade que existem em ambas as tabelas.

Exemplo:

```
use aula
go
Select A1.A1_COD, A1.A1_LOJA, A1.A1_NREDUZ, C5.C5_FILIAL, C5.C5_NUM
from SC5990 C5
Cross Join SA1990 A1
```

7.1.4. Exercicio Complementar Left, Right e Join

Exemplo:

```
use aula
go
Select A1.A1_COD, A1.A1_LOJA, A1.A1_NREDUZ, C5.C5_FILIAL, C5.C5_NUM
from SC5990 C5
Cross Join SA1990 A1
```

7.1.5. Cláusula WHERE

Especifica uma condição de procura para restringir as linhas devolvidas.

Sintaxe

[WHERE < search_condition >]

Argumentos

<search_condition>

Restringe as linhas devolvidas no resultado através de expressões lógicas. Não há nenhum limite de expressões que podem ser incluídos em uma condição de procura.

Exemplo:

```
Use aula
go
Select C5.C5 NUM, A1.A1 NOME, A1.A1 COD, B1.B1 DESC, C5.C5 EMISSAO
From SC6990 C6
Join SC5990 C5 ON C5.C5 NUM     = C6.C6 NUM AND C5.C5 FILIAL = C6.C6 FILIAL
Join SA1990 A1 ON C5.C5_CLIENTE = A1.A1_COD
Join SB1990 B1 ON C6.C6_PRODUTO = B1.B1_COD AND B1.B1_FILIAL = C6.C6_FILIAL
Where C6.D_E_L_E_T_ = ''
AND   C6.C6 FILIAL  = 01
And   C5.C5_NUM     IN ('000005','000019')
```

7.1.6. Cláusula GROUP BY

Especifica os grupos nos quais as linhas serão colocadas e, se são incluídas funções de totalização na cláusula SELECT, calcula um valor sumário por cada grupo. Quando GROUP BY é especificado, qualquer coluna em qualquer expressão que não seja de totalização que estiverem no SELECT deverá ser incluída na lista do GROUP BY.

Sintaxe

[GROUP BY [ALL] group_by_expression [,...n]
[WITH{CUBE|ROLLUP}]]

Argumentos

ALL
Inclui todos os grupos no resultado, até mesmo os que não tem linhas que satisfaçam a condição do WHERE. Quando ALL é especificado, são devolvidos valores NULL nas colunas sumárias de grupos que não satisfaçam a condição de procura. Você não pode especificar ALL com o CUBE ou operadores de ROLLUP.

group_by_expression

É uma expressão na qual o agrupamento é executado. Ela conterá os campos pelo qual serão montados os grupos, poderá conter colunas ou expressões não totalizadora que se referem a colunas. Alias de campos não podem ser colocados nessa clausula.

Exemplo:

```
Use aula
go

SELECT SUM(D2_QUANT) QTDE, D2_UM, D2_ITEM
FROM SD2990
GROUP BY D2_UM, D2_ITEM
```

CUBE

Com essa clausula especificada além das linhas habituais trazidas pelo GROUP BY, são introduzidas linhas sumárias no resultado. Com isso além da totalização por grupo também é mostrado um total por subgrupos e um total geral.

Exemplo:

```
Use aula
Go

-- CUBE TODOS OS TOTAIS POSSIVEIS
SELECT E1 CLIENTE, E1 LOJA, SUM(E1 VALOR) VAL, SUM(E1 SALDO) SLD
FROM   SE1990 SE1
WHERE SE1.D_E_L_E_T_ = ''
GROUP BY E1_CLIENTE, E1_LOJA WITH CUBE
```

Results | Messages

	E1_CLIENTE	E1_LOJA	VAL	SLD
1	000001	01	2232	1074
2	000001	NULL	2232	1074
3	000002	01	1615,16	0
4	000002	NULL	1615,16	0
5	000003	01	2033,65	1010
6	000003	NULL	2033,65	1010
7	000004	01	1490	0
8	000004	NULL	1490	0
9	000005	01	1890	1890
10	000005	NULL	1890	1890
11	000006	01	2457,33	2353,33
12	000006	NULL	2457,33	2353,33
13	000009	01	1500	1500
14	000009	NULL	1500	1500
15	C00001	01	1343,38	1343,38
16	C00001	NULL	1343,38	1343,38
17	NULL	NULL	14561,52	9170,71
18	NULL	01	14561,52	9170,71

Atenção:

Como ele retorna os valores totalizados, ou seja, mais de um recordset então este recurso até o momento não é suportado pelo Protheus.

ROLLUP

Com essa clausula especificada além das linhas habituais trazidas pelo GROUP BY, são introduzidas linhas sumárias no resultado. Porém serão totalizadas os grupos e subgrupos na ordem que aparecem as colunas no GROUP BY.

Exemplo:

```
Use aula
Go
-- ROLLUP - TOTALIZA DA EQ. PARA A DIREITA
SELECT E1_CLIENTE, E1_LOJA, E1_TIPO, SUM(E1_VALOR) VAL, SUM(E1_SALDO) SLD
FROM   SE1990 SE1
WHERE SE1.D E L E T  = ''
GROUP BY E1 CLIENTE, E1 LOJA, E1 TIPO WITH ROLLUP --TOTAL GERAL
```

Results | Messages

	E1_CLIENTE	E1_LOJA	E1_TIPO	VAL	SLD
1	000001	01	NF	2232	1074
2	000001	01	NULL	2232	1074
3	000001	NULL	NULL	2232	1074
4	000002	01	NF	1615,16	0
5	000002	01	NULL	1615,16	0
6	000002	NULL	NULL	1615,16	0
7	000003	01	IR-	10	10
8	000003	01	NF	2023,65	1000
9	000003	01	NULL	2033,65	1010
10	000003	NULL	NULL	2033,65	1010
11	000004	01	NF	1490	0
12	000004	01	NULL	1490	0
13	000004	NULL	NULL	1490	0
14	000005	01	NF	1890	1890
15	000005	01	NULL	1890	1890
16	000005	NULL	NULL	1890	1890
17	000006	01	DP	1353,33	1353,33
18	000006	01	NF	1104	1000
19	000006	01	NULL	2457,33	2353,33
20	000006	NULL	NULL	2457,33	2353,33
21	000009	01	NF	1500	1500
22	000009	01	NULL	1500	1500
23	000009	NULL	NULL	1500	1500
24	C00001	01	DP	1343,38	1343,38
25	C00001	01	NULL	1343,38	1343,38
26	C00001	NULL	NULL	1343,38	1343,38
27	NULL	NULL	NULL	14561,52	9170,71

Atenção:

Como ele retorna os valores totalizados, ou seja, mais de um recordset então este recurso até o momento não é suportado pelo Protheus.

7.1.7. Cláusula HAVING

Especifica uma condição de busca para um grupo ou um totalizador. HAVING é normalmente usado com a cláusula GROUP BY. Quando GROUP BY não é usado o HAVING se comporta como a cláusula WHERE.

Sintaxe

[HAVING < search_condition >]

Argumentos

<search_condition>

Especifica a condição de busca para o grupo ou o totalizador se encontrar. Quando HAVING é usado com GROUP BY ALL, a cláusula HAVING anula ALL.

Exemplo:

```
use aula
go

--HAVING RECEBE TODAS AS LINHAS DEPOIS FILTRA
SELECT E1 CLIENTE, SUM(E1 VALOR) VAL, SUM(E1 SALDO) SLD
FROM   SE1990 SE1
WHERE SE1.D E L E T  = ''
GROUP BY E1_CLIENTE
HAVING SUM(E1_VALOR) > 2000
```

Results | Messages

	E1_CLIENTE	VAL	SLD
1	000001	2232	1074
2	000003	2033,65	1010
3	000006	2457,33	2353,33

Atenção:

Como ele retorna os valores totalizados, ou seja, mais de um recordset então este recurso até o momento não é suportado pelo Protheus.

Operador UNION

Associa os resultados de duas ou mais querys em um único resultado que consiste em todas as linhas que pertencem a todas as querys. É diferente de usar JOIN que combina as colunas de duas ou mais tabelas.

Para utilizar o operador UNION o número e a ordem das colunas devem ser idênticas em todas as querys e os tipos de dados devem ser compatíveis.

Sintaxe

```
{<query specification>|(<query expression>)}
   UNION [ALL]
   <query specification|(<query expression>)
   [UNION [ALL]
 <query specification|(<query expression>)
   [...n]]
```

Argumentos

<query_specification> | (<query_expression>)

É uma especificação de query ou expressão de query que devolvem os dados serem combinados com os dados de outra especificação de query ou expressão de query. As definições das colunas que são parte de uma operação de UNION não têm que ser idênticas, mas elas devem ser compatíveis por conversão implícita.

UNION

Especifica que os resultados de múltiplas querys serão combinados e será devolvido como um único resultado.

Exemplo:

```
use aula
go
-- UNION sem ALL como se tivesse feito um distinct
SELECT C5_CLIENTE, C5_LOJACLI
FROM   SC5990
WHERE D E L E T  = ''
UNION -- NAO DUPLICA
SELECT E1_CLIENTE, E1_LOJA
FROM SE1990
WHERE D_E_L_E_T_ = ''
```

ALL

Devolve todas as linhas no resultado, inclusive duplicadas. Se não for especificado, serão removidas as linhas duplicadas.

Exemplo:

```
use aula
go

SELECT C6 PRODUTO, SUM(C6 QTDVEN) AS QTDVEN
FROM SC6990
WHERE D E L E T  = ''
GROUP BY C6_PRODUTO
--COLOCAR 1 LINHA COM TOTAL GERAL
UNION ALL
SELECT C6 PRODUTO = 'TOTAL', SUM(C6 QTDVEN)
FROM SC6990
WHERE D_E_L_E_T_ = ''
```

7.1.8. Cláusula ORDER BY

Especifica a ordem do resultado da query. A cláusula ORDER BY é nula em views, funções, tabelas derivadas, e subqueries, a menos que TOP seja especificado.

Sintaxe

[ORDER BY {order_by_expression [ASC|DESC]} [,...n]]

Argumentos

order_by_expression

Especifica uma coluna para utilizar na ordenação. A coluna pode ser especificada pelo nome, alias, uma expressão, ou um inteiro não negativo que representa a posição da coluna na lista do SELECT.

