# S Q L

## *Asserções, Visões e Técnicas de Programação*

Daniel Bordignon Cassanelli
Fernando Luiz Grando
Pedro Patitucci Finamore

# Resumo

- Apresentaremos os seguintes tópicos:

  - Especificação de restrições genéricas sobre o banco de dados através de asserções e gatilhos.

  - Facilidades SQL para a definição de visões (views) do banco de dados.

  - Visão geral das diversas técnicas de acesso a banco de dados por meio de programas.

# Introdução

- A SQL (Linguagem de Consulta Estruturada) é uma linguagem usada para o acesso e a manipulação de banco de dados.

- É implementada pela maioria dos bancos de dados existentes.

# Introdução

- A SQL foi desenvolvida originalmente no início dos anos 70 nos laboratórios da IBM em San Jose, dentro do projeto System R, que tinha por objetivo demonstrar a viabilidade da implementação do modelo relacional proposto por E. F. Codd.

# Introdução

- Embora a SQL tenha sido originalmente criada pela IBM, rapidamente surgiram vários "dialetos" desenvolvidos por outros produtores.

- Num esforço conjunto da ANSI e da ISO criou-se a primeira versão padrão da SQL, a SQL-86, substituída posteriormente pela SQL-92 e depois pela SQL-99.

# Divisão da SQL

- A SQL pode ser dividida em:

- DML - Linguagem de Manipulação de Dados

  - A DML é um subconjunto da linguagem usada para inserir (INSERT), atualizar (UPDATE) e apagar (DELETE) dados de uma tabela existente.

# Divisão da SQL

- DDL - Linguagem de Definição de Dados

  - A DDL permite ao utilizador definir tabelas novas e elementos associados. Os comandos básicos da DDL são poucos: CREATE - cria um objeto dentro da base de dados; DROP - apaga um objeto do banco de dados; e ALTER - permite ao usuário alterar um objeto.

# Divisão da SQL

- DCL - Linguagem de Controle de Dados

  - O DCL controla os aspectos de autorização de dados e licenças de usuários para controlar quem tem acesso para ver ou manipular dados dentro do banco de dados. Comandos importantes: GRANT - autoriza ao usuário executar ou setar operações; e REVOKE - remove ou restringe a capacidade de um usuário de executar operações.

# Divisão da SQL

- DQL - Linguagem de Consulta de Dados

  - Embora tenha apenas um comando, a DQL é a parte da SQL mais utilizada. O comando SELECT permite ao usuário especificar uma consulta como uma descrição do resultado desejado. Esse comando é composto de várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais elaboradas.

# Asserções

- Uma asserção (afirmação) é um predicado expressando uma condição que desejamos que o banco de dados sempre satisfaça.

- É utilizada como restrição de integridade.

# Asserções

- Restrições de integridade: fornecem meios para assegurar que mudanças feitas no banco de dados por usuários autorizados não resultem na inconsistência dos dados.

# Asserções

- Quando uma assertiva é criada, o sistema testa sua validade.

- Se a afirmação é válida, então qualquer modificação posterior no banco de dados será permitida apenas quando a asserção não for violada.

# Asserções

- Em SQL, os usuários podem especificar as restrições genéricas via asserções declarativas, usando a declaração CREATE ASSERTION.

- Em uma asserção, é dado um nome à restrição por meio de uma condição semelhante à cláusula WHERE de uma consulta SQL.

# Asserções

- Por exemplo, para especificar a restrição:
- “O salário de um empregado não pode ser maior que o salário do gerente do departamento em que ele trabalha”.
- Em SQL, precisará formular a seguinte asserção:

# Asserções

```
CREATE ASSERTION LIMITE_SALARIO
CHECK (NOT EXISTS
        (SELECT *
        FROM EMPREGADO E, EMPREGADO M, DEPARTAMENTO D
        WHERE       E.SALARIO>M.SALARIO AND
                    E.DNO=D.NUMERO AND
                    D.SSNGER=M.SSN));
```

# Asserções

- A restrição LIMITE_SALARIO é seguida pela palavra-chave CHECK, que é seguida por uma condição entre parênteses que precisa ser verdadeira em todos os estados do banco de dados para que a asserção seja satisfeita.

- O SGBD é responsável por garantir que a condição não seja violada.

# Asserções

- Sempre que alguma tupla no banco de dados fizer com que uma condição ASSERTION evolua para FALSE, a restrição é violada.

# Asserções

- Técnica básica para a formulação de asserções:

  - Especificar uma consulta que selecione as tuplas que violem a condição desejada.

