BANCO DE DADOS

Parte - 4

A Linguagem SQL

Introdução

- Desenvolvida pelo depto de pesquisa da IBM na década de 1970 (System R)
- Linguagem padrão de BD Relacionais;
- Apresenta várias padrões evolutivos: SQL86, SQL89(SQL1), SQL92 (SQL2), SQL99(SQL3)
- A última versão definida pela ANSI/ISO traz características novas como: store procedures, triggers, suporte à programação OO, entre muitas outras.
- Diferentes fornecedores de SGBDS apresentam versões de SQL com algumas particularidades

Características comuns

- Estilo declarativo, não procedimental;
- Permite otimizações;
- Utilizadas por várias classes de usuários
- Sintaxe simples e bem definida
- Presente em todos os SGBDs Relacionais
- É incorporada comumente a uma outra linguagem;
- Não é uma linguagem completa como C, Java ou Delphi;
- Portável entre sistema operacionais;

Algumas Funcionalidades

- Uma série de comandos DDL para definição de dados
- Uma série de comandos DML para manipulação de dados
- Uma versão de SQL embutida em linguagens de programação
- Instruções para definição de visões (tabelas virtuais)
- Instruções para controle de autorização de acesso
- Instruções para controle de transações e concorrência
- Instruções para especificação de restrições de integridade

Instruções SQL/DDL

CREATE TABLE - Permite a criação de uma tabela

```
CREATE TABLE Emp
   (EmpNume          integer(5) not null,
    EmpNome          char(30) not null,
    EmpFunc          char(20) not null,
    DepNume          integer(4) not null,
    EmpComi          integer(10,2),
    EmpSala          integer(10,2),
    primary key (EmpNume),
    foreign key has (DepNume) references Dept
    on delete restrict
    on update cascade
   );
```

ALTER TABLE - Permite a alteração de uma tabela

```
alter table dept ADD (depsala  integer(4));
```

DROP TABLE - Permite a exclusão de uma tabela

```
Drop table EMP
```

CREATE INDEX - Permite a criação de índice em uma tabela

```
CREATE unique INDEX EmpNum on Emp(EmpNume
asc);
```

A Instrução SELECT

Permite a recuperação de dados em uma ou mais tabelas.

Forma básica:

```
SELECT <lista de colunas>
FROM <lista de tabelas>
WHERE <critério de seleção>
```

Exemplos:

```
SELECT EmpNome, EmpSala
FROM EMP
WHERE DepNume > 10;

SELECT * FROM DEPT;

SELECT nome, matricula
FROM cadastro
WHERE nome like “%Silva%”;
```

Alguns operadores usados na cláusula WHERE:

- `BETWEEN .. AND ...` (Entre dois valores (inclusive))
- `IN (....)` (Lista de valores)
- `LIKE` (Com um padrão de caracteres)
- `IS NULL` (É um valor nulo)

Operadores negativos:

- `<>` (Diferente)
- `NOT BETWEEN` (Não entre 2 valores informados)
- `NOT IN` (Não existente numa lista de valores)
- `NOT LIKE` (Diferente do padrão de caracteres)
- `IS NOT NULL` (Não é um valor nulo)

Funções de agregação

- AVG () (Média do valor n, ignorando nulo)
- COUNT () (Número de registros)
- MAX () (Maior valor)
- MIN () (Menor valor)
- SUM () (Soma dos valores de n, ignorando nulo)

Estas funções podem ser usadas em conjunto com as seguintes cláusulas:

- GROUP BY (Divide tuplas em grupos menores)
- HAVING (Especifica grupos que são manipulados)

Exemplo:

```
SELECT cod_curso, count(*)
FROM matricula
GROUP BY cod_curso
HAVING count(*) > 10
```

(lista os cursos com mais de 10 alunos)

