UNIOESTE - Universidade Estadual do Oeste do Paraná
CCET - Centro de Ciências Exatas e Tecnológicas
Colegiado de Informática
Curso de Bacharelado em Informática

SQL
Asserções, Visões e Técnicas de Programação

Daniel Bordignon Cassanelli
Fernando Luiz Grando
Pedro Patitucci Finamore

CASCAVEL
2009

Daniel Bordignon Cassanelli
Fernando Luiz Grando
Pedro Patitucci Finamore

TÍTULO DO SEMINÁRIO

SQL
Asserções, Visões e Técnicas de Programação

Seminário apresentado como requisito de avaliação do terceiro bimestre na disciplina de Banco de Dados I.

CASCAVEL
2009

Daniel Bordignon Cassanelli
Fernando Luiz Grando
Pedro Patitucci Finamore

TÍTULO DO SEMINÁRIO

SQL
Asserções, Visões e Técnicas de Programação

Seminário apresentado como requisito de avaliação do terceiro bimestre na disciplina de Banco de Dados I.

Cascavel, 21 de Setembro de 2009.

Sumário

Resumo

Este seminário tem como objetivo aprofundar os conhecimentos já adquiridos em relação à linguagem de banco de dados SQL. Ele apresenta as facilidades adicionais da mesma, como a especificação de restrições gerais, utilizando-se das asserções. Foi discutido também o conceito de visão em SQL. Em particular, foi dada uma visão geral das técnicas mais importantes para a programação de um banco de dados, exemplificando algumas delas.

Palavras-chave: SQL, Asserção, Visão, SQL Embutida, SQLJ, SQL/PSM.

Capítulo 1

Introdução

A Linguagem de Consulta Estruturada (SQL) – do inglês Structured Query Language – é uma linguagem usada para o acesso e a manipulação de banco de dados, implementada pela maioria dos bancos de dados existentes. Através da SQL, podemos fazer desde uma pesquisa simples para termos como resultados os registros de uma tabela, até pesquisas complexas envolvendo grandes bases de dados e diversas tabelas.

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. O nome original da linguagem era SEQUEL, acrônimo para "Structured English Query Language" (Linguagem de Consulta Estruturada em Inglês), vindo daí o fato de, até hoje, a sigla, em inglês, ser comumente pronunciada "síquel" ao invés de "és-kiú-él", letra a letra. No entanto, em português, a pronúncia mais corrente é a letra a letra: "ése-quê-éle".

A linguagem SQL foi muito difundida na década de 70, e agora conta com boas interfaces gráficas e novos recursos que, sem dúvidas, fazem dela a melhor forma de acesso a banco de dados. A linguagem é um grande padrão de banco de dados. Isto decorre da sua simplicidade e facilidade de uso. Ela se diferencia de outras linguagens de consulta a banco de dados no sentido em que uma consulta SQL especifica a forma do resultado e não o caminho para chegar a ele. Ela é uma linguagem declarativa em oposição a outras linguagens procedurais. Isto reduz o ciclo de aprendizado daqueles que se iniciam na linguagem.

Embora a SQL tenha sido originalmente criada pela IBM, rapidamente surgiram vários "dialetos" desenvolvidos por outros produtores. Essa expansão levou à necessidade de ser criado e adaptado um padrão para a linguagem. Esta tarefa foi realizada pela American National Standards Institute (ANSI) em 1986 e ISO em 1987.

Grandes empresas que desenvolvem bancos de dados como a Inprise (Interbase), Oracle e Microsoft (SQL Server) desenvolveram sua própria implementação de SQL para melhorar o trabalho com seus bancos de dados. A idéia é facilitar o trabalho para o desenvolvedor, mas infelizmente algumas instruções SQL desenvolvidas em um banco de dados são completamente desconhecidas para outros, dificultando o trabalho em conjunto com diferentes bancos de dados.

A SQL foi revista em 1992 e a esta versão foi dada o nome de SQL-92. Foi revista novamente em 1999 e 2003 para se tornar SQL:1999 (SQL3) e SQL:2003, respectivamente. A SQL:1999 usa expressões regulares de emparelhamento, queries

recursivas e gatilhos (triggers). Também foi feita uma adição controversa de tipos não-escalados e algumas características de orientação a objeto. A SQL:2003 introduz características relacionadas ao XML, seqüências padronizadas e colunas com valores de auto-generalização (inclusive colunas-identidade).

