SGBD relacional

Paulino Ng

2020-08-19

SGBD relacional (Relational DBMS)

De acordo com BOWMAN, J, et al., Codd e Date definem um SGBD relacional como um sistema que deve:

- representar toda Informação no BD como tabelas;
- manter a representação lógica dos dados independente das características físicas de armazenamento;
- usar uma linguagem de alto nível para estruturar, consultar e modificar as informações no BD;
- dar suporte às principais operações relacionais (seleção, projeção e *join*) e operações em conjuntos como união, interseção, diferença e divisão;
- dar suporte a visões, que permitem ao usuário especificar maneiras alternativas de ver os dados nas tabelas;
- fornecer um método para diferenciar entre valores desconhecidos (**null**) de zeros e valores vazios; e
- dar suporte a mecanismos para controlar a integridade dos dados, a autorização de acesso, a realização de transações e a recuperação de dados.

Modelo Relacional: Tudo É Tabela

Neste modelo, os dados individuais estão na junção de uma linha com uma coluna. A primeira linha da tabela dá nome às colunas, estes nomes devem lembrar o que significam os dados nas linhas seguintes. Os dados numa linha descrevem a ocorrência de uma *entidade* e são, às vezes, chamados de *instâncias*.

Algumas questões de nomenclatura, a tabela a seguir apresenta alguns termos que consideramos "equivalentes" em BD:

tabela	relação	arquivo
linha	tupla	registro
coluna	atributo	campo

Cada coluna da tabela representa diferentes pessoas falando sobre BDs: a primeira é o pessoal de SGBDs relacionais, a segunda, pessoas do mundo acadêmico e a terceira, os programadores de SBDs não relacionais.

Exercício

1. Como sempre ocorre ao estudarmos um novo assunto, aparecem termos, jargões, nesta nova área. É importante nos familiarizarmos com este vocabulário. Comece a fazer um glossário de termos novos, ou com novos significados, num arquivo ou num pedaço de papel e mantenha sempre a mão para inserir novas entradas nele.

Comandos SQL usados nos módulos da aula passada

```
create table DEPARTMENTS (
  deptno          number,
  name            varchar2(50) not null,
  location        varchar2(50),
  constraint pk_departments primary key (deptno)
)
```

Este comando cria no banco de dados uma tabela com o nome **DEPARTMENTS** e as colunas **deptno**, **name** e **location**. As colunas são separadas por vírgulas. Após o nome da coluna vem o tipo do dado nas próximas vamos ver os tipos de dados mais usados em SQL, nas 3 colunas são usados os tipos `number`, `varchar2(50)` e `varchar2(50)`. Estes tipos de dados são usados no SQL da Oracle, eles são ligeiramente diferentes dos que usamos num SQL padrão.

No lugar de `number`, podemos usar `integer(n)` onde n é a quantidade de dígitos do número inteiro. No lugar de `varchar2`, usa-se normalmente `varchar`, mas o `varchar` do SQL sabe diferenciar uma *string* vazia (''') de um valor `null` (ausência de valor), o `varchar2` do SQL da Oracle não sabe a diferença entre estas duas situações. Não devemos usar `varchar` em programas SQL da Oracle pois atualmente o `varchar` se comporta como o `varchar2`, mas a Oracle se reserva o direito de um dia seguir o padrão SQL e o tipo `varchar` poderá diferenciar `null` e".

A palavra chave `constraint` (restrição) serve para introduzir o nome de uma restrição, `constraint` não é o nome de uma coluna. O nome da restrição vem depois do seu anúncio, no caso do comando, o nome da restrição é **pk_departments**. A restrição (de integridade) é que o valor da coluna **deptno** é uma chave primária (*primary key*). Ser uma **chave primária** significa que o valor identifica unicamente uma linha na tabela. Isto é, não existem duas linhas com o mesmo valor de chave primária, toda linha tem de ter um valor na coluna que servede chave primária. Em alguns casos, a chave primária pode ser composta por mais de uma coluna.