Produto Cartesiano

- Segue a mesma fundamentação da álgebra dos conjuntos

Exemplo:

```
SELECT nome_curso, nome_aluno
FROM matricula, curso
```

Produto Cartesiano com predicado de junção

- Permite associar tuplas de duas tabelas pela relação entre suas chaves primária e estrangeira;
- Produz resultado equivalente à cláusula JOIN que será abordada adiante;

Exemplo:

```
SELECT nome_curso, nome_aluno
FROM matricula, curso
WHERE matricula.cod_curso = curso.cod_curso
```

- `(matricula.cod_curso = curso.cod_curso)` é o predicado de junção

Junções

- Maneira eficiente de consulta para dados de múltiplas tabelas
- Tipos de Junção:

```
INNER JOIN
OUTER JOIN
LEFT OUTER JOIN
FULL OUTER JOIN
```

- Exemplo:

```
SELECT   customer.cust_id_n,
         customer.cust_name_s,
         phone.phone_phonenum_s,
         phone.phone_type_s
FROM     customer JOIN phone
ON       customer.cust_id_n =

         phone.phone_custid_fn
```

Combinando Resultados de Diferentes Consultas

- Permite que as operações básicas sobre conjunto possam ser aplicadas a conjuntos de tuplas.
- As cláusulas que define esta operação são:

```
UNION
INTERSECT
EXCEPT
```

- Exemplo:

```
SELECT status_code_s, status_desc_s
FROM   status_1
UNION
SELECT status_code_s, status_desc_s
FROM   status_1
```

- Todas as 3 cláusula suprimem tuplas repetidas da relação gerada.

A Instrução INSERT

Permite a inserção de dados em uma tabela.

Estrutura básica:

```
INSERT INTO <tabela> [<campos>]
VALUES <valores>;
```

Estrutura completa:

```
INSERT INTO <table_or_view_name>
[(<column_name>,...)]
{ {VALUES (<literal> |
    <expression> |
    NULL |
    DEFAULT,...)} |
  {<select_statement>} }
```

Exemplo:

```
INSERT INTO DEPT (DepNome, DepOrca, DepLoc, DepNume)
VALUES ('Informática', 100.000, 'Prédio A', 100);
```

A Instrução UPDATE

Permite a alteração de dados em uma tabela.

Estrutura básica:

```
UPDATE <tabela>
SET <campo> = <expressão>
WHERE <condição>;
```

Estrutura completa:

```
UPDATE <table_or_view_name>
SET {<column_name> = <literal> |
     <expression> |
     (<single_row_select_statement>) |
       NULL |
       DEFAULT,...}
[WHERE <predicate>]
```

Exemplo:

```
UPDATE EMP
SET EMPSALA = EMPSALA * 1,2
WHERE EMPSALA < 1000;
```

A Instrução DELETE

Permite a exclusão de dados em uma tabela.

Estrutura básica:

```
DELETE FROM <tabela>
WHERE <condição>;
```

Estrutura completa:

```
DELETE FROM <table_or_view_name>
WHERE <predicate>
```

Exemplo:

```
DELETE FROM EMP
WHERE EMPSALA
```

Outras Instruções

MERGE - Permite a combinação das instruções INSERT e UPDATE

Estrutura :

```
MERGE INTO [<qualifier>.]<table_name1>
USING [<qualifier>.]<table_name2> ON (<condition>)
WHEN MATCHED THEN
    UPDATE SET {<column> = {<expression> |
                DEFAULT},...}
WHEN NOT MATCHED THEN
    INSERT [(<column>,...)] VALUES (<expression> |
                        DEFAULT,...);
```

Outras Instruções

TRUNCATE - equivalente ao DELETE, porém mais rápido. Não pode ser utilizado com a cláusula WHERE. Não permite exclusão em quando a tabela é referenciada por uma chave estrangeira.