A cláusula ORDER BY pode incluir itens que não aparecem na lista do SELECT. Porém, se for especificado SELECT DISTINCT, ou se o SELECT contém o operador UNION, as colunas de ordenação têm que aparecer na lista do SELECT.

Além disso, quando a declaração SELECT inclui o operador UNION, devera ser usado os nomes ou alias do primeiro SELECT.

Exemplo:

```
use aula
go

SELECT A1.A1 COD, A1.A1 NOME, A1.A1 END,A1.A1 MUN
FROM SA1990 A1
ORDER BY A1.A1_NOME
```

ASC (Default)

Especifica que os dados serão ordenados de forma ascendente, do menor para o maior.

Exemplo:

```
use aula
go

SELECT A1.A1_COD, A1.A1_NOME, A1.A1_END,A1.A1_MUN
FROM SA1990 A1
ORDER BY A1.A1_NOME ASC
```

DESC

Especifica que os dados serão ordenados de forma descendente, do maior para o menor.

Os valores NULL são considerados como os valores mais baixos possíveis.

Exemplo:

```
use aula
go

SELECT A1.A1 COD, A1.A1 NOME, A1.A1 END,A1.A1 MUN
FROM SA1990 A1
ORDER BY A1.A1_NOME DESC
```

Para reforçar os conceitos de Left, Right e Join, segue um exemplo complementar:

Exemplo Complementar:

```
use aula
go

create table empregadosrodrigo(
 codigo_empregado int,
 nome varchar(50)
)
create table pagamentosrodrigo(
 codigo_pagto int,
 codigo_empregado int,
 valor decimal(10,2)
)
create table descontosrodrigo(
 codigo_desconto int,
```

```
 codigo_empregado int,
 valor decimal(10,2)
)

insert into empregadosrodrigo(codigo_empregado,nome) values(1,'Rodrigo')
insert into empregadosrodrigo(codigo_empregado,nome) values(2,'Daniella')
insert into empregadosrodrigo(codigo_empregado,nome) values(3,'Maria Luiza')
insert into empregadosrodrigo(codigo empregado,nome) values(4,'Walter')
insert into empregadosrodrigo(codigo empregado,nome) values(5,'Cris')
insert into pagamentosrodrigo(codigo_empregado,valor) values(1,100)
insert into pagamentosrodrigo(codigo_empregado,valor) values(1,200)
insert into pagamentosrodrigo(codigo_empregado,valor) values(3,300)
insert into pagamentosrodrigo(codigo empregado,valor) values(5,400)
insert into pagamentosrodrigo(codigo empregado,valor) values(5,500)
insert into descontosrodrigo(codigo empregado,valor) values(1,50)
insert into descontosrodrigo(codigo_empregado,valor) values(2,20)
insert into descontosrodrigo(codigo_empregado,valor) values(5,30)

select e.nome, p.valor as pagamento
from empregadosrodrigo as e INNER JOIN pagamentosrodrigo as p
ON e.codigo empregado = p.codigo empregado

select e.nome, p.valor as pagamento, d.valor as desconto
from empregadosrodrigo as e
INNER JOIN pagamentosrodrigo as p ON e.codigo empregado = p.codigo empregado
INNER JOIN descontosrodrigo as d ON e.codigo empregado = d.codigo empregado

select e.nome, p.valor pg, d.valor dc
from empregadosrodrigo e
JOIN pagamentosrodrigo p ON e.codigo empregado = p.codigo empregado
JOIN descontosrodrigo  d ON e.codigo empregado = d.codigo empregado

select e.nome, p.valor pg, d.valor dc
from empregadosrodrigo e,
     pagamentosrodrigo p,
     descontosrodrigo  d
where e.codigo empregado = p.codigo empregado
and   e.codigo_empregado = d.codigo_empregado

select e.nome, p.valor as pg
from empregadosrodrigo e
LEFT JOIN pagamentosrodrigo p ON e.codigo empregado = p.codigo empregado

select e.nome, p.valor as pg
from empregadosrodrigo e
LEFT JOIN pagamentosrodrigo p ON e.codigo_empregado = p.codigo_empregado

select e.nome, p.valor as pagamento, d.valor as desconto
from empregadosrodrigo as e
LEFT JOIN pagamentosrodrigo as p ON e.codigo_empregado = p.codigo_empregado
LEFT JOIN descontosrodrigo as d ON e.codigo_empregado = d.codigo_empregado

select e.nome, p.valor pg
from pagamentosrodrigo p
LEFT JOIN empregadosrodrigo e ON p.codigo_empregado = e.codigo_empregado

--traz todos os empregados independente se tenham pagamentos ou não
select e.nome, p.valor pg
from empregadosrodrigo e
LEFT joIN pagamentosrodrigo p ON e.codigo_empregado = p.codigo_empregado

--trazer todos os empregados independente se tiveram desconto ou não
select e.nome, d.valor pg
from empregadosrodrigo e
LEFT joIN descontosrodrigo d ON e.codigo empregado = d.codigo empregado

--trazer somente os empregados que tiveram descontos
select e.nome, d.valor pg
from  descontosrodrigo d
```

```
LEFT joIN  empregadosrodrigo e ON d.codigo_empregado = e.codigo_empregado

drop table empregadosrodrigo
go

drop table pagamentosrodrigo
go

drop table descontosrodrigo
go
```

7.1.9. Cláusula COMPUTE

Esta cláusula gera totais que aparecem como colunas sumárias ao final do resultado da query. Quando usado com BY, o COMPUTE gera quebras e subtotais no resultado. Você pode especificar COMPUTE BY e COMPUTE na mesma query.

Sintaxe

```
[COMPUTE
{{AVG|COUNT|MAX|MIN|STDEV|STDEVP|VAR|VARP|SUM}(expression)}
    [,...n]
    [BY expression [,...n]]
]
```

Argumentos

AVG | COUNT | MAX | MIN | STDEV | STDEVP | VAR | VARP | SUM

Especifica a totalização a ser executada.

AVG Média dos valores na expressão numérica.

COUNT Número de linhas selecionadas.

MAX Valor maior na expressão.

MIN Valor menor na expressão.

STDEV Desvio Padrão para todos os valores na expressão. (Estatística)

STDEVP Desvio Padrão para a população para todos os valores na expressão. (Estatística)

SUM Total dos valores na expressão numérica.

VAR Variância para todos os valores na expressão. (Estatística)

VARP Variância para a população para todos os valores na expressão. (Estatística)

Estas funções ignoram valores nulos.

(expression)

Uma expressão, como o nome de uma coluna na qual o cálculo será executado. A expressão tem que aparecer na lista do SELECT e deve ser especificada exatamente igual. Alias de campo não pode ser usado dentro da expressão.

BY expression

Gera quebra e subtotais no resultado. A expressão é uma cópia exata de uma expressão do ORDER BY associado. Podem ser especificadas varias expressões conforme aparecem no ORDER BY, porém na mesma ordem. Essas expressões fazem com que o resultado de um grupo seja quebrado e subgrupos e aplica a função de totalização em cada nível de agrupamento.

Exemplo:

```
Use aula
go

--COMPUTE BY
SELECT E1 CLIENTE, E1 VALOR, E1 SALDO
FROM   SE1990 SE1
WHERE SE1.D_E_L_E_T_ = ''
ORDER BY E1_CLIENTE, E1_LOJA
COMPUTE SUM(E1_VALOR), COUNT(E1_CLIENTE) by E1_CLIENTE
```

7.2.SUBQUERYS

Subquerys são querys executadas dentro de outra query, ou seja, cria-se um select e utiliza o seu resultado como se fosse uma tabela que pode ser referenciado no FROM ou JOINS ou como um único resultado em uma coluna.

Exemplo:

```
Use aula
go

SELECT SA1.A1 NOME,
       ( SELECT SUM(E1 SALDO)
           FROM SE1990 AS SE1
          WHERE SE1.E1 CLIENTE = SA1.A1 COD ) AS TOTAL
  FROM SA1990 AS SA1



SELECT SA1.A1_NOME, TOT.TOTAL
  FROM SA1990 AS SA1
 INNER JOIN ( SELECT E1_CLIENTE, SUM(E1_SALDO) AS TOTAL
                FROM SE1990
               GROUP BY E1 CLIENTE ) as TOT
    ON TOT.E1_CLIENTE = SA1.A1_COD
```

A segunda forma é mais rápida, porque na primeira forma, para cada linha da tabela CLIENTE o SGBD deve fazer um SELECT na tabela TITULOS, ou seja se existirem 200 cliente, serão 200 Select na tabela TITULOS. No segundo caso o SGBD só precisa de um único SELECT na Tabela TITULOS agrupando por Cod_Cli e depois que os dados estão prontos ele junta com a tabela CLIENTE.

8. VISOES EM SQL

Visões ou VIEW é uma tabela virtual que representa os dados de uma ou mais tabela de um modo alternativo.

CREATE VIEW / DROP

8.1.1. Criação de View

A sintaxe para criar uma VIEW é a seguinte:

```
CREATE VIEW [<database_name>.][<owner>.]view_name[(column
[,...n])]
        [WITH ENCRYPTION]
AS
select_statement
[WITH CHECK OPTION]
```

Argumentos

view_name

É o nome da VIEW, o dono, owner, pode ser omitido.

Column

É o nome a ser usado para uma coluna na VIEW. Nomear colunas no CREATE VIEW só é necessário quando uma coluna é derivada de uma expressão aritmética, uma função ou uma constante, quando duas ou mais colunas podem ter o mesmo nome (normalmente por causa de um JOIN), ou quando se deseja que o nome da coluna na VIEW seja diferente da coluna de qual derivou. Também podem ser declarados nomes de colunas no próprio SELECT.

Se "colums" não é especificado, a VIEW assume o nome retornado no SELECT.

AS

Indica a ação que VIEW irá executar.

select_statement
É a declaração do SELECT que define a VIEW. Podem ser usadas mais de uma tabela e outras views. O SELECT pode ser complexo.

Existem algumas restrições no SELECT que defini a VIEW.

Atenção:

Não pode conter:

- Clausula COMPUTE ou COMPUTE BY.
- Incluir ORDER BY, a menos que exista a clausula TOP no SELECT.

- Incluir a palavra chave INTO.

-Referenciar uma tabela temporária ou variável.

WITH CHECK OPTION

Força que todas as atualizações executadas através de uma VIEW não façam com que a linha alterada desapareça da VIEW, devido ao critério utilizado do SELECT. Ou seja se depois da alteração a coluna não satisfazer mais o critério do SELECT a alteração não será permitida, o SGBD retornará um erro e as linhas não serão alteradas.

