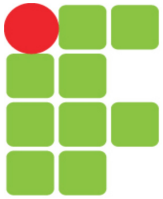

INSTITUTO FEDERAL
PARANÁ

Ministério da Educação

Banco de Dados II

Prof. Alcione Benacchio

Referências Bibliográficas

http://www.postgresql.org/docs/9.1
http://www.postgresql.org.br/

PostgreSQL

Características:

- Banco de dados relacional
- Suporte a transações
- Lock por registro
- Integridade referencial
- Extensão para GIS (Dados georreferenciados)
- Acesso via drivers ODBC e JDBC
- Interface de Gerenciamento Gráfico e via Shell
- Uso otimizado de recursos do Sistema Operacional
- Recursos de Triggers, views e functions

PostgreSQL

Características:

- Tablespace
- Backup Online
- Backup Offline
- Transações com Savepoint e Two-Phase Commit
- Mecanismo próprio de logs
- Autovacuum integrado

PostgreSQL

Limitações:

Tamanho máximo de um banco de dados: Ilimitado
Tamanho máximo de uma tabela: 32 TB
Tamanho máximo de um registro: 1.6 TB
Tamanho máximo de um campo: 1 GB
Máximo de linhas numa tabela: Ilimitado
Máximo de colunas numa tabela: 1600
Máximo de índices numa tabela: Ilimitado

PostgreSQL

Interfaces de Acesso ao Banco de Dados:

- C/C++
- Java (JDBC)
- PHP, JSP, ColdFusion
- Perl
- Python
- Delphi

PostgreSQL

Para que um sistema possa se conectar a um servidor de banco de dados, ele precisa montar uma url de conexão, que deve conter:

- **USER**: Usuário para conexão
- **PASSWORD**: Senha para a conexão
- **DATABASE**: Nome do Banco de dados a ser acessado
- **PORT**: Porta de conexão
- **HOSTNAME**: Endereço IP ou nome do servidor

Interface de Acesso ao BD via Shell

psql: é o modo interativo do PostgreSQL para acesso e manipulação dos banco de dados via shell.
-h: hostname (localhost) máquina local
-U: user (postgres) usuário administrador do banco
-p: port (5432) porta padrão do postgres
-W: força o usuário a digitar a senha
nomebd: nome do banco de dados a qual irá se conectar

psql -h localhost -U postgres -p 5432 -W nomebd

SQL - DDL

DDL - Data Definition Language (Linguagem de Definição de Dados)

É um subconjunto de instruções SQL, que são declarações utilizadas para definir estruturas de um banco de dados, como adicionar colunas, tabelas, índices entre outros.

Comandos DDL SQL:

Create: Cria objetos em um SGBD. Como por exemplo, tabelas, índices ou visões.

Drop: Remove objetos um banco de dados, tabelas, índices ou visões.

Alter: Altera as definições de um objeto existente no banco de dados.

SQL - DDL

CREATE DATABASE: Cria um banco de dados.

Para criar um banco de dados, você deve ser um superusuário ou ter o privilégio CREATEDB.

Declarações:

OWNER: Especifica o dono do banco de dados.

TABLESPACE: Especifica o Tablespace onde será armazenado o banco de dados.

Exemplos:

```
CREATE DATABASE ifpr;
CREATE DATABASE ifpr OWNER postgres;
CREATE DATABASE ifpr TABLESPACE tb_home;
```

SQL - DDL

DROP DATABASE: Remove um banco de dados.

Só pode ser executado pelo proprietário do banco ou pelo superusuário *postgres*. Além disso, não pode ser executado enquanto você ou alguém está conectado ao banco de dados que deseja remover.

Exemplos:

DROP DATABASE ifpr; *(Remove o banco de dados ifpr)*

DROP DATABASE IF EXISTS ifpr; (Apenas se existir, remove o banco de dados ifpr, muito utilizado na construção de scripts).

SQL - DDL

CREATE TABLESPACE: Cria um novo espaço de armazenamento(Cluster de dados) no destino especificado.