Tal como dito anteriormente, a SQL, embora padronizada pela ANSI e ISO, possui muitas variações e extensões produzidos pelos diferentes fabricantes de sistemas gerenciadores de bases de dados. Tipicamente a linguagem pode ser migrada de plataforma para plataforma sem mudanças estruturais principais.

Outra aproximação é permitir para código de idioma procedural ser embutido e interagir com o banco de dados. Por exemplo, o Oracle e outros incluem Java na base de dados, enquanto o PostgreSQL permite que funções sejam escritas em Perl, Tcl, ou C, entre outras linguagens.

A SQL pode ser dividida em:

- DML - Linguagem de Manipulação de Dados

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

- DDL - Linguagem de Definição de Dados

A DDL (Data Definition Language) permite ao utilizador definir tabelas novas e elementos associados. A maioria dos bancos de dados de SQL comerciais tem extensões proprietárias no DDL. 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, por exemplo, adicionando uma coluna a uma tabela existente.

- DCL - Linguagem de Controle de Dados

O DCL (Data Control Language) 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. Seus comandos mais importantes são: GRANT - autoriza ao usuário executar ou setar operações; e REVOKE - remove ou restringe a capacidade de um usuário de executar operações.

- DQL - Linguagem de Consulta de Dados

Embora tenha apenas um comando, a DQL (Data Query Language) é a parte da SQL mais utilizada. O comando SELECT permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado. Esse comando é

composto de várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais elaboradas.

Algumas das características da linguagem SQL são:

- Permite trabalhar com várias tabelas;
- Permite utilizar o resultado de uma instrução SQL em outra instrução SQL (sub-queries);
- Não necessita especificar o método de acesso ao dado;
- É uma linguagem para vários usuários como:
    - ➔ Administrador do sistema;
    - ➔ Administrador do banco de dados (DBA);
    - ➔ Programadores de aplicações;
    - ➔ Pessoal da agência e tomada de decisão;
- É de fácil aprendizado;
- Permite a utilização dentro de uma linguagem procedural como C, COBOL, FORTRAN, Pascal e PL/I - SQL embutida.

Capítulo 2

Asserções (Assertions)

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

Há vários tipos de restrições de integridade que seriam cabíveis a bancos de dados. Entretanto as regras de integridade são limitadas às que podem ser verificadas com um mínimo de tempo de processamento.

Uma asserção (afirmação) é um predicado expressando uma condição que desejamos que o banco de dados sempre satisfaça. Restrições de domínio, dependências funcionais e restrições de integridade referencial são formas especiais de 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.

O alto custo de testar e manter asserções tem levado a maioria de desenvolvedores de sistemas a omitir o suporte para asserções gerais. Em SQL, os usuários podem especificar as restrições genéricas via asserções declarativas, usando a declaração CREATE ASSERTION da DDL. Em uma asserção, é dado um nome à restrição por meio de uma condição semelhante à cláusula WHERE de uma consulta SQL. 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:

```
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));
```

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 nome da restrição pode ser usado posteriormente para sua referência, modificação ou eliminação. O SGBD é responsável por garantir que a condição não seja violada. Pode ser usada qualquer condição na cláusula WHERE, mas muitas condições podem ser especificadas

utilizando-se estilo EXISTS e NOT EXISTS das condições SQL. Sempre que alguma tupla no banco de dados fizer com que uma condição ASSERTION evolua para FALSE, a restrição é violada. A restrição será satisfeita para um dado estado no banco de dados, se nenhuma combinação de tuplas no banco de dados violar essa restrição.

A técnica básica para a formulação de asserções é 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. 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. Se o resultado dessa consulta não for vazio, a asserção será violada.