WITH ENCRYPTION

Indica que o Servidor irá criptografar as colunas nas tabelas de sistema onde é guardado o conteúdo do CREATE VIEW.

Atenção:

Tenha certeza que deseja criptografar sua view. Se esquecer a senha não conseguirá visualizá-la.

8.1.2. Exclusão de View

A sintaxe para excluir uma VIEW é a seguinte:

DROP VIEW [<database_name>.][<owner>.]view_name

Exemplo:

```
Use aula
go

Drop View V_DEL_SA1990
```

Atenção:

Tenha certeza que deseja apagar sua view.

8. MODIFICAÇÃO DE DADOS

Os comando para manipulação de dados são três, Inclusão através do INSERT, alteração com o UPDATE e exclusão com DELETE

8.1.INSERT

Utilizado para inserir dados nas tabelas. Abaixo segue sua sintaxe:

```
INSERT [INTO]
   { table_name
     |view_name
     |rowset_function_limited  }
   {[(column_list)]
    { VALUES( {DEFAULT|NULL|expression} [ ,...n])
     |derived_table
     |execute_statement  }  }
   |DEFAULT VALUES
```

Argumentos

[INTO]

É uma palavra chave opciona que pode ser usado entre o INSERT e a tabela alvo.

table_name

É o nome da Tabela ou Tabela Variável que ira receber os dados

WITH (<table_hint_limited> [...n])

Especifica uma ou mais "table hints" que são permitidas na tabela destino.

Não são permitidos os hints READPAST, NOLOCK E READUNCOMMITTED.

view_name

É o nome da VIEW. A VIEW referenciada deve ser alterável. As modificaçõe não podem afetar mais de uma tabela referenciada na clausula FROM da VIEW.

rowset_function_limited

É qualquer uma das funções OPENQUERY ou OPENROWSET

(column_list)

É uma lista de uma ou mais colunas em qual serão inseridos os dados. A lista deverá estar entre parêntese e separados por virgulas

VALUES
É a lista de dados a serem inseridos.

DEFAULT
Força o SGBD inserir os DEFAULTs especificados nos campos.

expression
É uma constante, variável ou uma expressão. A expressão não pode conter uma declaração de SELECT ou EXECUTE.

derived_table
É alguma declaração de SELECT que retorna linhas para incluir na tabela.

execute_statement
É alguma declaração de EXECUTE que retorna um SELECT.

DEFAULT VALUES
Cria uma nova linha com os valores DEFAULTs definidos para as colunas.

Atenção:

Pensando na integridade dos seus dados, cuidado ao inserir os dados diretamente nas tabelas do Protheus. A TOTVS não aconselha que se execute "por fora".

Exemplo:

```
Use aula
go


-- Verificando se já está em meu database
If exists(Select 'x' From sysobjects where xtype = 'U' and name = 'TAB01')
       drop table TAB01

-- Criando a Tabela
Create Table TAB01(
                  COD int,
                  NOME VARCHAR(50),
                  CIDADE VARCHAR(30)
                  )
go

-- Inserindo dados em minha tabela a partir da SA1990
INSERT TAB01(COD, NOME, CIDADE)
SELECT R E C N O , LEFT(A1 NOME, 20), LEFT(A1 MUN, 20)
  FROM SA1990

Select * From TAB01
```

```
Select * From SA1990

-- Inserindo na Temporária -- O F coloquei para não duplicar os códigos
SELECT 'R' + RIGHT('000000' + CAST(COD AS VARCHAR(5)) ,5) COD, NOME, CIDADE,
  REC = Identity(int,1,1) --so pode ser usado para incluir em uma tabela
  INTO #SA1
  FROM TAB01

SELECT * FROM  #SA1

-- Acertando o RECNO com os novos clientes
Insert SA1990(A1 COD, A1 NOME, A1 MUN, R E C N O )
Select COD, NOME, CIDADE,
       REC + (Select MAX(R E C N O ) from SA1990 (NOLOCK) )
From #SA1

-- Checando Clientes Inseridos
Select   A1.R E C N O , A1.A1 NOME, A1.A1 NREDUZ, A1.A1 END, A1.A1 MUN, A1.A1 EST
From     SA1990 A1
Order By A1.R E C N O


---------------------------------
-- Para Testar Novamente      --
-- Se for testar o update não --
-- execute o Drop e o Delete  --
---------------------------------
-- Limpando Temporária
DROP TABLE #SA1
go

-- Excluindo clientes inseridos
Delete From SA1990 Where A1_COD LIKE 'R%'
go

-- Verificando Exclusao efetuada
Select * From SA1990
go

------------------------| FIM |------------------------------
```

8.2.UPDATE

Altera os dados de uma tabela.

```
UPDATE
  { table_name
    |view_name
    |rowset_function_limited}

  SET { column_name={expression|DEFAULT|NULL}
       |@variable=expression
       |@variable=column=expression }[ ,...n ]

  { { [FROM {<table_source>}[ ,...n ]]
     [WHERE <search_condition>] }
    |[WHERE CURRENT OF{ { [GLOBAL]cursor_name }
                      |cursor_variable_name }] }


table_source

table_name[[AS]table_alias]
|view_name[[AS]table_alias]
|rowset_function[[AS]table_alias]
|derived_table[AS]table_alias
|<joined_table>


joined_table

<table_source><join_type><table_source>ON<search_condition>
|<table_source>CROSS JOIN<table_source>
|<joined_table>

join_type

[INNER|{{LEFT|RIGHT|FULL}[OUTER]}]JOIN
```

Argumentos

table_name
É o nome da tabela para alteração.

view_name
É o nome da VIEW para alteração.

rowset_function_limited
É uma das funções OPENQUERY ou OPENROWSET, sujeitos a capacidade do provider

SET
Especifica a lista de colunas ou variáveis para alteração.

column_name
É o nome da coluna que será alterada.

A table alias specified in a FROM clause cannot be used as a qualifier in SET column_name. For example, this is not valid:

expression
É uma variável, valor literal, expressão ou um subselect entre parênteses que retorna um único valor.

DEFAULT
Especifica que o default do campor irá substituir o valor existente na coluna.

@variable
É uma variável declarada que receberá o valor da expressão.

SET @variable = column = expression atualiza a variável com o mesmo valor que a coluna recebeu. É diferente de SET @variable = column, column = expression, com isso a variável recebe o valor anterior da coluna e depois a coluna é atualizada

FROM < table_source >
Especifica que uma tabela será usada para prover o critério para a alteração

table_name [[AS] table_alias]
É a tabela que irá prover o critério de atualização com o seu respectivo alias

view_name [[AS] table_alias]
É o nome e alias da view que será utilizado para prover critério de atualização

rowset_function [[AS] table_alias]
É o nome de alguma função que retorna dados com seu respectivo alias.

derived_table
É uma subquery retorna linha do database. derived_table é usada como entrada de dados para outra query.

< joined_table >
É o resultado da junção de duas ou mais tabelas

Os comandos de junção são similares aos da Clausula FROM na Declaração do SELECT

WHERE
Especifica a condição utilizada para limitar as linhas que serão alteradas

< search_condition >
É a condição que as linhas devem satisfazer para serem alteradas. Podem ser usados campos de todas as tabelas envolvidas.

CURRENT OF
Especifica que a atualização será realizada na posição atual do cursor.

GLOBAL
Especifica que o nome do cursor refere-se a um cursor global

cursor_name
É o nome do cursor aberto do qual a busca deverá ser feita.

cursor_variable_name
É o nome da variável cursor

Exemplo:

```
Use aula
go

-- Utilizando a tabela criada no item Insert

-- Altera o nome com um (-)hifen antes do nome
UPDATE #SA1
SET NOME = LEFT('-' + NOME,20) --PARA DEIXAR COM 20, SENAO ESTOURA CAPACIDADE

Select * From #SA1

-- Troca por *
UPDATE #SA1
SET NOME = '*' + SUBSTRING(NOME, 2,20)
WHERE COD > 'R00005'

Select * From #SA1

-- Troca por sinal de +
UPDATE S
SET NOME = '+' + SUBSTRING(NOME, 2,20)
FROM #SA1 S
WHERE COD > 'R00005'

Select * From #SA1

-- Alterando Totalmente
UPDATE S
SET NOME = '+)))))))0)0'
FROM #SA1 S
WHERE COD > 'R00005'

Select * From #SA1

-- Voltando conforme SA1990
UPDATE S
SET NOME = LEFT(A1 NOME,20)
FROM #SA1 S
INNER JOIN SA1990
ON A1_COD = COD

Select * From #SA1
```

Exemplo:

```
Use aula
go

-- Utilizando a tabela TAB01 criada no item Insert
Begin Transaction

SELECT * FROM TAB01

DECLARE @REC INT
SET @REC = 0

UPDATE TAB01
   SET CIDADE =  CASE WHEN @REC % 2 = 0 THEN 'PAR' ELSE 'IMPAR' END,
       @REC = @REC + 1

SELECT * FROM TAB01

Rollback Transaction
```

8.3.DELETE

O comando utilizado para excluir dados de uma tabela é o DELETE.

Sintaxe

```
DELETE [FROM]
   {table_name
    |view_name
    |rowset_function_limited }

   [FROM{<table_source>}[ ,...n ]]
   [WHERE {<search_condition>
         |{[CURRENTOF { { [GLOBAL] cursor_name }
                     |cursor_variable_name  } ] } } ]
```

Argumentos

FROM

É opcional utilizado entre o DELETE e a tabela, view ou rowset_function_limited.

table_name

É o nome da tabela de onde os dados serão excluídos.

view_name

É o nome da VIEW.

rowset_function_limited

É uma das funções OPENQUERY ou OPENROWSET, sujeitos a capacidade do provider

FROM <table_source>

Especifica a Clausula FROM semelhante a Declaração do SELECT

Exemplo:

```
Use aula
go

-- Utilizando a tabela TAB01 criada no item Insert
Begin Transaction

Select * From TAB01

Declare @REC INT

Set @REC = 10 -- Ou Select @REC = 10 também atribui

Delete From TAB01 Where COD = @REC

Select * From TAB01

Rollback Transaction
```

Atenção:

Não apague registros diretamente no database "por fora", somente se tiver certeza, absoluta, do que está fazendo.

