Faculdades Integradas de Bauru - FIB

Banco de Dados

Aula 2

MBA em Engenharia de Software com ênfase em aplicações WEB e Mobile

Prof. Me. Marco Aurelio M. Antunes

Pesquisa em múltiplas tabelas

É possível realizar pesquisas em múltiplas tabelas, para isso é necessário saber vincular a informação de forma a mostrar a informação de maneira correta. A isso é dado o nome de união de tabelas (join).

União de tabelas

Para realizar a união de tabelas devemos acrescentar após a cláusula FROM do comando SELECT às tabelas que queremos unir e colocar na cláusula WHERE a condição de união da(s) tabela(s), ou seja, as respectivas chaves primárias e estrangeiras.

Sintaxe:

select [tabela.]coluna,[tabela.]coluna

from tabela1,tabela2

where tabela1.chave_primária,tabela2.chave_estrangeira

É opcional colocar a identificação da tabela antes do nome das colunas na lista de campos do comando SELECT. Contudo é uma prática recomendada para facilitar o entendimento do comando.

União Regular (inner join ou equi-join)

Denomina-se união regular, as uniões que tem a cláusula WHERE unindo a chave primária à chave estrangeira das tabelas afetadas pelo comando SELECT.

Exemplos:

```
select  * from cd;
select cd.codigo_cd, gravadora.nome_gravadora
from cd, gravadora
where cd.codigo_gravadora = gravadora.codigo_gravadora;

-- inner join / on
select  * from cd;
select cd.codigo_cd,cd.nome_cd,cd.codigo_gravadora,gravadora.nome_gravadora
from cd inner join gravadora
on cd.codigo_gravadora = gravadora.codigo_gravadora;
```

```
-- inner join / on / where(filtro)
select  * from cd;
select cd.codigo_cd,cd.nome_cd,cd.codigo_gravadora,gravadora.nome_gravadora
from cd inner join gravadora
on cd.codigo_gravadora = gravadora.codigo_gravadora
where cd.codigo_gravadora > 2;
```

Apelidos em tabelas

Para evitar que o comando fique extremamente extenso, é possível atribuir apelidos às tabelas utilizadas no comando SELECT. Devemos fazer isso, colocando o apelido após o nome da tabela na cláusula FROM.

Exemplo1:

```
select a.codigo_cd, b.nome_gravadora
from cd a inner join gravadora b
on a.codigo_gravadora = b.codigo_gravadora;
```

Exemplo2:

```
create table empregados(
codigo_empregado int,
nome varchar(50)
);

create table pagamentos(
codigo_pagto int,
codigo_empregado int,
valor decimal(10,2)
) ;

create table descontos(
codigo_desconto int,
codigo_empregado int,
valor decimal(10,2)
);
```

```
insert into empregados(codigo_empregado,nome) values(1,'Luis')
insert into empregados(codigo_empregado,nome) values(2,'Marina')
insert into empregados(codigo_empregado,nome) values(3,'Letícia')
insert into empregados(codigo_empregado,nome) values(4,'Gustavo')
insert into empregados(codigo_empregado,nome) values(5,'Mateus')

insert into pagamentos(codigo_empregado,valor) values(1,100)
insert into pagamentos(codigo_empregado,valor) values(1,200)
insert into pagamentos(codigo_empregado,valor) values(3,300)
insert into pagamentos(codigo_empregado,valor) values(5,400)
insert into pagamentos(codigo_empregado,valor) values(5,500)

insert into descontos(codigo_empregado,valor) values(1,50)
insert into descontos(codigo_empregado,valor) values(2,20)
insert into descontos(codigo_empregado,valor) values(5,30)
```

-- verificar o salário dos funcionários quem TEM pagamento a receber.