Observe que a cláusula CHECK e a condição da restrição podem ser usadas também para especificar restrições nos atributos e nos domínios e nas tuplas. A principal diferença entre uma CREATE ASSERTION e as outras duas é que a cláusula CHECK da SQL em um atributo, um domínio ou uma tupla fará a checagem somente quando as tuplas forem inseridas ou atualizadas. Assim, a verificação poderá ser implementada com maior eficiência pelo SGBD nesses casos. O projetista do esquema poderia usar CHECK em atributos, nos domínios e nas tuplas apenas quando estiver certo de que a restrição só poderá ser violada pela inserção ou atualização das tuplas. Porém, o projetista deveria usar CREATE ASSERTION somente nos casos em que não for possível usar CHECK nos atributos, nos domínios ou nas tuplas, assim a verificação será implementada com maior eficiência pelo SGBD.

Outra declaração relacionada ao CREATE ASSERTION na SQL é a CREATE TRIGGER, mas os gatilhos serão 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. Ao invés de oferecer aos usuários apenas a opção para a interrupção de uma operação que cause violação – como pela CREATE ASSERTION –, o SGBD poderia dispor de outras opções. Por exemplo, pode ser útil especificar uma condição em que, quando houver uma violação, os outros usuários sejam informados. Um gerente gostaria de ser informado, por meio do envio de mensagem, se as despesas de um empregado em viagem excedessem certo limite. A ação que o SGBD poderia tomar, nesse caso, é enviar a mensagem apropriada ao usuário em questão. A condição será, assim, usada para monitorar o banco de dados. Outras ações poderiam ser especificadas, como executar um determinado procedimento armazenado ou engatilhar outras atualizações. A declaração CREATE TRIGGER, na SQL, é usada para implementar essas ações. Um gatilho (trigger) especifica um evento (como uma operação de atualização no banco de dados em particular), uma condição e uma ação. A ação será executada automaticamente se uma condição for satisfeita quando ocorrer um determinado evento.

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))
```

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)))
```

Capítulo 3

Visões (Views)

Podemos definir uma view como qualquer relação que não faz parte do modelo lógico do banco de dados, mas que é visível ao usuário, como uma relação virtual, ou seja, é uma tabela derivada a partir das tabelas do banco de dados.

O conjunto de tuplas de uma relação visão é resultado de uma expressão de consulta que já foi definido no momento de sua execução.

De um modo prático, as visões permitem prover foco, simplificação e personalização no banco de dados, além de garantir segurança através das restrições já inclusas.

De um modo geral, as views são como as operações join, no que dizem respeito à marca registrada do modelo relacional, pois criam tabelas virtuais a partir de declarações SELECT e abrem um mundo de flexibilidade para a análise e manipulação de dados.

Muitos pensam que as views são cópias dos dados das tabelas ou de outras visões das quais esta é derivada, o que é um erro, pois as visões são tabelas virtuais e são assim chamadas porque não existem como entidades independentes no banco de dados do mesmo modo que as tabelas reais. Visto isso, pode-se concluir que é permitido consultar visões da mesma forma que consultamos tabelas. No entanto, a modificação de dados através de visões é restrita.

Declaração de Views

Assim como as tabelas, as visões possuem comandos CREATE e DROP. Segue abaixo a sintaxe simplificada de uma declaração de definição de visão.

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

AS

SELECT_statement
```

Sintaxe de uma declaração de visão

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.

```
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'
```

Exemplo 01: Declaração de uma visão

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;
```

Exemplo 02: Chamada de uma visão

Exclusão de Views

A sintaxe de exclusão de uma view se dá da seguinte forma:

```
DROP VIEW view_name;
```

Sintaxe de exclusão de uma visão

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

```
DROP VIEW autores_campinas
```

Exemplo 03: Exclusão de uma visão

Uma colocação interessante a se fazer e que sempre é matéria de questionamentos é: no caso de se apagar uma tabela, 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.

Atualização de Views

Para se alterar uma view, altera-se o nome do retorno desejado. 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 proposta pelo exemplo 04. A seguir será mostrado a sintaxe do comando de atualização de views.

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

Sintaxe de Atualização de Views

Pode-se atualizar o dado em uma view se para isso, o usuário tiver os privilégios devidos.

```
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'
```

Exemplo 04: Atualização de uma View

Visto todas estas possibilidades de se trabalhar com dados através das views, pode-se perceber que se trata de uma ferramenta poderosa e que, se for usada de forma sensata e coerente, pode ser de grande ajuda para o manipulador de banco de dados.