Estrutura :

```
TRUNCATE TABLE <table_name>
```

Como é mais eficiente que o DELETE, uma dica é excluir as integridades de referência a chave estrangeira para executar o TRUNCATE. Exemplo:

```
ALTER TABLE ORDER_LINE DISABLE CONSTRAINT
FK_ORDLINE_PRODUCT;
TRUNCATE TABLE PRODUCT
```

SUBCONSULTAS ANINHADAS

Mecanismos para aninhamento de consultas são suportados pelo SQL:

IN

```
select nome
from piloto p
where cod_piloto in (select cod_piloto
                     from resultado)
```

(Relação do pilotos que pontuaram)

NOT IN

```
select nome
from equipe p
where cod_equipe not in (select cod_equipe
                         from piloto)
```

(Relação de equipes sem pilotos)

SOME

```
select nome
from aluno
where cod_turma = 1 and
      nota > some(select nota
                  from aluno
                  where cod_turma = 2)
```

(Relação dos alunos da turma 1 cuja nota é maior que alguma nota da turma 2)

ALL

```
select nome
from aluno
where cod_turma = 1 and
      nota > all(select nota
                 from aluno
                 where cod_turma = 2)
```

(Relação dos alunos da turma 1 cuja nota é maior que todas as notas da turma 2)

EXISTS

```
select nome
from piloto p
where exists (select *
              from resultado r
              where p.cod_piloto = r.cod_piloto)
```

(Relação do pilotos que pontuaram)

NOT EXISTS

```
select nome
from piloto p
where not exists (select * from resultado r
                  where p.cod_piloto=r.cod_piloto and
                        r.colocaca_final= 1)
```

(Relação do pilotos que não venceram corridas)

Vale observar que a cláusula EXISTS pode ser facilmente substituída pela cláusula IN. Porém, deve-se estar atento para as particularidades de cada uma delas.

PARTE - II

PostgreSQL

INTRODUÇÃO

- É um sistema de gerenciamento de banco de dados objeto-relacional (SGBDOR);
- Desenvolvido no Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley;
- Aborda conceitos como:
  Herança
  Tipos de dado
  Funções
- Outras funcionalidades fornecem poder e flexibilidade adicionais:
  Restrições
  Gatilhos
  Regras
  Integridade da transação

