Práticas de Banco de Dados

Parte 4 – Views, Procedures e Funções

Professor Eduardo Xavier

Views (Visões)

- **Definição**
  - Visões (VIEWS) são modos personalizados de se enxergar os dados armazenados no banco de dados, geradas a partir de um comando SELECT feito em uma ou mais tabelas existentes.
- **Características**
  - Embora uma VIEW seja armazenada no catálogo e se apresente a quem a acessa de modo semelhante a uma tabela, existe uma diferença crucial: os dados da visão não são armazenados no BD.
  - Em lugar dos dados, existem uma série de referências que apontam para os dados originais das tabelas onde foi feito o SELECT que compõe a VIEW.
  - Isso dá a VIEW a garantia de que quaisquer alterações sofridas nas tabelas relacionadas a ela estão automaticamente refletidas

Views (Visões)

- Por que adotar o uso de visões?
  - **Para restringir o acesso a dados** – o comando SELECT pode filtrar linhas e colunas que não devam ser mostradas
  - **Para simplificar o acesso a dados** – a VIEW pode encapsular comandos de acesso complexos, facilitando a obtenção de informações e padronizando a forma de acesso
  - **Para garantir independência dos dados** - uma VIEW pode ser usada para encobrir alterações de estruturas, bem como a distribuição física dos dados
  - **Para prover diferentes apresentações dos mesmos dados** – usuários diferentes podem ter necessidades distintas de visualização da mesma informação e o uso de VIEWS pode ser uma alternativa para isso
  - **Para ocultar e padronizar códigos** – a existência de uma VIEW garante que todos os usuários que fazem uso da mesma acessam os dados da mesma maneira e de forma transparente

Views (Visões)

- As **views** também são chamadas de **tabelas virtuais** ou **derivadas**;
  - Os dados nessas tabelas virtuais são derivados de tabelas da base de dados ou outras views previamente definidas;
  - Há possíveis limitações nas operações de atualização que podem ser aplicadas à views, mas não existe quaisquer limitações sobre a consulta de uma view.
    - É preciso estar atendo a colunas “calculadas”, permissões de acesso concedidas à view e às tabelas que a compõem e, finalmente, a restrições de sintaxe do SGBD utilizado.

Views (Visões)

- **Sintaxe de criação**

  CREATE VIEW <nome>
  AS
  <subquery>

- **Sintaxe de eliminação**

  DROP VIEW <nome>

- Onde:
  - <nome> é o nome da VIEW
  - <subquery> é o comando SELECT que gera a VIEW

Views (Visões)

- **Exemplo:**

```
CREATE VIEW empregados-com-dependentes
AS
SELECT emp.matricula, emp.nome, emp.CPF
        FROM empregados
        WHERE emp.matricula IN
                (SELECT DISTINCT dep.mat-empregado
                 FROM dependentes dep )
```

Views (Visões)

- **Outros exemplos:**

```
CREATE VIEW alunos-banco-de-dados
AS
SELECT a.matricula, a.nome FROM alunos
INNER JOIN matriculas m ON a.matricula = m.matric-aluno
WHERE m.codigo-disciplina = 'BD'
```

```
CREATE VIEW produtos-mais-vendidos
AS
SELECT prd.descricao AS 'Produto', vd.COUNT(*) AS 'Quantidade'
FROM produtos prd, vendas vd
WHERE prd.codigo = vd.cod-produto
GROUP BY prd.descricao
```

Procedimentos e Funções Armazenadas

- É possível armazenar no banco de dados blocos de comandos SQL para execução posterior.
  - Por serem armazenados no servidor, a tendência é que este blocos executem com maior rapidez.
  - O armazenamento dos blocos também contribui para reduzir o tráfego de dados na rede (se forem bem planejados), melhorando o desempenho das aplicações.
  - Facilitam o reuso de códigos
  - Porém é preciso lembrar que a carga de processamento agora está no servidor

Procedimentos e Funções Armazenadas

- Estes blocos de comandos SQL podem ser:
  - **STORED PROCEDURES**
    - Procedimentos armazenados que podem ser acionados via comando EXEC <nome-procedimento>
    - Geralmente usadas para encapsular procedimentos complexos.
  - **USER DEFINED FUNCTIONS**
    - Função armazenada que pode ser acionada dentro de um SELECT (como campo a ser exibido ou em cláusulas de controle como WHERE, GROUP BY, ORDER BY, HAVING,...)
  - Geralmente usadas para encapsular lógicas mais simples e de uso comum na aplicação, permitindo que seja adotada por vários processos
- **IMPORTANTE: Estas definições podem variar de acordo com o produto/fabricante adotado. As sintaxes que serão vistas a seguir se encaixam na mesma situação.**

Procedimentos e Funções Armazenadas

Sintaxe PROCEDURE:

CREATE PROCEDURE <nome>
(<tipo-parametro><nome-parâmetro> <tipo-de-dado>, ...)
<bloco de comandos SQL>

Onde:

<nome> é o nome da STORED PROCEDURE

<tipo-parametro> Os parâmetros de um procedimento podem ser de IN (entrada), OUT (saída), ou INOUT (entrada/saída).