Capítulo 4

Técnicas de Programação

Uma interface interativa é muito conveniente para a criação de esquemas e restrições ou para consultas ad hoc eventuais. Entretanto, a maioria das interações com o banco de dados na prática é executada por meio de programas que tenham sido cuidadosamente projetados e testados. Esses programas são normalmente conhecidos como programas de aplicação ou aplicações com banco de dados, e são usados como transações customizadas pelos usuários finais. Outro uso muito comum da programação com o banco de dados é acessá-lo por meio de um programa de aplicação que implementa uma interface Web; por exemplo, para reservas de passagens aéreas ou para compras em uma loja. De fato, a maioria das aplicações na Web em comércio eletrônico inclui algum comando de acesso a um banco de dados.

Abordagens para a Programação com o Banco de Dados

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:

1. *Embutindo os comandos de banco de dados em uma linguagem de programação de propósito geral*: nessa abordagem, as declarações para o banco de dados ficam embutidas na linguagem de programação hospedeira, e elas são identificadas por um prefixo especial. Por exemplo, o prefixo embutido para a SQL é a cadeia EXEC SQL, que precede todos os comandos SQL na linguagem hospedeira. Um pré-compilador ou pré-processador inspeciona primeiro o código-fonte do programa para identificar as declarações do banco de dados e extraí-los para o processamento pelo SGBD. Eles serão recolocados no programa como chamadas de funções para o gerador de código do SGBD.

2. *Usando uma biblioteca de funções para o banco de dados*: deixa-se uma biblioteca de funções disponível para que a linguagem de programação hospedeira possa fazer chamadas para o banco de dados. Por exemplo, pode haver funções para conectar o banco de dados, fazer uma consulta, executar uma atualização, e assim por diante. A consulta real ao banco de dados e os comandos de atualização, bem como outras informações necessárias, são inseridos como parâmetros nas chamadas de funções. Essa abordagem proporciona o que é conhecido como uma Interface para o

Programa de Aplicação (API) pelo programa de aplicação para o banco de dados.

3. *Projetando uma nova linguagem*: uma linguagem de programação de um banco de dados é projetada especialmente para ser compatível com o modelo do banco de dados e com a linguagem de consulta. As estruturas de programação adicionais, como laços e declarações condicionais, são acrescentadas à linguagem de um banco de dados para convertê-la em uma linguagem de programação completa.

Na prática, as duas primeiras abordagens são mais comuns, uma vez que as muitas aplicações existentes em linguagens de programação genéricas exigem algum acesso a um banco de dados. A terceira abordagem é a mais apropriada para as aplicações que tenham interação intensa com o banco de dados. Um dos problemas principais das duas primeiras abordagens é a impedância de correspondência, que não ocorre na terceira.

Impedância de Correspondência (Impedance Mismatch)

A impedância de correspondência é o termo usado para se referir aos problemas que ocorrem em decorrência das diferenças entre os modelos de um banco de dados e da linguagem de programação. Por exemplo, o modelo relacional possui três construtores principais: os atributos e seus tipos de dados, as tuplas (registros) e as tabelas (conjunto de diversos registros). O primeiro problema possível é que os tipos de dados da linguagem de programação difiram dos tipos de dados do modelo de dados. Daí será necessário estabelecer uma forma de correspondência para cada linguagem de programação hospedeira que determine, para cada tipo de atributo, os tipos compatíveis na linguagem de programação. É necessário determinar paridade para cada linguagem de programação porque cada uma possui seus próprios tipos de dados; por exemplo, os tipos de dados disponíveis em C e JAVA são diferentes e ambos diferem dos tipos de dados da SQL.