  - Por meio da inclusão dessa consulta em uma cláusula NOT EXISTS, a asserção especificará que o resultado dessa consulta deverá ser vazio.

# Asserções

- Logo, a asserção será violada se o resultado da consulta não for vazio.

- Em nosso exemplo, a consulta seleciona todos os empregados cujos salários sejam maiores do que o do gerente de seu departamento.

# Asserções

```
CREATE ASSERTION LIMITE_SALARIO
CHECK (NOT EXISTS
        (SELECT *
        FROM EMPREGADO E, EMPREGADO M, DEPARTAMENTO D
        WHERE       E.SALARIO>M.SALARIO AND
                    E.DNO=D.NUMERO AND
                    D.SSNGER=M.SSN));
```

- Se o resultado dessa consulta não for vazio, a asserção será violada.

# Exemplos

1) Definir uma restrição de integridade que não permita saldos negativos.

```
CREATE ASSERTION SALDO_RESTRICAO1
CHECK (NOT EXISTS
        (SELECT *
        FROM CONTA
        WHERE SALDO < 0))
```

# Exemplos

**2)** Só permitir a inserção de saldos maiores que quantias emprestadas para aquele cliente.

```
CREATE ASSERTION SALDO_RESTRICAO2
CHECK (NOT EXISTS
        (SELECT *
                FROM CONTA
                WHERE SALDO <
                        (SELECT MAX(quantia)
                        FROM EMPRESTIMO
                        WHERE EMPRESTIMO.CLIENTE_COD = CONTA.CLIENTE_COD)))
```

# Gatilhos

- Outra declaração relacionada ao CREATE ASSERTION na SQL é a CREATE TRIGGER (gatilhos).

- Porém os gatilhos são usados de maneira diferente. Em muitos casos é conveniente especificar o tipo de ação a ser tomada quando certos eventos ocorrerem ou quando certas condições forem satisfeitas.

# Gatilhos

- Um gatilho (trigger) utiliza o modelo ECA – evento-condição-ação.

- A ação será executada automaticamente se uma condição for satisfeita quando ocorrer um determinado evento.

# Gatilhos

- Evento: normalmente são operações de atualização de banco de dados aplicadas explicitamente.

- Condição: uma vez ocorrido o evento, uma condição opcional pode ser avaliada. Se nenhuma condição for especificada, a ação será disparada pelo evento.

# Gatilhos

- Ação: normalmente é uma sucessão de declarações SQL, mas também poderia ser uma transação de banco de dados ou um programa externo que será executado automaticamente.

# Gatilhos

- Sintaxe (Oracle – PL/SQL):

<trigger> ::= **create trigger** <nome gatilho>
(**after** | **before**) <eventos ativadores> **on** <nome tabela>
[ **for each row** ]
[ **when** <condição> ]
<ações disparadas>

<eventos ativadores> ::= <evento ativador> {**or** <evento ativador>}
<evento ativador> ::= **insert** | **delete** | **update** [**of** <nome coluna> {, <nome coluna>}]
<ação disparada> ::= <bloco PL/SQL>

# Gatilhos

- Por exemplo, pode ser útil especificar uma condição em que, quando houver uma violação, os outros usuários sejam informados.

- Exemplo: Um gerente gostaria de ser informado, por meio do envio de mensagem, se as despesas de um empregado em viagem excedessem certo limite.

# Gatilhos

- Considere o seguinte esquema simplificado:
  - Empregado (nome, ssn, salario, *dno, *supervisor_ssn)
    - Considere que é permitido o valor **null** para dno, indicando que um empregado pode estar temporariamente sem departamento.
  - Departamento (dnome, dno, total_sal, *gerente_ssn)
    - Note que o atributo total_sal é derivado, e seu valor corresponde à soma de salários dos empregados locados naquele departamento em particular.

# Gatilhos

- Atualização do total_sal.
- Evento: inserção de um novo empregado.
- Condição: o empregado novo deverá ser nomeado para um departamento - isto é, o valor do atributo dno para a tupla do novo empregado não for null.

# Gatilhos

- Ação: atualizar o valor de total_sal automaticamente para que o departamento do empregado reflita as recentes atualizações nos salários dos empregados .

# Gatilhos

```
create trigger totalsal1
 after insert on empregado
  for each row
  when (new.dno is not null)
          update departamento
          set total_sal = total_sal + new.salario
          where dno = new.dno;
```

**New**: palavra chave usada para se referir à tupla que foi inserida ou atualizada.

# Visões

- Uma visão em SQL é uma tabela única derivada de outra tabela, que pode ser uma tabela básica ou uma visão previamente definida.
- Uma visão não existe de forma física, ela é considerada uma tabela virtual.
- Isso limita as operações de atualização para as visões.