CONTROLE DE FLUXO – SCRIPTS

VARIAVEIS

As variáveis são criadas através da clausula DECLARE, elas podem ser utilizadas em Scripts, Procedures, Triggers e Functions.

Sintaxe

DECLARE {@local_variable data_type}[,...n]

Argumentos

@local_variable
É o nome da variável. Nomes de Variáveis devem começar com o sinal "arroba" @.

data_type
É um tipo de dado. Não são aceitos os tipos text, ntext ou image.

Exemplo:

```
Declare @REC INT

Set @REC = 10 -- Ou Select @REC = 10 também atribui
```

8.2.BEGIN – END

Begin e End é utilizado para delimitar um bloco de comandos.

Sintaxe

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

Argumentos

{ sql_statement | statement_block }
É alguma declaração Transact-SQL válida ou varias Declarações agrupadas em um bloco.

Exemplo:

```
Declare @nVar Int

Set @nVar = 2 -- Troque para Testar
```

```
-- Com mais de um comando utiliza-se Begin End
If @nVar = 1
Begin
    Print 'Primeiro Comando - If'
    Print 'Segundo Comando - If'
End
Else
    Begin
       Print 'Primeiro Comando - Else'
       Print 'Segundo Comando - Else'
    End

-- Somente um Comando
If @nVar = 1
    Print 'Somente um Comando - If'
Else
    Print 'Somente um Comando - Else'
```

8.3.GOTO

Desvia o fluxo de execução para um "label". A Declaração GOTO e "label" podem ser usados em qualquer lugar dentro de Procedure, scripts, ou bloco de códigos. Declarações GOTO podem ser aninhadas.

Sintaxe

Definir o label:

label :

Desviar a execução:

GOTO label

Argumentos

label

É o ponto onde o fluxo de processamento continua executando quando um GOTO devia para essa "label".

Exemplo:

```
DECLARE @OPCAO VARCHAR(1)

SET @OPCAO = '2'

IF @OPCAO = '1'
   BEGIN
      PRINT 'OPCAO1'
      GOTO OPT2
   END
ELSE IF @OPCAO = '2'
   PRINT 'OPCAO 2'
ELSE
   GOTO SAIR

OPT1:
PRINT 'OPCAO1'

OPT2:
PRINT 'OPCAO2'

SAIR:
PRINT 'SAIR'
```

8.4.IF – ELSE

Impõe condições na execução de um script, o fluxo segue após o IF se a condição é satisfeita, ou seja, quando retorna "TRUE". Caso contrário o fluxo do comando pula a execução para o próximo comando. É opcional utilizar o ELSE,

com ele declarado, quando a condição não é satisfeita, ou seja, retorna "FALSE", o fluxo segue a execução após o ELSE, pulando os comandos após o IF, com os comandos alternativos.

Sintaxe

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

Argumentos

Boolean_expression
É uma expressão que retorna "TRUE" ou "FALSE". Se a expressão contem uma Declaração SELECT, ela deve estar entre parenteses.

{sql_statement | statement_block}
É qualquer declaração Transact-SQL ou declarações que se agrupam como dum bloco de declaração. Para definir um bloco de declaração, use o BEGIN - END.

Exemplo:

```
Declare @nVar Int

Set @nVar = 2 -- Troque para Testar

-- Com mais de um comando utiliza-se Begin End
If @nVar = 1
Begin
    Print 'Primeiro Comando - If'
    Print 'Segundo Comando - If'
End
Else
    Begin
       Print 'Primeiro Comando - Else'
       Print 'Segundo Comando - Else'
    End

-- Somente um Comando
If @nVar = 1
    Print 'Somente um Comando - If'
Else
    Print 'Somente um Comando - Else'
```

8.5.WHILE - BREAK – CONTINUE

Executa repetidamente o comando ou bloco de comandos enquanto a condição for verdadeira. A execução do WHILE pode ser controlada através das palavras chaves BREAK e CONTINUE.

Sintaxe

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

Argumentos

Boolean_expression

É uma expressão que retorna "TRUE" ou "FALSE". Se a expressão contem uma Declaração SELECT, ela deve estar entre parenteses.

{sql_statement | statement_block}

É qualquer declaração Transact-SQL ou declarações que se agrupam como dum bloco de declaração. Para definir um bloco de declaração, use o BEGIN - END.

BREAK

Força o fluxo de execução a sair do WHILE e desvia ele para o proximo comando depois do fim do WHILE.

CONTINUE

Desvia o fluxo de execução para o inicio do bloco de comandos do WHILE, ignorando os comandos que estão depois do CONTINUE.

Exemplo:

```
Declare @val int

Set @val = 0

While @val < 10
Begin
   Set @val = @val + 1

   Print @val

   If @val = 5
      BREAK -- EXIT em ADVPL
```

```
   Print 'END'

End

----
Declare @val int

Set @val = 0

While @val < 10
Begin
   Set @val = @val + 1

   Print @val

   If @val = 5
      Continue -- LOOP em ADVPL

   Print 'END'

End
```

8.6.WAITFOR

Este comando faz com que o fluxo de execução pare durante um determinado tempo ou até uma determinada hora.

Sintaxe

WAITFOR { DELAY 'time' | TIME 'time' }

Argumentos

DELAY
Faz com que o fluxo de execução para pelo tempo determinado no 'time' até o máximo de 24 horas.

'time'
É a quantidade de tempo que a execução estará parada.

TIME
É a instrução para que a execução pare até que o Servidor chegue na hora informada.

Exemplo:

```
WAITFOR DELAY '00:10:00' -- Tem q. deletar um registro e tem q. fazer para esperar

--SE ESTIVER NO MEIO DE UMA TRANSACAO FICARA BLOQUEADA POR ESTE TEMPO
```

Exemplo:

```
WAITFOR TIME '17:35:00'

-- ESPERARA ATE CHEGAR ESTA HORA SUA CONEXAO ESTARA ESPERANDO.
```

8.7.CURSOR

Cursor é utilizado para executar comando linha a linha em no banco de dados, os outros comandos sempre executam em bloco, não permitindo o usuário correr a tabela. Existem duas Sintaxe, a definida na norma SQL-92 e a do Transact-SQL:

8.7.1. Declaração de Cursor

Sintaxe SQL-92

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

Sintaxe Transact-SQL

```
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 ] ] ]
```

Argumentos SQL-92

cursor_name

É o nome do cursor definido.

INSENSITIVE

Define que será criada uma copia temporária dos dados para ser usada no CURSOR. Todas requisições feitas para o CURSOR são respondidas com os dados temporários, conseqüentemente, modificações feitas nas tabelas em que o cursor é baseado, não refletira nos retornos do CURSOR . Se

INSENSITIVE é omitido, os dados alterados nas tabelas por qualquer usuário irão refletir nos retornos do CURSOR

SCROLL

Especifica que os comandos de busca estarão disponíveis (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Se SCROLL não for especificado então somente NEXT estará disponível.

select_statement

É a query que irá definir os dados do CURSOR. A palavras chaves COMPUTE, COMPUTE BY e INTO não são suportados no SELECT utilizado no CURSOR

READ ONLY

Previne atualizações feitas pelo CURSOR. O CURSOR não pode ser referenciado em um UPDATE ou DELETE.

UPDATE [OF column_name [,...n]]

Defines as colunas que podem ser alteradas com o CURSOR. SE "OF column_name [,...n]" é especificado, só as colunas listadas podem sofrer atualizações. Se não for informada a lista de colunas, todas colunas podem ser alteradas.

Argumentos Transact-SQL

cursor_name

É o nome do cursor definido.

LOCAL

Especifica que o escopo do cursor é local, dentro do Script, Procedure ou Trigger.

GLOBAL

Especifica que o escopo do cursor é local global.

FORWARD_ONLY

Especifica que o cursor só pode rolar do começo para o fim. FETCH NEXT é o unico comando aceito para fazer buscas no CURSOR. Se FORWARD_ONLY e especificado sem as chaves STATIC, KEYSET ou DYNAMIC, o cursor opera com DYNAMIC.

Quando FORWARD_ONLY nem SCROLL é especificado, FORWARD_ONLY é o padrão, a menos que as chaves STATIC, KEYSET ou DYNAMIC estão especificadas.

STATIC

Define que será criada uma copia temporária dos dados para ser usada no CURSOR. Todas requisições feitas ao CURSOR são respondidas com os dados temporários, conseqüentemente, modificações feitas nas tabelas em que o cursor é baseado, não refletira nos retornos do CURSOR .

KEYSET

Especifica que o conteúdo e a ordem das linhas no cursor são fixadas quando o cursor é aberto. As chaves que identificam exclusivamente as linhas é construída em uma tabela temporária conhecida como keyset. Alterações dos valores que não são chaves feitas na tabela, feitas através do cursor ou por outros usuários, são visíveis no cursor. Inserções feitas não são visíveis.

Se uma linha é excluída, quando for buscar essa linha pelo cursor o @@FETCH_STATUS retorna -2.

Alterações feitas nos valores chaves irão funcionar como a exclusão e @@FETCH_STATUS retorna -2.

DYNAMIC

Define que o cursor irá refletir todas as alterações feitas nas linhas que estão no resultado das buscas do cursor

FAST_FORWARD

Especifica FORWARD_ONLY, READ_ONLY com otimização de performance. FAST_FORWARD não pode ser especificado com SCROLL ou FOR_UPDATE. FAST_FORWARD e FORWARD_ONLY são exclusivos, se um for especificado o outro não pode ser especificado.

READ_ONLY

Especifica que o cursor é somente de leitura, não pode ser alterado.

SCROLL_LOCKS

Garante as alterações e exclusões nos dados posicionados, pois o SGBD trava as linhas quando são lidas pelo cursor e assegura que estão com as ultimas atualizações. SCROLL_LOCKS não pode ser especificada se FAST_FORWARD estiver.

OPTIMISTIC

O SGBD não trava as linhas portanto não garante as alterações e exclusões. OPTIMISTIC não pode ser especificada se FAST_FORWARD estiver.

TYPE_WARNING

Especifica que uma mensagem de advertencia será enviada ao cliente se o cursor for implicitamente convertido do tipo requerido para outro.

select_statement

É a query que irá definir os dados do CURSOR. A palavras chaves COMPUTE, COMPUTE BY e INTO não são suportados no SELECT utilizado no CURSOR

UPDATE [OF column_name [,...n]]

Defines as colunas que podem ser alteradas com o CURSOR. SE "OF column_name [,...n]" é especificado, só as colunas listadas podem sofrer atualizações. Se não for informada a lista de colunas, todas colunas podem ser alteradas.