Outro problema é que o resultado da maioria das consultas é um conjunto (ou diversos conjuntos) de tuplas e cada uma delas é formada por uma seqüência de valores de atributos. Em alguns programas, freqüentemente é necessário acessar o valor de um dado em particular, dentro de uma determinada tupla, para a impressão ou o processamento. Assim, os valores de paridade são necessários para mapear a estrutura de dados resultante da consulta, que é uma tabela apropriada para a estrutura de dados na linguagem de programação. É necessário um mecanismo de laço para varrer as tuplas do resultado de uma consulta, de forma a ter acesso a uma única tupla de cada vez e dela extrair os valores em particular. Um cursor ou uma variável

interativa é usado para varrer as tuplas do resultado de uma consulta. Os valores individuais dentro de cada tupla são, em geral, extraídos por meio de variáveis, com o tipo apropriado dentro do programa.

A impedância de correspondência é minimizada quando uma linguagem de programação é especialmente projetada para usar os mesmos modelos de dados e os mesmos tipos de dados do modelo do banco de dados. Um exemplo disso é a PL/SQL da ORACLE. Para banco de dados de objetos, o modelo de objetos é muito similar ao modelo de dados da linguagem de programação JAVA, assim a impedância de correspondência é amplamente reduzida quando JAVA é usada como linguagem hospedeira para o acesso a um banco de dados de objetos JAVA - compatíveis. Diversas linguagens de programação foram implementadas como protótipos de pesquisa.

Seqüência Típica de Interação em Programação com o Banco de Dados

Quando um programador ou um engenheiro de software cria um programa que exige acesso a um banco de dados, é muito comum que o programa rode em um computador enquanto o banco de dados está instalado em outro. Quando codificamos esse tipo de programa, uma seqüência de interação comum é a seguinte:

1. Quando um programa cliente exige acessão a um banco de dados em particular, o programa precisa primeiro estabelecer ou abrir uma conexão com o servidor de banco de dados. Normalmente isso envolve a especificação de um endereço na Internet de uma máquina na qual o servidor de banco de dados está localizado, além do fornecimento do login de uma conta e uma senha de acesso ao banco de dados.

2. Uma vez estabelecida a conexão, o programa pode interagir com o banco de dados submetendo-o a consultas, atualizações e outros comandos. Em geral, a maioria das declarações SQL pode ser inserida nos programas aplicativos.

3. Quando o programa não precisar mais do acesso ao banco de dados em questão, pode terminar ou fechar a conexão.

Um programa pode acessar diversos bancos de dados se necessário. Em algumas abordagens de programação com o banco de dados, somente uma conexão pode ser ativada por vez, embora, em outras, diversas conexões possam ser estabelecidas ao mesmo tempo.

SQL Embutida

Conforme a primeira abordagem para a programação com o banco de dados, apresentada anteriormente, as declarações SQL podem ser embutidas em uma linguagem de programação de propósito geral como C, COBOL ou PASCAL. A linguagem de programação é chamada linguagem hospedeira. A maioria das declarações SQL – incluindo a definição de dados ou de restrições, as consultas, atualizações ou a definição de visões – pode ser embutida em uma linguagem de programação hospedeira. 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. As declarações SQL podem terminar com um ponto e vírgula ou até encontrar um END-EXEC.

Para ilustrar os conceitos de SQL embutida, vamos usar C como linguagem de programação hospedeira. Dentro dos comandos SQL embutidos, podemos nos referir especificamente a variáveis declaradas no programa C. Estas são chamadas variáveis compartilhadas porque são usadas em ambos os programas, no C e nos comandos embutidos. As variáveis compartilhadas têm como prefixo dois pontos (:) quando elas aparecem em uma declaração SQL. Isso distingue os nomes das variáveis dos nomes dos construtores do esquema do banco de dados, como os atributos e as relações. Também permite que as variáveis de programa tenham os mesmos nomes que os atributos, desde que eles sejam diferenciados, na declaração SQL, pelo prefixo ‘:’. Os nomes dos construtores do esquema do banco de dados – como os atributos e as relações – podem ser usados apenas dentro de comandos SQL, mas as variáveis compartilhadas podem ser utilizadas em qualquer parte do programa C sem o prefixo ‘:’.