# Visões

- Podemos imaginar uma visão como um meio para a especificação de uma tabela que precisa ser consultada freqüentemente, embora ela não exista fisicamente.

# Visões

- Especificação em SQL:
- Assim como as tabelas, as visões possuem comandos CREATE e DROP.
- A visão recebe um nome (virtual) de tabela (ou nome da visão), uma lista de nomes de atributos e uma consulta para especificar o conteúdo dessa visão.

# Visões

```
CREATE VIEW view_name [(column_name [, column_name]...)]

AS

SELECT_statement
```

- Sintaxe de uma declaração de visão.

# Exemplo

- Para exemplificar a criação de uma view, será mostrado a seguir uma visão que exibe os nomes de autores que moram em Campinas, juntamente com o nome dos seus livros publicados.

# Exemplo

```
create view autores_campinas (NomeAutor, TituloLivro)

as

select autores.nome, livros.titulo

from autores, livros, livrosautores

where autores.id = livrosautores.au_id

 and livros.id = livrosautores.livro_id

 and autores.cidade = 'CAMPINAS'
```

- Declaração de uma visão.

# Visões

- A visão é chamada autores_campinas.
- A sua declaração SELECT puxa dados de três tabelas.
- Agora, pode-se fazer a consulta da seguinte maneira:

```
SELECT * FROM autores_campinas;
```

- Chamada de uma visão.

# Visões

```
DROP VIEW view_name;
```

- Exclusão de uma visão.

- Para o exemplo da view declarada anteriormente (autores_campinas), a exclusão desta view se daria da seguinte forma:

```
DROP VIEW autores_campinas
```

# Visões

- Uma observação interessante:
- Caso uma tabela tenha sido apagada, a view ainda existirá?!
- A resposta é sim! Mesmo que uma tabela seja dropada, a view ainda permanecerá lá. Entretanto, se uma consulta for feita, será retornado uma mensagem indicando que a view possui erros.

# Visões

- Para se alterar uma view, altera-se o nome do retorno desejado.

```
CREATE OR REPLACE VIEW view_name AS
SELECT columns
FROM table
WHERE predicates;
```

- Atualização de uma visão.

# Visões

- No caso do exemplo, se fosse desejado que o retorno para o usuário fosse os autores que moram em Itu, a consulta ficaria da forma:

```
CREATE OR REPLACE view autores_campinas (NomeAutor,
TituloLivro)
as

select autores.nome, livros.titulo

from autores, livros, livrosautores

where autores.id = livrosautores.au_id

and livros.id = livrosautores.livro_id

and autores.cidade = 'ITU'
```

# Técnicas de Programação

- Existem diversas técnicas para as interações entre um banco de dados e os programas de aplicação.

- As principais abordagens para a programação com o banco de dados são as seguintes:

# Técnicas de Programação

- Embutindo os comandos de banco de dados em uma linguagem de programação de propósito geral;
- Usando uma biblioteca de funções para o banco de dados;
- Projetando uma nova linguagem.

# SQL Embutida

- As declarações SQL podem ser embutidas em uma linguagem de programação de propósito geral como JAVA, C, COBOL ou PASCAL, as quais são chamadas de linguagens hospedeiras.

# SQL Embutida

- Uma declaração SQL embutida se distingue de uma declaração da linguagem pelas palavras-chave EXEC SQL, usadas como prefixo, de modo que o pré-processador poderá separar as declarações SQL embutidas do código da linguagem hospedeira.

# SQL Embutida

- Para ilustrar os conceitos de SQL embutida, vamos usar C como linguagem de programação hospedeira.

- Exemplo: Um laço que lê o número do seguro social de um empregado e imprime algumas informações do registro do empregado correspondente no banco de dados.