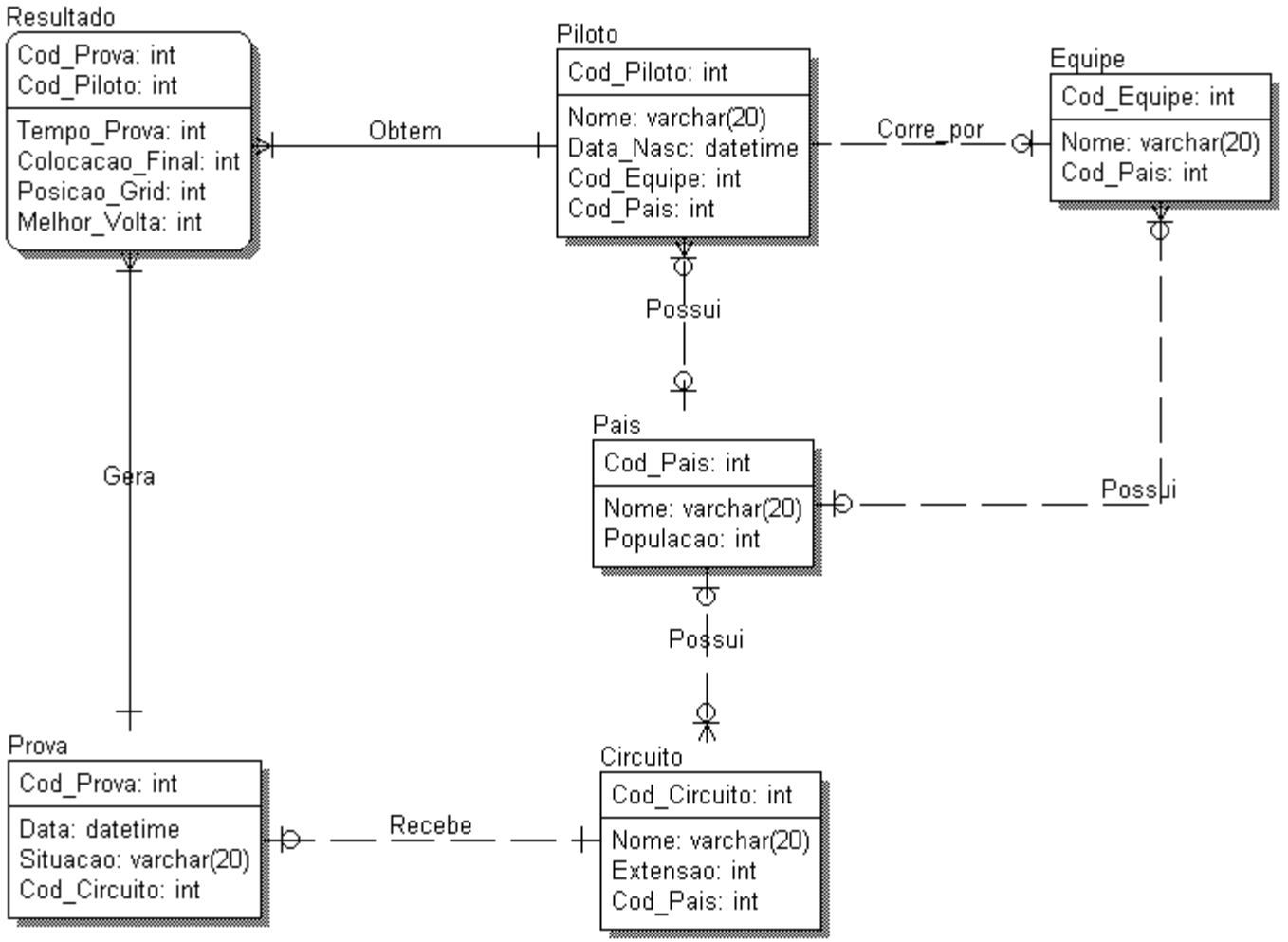
ARQUITETURA

- O PostgreSQL utiliza o modelo cliente-servidor
- Um processo servidor, o qual gerencia os arquivos do banco de dados, recebe as conexões das aplicações cliente com o banco de dados, e executa as ações no banco de dados em nome dos clientes.
- O programa servidor de banco de dados chama-se postmaster.
- A aplicação cliente do usuário (frontend) que executam operações de banco de dados podem ter naturezas muito diversas: o cliente pode ser uma ferramenta no modo caractere, uma aplicação gráfica, um servidor Web que acessa o banco de dados para mostrar páginas na Web, ou uma ferramenta especializada para manutenção do banco de dados.
- É típico em aplicações cliente-servidor o cliente e o servidor estarem em máquinas diferentes. Neste caso se comunicam através de uma conexão de rede TCP/IP

SQL - Prática

EXERCÍCIO

Baseado no modelo E-R, abordado em sala, para a Fórmula 1. Aplique todos os passos para o mapeamento E-R -> Relacional.



Exercícios

Baseado no modelo que representa um campeonato de Fórmula 1, elabore consultas em SQL para prover as seguintes informações:

-- Pilotos por Equipe
-- Pilotos por Pais
-- Pilotos por Pais utilizando a cláusula
-- Quantidade de Pilotos por Pais
-- Os 3 países com maior quantidade de pilotos
-- Relação de Pilotos com seus respectivos países
-- Quantidade de pontos por piloto
-- Quantidade de pontos por equipe
-- Vencedor do GP da Malásia
-- Pilotos que mais pontuaram
-- Pilotos que pontuaram
-- Pilotos que ainda não pontuaram
-- Pilotos com maior quantidade de pontos
-- Pais como maior número de pilotos
-- Equipe que nunca corre em casa
-- Pilotos que pontuaram em casa