-- Apesar de termos cinco empregados na tabela, ele mostrou apenas três, o motivo é que apenas estes três tem pagamentos.

```
select e.codigo_empregado,e.nome,p.valor
from empregados e INNER JOIN pagamentos p
ON e.codigo_empregado = p.codigo_empregado
```

-- verificar em 3 tabelas os funcionários que tem descontos.

-- Neste caso apenas dois empregados foram mostrados já que incluímos na consulta os descontos, ou seja, a leitura que esta consulta fez é: mostrar empregados quem tem pagamentos e descontos.

```
select e.nome, p.valor 'pagamento', d.valor 'desconto'
from empregados e INNER JOIN pagamentos p
ON e.codigo_empregado = p.codigo_empregado
INNER JOIN descontos d
ON e.codigo_empregado = d.codigo_empregado
```

Atividade 1

clientes
codcli
nome
fone

vendedor
codven
nome
fone

produtos
codprod
produto
preco
codfornec

fornecedor
codfor
empresa
fone
contato

pedido
codped
codprod

venda
codcli
codven
codped
valorvenda

Baseado no relacionamento entre as tabelas mostrar o nome do cliente, o nome do vendedor e o produto adquirido na venda.

O que são Visões?

A view pode ser definida como uma tabela virtual composta por linhas e colunas de dados vindos de tabelas relacionadas em uma query (um agrupamento de SELECT's, por exemplo). As linhas e colunas da view são geradas dinamicamente no momento em que é feita uma referência a ela.

Como já dito, a query que determina uma view pode vir de uma ou mais tabelas, ou até mesmo de outras views.

Ao criarmos uma view, podemos filtrar o conteúdo de uma tabela a ser exibida, já que a função da view é exatamente essa: filtrar tabelas, servindo para agrupá-las, protegendo certas colunas e simplificando o código de programação.

É importante salientar que, mesmo após o servidor do SQL Server ser desligado, a view continua **"viva"** no sistema, assim como as tabelas que criamos normalmente. As views não ocupam espaço no banco de dados.

Por que utilizar Visões?

- Restringir acesso a dados, utilizando o comando SELECT, podemos filtrar linhas e colunas que não devam ser mostrados a todos os usuários.
- Buscas complexas tornam-se simples: usuários não precisam realizar complexos comandos SELECTs para localizar as informações de que eles necessitam, assim quando precisarem da informação, farão a busca na visão.
- Independência de dados.
- Dados mostrados de maneira diferente (customizada).

As visões podem se classificadas por:

Simples

- Os dados são extraídos de uma única tabela
- Não contém funções
- Não possuem dados agrupados

Complexas

- Os dados são extraídos de várias tabelas
- Podem conter funções
- Podem conter dados agrupados

Vantagens das Views

•**Reuso**: as views são objetos de caráter permanente. Pensando pelo lado produtivo isso é excelente, já que elas podem ser lidas por vários usuários simultaneamente.

•**Segurança**: as views permitem que ocultemos determinadas colunas de uma tabela. Para isso, basta criarmos uma view com as colunas que acharmos necessário que sejam exibidas e as disponibilizarmos para o usuário.

•**Simplificação do código**: as views nos permitem criar um código de programação muito mais limpo, na medida em que podem conter um SELECT complexo, é uma forma de aumentar a produtividade da equipe de desenvolvimento.

Desvantagens das Views

- Esconde uma complexidade da *query* podendo enganar o desenvolvedor quanto à performance necessária para acessar determinada informação. E pode ser pior quando *views* usam outras*views*. Em alguns casos você pode estar fazendo consultas desnecessárias sem saber de forma muito intensiva.

- Cria uma camada extra. mais objetos para administrar. Algumas pessoas consideram isto um aumento de complexidade. Uma outra forma de ver isto é que uma *view* pode ser mal usada.

- Pode limitar exageradamente o que o usuário pode acessar impedindo certas tarefas.