<parâmetro> é uma variável que contém informação relevante para o processamento da STORED PROCEDURE.

Parâmetros não são obrigatórios. É possível criar procedimentos e funções que executam sem necessitar de nenhum tipo de informação externa.

<tipo-de-dado> é o formato que o parâmetro deve possuir. O tipo de dado dos parâmetros podem ser de algum tipo de dado válido, por exemplo, INT, CHAR, DATE, etc

<bloco de comandos SQL> são os comandos que serão armazenados

Procedimentos e Funções Armazenadas

- **Exemplo:**

```
CREATE PROCEDURE notas-aluno (IN mat-aluno INT)
BEGIN
        SELECT 'Relatório de Notas do Aluno:', nome
        FROM alunos WHERE matricula = mat-aluno;

        SELECT disciplinas.nome, notas.data-prova, notas.nota-prova
        FROM notas
        INNER JOIN disciplinas
        ON notas.cod-disciplina = disciplinas.cod-disciplina
        WHERE notas.matricula-aluno = mat-aluno
        ORDER BY disciplinas.nome, notas.data-prova;
END;
```

Esta STORED PROCEDURE exibe todas as notas de determinado aluno em todas as disciplinas que o mesmo está ou foi matriculado a partir de seu número de matrícula

Para acioná-la, por exemplo, para o aluno cuja matrícula é 278, basta executar o seguinte comando:

```
CALL notas-aluno(278)
```

Procedimentos e Funções Armazenadas

- **Sintaxe FUNCTION:**

```
CREATE FUNCTION <nome>
(<parâmetro> <tipo-de-dado>, <parâmetro> <tipo-de-dado>, ...)
RETURNS <tipo-de-dado>
[NOT] DETERMINISTIC
<bloco de comandos SQL>
```

Onde:

<nome> é o nome da USER FUNCTION

<parâmetro> é uma variável que contém informação relevante para o processamento da USER FUNCTION.

Parâmetros não são obrigatórios. É possível criar procedimentos e funções que executam sem necessitar de nenhum tipo de informação externa.

<tipo-de-dado> é o formato que o parâmetro deve possuir ou que a USER FUNCTION deve retornar com resposta. Os parâmetros de uma função podem ser apenas IN (entrada). A exemplo dos procedimentos, o tipo dos parâmetros podem ser de algum tipo de dado válido, por exemplo, INT, CHAR, DATE, etc.

[NOT] DETERMINISTIC indica se a função é determinística ou não.

<bloco de comandos SQL> são os comandos que serão armazenados

Procedimentos e Funções Armazenadas

- **Exemplo:**

```
CREATE FUNCTION media-aluno (mat-aluno INT,disciplina CHAR(3))
RETURNS DECIMAL(4,2)
DETERMINISTIC
BEGIN
    DECLARE resposta DECIMAL(4,2);
    SELECT AVG(notas.nota-prova) INTO resposta
    FROM notas
    WHERE          (notas.matricula-aluno = mat-aluno)
    AND                    (notas.codigo-disciplina = disciplina);
    RETURN resposta;
END;
```

Esta USER FUNCTION calcula a média de um determinado aluno em uma determinada disciplina
Para acioná-la, por exemplo, para calcular a média de todos os alunos na disciplina ‘BD1’, poderíamos usar seguinte comando:

```
SELECT nome, media-aluno(matricula,’BD1’) FROM alunos ORDER BY nome
```

Leituras Recomendadas

- Documentação oficial do MySQL:
  - http://dev.mysql.com/doc/refman/5.5/en/create-view.htm
  - http://dev.mysql.com/doc/refman/5.5/en/create-procedure.html
  - http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html
  - http://dev.mysql.com/doc/refman/5.5/en/set-statement.html
- Elmasri, Ramez. Sistemas de Banco de Dados. 6a ed.São Paulo: Pearson Addison Wesley, 2011.
- Silberschatz, Abraham; Korth, Henry F.; Sudarshan, S. Sistema de Banco de Dados. São Paulo: Makron Books, 1999.