8.7.2. Abertura de Cursor

Sintaxe

OPEN { { [GLOBAL] cursor_name } | cursor_variable_name }

Argumentos

GLOBAL

Especifica que o cursor_name se refere a um cursor global.

cursor_name

É o nome do cursor declarado. Se ambos, um global e um cursor local, existir com cursor_name como o nome deles, cursor_name se refere ao cursor global se GLOBAL é especificado; caso contrário, cursor_name se refere ao cursor local.

cursor_variable_name

É o nome de uma variável tipo cursor que faz referência a um cursor.

8.7.3. Obter dados do Cursor

Sintaxe

```
FETCH [[NEXT|PRIOR|FIRST|LAST|
      ABSOLUTE{n|@nvar}|
      RELATIVE{n|@nvar}]
      FROM] {{[GLOBAL] cursor_name} |
            @cursor_variable_name}
[INTO @variable_name[,...n]]
```

Argumentos

NEXT

Retorna o próximo resultado imediatamente posterior a linha corrente do cursor e incrementa a linha do cursor para a linha retornada. NEXT é a opção default.

PRIOR

Returna a linha que precede a posição do cursor e decrementa a linha corrente.

FIRST

Retorna a primeira linha do cursor e marca-a como linha corrente.

LAST

Retorna a última linha do cursor e marca-a como linha corrente.

ABSOLUTE {n | @nvar}

Se n ou @nvar é positivo, retorna a n linhas depois do inicio do cursor e a torna linha corrente, se n ou @nvar é negativo, retorna n linhas antes do fim do cursor e torna essa linha como linha corrente. Se n ou @nvar for 0, nenhuma linha é retornada. n deve ser uma constante inteira e @nvar deve ser smallint, tinyint, ou int.

RELATIVE {n | @nvar}

Se n ou @nvar é positivo, retorna a n linhas depois da posição atual do cursor e a torna linha corrente, se n ou @nvar é negativo, retorna n linhas antes da posição atual do cursor e torna essa linha como linha corrente. Se n ou @nvar for 0, retorna a linha corrente. n deve ser uma constante inteira e @nvar deve ser smallint, tinyint, ou int.

GLOBAL

Especifica que o cursor_name se refere a um cursor global.

cursor_name

É o nome do cursor declarado. Se ambos, um global e um cursor local, existir com cursor_name como o nome deles, cursor_name se refere ao cursor global se GLOBAL é especificado; caso contrário, cursor_name se refere ao cursor local.

@cursor_variable_name

É o nome de uma variável tipo cursor que faz referência a um cursor.

INTO @variable_name[,...n]

Faz com que os dados das colunas do cursor sejam armazenados nas variáveis locais. Cada variável na lista, da esquerda para direita, é associada com a coluna correspondente no resultado do cursor. Os dados de cada variável têm que ser do mesmo tipo ou deve ser possível fazer uma conversão implícita. O número de variáveis tem que ser igual ao número de colunas do cursor.

8.7.4. Verificar retorno do comando FETCH

Para verificar se o retorno do ultimo comando FETCH é valido, utilize a função @@FETCH_STATUS.

Sintaxe

@@FETCH_STATUS

Tipo de Retorno

Integer

Valores possíveis de retorno

0	Comando executado com êxito
-1	Comando falhou ou fim do Cursor
-2	Retorno perdido

8.7.5. Fechamento de Cursor

Para fechar o Cursor utiliza-se o Comando CLOSE. CLOSE, fecha o cursor liberando o resultado corrente e libera quaisquer Locks que possam existir nas linhas posicionadas. CLOSE deixa a estrutura acessivel para reabertura, mas buscas e updates não são permitidos até a reabertura.

Sintaxe

CLOSE { { [GLOBAL] cursor_name } | cursor_variable_name }

Argumentos

GLOBAL

Especifica que o cursor_name se refere a um cursor global.

cursor_name

É o nome do cursor declarado. Se ambos, um global e um cursor local, existir com cursor_name como o nome deles, cursor_name se refere ao cursor global se GLOBAL é especificado; caso contrário, cursor_name se refere ao cursor local.

cursor_variable_name

É o nome de uma variável tipo cursor que faz referência a um cursor.

8.7.6. Encerrar Cursor

Através do comando DEALLOCATE é possível encerrar o Cursor liberando recursos do Servidor

Sintaxe

DEALLOCATE { { [GLOBAL] cursor_name } | @cursor_variable_name }

Argumentos

GLOBAL

Especifica que o cursor_name se refere a um cursor global.

cursor_name

É o nome do cursor declarado. Se ambos, um global e um cursor local, existir com cursor_name como o nome deles, cursor_name se refere ao cursor global se GLOBAL é especificado; caso contrário, cursor_name se refere ao cursor local.

cursor_variable_name

É o nome de uma variável tipo cursor que faz referência a um cursor.

Exemplo:

```
Begin transaction

SELECT E1_NOMCLI FROM SE1990 (NOLOCK) ORDER BY E1_NUM

Declare @LOJA VARCHAR(2), @COD VARCHAR(6), @NOME VARCHAR(20)

Declare Cliente CURSOR LOCAL FORWARD ONLY FAST FORWARD
FOR
   SELECT A1_LOJA, A1_COD, A1_NOME -- Troque por A1_NREDUZ e Troque novamente
   FROM SA1990 (NOLOCK)
   WHERE D E L E T  = ''

OPEN Cliente
FETCH NEXT FROM Cliente INTO @LOJA, @COD, @NOME

WHILE @@FETCH STATUS = 0
BEGIN
    UPDATE SE1990
       SET E1_NOMCLI  = @NOME
     WHERE E1_CLIENTE = @COD
       AND E1 LOJA    = @LOJA

    FETCH NEXT FROM Cliente INTO @LOJA, @COD, @NOME

END
CLOSE Cliente
DEALLOCATE Cliente
SELECT E1 NOMCLI FROM SE1990 (NOLOCK) ORDER BY E1 NUM
go
Rollback Transaction
```

9. TRIGGER

Trigger é um tipo de "stored procedure" especial que é executada automaticamente quando um usuário tenta alterar os dados da tabela especificada. É permitida a criação de gatilhos múltiplos para qualquer Declaração de INSERT, UPDATE ou DELETE.

9.1.CREATE TRIGGER

Sintaxe

```
CREATE TRIGGER trigger_name
ON {table|view}
[ WITH ENCRYPTION ]
{ { {FOR|AFTER|INSTEAD OF} {[DELETE][,][INSERT][,][UPDATE]}
AS
[{IF UPDATE (column) [{AND|OR} UPDATE(column)][...n ]}]
sql_statement [ ...n ] } }
```

Argumentos

trigger_name

É o nome da Trigger

Table | view

É a Tabela ou View onde a Trigger é executada. É opcional especificar o owner da tabela ou view.

WITH ENCRYPTION

Criptografa os dados na tabela syscomments onde a Trigger é gravada.

AFTER

Especifica que a trigger sé será disparada quando todas as operações que ativam a Trigger forem executadas com sucesso. Todas as checagens de integridade referencial e de constraints devem ser executadas com sucesso antes da execução da Trigger.

AFTER é padrão se FOR for a unica chave declarada.

AFTER não pode ser definida em Views.

INSTEAD OF

Especifica que a Trigger é executada em vez da declaração que a disparou, dessa forma anulando as ações.

Só é possivel criar uma trigger INSTEAD OF por declaração de INSERT, UPDATE ou DELETE em uma tabela ou view. Porém, é possível definir Views sobre Views onde cada view tenha sua propria trigger INSTEAD OF.

Cuidado:
Significa "ao invés de" e não "no instante de".

{[DELETE] [,] [INSERT] [,] [UPDATE]}

São chaves que especifica qual declaração de modificação de dados, que quando afetar a tabela ou view ativara a Trigger. Pelo menos uma opção deve ser especificada. Pode-se fazer qualquer combinação em quaquer ordem dessas opções. Se mais de uma opção for especificada, elas deveram ser separadas por virgulas.

Para Trigger INSTEAD OF, a opção DELETE não é suportada em tabelas ou views que tem integridade referencial especificando a ação em cascata ON DELETE. Similarmente a opção UPDATE não é suportada em tabelas ou views que tem integridade referencial especificando a ação em cascata ON UPDATE.

AS

São as ações que a Trigger irá executar.

sql_statement

São as condições e ações da Trigger. Condições são critérios adicionais que determina se as ações da Trigger serão executadas ou se ela será encerrada.

A Trigger pode executar qualquer numero e tipo de declarações Transact-SQL. Elas são usadas para validar ou alterar o banco de dados quando uma declaração de modificação for executada (UPDATE, INSERT e DELETE). Existem a tabelas especiais nas Triggers que podem ser usadas em sua Declaração:

deleted e inserted

A tabela deleted guarda os dados excluidos ou antes das alterações feitas pelo comando que disparou a Trigger, quando as chaves DELETE e UPDATE estiverem declaradas.

A tabela inserted guarda os dados inseridos ou depois das alterações feitas pelo comando que disparou a Trigger , quando as chaves INSERT e UPDATE estiverem declaradas.

IF UPDATE (*column*)

Testa se a coluna "*column*" foi alterada.