Criando uma View

```
create view vwprodutos as
  select codprod      as 'Codigo’,
         produto,
         preco        as 'Preco Atual’,
         preco * 1.1  as 'Preco com reajuste',
         codfor
  from produtos
  where codprod > 2;
```

Para consultarmos os dados na view usamos o comando SELECT, da mesma forma que se estivéssemos fazendo uma consulta em uma tabela comum.

```
select * from vwprodutos;
```

Alterando uma View

```
alter view vwprodutos as
  select codprod, produto, preco * 1.1 as 'Preco com reajuste'
  from produtos;
```

O comando **ALTER VIEW** é utilizado para atualizar uma view, após ela já ter sido criada e necessitar de alterações.

```
select * from vwprodutos;
```

Excluindo uma View

Para excluirmos uma view é bem simples: é só usar o comando **DROP VIEW.**

A exclusão de uma view implica na exclusão de todas as permissões que tenham sido dadas sobre ela. Dito isso, devemos usar o comando DROP VIEW apenas quando desejamos de fato retirar a view do sistema.

```
drop view vwprodutos;
```

Atividade 2

Transformar em Views alguns select mais complexos utilizados em exercícios ou exemplos anteriores

O que são Triggers?

O termo trigger (gatilho em inglês) define uma estrutura do banco de dados que funciona, como o nome sugere, como uma função que é disparada mediante alguma ação. Geralmente essas ações que disparam os triggers são alterações nas tabelas por meio de operações de inserção, exclusão e atualização de dados (insert, delete e update).

Um gatilho está intimamente relacionado a uma tabela, sempre que uma dessas ações é efetuada sobre essa tabela, é possível dispará-lo para executar alguma tarefa.

As Triggers são usadas para realizar tarefas relacionadas com validações, restrições de acesso, rotinas de segurança e consistência de dados, desta forma, estes controles deixam de ser executados pela aplicação e passam a ser executados pelos Triggers em determinadas situações onde o controle e a consistência de dados se faz necessária.

Dicas:

- Utilize TRIGGER somente quando necessário;
- Quando for utilizar, que seja o mais simples possível;
- TRIGGERS não tem um bom suporte quando há muitas transações;
- Dependendo de como forem definidas, podem originar problemas com performances;
- Minimize o quanto puder instruções ROLLBACK em TRIGGERS;

Sintaxe

```
CREATE TRIGGER [NOME DO TRIGGER]
ON [NOME DA TABELA]
[FOR/AFTER/INSTEAD OF] [INSERT/UPDATE/DELETE]
AS
--CORPO DO TRIGGER
```

•**NOME DO TRIGGER**: nome que identificará o gatilho como objeto do banco de dados. Deve seguir as regras básicas de nomenclatura de objetos.

•**NOME DA TABELA:** tabela à qual o gatilho estará ligado.

•**FOR/AFTER/INSTEAD OF**: uma dessas opções deve ser escolhida para definir o momento em que o trigger será disparado. FOR é o valor padrão e faz com o que o gatilho seja disparado junto da ação. AFTER faz com que o disparo se dê somente após a ação que o gerou ser concluída. INSTEAD OF faz com que o trigger seja executado no lugar da ação que o gerou.

•**INSERT/UPDATE/DELETE:** uma ou várias dessas opções (separadas por vírgula) devem ser indicadas para informar ao banco qual é a ação que disparará o gatilho.

Atividade 3

Incluir clientes

Lançar movimentação financeira de depósito – inserindo o movimento e atualizando o saldo da conta do cliente

Lançar movimentação financeira de saque – inserindo o movimento e atualizando o campo saldo da conta do cliente

Apagar movimentos e ajustar o saldo.

Impedir que clientes sejam excluídos

Alterar o nome do cliente

Criar um gatilho de auditoria para as operações de inclusão, alteração e exclusão.

O que é Stored Procedure?

Stored Procedure, é um recurso valioso para o desenvolvimento de aplicações e desempenho. Nada mais são do que um conjunto de instruções Transact-SQL, que são executadas dentro do banco de dados. É como escrever um programa dentro do próprio banco de dados para executar tudo lá dentro.