## Exemplo de Programação com a SQL Embutida

```
0)  int loop;
1)  EXEC SQL BEGIN DECLARE SECTION ;
2)  varchar dnome [16], pnome [16], unome [16], endereço [31] ;
3)  char ssn [10], datnasc [11], sexo [2], minicial [2] ;
4)  float salario, aumento ;
5)  int dno, dnumero ;
6)  int SQLCODE ; char SQLSTATE [6] ;
7)  EXEC SQL END DECLARE SECTION ;
```

*Variáveis do programa C usadas no exemplo de SQL embutida.*

```
//segmento de Programa Exemplo:
0)  loop = 1 ;
1)  while (loop) {
2)      prompt("Entre com o Número do Seguro Social: ", ssn) ;
3)      EXEC SQL
4)              select PNOME, MINICIAL, UNOME, ENDEREÇO, SALARIO
5)              into :pnome, :minicial, :unome, :endereço, :salário
6)              from EMPREGADO where SSN = :ssn ;
7)      if (SQLCODE == 0) printf(pnome, minicial, unome, endereço, salário)
8)              else printf("Número do Seguro Social não existe: ", ssn) ;
9)      prompt("Mais Números de Seguro Social (entre 1 para Sim, 0 para Não): ",
    loop) ;
10) }
```

# SQLJ – SQL Embutida com JAVA

- A SQLJ é um padrão que tem sido adotado por alguns vendedores para embutir a SQL em JAVA.

- Geralmente, um tradutor de SQLJ converte as declarações SQL em JAVA, as quais podem ser executadas, então, por uma interface JDBC.
- Conseqüentemente, é necessário instalar um driver JDBC para usar a SQLJ.

# SQLJ – SQL Embutida com JAVA

- Focalizaremos a SQLJ da forma como ela é usada no SGBDR ORACLE.
- Antes de poder processar a SQLJ com JAVA em ORACLE, é necessário importar várias bibliotecas de classes.
- Além disso, o programa deve, primeiro, conectar-se ao banco de dados desejado.

## Exemplo de Programação com a SQLJ

```
1)  String dnome, ssn, pnome, fn, unome, ln, datanasc, endereço ;
2)  char sexo, minicial, mi ;
3)  double salario, sal ;
4)  integer dno, dnumero ;
```

*Variáveis de programa JAVA usadas no exemplo a seguir.*

```
//segmento de Programa Exemplo
1)  ssn = readEntry("Entre com o Número do Seguro Social: ") ;
2)  try {
3)      #sql {select PNOME, MINICIAL, UNOME, ENDEREÇO, SALARIO
4)              into :pnome, :minicial, :unome, :endereço, :salario
5)              from EMPREGADO where SSN = :snn} ;
6)  }   catch (SQLException se) {
7)      System.out.println("Número do Seguro Social Inexistente: " + ssn) ;
8)      Return ;
9)      }
10) System.out.println(nome + " " + minicial + " " + unome + " " + endereço + " " +
    salario) ;
```

*Segmento de programa JAVA com a SQLJ.*

# SQL/PSM

- A SQL/PSM (Structured Query Language/Persistent Stored Modules) foi desenvolvida pela ANSI como uma extensão da SQL.

- Foi adotada em 1996 e oferece programação procedural, além dos comandos de consulta da SQL.

# SQL/PSM

- A SQL/PSM não é considerada uma linguagem de programação completa.

- Apenas ilustra como os construtores típicos das linguagens de programação comuns – como os laços e as estruturas de condição – podem ser incorporados à SQL.

# SQL/PSM

- A seguir temos um exemplo para ilustrar como esses construtores podem ser usados.
- A declaração para desvio condicional em SQL/PSM tem a seguinte forma:

```
IF <condição> THEN <lista de declarações>
        ELSEIF <condição> THEN <lista de declarações>
        ...
        ELSEIF <condição> THEN <lista de declarações>
        ELSE <lista de declarações>
        END IF;
```

# SQL/PSM

- A SQL/PSM possui vários construtores para laços.
- Há as estruturas-padrão while (enquanto) e repeat (repetição), que têm as seguintes formas:

```
WHILE <condição> DO
        <lista de declarações>
END WHILE;


REPEAT
        <lista de declarações>
UNTIL <condição>
END REPEAT;
```

# SQL/PSM

- Considerando o exemplo a seguir, ele ilustra como a estrutura de desvio condicional pode ser usada em função SQL/PSM.

## Exemplo

```
// Função PSM
0)  CREATE FUNCTION DeptTamanho(IN deptnro INTEGER)
1)  RETURNS VARCHAR [7]
2)  DECLARE NroDeEmps INTEGER;
3)  SELECT COUNT(*) INTO NroDeEmps
4)  FROM EMPREGADO WHERE DNO = deptnro;
5)  IF NroDeEmps > 100 THEN RETURN “ENORME”
6)      ELSEIF NroDeEmps > 25 THEN RETURN “GRANDE”
7)      ELSEIF NroDeEmps > 10 THEN RETURN “MÉDIO”
8)      ELSE RETURN “PEQUENO”
9)  END IF;
```

*Declarando uma função em SQL/PSM.*

# F I M.