Sintaxe do comando de criação de tabelas

Para criar uma tabela, você precisa definir 3 coisas:

1. Seu nome
2. Suas colunas
3. Os tipos de dados destas colunas

A sintaxe básica para criar uma tabela é:

```
create table <nome_da_tabela> (
  <nome_da_coluna1> <tipo_de_dado>,
  <nome_da_coluna2> <tipo_de_dado>,
  <nome_da_coluna3> <tipo_de_dado>,
  ...
)
```

Não estão incluídos nesta sintaxe as restrições.

Segundo create table

```
create table EMPLOYEES (
  empno                 number,
  name                  varchar2(50) not null,
  job                   varchar2(50),
  manager               number,
  hiredate              date,
  salary                number(7,2),
  commission            number(7,2),
  deptno                number,
  constraint pk_employees primary key (empno),
  constraint fk_employees_deptno foreign key (deptno)
      references DEPARTMENTS (deptno)
)
```

Análise do 2º comando

Temos 3 novidades:

1. `number(7,2)` diz que o número tem 7 casas (dígitos) sendo duas delas depois da vírgula.
2. O tipo de dados `date` usado para guardar datas.
3. Além da restrição de chave primária, temos agora, a chave estrangeira.

Chave Estrangeira (`foreign key`)

Uma **chave estrangeira** é um dado que referencia um dado que é uma chave primária numa outra tabela. Assim, os dados da coluna **deptno** da tabela **EMPLOYEES** referenciam os dados da coluna **deptno** da tabela **DEPARTMENTS**. Observe que o nome das colunas nas duas tabelas é o mesmo, mas não precisam ser.

Observe que os dados da chave primária da tabela **DEPARTMENTS** aparecem de novo na tabela **EMPLOYEES**, chamamos esta repetição de redundância. É uma forma de redundância necessária, mas que torna difícil manipular com os dados. Imagine que você remove uma linha da tabela **DEPARTMENTS**, o que deve acontecer com as linhas da tabela **EMPLOYEES** que referenciam o valor de **deptno** da linha apagada?

Terceiro e quarto comandos, *Trigger*

```
create or replace trigger  DEPARTMENTS_BIU
    before insert or update on DEPARTMENTS
    for each row
begin
    if inserting and :new.deptno is null then
        :new.deptno := to_number(sys_guid(),
          'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX');
    end if;
end;
create or replace trigger EMPLOYEES_BIU
    before insert or update on EMPLOYEES
    for each row
begin
    if inserting and :new.empno is null then
        :new.empno := to_number(sys_guid(),
            'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX');
    end if;
end;
```

O que é um *Trigger*?

Um *trigger* em SQL é um *gatilho* que lança uma ação. No caso das duas instruções anteriores, elas criam um gatilho que é disparado antes de uma operação de `insert` ou `update` (operações de inserção ou modificação de dados) nas tabelas.

O código que vem depois `for ... end;` diz o que fazer (a ação). Ele está em PL/SQL, *Procedural Language/SQL*. Não vamos nos aprofundar nesta linguagem de programação. Ela é a linguagem que a Oracle utiliza para programar as ações nos *triggers*.

Nos 2 *triggers*, o código diz para cada linha inserida ou modificada, se o valor da coluna de chave primária de cada uma das tabelas não estiver definido no comando, é para chamar uma função de sistema `sys_guid()` que vai calcular um número com a quantidade dígitos dado pela quantidade de `X`. Esta função é chamada de *General User ID* e serve para gerar um número único.

Comando `insert`

O comando para inserir dados na tabela (inserir uma linha na tabela) é o `insert`.

A sintaxe do insert é:

```
insert into <nome_da_tabela>
(lista_de_colunas)
values
(lista_de_valores)
```

ou

```
insert into <nome_da_tabela> values
(valores_para_todas_as_colunas_na_ordem_do_create)
```

Comando `select`

O comando `select` serve para visualizar os dados de uma tabela. Sua forma mais simples é:

```
select * from <nome_da_tabela>
```

Este comando mostra o conteúdo de uma tabela. O `*` significa todas as colunas. Podemos mudar a ordem das colunas e escolher quais colunas com:

```
select <lista_de_colunas> from <nome_da_tabela>
```