Dentro da Stored Procedure podemos utilizar comandos Transact-SQL que não deixam nada a desejar a comandos de uma linguagem de programação qualquer. O Transact-SQL possui instruções de comparação (if), loops (while) operadores, variáveis e funções.

Uma vantagem da Stored Procedures é que só precisa chamar o nome da Stored Procedure, que pode conter diversos comandos Transact-SQL embutidos dentro dela, evitando assim um tráfego de rede maior, resultando em resposta mais rápida.

Uma Stored Procedure pode ainda retornar valores para a aplicação. O SQL Server permite o retorno de dados em forma de uma tabela após a execução ou um valor de retorno normal

O uso de Stored Procedure é encorajado, mas deve-se utilizar este recurso com cuidado, pois se utilizado em excesso o SQL Server pode ser sobrecarregado, mas ao mesmo tempo podemos obter um ganho de desempenho considerável, dependendo do caso.

.

O que é Stored Procedure?

As vantagens do uso de Stored Procedures são claras:

- Modularidade: passamos a ter o procedimento divido das outras partes do software, basta alterar somente as suas operações para que se tenha as modificações por toda a aplicação;

- Diminuição de I/O: uma vez que é passado parâmetros para o servidor, chamando o procedimento armazenado, **as operações se desenvolvem usando processamento do servidor** e no final deste, é retornado ou não os resultados de uma transação, sendo assim, não há um tráfego imenso e rotineiro de dados pela rede;

- Rapidez na execução: os stored procedures, após salvos no servidor, ficam somente aguardando, já em uma posição da memória cache, serem chamados para executarem uma operação, ou seja, como estão pré-compilados, as ações também já estão pré-carregadas, dependendo somente dos valores dos parâmetros. Após a primeira execução, elas se tornam ainda mais rápidas;

- Segurança de dados: podemos também, ocultar a complexidade do banco de dados para usuários, deixando que sejam acessados somente dados pertinentes ao tipo de permissão atribuída ao usuário.

O que é Stored Procedure?

Algumas ‘regras’ para o uso de Stored Procedures:

- Não faça Stored Procedures que somente fazem um Select ou Update ou Delete. Para isso envie a instrução diretamente.

- Use sempre transações, para poder ‘voltar’ os dados em caso de problemas.

- Retorne somente o necessário, evitando tráfego na rede desnecessário.

- SEMPRE idente seu código ao entrar em uma estrutura de bloco.

- Comente o máximo possível do seu código, através do -- ou do /* e */

- Procedimentos armazenados, podem fazer referência à VIEWS e TRIGGERS, bem como à tabelas temporárias.

Utilidade Pública ☺

```
-- Localizando Stored Procedures
Select name From sys.objects Where type = 'P';


-- Localizando Tables
Select name From sys.objects Where type = 'U';


-- Localizando Views
Select name From sys.objects Where type = 'V';


-- Localizando Triggers
Select name From sys.objects Where type = 'T';
```

Atividade 4

Fazer stored procedures para:

Inserir clientes verificando se os dados são consistentes

Verificar dados cadastrais do cliente

Fazer lançamentos de Deposito na tabela Movimento e atualizar o saldo na tabela Conta Corrente

Fazer lançamentos de Saque na tabela Movimento e atualizar o saldo na tabela Conta Corrente

Mostrar o movimento da conta em um determinado dia ou período

Fazer uma projeção de Saldo com uma rentabilidade de 30%

Fazer uma projeção de Saldo com uma rentabilidade fornecida pelo usuário

Transact-SQL (T-SQL)

Transações SQL-Server

O que é transação?

É uma unidade lógica de processamento que tem por objetivo preservar a integridade e a consistência dos dados. Esse processamento pode ser executado todo ou não, garantindo a atomicidade (Consistência, Isolamento e Durabilidade) das informações.