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)  }
```

O exemplo a ilustrar a programação com a SQL embutida é um segmento de repetição de programa (laço) que lê o número do seguro social de um empregado e imprime algumas informações do registro do empregado correspondente no banco de dados. O código do programa C é mostrado logo abaixo. O programa lê (entrada) o valor do número social e então recupera, do banco de dados, a tupla empregado com o número do seguro social via comando SQL embutido. A cláusula INTO (linha 5) especifica as variáveis de programa nas quais os valores dos atributos do banco de dados foram recuperados. As variáveis do programa C na cláusula INTO são prefixadas com dois pontos (:), conforme foi dito anteriormente.

A linha 7 ilustra a comunicação entre o banco de dados e o programa por meio da variável especial SQLCODE. Se o valor devolvido pela SGBD para a SQLCODE for 0, a declaração anterior foi executada sem ocorrência de erro ou exceção. Na linha 7 isso pode ser verificado e pressupõe-se que, caso tenha ocorrido erro, é porque não existe nenhuma tupla empregado com o valor do número do seguro social em questão; o que resulta na mensagem que contém essa informação (linha 8).

No exemplo, é selecionada uma única tupla pela consulta SQL embutida; e somente assim poderemos associar esses valores de atributos diretamente às variáveis do programa C por meio de uma cláusula INTO na linha 5. Normalmente, uma consulta SQL pode recuperar muitas tuplas. Nesse caso, o programa C vai tratar e processar as tuplas recuperadas uma por vez. Um cursor pode ser usado para permitir o processamento de uma tupla de cada vez dentro do programa na linguagem hospedeira.

SQLJ – SQL Embutida em Comandos JAVA

A SQLJ é um padrão que tem sido adotado por alguns vendedores para embutir a SQL em JAVA. Historicamente, a SQLJ foi desenvolvida depois de JDBC, que é usado para acessar os bancos de dados com a SQL por meio de JAVA, usando chamadas de função. Focalizaremos a SQLJ da forma como ela é usada no SGBDR ORACLE.

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.

Antes de poder processar a SQLJ com JAVA em ORACLE, é necessário importar várias bibliotecas de classe, mostradas a seguir. Elas incluem as classes JDBC e IO (linhas 1 e 2), mais as classes adicionais relacionadas nas linhas 3, 4 e 5. Além disso, o programa deve, primeiro, conectar-se ao banco de dados desejado, usando a chamada função getConnection, que é um dos métodos da classe Oracle, linha5. O formato dessa chamada função, que retorna a um objeto do tipo default context, é o seguinte:

```
public static DefaultContext
getConnection(String url, String usuario, String senha, Boolean autoCommit)
throws SQLException ;
```

Podemos escrever, por exemplo, as declarações das linhas 6 a 8 para conectar um banco de dados ORACLE, localizado na URL <nome url>, usando o login <nome do usuário> e <senha>, com efetivação automática de cada comando, e então estabelecer essa conexão como default context para os comandos subseqüentes.

```
1) import java.sql.* ;
2) import java.io.* ;
3) import sqlj.runtime.* ;
4) import sqlj.runtime.ref.* ;
5) import oracle.sqlj.runtime.* ;
   ...
6) DefaultContext cntxt =
7)       oracle.getConnection("<nome url>", "<nome usuário>", "<senha>", true) ;
8) DefaultContext.setDefaultContext(cntxt) ;
```

Importando as classes necessárias para inserir a SQLJ em um programa JAVA no ORACLE, estabelecendo uma conexão e default context.

```
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.