Exemplo:

```
/*
1 Será criado uma tabela de log com campos DATAHORA, DESCRIÇÃO(200)
No qual guardará as alterações da tabela SA1990 campos:
Nome, tel, endereço. Ex Descr. Nome De: "" Para: "".
*/
use aula
drop table TAB LOG
Create Table TAB LOG( Data  Datetime,
                      Descr Varchar(400))
GO

IF  EXISTS (SELECT * FROM sys.triggers WHERE object id =
OBJECT ID(N'[dbo].[T SA1990 A]'))
       DROP TRIGGER [dbo].[T SA1990 A]

CREATE TRIGGER T_SA1990_A
ON SA1990
FOR UPDATE
AS
  INSERT TAB LOG(Data, Descr)
  SELECT getdate(),
         'Alterado' +
         Case When d.A1 NOME <> i.A1 NOME
           Then ' Nome de: '+RTrim(d.A1 NOME)+' Para: '+RTrim(i.A1 NOME)+'|'
           Else       '' End +
         Case When d.A1_END  <> i.A1_END
           Then ' End de: '+RTrim(d.A1_END )+' Para: '+RTrim(i.A1_END )+'|'
           Else '' End +
         Case When d.A1 TEL  <> i.A1 TEL
           Then ' Tel de: '+RTrim(d.A1 TEL )+' Para: '+RTrim(i.A1 TEL )+'|'
           Else '' End
    FROM inserted i
   inner join deleted d
      on i.R_E_C_N_O_ = d.R_E_C_N_O_
   where d.A1 NOME <> i.A1 NOME
      or d.A1 END  <> i.A1 END
      or d.A1 TEL  <> i.A1 TEL


-- Checando Antes
select *
  from TAB LOG

SELECT *
 FROM SA1990

UPDATE SA1990
   SET A1 TEL  = '(11)21135096',
       A1_NOME = 'NOVO NOME'
  WHERE R_E_C_N_O_ = 1

-- Checando Depois
select *
  from TAB_LOG

SELECT *
 FROM SA1990

-------------------------| FIM |--------------------
/*2 Criar Trigger de Exclusão impedindo a exclusão das linhas da tabela SE1990
```

```
executando ao invés disso os seguintes comandos:
rollback tran
raiserror('Teste',1,16)
*/

USE [aula]
GO
IF  EXISTS (SELECT * FROM sys.triggers WHERE object id =
OBJECT ID(N'[dbo].[T SE1990 D]'))
       DROP TRIGGER [dbo].[T_SE1990_D]

CREATE TRIGGER T_SE1990_D
ON SE1990
FOR DELETE
AS
  ROLLBACK TRAN
  RAISERROR('Não é permitido fazer exclusão de dados desta tabela',16,1)


DELETE SE1990 WHERE R E C N O  = 1

SELECT *
  FROM SE1990
WHERE R_E_C_N_O_ = 1
```

9.2.DROP TRIGGER

Sintaxe

DROP TRIGGER { trigger } [,...n]

Argumentos

trigger

É o nome da Trigger a ser excluida

> **Cuidado:**
>
> Tenha certeza que deseja exclui-la.

> **Atenção:**

```
DROP TRIGGER T_SE1990_D
```

10.PROCEDURE

São procedimentos armazenados no banco de dados que executam varios comandos Transact-SQL.

10.1. CREATE PROC

Sintaxe.

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
    [ { @parameter data_type }
        [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]

[ WITH
    { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]

AS sql_statement [ ...n ]
```

Argumentos

procedure_name

É o nome da Procedure, pode ser criar uma Procedure Local com a sinal "#" antes do nome ou uma Global "##".

;number

É um numero inteiro opcional usado para agrupar procedures com o mesmo nome que podem ser excluidos com uma unica declaração DROP PROCEDURE.

@parameter

É o parametro da Procedure. Um ou mais parametros podem ser declarados. O valor de cada um dos parâmetros deve ser fornecido pelo usuário que chamar a procedure (a menos que um "default" do parametreo estiver definido). A Procedure pode ter no máximo 2100 parametros.

Especificar o nome do parâmetro com o arroba(@) no primeiro caractere.

data_type
É o tipo de dados do parâmetro. Todo o tipos de dados podem ser usados como um parâmetro. Porém, Cursor só pode ser usado em parâmetros VARYING com OUTPUT.

VARYING

Especifica que o parametro é de saída e que é contruido dinamicamente, cujo conteúdo pode variar. Aplicado somente para Cursor

Default

É o valor padrão do parâmetro quando o mesmo não for passado na chamada da função.

OUTPUT

Indica que o parametro retorna valor para o execução que chamou a Procedure.

{RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION}

Atenção:

RECOMPILE indica que o Server não faz cache do plano de execução para esta Procedure e a procedure é re-compilada em tempo de execução.
ENCRYPTION indica que o Server irá criptografar o conteudo da procedure guardada na tabela syscomments

AS

Especifica as ações da Procedure

sql_statement

É qualquer numero e tipo de comandos Transact-SQL que definem as ações que a procedure irá executar.

Exemplo:

```
----------------------------------------
use aula
drop proc S PRINT
go

Create Procedure S_PRINT @Valor varchar(10) = 'Vazio',
                         @Valor2 varchar(10)
as
  Print '@Valor='+@Valor
  Print '@Valor2='+@Valor2
--- Fim da Proc

------ Testando
Declare @texto varchar(10)
Set @texto = 'AAAAA'
```

```
EXEC S_PRINT @Valor2 = @texto, @Valor = 'Texto1'

EXEC S PRINT 'BBBB1', 'CCCC2'

EXEC S_PRINT @Valor2 = 'TEXTO2', @Valor = 'Texto1'
-----------------------------------------------
use aula
Drop Proc S CLIENTE
go

Create Procedure S_CLIENTE
as
select * from SA1990

--- Testando
EXEC S_CLIENTE
----------------------
```

10.2. DROP PROCEDURE

Utilizado para excluir uma procedure.

Sintaxe

DROP PROCEDURE { *procedure* }

Argumento

procedure

Nome da procedure que será excluída.

Cuidado:

```
Drop Proc S_CLIENTE
```

11.FUNCTION

São funções armazenadas no servidor que podem ser chamadas em scripts e em querys. Elas retornam valores.

11.1. CREATE FUNCTION

Sintaxe

Função Tipo "Scalar"

```
CREATE  FUNCTION [owner_name.] function_name
   ([{@parameter_name[AS]
scalar_parameter_data_type[=default]}
    [ ,...n ] ])
RETURNS scalar_return_data_type
[ WITH < function_option> [ [,] ...n] ]
[ AS ]
BEGIN
   function_body
   RETURN scalar_expression
END
```

Função Tipo "Inline Table-valued"

```
CREATE FUNCTION [owner_name.] function_name
   ([{@parameter_name[AS]
scalar_parameter_data_type[=default]}
    [ ,...n ] ] )
RETURNS TABLE
[ WITH < function_option > [ [,] ...n ] ]
[ AS ]
RETURN [ ( ] select-stmt [ ) ]
```

Função tipo "Multi-statement Table-valued"

```
CREATE FUNCTION [owner_name.] function_name
   ([{@parameter_name[AS]
scalar_parameter_data_type[=default]}
    [ ,...n ] ] )
RETURNS @return_variable TABLE < table_type_definition >
[ WITH < function_option > [ [,] ...n ] ]
[ AS ]
BEGIN
   function_body
   RETURN
END
```

```
function_option
```

{ ENCRYPTION }

table_type_definition

({ column_definition | table_constraint } [,...n])

Argumentos

owner_name

É o nome do usuário que é owner (dono) da Função. Pode ser omitido e o Server irá considerar o dbo da Base onde a mesma esta sendo criada.

function_name

É o nome da função

@parameter_name

É o nome do parametro da função, iniciado com arroba (@). Um ou mais parâmetros podem ser declarados com limite máximo de 1024 parâmetros. Se for omitido será utilizado o valor DEFAULT.

scalar_parameter_data_type

É o tipo de dados do parametro. Todos os tipos de dados "scalar", incluindo bigint e sql_variant, podem ser usados nos parâmetros da função. O tipo timestamp não são suportados. Tipos "não scalar" como Cursor e tabela não são suportados.

scalar_return_data_type

É o tipo de dado "scalar" que a função retorna. Exceto text, ntext, image, and timestamp.

scalar_expression

Expressão que retorna um valor tipo "scalar" para a função retornar.

TABLE

Especifica que o retorno da função é uma tabela com valores.

Em uma função "inline table-valued", a tabela retornada é definida em uma simples Declaração de SELECT. Funções "Inline" não tem variáveis de retorno associadas.

Em uma função "multi-statement table-valued", @return_variable é uma variável tabela, usada para guardar as linhas que deverão ser retornadas pela função

function_body

Especifica um série de comandos Transact-SQL, que juntos definem o valor que a função irá retornar. A function_body só é usado em função "scalar " e "multi-statement table-valued"

Em função "scalar", function_body é uma serie de comandos Transact-SQL que atualizam o valor retornado pela função.

Em função "multi-statement table-valued", function_body é uma serie de comandos Transact-SQL que populam a tabela de retorno.

select-stmt

É uma Declaração de SELECT simples que a função irá retornar.

ENCRYPTION

Indica que o Server irá criptografar o conteúdo da função guardada na tabela syscomments

Exemplo:

```
DROP FUNCTION F_TITULO_ABERTO
GO
-- Inicio
CREATE FUNCTION F TITULO ABERTO(@ForINI  varchar(6)='',
                               @LojaIni Varchar(2)='',
                               @ForFIM  varchar(6)='ZZZZZZ',
                               @LojaFIM Varchar(2)='ZZ' )
RETURNS TABLE
AS
RETURN ( SELECT E2_PREFIXO, E2_NUM, E2_PARCELA, E2_TIPO, E2_FORNECE, E2_LOJA,
                E2_VALOR, E2_SALDO, E2_VENCTO
           FROM SE2990 (NOLOCK)
          WHERE D E L E T  = ''
            AND E2 FORNECE BETWEEN @ForINI AND @ForFIM
            AND E2_LOJA    BETWEEN @LojaIni AND @LojaFIM )
GO
-- Fim da Function

--- Testando
SELECT *   FROM dbo.F_TITULO_ABERTO(default,default,default,default)
```

Exemplo:

```
use aula
Drop Function F_SOMA
go
-- Inicio
CREATE FUNCTION F_SOMA(@Val1 int, @Val2 int)
RETURNS int
AS
BEGIN
    Declare @Result int

    Select @Result = @Val1 + @Val2

    RETURN @Result
END
-- Fim da Function

------------------| Testando |-------
Select dbo.F_SOMA(2,1)
```

11.2. DROP FUNCTION

Utilizado para excluir uma Função

Sintaxe

DROP FUNCTION { [owner_name .] function_name }

Argumentos

owner_name
É o nome do usuário que é owner (dono) da Função.

function_name
É o nome da função a ser excluida.

Cuidado:

```
Drop Function F_SOMA
```

12.CONTROLE DE TRANSAÇÃO

O controle de transação é utilizado quando existe a necessidade de integridade de dados, onde se tem uma série de comandos (Update, Insert, Delete) e que todos devem ser executados e caso ocorra falha em um deles, seja necessário retornar todos os outros comando que foram executados com êxito. Abaixo seguem os comando necessários.

12.1. BEGIN TRANSACTION

Marca o começo de uma transação. BEGIN TRANSACTION incrementa @@TRANCOUNT com 1.

Sintaxe