A sintaxe básica de uma transação é:

```
Begin Transaction
--Corpo de comando
Commit ou Rollback
```

Transações SQL-Server

Begin Transaction: é a Tag inicial para o inicio de uma transação.

--Corpo de comando: é o conjunto de comandos a serem executados dentro de uma transação.
Commit ou Rollback: são comandos que finalizam a transação onde: o **'commit'** confirma o conjunto de comandos e o **'rollback'** desfaz todo o processo executado pelo corpo de comandos, caso tenha ocorrido algum evento contrario ao desejado.

```
-- exemplo com um comando
select * from descontos
begin transaction
insert into descontos values (4,1,20)
select * from descontos

rollback
select * from descontos
```

Transações SQL-Server

```
-- exemplo com vários comandos
select * from descontos
begin transaction
insert into descontos values (4,1,20)
insert into descontos values (5,1,20)
select * from descontos
delete from descontos where codigo_desconto = 5
select * from descontos

rollback
select * from descontos
```

Transações SQL-Server

```
-- exemplo de confirmação
select * from descontos
begin transaction
insert into descontos values (4,1,20)
commit
select * from descontos

rollback  -- Não funciona --
```

Verificando erros dentro de uma transação

No SQL SERVER, existe uma função de sistema, que faz a identificação de um erro dentro de uma transação chamada de @@ERROR, função essa que por padrão, recebe o valor 0 (zero) caso não ocorra nenhum erro, e o valor 1 caso ocorra algum erro.

Exemplo de @@ERROR:
Neste exemplo, estamos iniciando uma transação, para executar a mudança de saldo de algumas contas, onde, em caso de um erro, executará o comando rollback para finalizar a transação e retornar os valores dos saldos, exatamente como eram antes da execução da transação, caso tudo ocorra sem erro, executará a transação, confirmando a alteração.

Verificando erros dentro de uma transação

```
create table Contas (
nome varchar(40),
valor numeric(8,2));

insert into Contas values ('Ze',100);
insert into Contas values ('Maria',99999);
select * from Contas;

-- COM erro
create procedure teste
as
Begin Transaction
UPDATE Contas SET valor = valor * 99999
if @@ERROR = 0
  COMMIT
else
  ROLLBACK

exec teste
select * from Contas
```

Verificando erros dentro de uma transação

```
-- SEM erro
create procedure teste2
as
Begin Transaction
UPDATE Contas SET valor = 200
if @@ERROR = 0
  COMMIT
else
  ROLLBACK

exec teste2
select * from Contas
```

Verificando erros dentro de uma transação

As vezes, as mensagens de erro não são muito claras para os usuários e pensando nisso, procurando ajudar os desenvolvedores e administradores de sistemas, foi introduzida no SQL a função RAISERROR, que permite que você personalize as mensagens de erro facilitando o suporte.

O RAISERROR retorna uma mensagem, assim como a função PRINT, aos aplicativos. Essa função pode ser usada para várias finalidades como: verificar problemas no código T-SQL e mostrar textos.

A construção de uma função do RAISERROR é a seguinte:
ERRO: Texto definido pelo usuário.
GRAVIDADE: Indica uma faixa de gravidade.
ESTADO (0-1): Utilizado para ajudar o usuário a encontrar onde o código está gerando erros. (Raramente diferente de 1).

```
-- SINTAXE:
RAISERROR('Você gerou um Erro', 16, 1);
```

Verificando erros dentro de uma transação

```
-- Exemplo
 select * from Contas;

create procedure teste3
as
Begin Transaction
UPDATE Contas SET valor = valor * 999999
if @@ERROR = 0
  COMMIT
else
  begin
  RAISERROR('ERRO NA ALTERAÇÃO DO SALDO DA TABELA CONTA', 16, 1)
  ROLLBACK
  end

exec teste3

select * from Contas
```

Atividade 5

Testar os exemplos de transação