Exemplo

```
//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.

Observe que, como o JAVA já usa o conceito de controlar exceções por erro, uma exceção especial, chamada SQLException, é usada para devolver condições de erro ou exceções após a execução de um comando SQL pelo banco de dados. Ela tem um papel semelhante à da SQLCODE e da SQLSTATE, da SQL embutida. O programa JAVA tem muitos tipos de exceções predefinidas. Cada operação (função) de JAVA tem de estabelecer as exceções que podem ser emitidas – isto é, as condições de exceção que podem ocorrer enquanto se executa o código JAVA daquela operação. No exemplo, o controle de exceção para um SQLException é especificado nas linhas 7 e 8.

Em SQLJ, os comandos de SQL embutidos dentro de um programa JAVA são precedidos por #sql, como ilustrado na linha 3 do exemplo, assim poderão ser identificados pelo pré-processador. A SQLJ usa uma cláusula INTO – similar à usada em SQL embutida – para devolver os valores dos atributos recuperados do banco de dados por uma consulta SQL em variáveis de programa JAVA. Da mesma forma que na SQL embutida, as variáveis de programa são precedidas de dois pontos (:) na declaração SQL.

No exemplo, é selecionada uma tupla isolada pela consulta SQLJ embutida; é por isso que podemos marcar os valores desses atributos diretamente nas variáveis do programa JAVA, da cláusula INTO, na linha 4. Para as consultas que recuperam muitas tuplas, a SQLJ usa o conceito de um iterator, semelhante ao de um cursor da SQL embutida.

SQL/PSM

A SQL/PSM (Structured Query Language/Persistent Stored Modules) foi desenvolvida pela American National Standards Institute (ANSI) como uma extensão da SQL. Foi adotada em 1996 e oferece programação procedural, além dos comandos de consulta da SQL.

Embora a SQL/PSM não seja considerada uma linguagem de programação completa, isso ilustra como os construtores típicos das linguagens de programação

comuns – como os laços (loops) e as estruturas de condição – podem ser incorporados à SQL.

A SQL/PSM é a parte do padrão SQL que especifica como escrever módulos armazenados de modo persistente, incluindo declarações para a criação de funções e procedimentos. Também inclui os construtores adicionais de programação para aumentar o poder da SQL para a escrita do código (ou do corpo) dos procedimentos armazenados e das funções.

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;
```

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.

Considerando o exemplo acima, ele ilustra como a estrutura de desvio condicional pode ser usada em função SQL/PSM. A função devolve um valor de cadeia de caracteres (linha 1) que descreve o tamanho de um departamento baseado no número de empregados. Há um parâmetro inteiro IN, deptnro, que fornece o número do departamento. Uma variável local NroDeEmps é declarada na linha 2. A consulta, das linhas 3 e 4, devolve o número de empregados do departamento, e o desvio condicional das linhas 5 a 8 devolverá, então, um dos valores {“ENORME”, “GRANDE”, “MÉDIO”, “PEQUENO”}, baseado no número de empregados.

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;
```

Também há uma estrutura de laço baseada em cursor. A lista de declaração, em cada volta, é executada para cada tupla do resultado da consulta, uma por vez. Possui a seguinte forma:

```
FOR <nome do laço> AS <nome do cursor> CURSOR FOR <consulta> DO
        <lista de declarações>
END FOR;
```

Os laços podem ter nomes e há uma declaração LEAVE <nome do laço> para quebrar um laço quando uma condição for satisfeita.

Referências Bibliográficas

1. Elmasri, Ramez. “Fundamentals Of Database Systems” / Ramez Elmasri, Shamkant B. Navathe. -- 4th ed. ISBN 0-321-12226-7.
2. Cristovão, Leandro. “Usando SQL no Delphi”, Visual Books, 2001. ISBN: 8575020544.
3. SQL – Wikipédia, a enciclopédia livre - http://pt.wikipedia.org/wiki/SQL (acessado em Setembro/2009).
4. Artigo da SQL Magazine 28: “Visões – Uma abordagem prática.” - http://www.devmedia.com.br (acessado em Setembro/2009).
5. SQL: VIEWS - http://www.techonthenet.com/sql/views.php (acessado em Setembro/2009).
6. Chamberlin, D. D., Astrahan, M. M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay, B. G., Lorie, R., Mehl, J. W., Price, T. G., Putzolu, F., Selinger, P. G., Schkolnick, M., Slutz, D. R., Traiger, I. L., Wade, B. W., and Yost, R. A. 1981. “A history and evaluation of System R. Commun.” ACM 24, 10 (Oct. 1981), 632-646. ISSN: 0001-0782. http://doi.acm.org/10.1145/358769.358784
7. Ramez E. Elmasri & Shamkant Navathe. “Sistemas de Banco de Dados”, Addison-Wesley, 2005. ISBN: 8588639173.
8. Korth, Henry F. “Sistemas de bancos de dados” / Henry F. Korth, Abraham Silberschatz; tradução Maurício Heihachiro Galvan Abe; revisão técnica Sílvio Carmo Palmieri. – 2ª Ed. – São Paulo : MAKRON Books, 1993.