```
BEGIN TRAN [ SACTION ] [ transaction_name |
@tran_name_variable
   [ WITH MARK [ 'description' ] ] ]
```

Argumentos

transaction_name

É o nome associado a transação, pode ter até 32 caracteres. Utilize nome em transação somente se externamente existir um para aninhado de BEGIN...COMMIT ou BEGIN...ROLLBACK.

@tran_name_variable

É o nome de uma variável declarada pelo usuário com o nome da transação, a variável pode ser do tipo char, varchar, nchar ou nvarchar.

WITH MARK ['description']

Especifica que a transação será marcada no log e 'description' é a "string" que descreve a marca.

> **Cuidado:**
>
> Se WITH MARK for usado, a transação deverá ter um nome.

12.2. COMMIT TRANSACTION

Marca o fim e o sucesso da transação. Se @@TRANCOUNT é 1, COMMIT TRANSACTION grava todas as modificações feitas desde o começo da transação permanentemente na base de dados e libera os recursos bloqueados durante as operações e decrementa @@TRANCOUNT para 0. Se @@TRANCOUNT é maior que 1, COMMIT TRANSACTION somente decrementa com 1 @@TRANCOUNT.

Sintaxe

COMMIT [TRAN [SACTION] [transaction_name | @tran_name_variable]]

Argumentos

transaction_name

É o nome associado a transação. É ignorado pelo Server. É utilizado somente para deixar o código mais legível.

@tran_name_variable

É o nome de uma variável declarada pelo usuário com o nome da transação, a variável pode ser do tipo char, varchar, nchar ou nvarchar.

12.3. SAVE TRANSACTION

Marca um savepoint dentro de uma transação. Com isso é possível em um rollback, desfazer somente parte de um transação.

Sintaxe

SAVE TRAN [SACTION] { savepoint_name | @savepoint_variable }

Argumentos

savepoint_name

É o nome do SAVE TRANSACTION. Utilize savepoint_name quando o rollback deverá afetar somente parte da transação

@savepoint_variable

É o nome de uma variável declarada pelo usuário com o nome da trado savepoint, a variável pode ser do tipo char, varchar, nchar ou nvarchar.

12.4. ROLLBACK TRANSACTION

Volta a transação para o começo ou para o savepoint dentro da transação, desfazendo as modificações.

Sintaxe

```
ROLLBACK [ TRAN [ SACTION ]
    [ transaction_name | @tran_name_variable
    | savepoint_name | @savepoint_variable ] ]
```

Argumentos

transaction_name

É o nome associado a transação. É ignorado pelo Server. É utilizado somente para deixar o código mais legível.

@tran_name_variable

É o nome de uma variável declarada pelo usuário com o nome da transação, a variável pode ser do tipo char, varchar, nchar ou nvarchar.

savepoint_name

É o nome do SAVE TRANSACTION. Utilize savepoint_name quando o rollback deverá afetar somente parte da transação

@savepoint_variable

É o nome de uma variável declarada pelo usuário com o nome da trado savepoint, a variável pode ser do tipo char, varchar, nchar ou nvarchar.

Exemplo:

```
-- TRANSACAO
SELECT @@TRANCOUNT

BEGIN TRAN

SELECT A1_NREDUZ, * FROM SA1990
WHERE A1_COD IN ('000002','000003','000004')

UPDATE SA1990
SET A1_NREDUZ = 'RODRIGO'
WHERE A1_COD = '000002'

SAVE TRAN RODRIGO

UPDATE SA1990
SET A1_NREDUZ = 'DANIELLA'
WHERE A1_COD = '000003'

SAVE TRAN DANIELLA

UPDATE SA1990
SET A1_NREDUZ = 'MALUIZA'
WHERE A1_COD = '000004'

ROLLBACK TRAN RODRIGO

COMMIT TRAN
-- As tabelas enquanto nao tiver o rollback ou commit ficarao travadas
SELECT @@TRANCOUNT
```

13.FUNÇÕES DO SQL

13.1. CASE

Avalia uma lista de condições e devolve uma das possíveis expressões de múltiplas opções.

CASE tem dois formatos:

A função CASE simples, compara uma expressão com um jogo de expressões simples, para determinar o resultado.

A função CASE de busca avalia um jogo de expressões booleanas para determinar o resultado.

Ambos formatos suportam o argumento opcional ELSE.

Sintaxe

Função CASE simples:

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

Função CASE de busca:

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

Argumentos

input_expression

É a expressão avaliada quando é usada a função CASE simples.

WHEN when_expression

É uma expressão que é comparada com input_expression quando é usada a função CASE simples. O tipo de dado da expressão input_expression e da when_expression precisa ser o mesmo ou aceitar uma conversão implícita.

n

Indica que podem haver multiplas clausulas "WHEN when_expression THEN result_expression", ou multiplas clausulas "WHEN Boolean_expression THEN result_expression".

THEN result_expression

É a expressão retornada quando input_expression é igual à when_expression, ou a Boolean_expression retorna Vedadeiro.

ELSE else_result_expression

É a expressão retornada quando a comparação não retornar Verdadeiro. Se este argumento for omitido o CASE irá retornar NULL.

WHEN Boolean_expression

É a expressão booleana avaliada quando é utilizado o CASE de busca

Exemplo:

```
use aula
select TIPO = CASE A1 TIPO When 'F' Then 'Fisica'
                            When 'J' Then 'Juridica'
                Else
                            'NAO ENCONTRADO'
                End, A1_PESSOA,A1_TIPO,*

From SA1990
```

```
------------- Outro
SELECT CAMPO = CASE WHEN A1 MCOMPRA < 800                THEN 'FRACO'
                    WHEN A1 MCOMPRA BETWEEN 800 AND 1000 THEN 'MEDIO'
                    WHEN A1_MCOMPRA > 1000               THEN 'BOM'
               END, *
FROM SA1990
ORDER BY CASE WHEN A1_MCOMPRA < 800                THEN 'FRACO'
              WHEN A1 MCOMPRA BETWEEN 800 AND 1000 THEN 'MEDIO'
              WHEN A1 MCOMPRA > 1000               THEN 'BOM'
         END


------------- Outro
SELECT CAMPO = CASE WHEN A1 MCOMPRA < 800                THEN A1 NOME
                    WHEN A1 MCOMPRA BETWEEN 800 AND 1000 THEN A1 MUN
                    WHEN A1_MCOMPRA > 1000               THEN A1_EST
               END, A1_MCOMPRA,*
FROM SA1990


--------------- Outro
SELECT CAMPO = CASE WHEN A1_MCOMPRA < 800                THEN A1_NOME
                    WHEN A1_MCOMPRA BETWEEN 800 AND 1000 THEN A1_MUN
                    WHEN A1 TIPO = 'F'                   THEN A1 EST
               END, A1 MCOMPRA,A1 TIPO,*
FROM SA1990


--------------- Outro
Select  B1 COD, MES = CASE MONTH(B1 DATREF) When 1  Then 'Janeiro'
                                            when 2  Then 'Fevereiro'
                                            When 3  Then 'Março'
                                            When 4  Then 'Abril'
                                            When 5  Then 'Maio'
                                            When 6  Then 'Junho'
                                            When 7  Then 'Julho'
                                            When 8  Then 'Agosto'
                                            When 9  Then 'Setembro'
                                            When 10 Then 'Outubro'
                                            When 11 Then 'Novembro'
                                            When 12 Then 'Dezembro'
              END
From SB1990

--------------- Outro
/* Trazer 3 colunas: unidade_CX, unidade_PC, OUTRAS */
Select D2 DOC, 'CX' = SUM(CASE WHEN D2 UM = 'CX'           THEN D2 TOTAL ELSE 0 END),
               'PC' = SUM(CASE WHEN D2 UM = 'PC'           Then D2 TOTAL Else 0 END),
         'OUTRAS'   = SUM(CASE WHEN D2_UM NOT IN('CX','PC') Then D2_TOTAL ELSE 0 END)
From SD2990
GROUP BY D2_DOC
```

13.2. CAST e CONVERT

Converte explicitamente um dado de algum tipo para outro. CAST e CONVERT são similares.

Sintaxe

Usando CAST:

CAST (expression AS data_type)

Usando CONVERT:

CONVERT (data_type [(length)] , expression [, style])

Argumentos

expression
É qualquer expressão SQL válida.

data_type
É o tipo de dado destino da conversão,

length
É um parametro opcional para os tipos nchar, nvarchar, char, varchar, binary ou varbinary.

style
É o estilo de data para converter datetime ou smalldatetime para dado tipo caractere (nchar, nvarchar, char, varchar, nchar ou nvarchar)

A Tabela abaixo mostra os possíveis estilos para o CONVERT

Sem século (yy)	**Com século (yyyy)**	**Entrada/Saída**
-	0 ou 100	mon dd yyyy hh:miAM (or PM)
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	Mon dd, yy
8	108	hh:mm:ss
-	9 ou 109	mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	Yymmdd
-	13 ou 113	dd mon yyyy hh:mm:ss:mmm(24h)
14	114	hh:mi:ss:mmm(24h)
-	20 ou 120	yyyy-mm-dd hh:mi:ss(24h)
-	21 ou 121	yyyy-mm-dd hh:mi:ss.mmm(24h)
-	126	yyyy-mm-dd Thh:mm:ss:mmm(no spaces)
-	130	dd mon yyyy hh:mi:ss:mmmAM
-	131	dd/mm/yy hh:mi:ss:mmmAM

Exemplo:

```
SELECT Getdate(), CAST( GetDate() as varchar(20) )

------- Outro
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20),Getdate())

------- Outro
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20),Getdate(),8)

------- Outro
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20),Getdate(),112)

------- Outro
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20), 100.0000000)

------- Outro
--- erro tamanho overflow
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20), 100.0000006666666666666660)

------- Outro
Select Len(100.0000006666666666666660) --26

------- Outro
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20), 100.006666660),
       CONVERT(DECIMAL(17,2), 1000.866332) -- ARREDONDA

------- Outro
SELECT Getdate(),
       CAST( GetDate() as varchar(20) ),
       CONVERT(VARCHAR(20), 100.006666660),
       CONVERT(DECIMAL(17,2), '1000.866332') - ARREDONDA
```

13.3. COALESCE

Retorna a primeira expressão não nula do conjunto de argumentos.

Sintaxe

COALESCE (expression [,...n])

Argumentos

Expression

É uma expressão de qualquer tipo.

N
Indica que podem ser especificadas varias expressões. Todas as expressões devem ser do mesmo tipo ou aceitarem conversão implicita.

Exemplo:

```
SELECT COALESCE(NULL, NULL, 2)

------- Outro
SELECT COALESCE(1, NULL, 2)

------- Outro
SELECT CAMPO =
      COALESCE(CASE WHEN A1_MCOMPRA < 800                THEN A1_NOME
                    WHEN A1_MCOMPRA BETWEEN 800 AND 1000 THEN A1_MUN
                    WHEN A1_MCOMPRA > 1000               THEN A1_EST
                END, 'AFSFSFSF' )
FROM SA1990
```

13.4. ISDATE

Determina se a expressão informada é uma data válida.

Sintaxe

ISDATE (expression)

Argumentos

Expression

É uma expressão a ser validada como uma data. Expression é qualquer expressão que devolve um tipo de dados varchar.

Tipo de Retorno

Int. 1 para Verdadeiro e 0 para Falso.

Exemplo:

```
SELECT ISDATE(Getdate())
```

13.5. ISNUMERIC

Determina se a expressão informada é um numérico válido.

Sintaxe

ISNUMERIC (expression)

Argumentos
expression

Tipo de Retorno

Int. 1 para Verdadeiro e 0 para Falso.

Exemplo:

```
select isnumeric(1)
select isnumeric('a')
```

13.6. NEWID

Cria um valor único do tipo uniqueidentifier.

Sintaxe

NEWID( )

Tipo de Retorno

uniqueidentifier

Exemplo:

```
select newid()
```

13.7. NULLIF

Retorna NULL se as duas expressões são equivalentes

Sintaxe

NULLIF (expression , expression)

Argumentos

expression
É uma constante, nome de coluna, função, subquery ou qualquer combinação de operadores matemáticos, strings e de igualdade.

Tipo de Retorno

Retorna o mesmo tipo da primeira expressão caso elas não seja equivalentes, caso contrario retorna valor nulo (NULL) do tipo da primeira expressão.

13.8. @@ROWCOUNT

Retorna o numero de linhas afetadas no ultimo comando.

Sintaxe

@@ROWCOUNT

Tipo de Retorno

Inteiro

13.9. @@TRANCOUNT

Retorna o numero de transações ativas na conexão corrente.

Sintaxe

@@TRANCOUNT

Tipo de Retorno

Integer

13.10. Funções de Date e Hora

13.10.1. DATEADD

Retorna um novo valor tipo datetime baseado na soma de um intervalo para a data especificada.

Sintaxe

DATEADD (datepart , number, date)

Argumentos

datepart

Especifica com qual parte da data será realizada a soma para o novo valor.

Datepart	**Abreviação**	**Parte**
Year	yy, yyyy	Ano
Quarter	qq, q	Trimestre
Month	mm, m	Mês
Dayofyear	dy, y	Dia do Ano
Day	dd, d	Dia
Week	wk, ww	Semana
Hour	hh	Hora
Minute	mi, n	Minuto
Second	ss, s	Segundo
Millisecond	ms	Milessegundo

number

É o valor utilizado para incrementar a parte da data. Se for especificado um valor não inteiro o valor será truncado.

date

É uma expressão que retorna um valor tipo datetime ou smalldate.

Tipo de Retorno

Retorna datetime ou smalldate conforme a data passada como parâmetro.

13.10.2. DATEDIFF

Retorna o número de data e hora entre duas datas especificadas referente a parte da data informada.

Sintaxe

DATEDIFF (datepart , startdate , enddate)

Argumentos

datepart

Especifica qual parte da data será realizada a diferença.

startdate

Data e hora inicio para o calculo

Enddate

Data e hora fim para o calculo.

Tipo de retorno

Integer

13.10.3. DATENAME

Retorna uma string representando a parte da data especificada da data passada como parâmetro.

Sintaxe

DATENAME (datepart , date)

Argumentos

datepart

É o parâmetro que define a parte da data para retorno.

Tipo de retorno

Nvarchar

13.10.4. DATEPART

Retorna um inteiro representando a parte espeficada da data .

Sintaxe

DATEPART (datepart , date)

Argumentos

datepart

É o parâmetro que define a parte da data para retorno.

Tipo de Retorno

Integer

13.10.5. DAY

Retorna um inteiro representado o dia da data passada como parametro

Sintaxe

DAY (date)

Argumento

date

É uma expressão do tipo datetime ou smalldatetime.

Return Type

Integer

13.10.6. GETDATE

Retorna a Data e Hora corrente do Servidor.

Sintaxe

GETDATE ( )

Tipo de retorno

datetime

13.10.7. GETUTCDATE

Retorna a Data e Hora atual representando o UTC Time (Universal Time Coordinate or Greenwich Mean Time). O UTC Time atual é derivado da hora local atual e do time zone configurado no Sistema Operacional do Servidor SQL.

Sintaxe

GETUTCDATE()

Tipo de retorno.

datetime

13.10.8. MONTH

Retorno um inteiro representando o mes da data especificada.

Sintaxe

MONTH (date)

Argumentos

Date

É uma expressão do tipo datetime ou smalldatetime.

Tipo de retorno

Int

13.10.9. YEAR

Retorno um inteiro representando o ano da data especificada.

Sintaxe

YEAR (date)

Argumentos

Date
An expression of type datetime or smalldatetime.

Tipo de retorno

Int

13.11. Funções Matemáticas

ABS (numeric_expression)
Retorna o valor absoluto e positivo da expressão numérica determinada.

ACOS (float_expression)
Retorna o ângulo, em radianos, cujo co-seno é a expressão float determinada, também chamada de arco-coseno.

ASIN (float_expression)
Retorna o ângulo, em radianos, cujo seno é a expressão float determinada, também chamada de arco-seno.

ATAN (float_expression)
Retorna o ângulo, em radianos, cuja tangente é a expressão float determinada, também chamada de arco- tangente.

ATN2 (float_expression , float_expression)
Retorna o ângulo, em radianos cuja tangente está entre as duas determinadas expressões float (também chamado de arco-tangente).

COS (float_expression)
Função matemática que devolve o co-seno trigonométrico do determinado ângulo (em radianos).

SIN (float_expression)
Função matemática que devolve o seno trigonométrico do determinado ângulo (em radianos).

TAN (float_expression)
Retorna a tangente da expressão.

COT (float_expression)
Função matemática que devolve a cotangente trigonométrica do determinado ângulo (em radianos).

DEGREES (numeric_expression)
Transforma um ângulo em radianos para graus.

RADIANS (numeric_expression)
Transforma um ângulo em graus para radianos.

CEILING (numeric_expression)
Retorna o menor inteiro maior ou igual à expressão numérica determinada.

FLOOR (numeric_expression)
Retorna o maior inteiro menor ou igual à expressão numérica determinada.

EXP (float_expression)

Retorna o valor exponencial da expressão float determinada.
LOG (float_expression)
Retorna o Logaritmo Natural da expressão numérica determinada.

LOG10 (float_expression)
Retorna o Logaritmo Base-10 da expressão numérica determinada.

PI ( )
Retorna a constante pi "π"

POWER (numeric_expression , y)
Retorna a potencia y da expressão determinada.

SQRT (float_expression)
Retorna a Raiz Quadrada da expressão.

RAND ([seed])
Retorna um numero randômico float entre 0 e 1. seed especifica um determinado numero.

ROUND (numeric_expression , length [, function])
Retorna a expressão passada, arredondando conforme o tamanho (length). Para (function) igual a 1 o resultado sai truncado.

SIGN (numeric_expression)
Retorna (+1) se valor positivo, (0) se for Zero ou (-1) caso valor passado seja negativo.

13.12. Funções de String

ASCII (character_expression)
Retorna o código ASCII do caractere mais à esquerda da expressão.

CHAR (integer_expression)
Retorna o Caractere correspondente ao código ASCII informado.

CHARINDEX (expression1 , expression2 [, start_location])
Retorna a posição inicial da expression1 na expression2, start_location determina a posição a partir da qual será feita a busca.

DIFFERENCE (character_expression , character_expression)
Retorna a diferença em inteiro entre o valor SOUNDEX de duas expressões.

LEFT (character_expression , integer_expression)
Retorna parte da expressão (character_expression) começando da esquerda trazendo o numero de caracteres especificado (integer_expression).

LEN (string_expression)
Retorna o numero de caracteres que a expressão (string_expression) contém, desconsiderando os espaços do final.

LOWER (character_expression)

Retorna a expressão passada em minúsculo.

LTRIM (character_expression)
Retorna a expressão passada retirando os espaços da esquerda.

NCHAR (integer_expression)
Retorna o caractere unicode correspondente ao código passado.

PATINDEX ('%pattern%' , expression)
Retorna a posição inicial da primeira ocorrência de '%pattern%' na expression, pode ser utilizado caractere coringa.

REPLACE ('string_searched' , 'string_find' , 'string_replacement')
Substitui todas as ocorrencias da 'string_find' na 'string_searched' por 'string_replacement'.

QUOTENAME ('character_string' [, 'quote_character'])
Retorna uma string Unicode delimitado por 'quote_character'

REPLICATE (character_expression , integer_expression)
Repete a expressão character_expression pela quantidade de vez informada em integer_expression

REVERSE (character_expression)
Retorna a expressão character_expression invertida.

RIGHT (character_expression , integer_expression)
Retorna parte da expressão character_expression começando da direita e trazendo o numero de caracteres especificado em integer_expression.

RTRIM (character_expression)
Retorna a expressão removendo os espaços da direita.

SOUNDEX (character_expression)
Retorna código SOUNDEX para avaliar a semelhança entre expressões.

SPACE (integer_expression)
Retorna uma string com a quantidade de espaços informados.

STR (float_expression [, length [, decimal]])
Converte um numero e caractere.

STUFF (character_expression1 , start , length , character_expression2)
Exclui um número especifico de caractere (length), começando em (start) da character_expression1 e inserindo a character_expression2.

SUBSTRING (expression , start , length)
Retorna um número especifico de caractere (length), começando em (start) da expressão.

UNICODE ('ncharacter_expression')
Retorna um valor inteiro representando o código em padrão Unicode do primeiro caractere da expressão.

UPPER (character_expression)
Converte a expressão em Maiúsculo.