Declarações:

LOCATION: Endereço destino de criação do tablespace.

OWNER: Proprietário do tablepsace.

Exemplos:

CREATE TABLESPACE nometbs LOCATION '/home/postgres/nometbs';

CREATE TABLESPACE nometbs LOCATION '/home/postgres/nometbs' OWNER postgres;

SQL - DDL

DROP TABLESPACE: Remove um espaço de armazenamento.

- Um espaço de armazenamento só pode ser removido pelo seu dono ou um superusuário.
- Este deve estar vazio de todos os objetos de banco de dados antes que ele possa ser descartado.
- É possível que objetos em outros bancos de dados possam residir no tablespace mesmo se não há objetos no banco de dados atual está usando a tabela.

```
DROP TABLESPACE nomtbs;
```

CREATE TABLE

Tipos de Dados
Inteiros: smallint, integer, bigint
Automáticos: serial, bigserial
Reais: real, float, double precison
Data e Hora: date, time, datetime, timestamp

CRIA SEQUENCE AUTOMÁTICAMENTE

```
CREATE TABLE distributors (
    did    serial primary key,
   name varchar(40) DEFAULT 'Luso Films',
   modtime timestamp DEFAULT current_timestamp
);
```

DEFINE SEQUENCE JÁ EXISTENTE

```
CREATE TABLE distributors ( did integer DEFAULT
nextval('distributors_serial'),     name      varchar(40) DEFAULT 'Luso
Films', modtime timestamp DEFAULT current_timestamp );
```

CREATE TABLE

CHAVE PRIMÁRIA NO INÍCIO
CREATE TABLE distributors (did integer PRIMARY KEY, name varchar(40));

CHAVE PRIMÁRIA NO FINAL
CREATE TABLE distributors (did integer, name varchar(40), PRIMARY KEY(did));

CHAVE PRIMÁRIA COMPOSTA
CREATE TABLE films (code char(5), title varchar(40), did integer, date_prod date, kind varchar(10), len interval hour to minute, CONSTRAINT code_title PRIMARY KEY(code,title));

CREATE TABLE

CHAVE PRIMÁRIA NO FINAL
CREATE TABLE distributors (did serial, name varchar(40), PRIMARY KEY(did));

DEFININDO INTEGRIDADE REFERENCIAL
CREATE TABLE product (pid serial, did integer, description varchar(40), FOREIGN KEY(did) REFERENCES distribuitors(did));

DEFININDO INTEGRIDADE REFERENCIAL COM PREENCHIMENTO OBRIGATÓRIO
CREATE TABLE product (pid serial, did integer NOT NULL, description varchar(40), FOREIGN KEY(did) REFERENCES distribuitors(did));

CREATE TABLE

UNIQUE CONSTRAINT
UNIQUE garante que o conteúdo de uma coluna, ou grupo de colunas é único em relação à tabela. É criado um índice implícito para garantir a unicidade. UNIQUE admite valores nulos.

```
CREATE TABLE pessoa (
    pessoa_id serial primary key,
    nome varchar(80) not null,
    rg varchar(10) UNIQUE);
```

ALTER TABLE

Adiciona a coluna ADDRESS a tabela distributors
ALTER TABLE distributors ADD COLUMN address varchar(30);
ALTER TABLE distributors ADD COLUMN ifpr varchar(20);

Remove a coluna IFPR da tabela distributors
ALTER TABLE distributors DROP COLUMN ifpr;

Modifica o tipo de dados das colunas ADDRESS e NAME ALTER TABLE distributors ALTER COLUMN address TYPE varchar(80), ALTER COLUMN name TYPE varchar(100);

Modifica o nome da coluna ADRESS para CITY ALTER TABLE distributors RENAME COLUMN address TO city;

Troca o nome tabela DISTRIBUTORS para SUPPLIERS ALTER TABLE distributors RENAME TO suppliers;

DROP TABLE

Apaga a tabela FILMS
DROP TABLE films;

Apaga a tabela FILMS e DISTRIBUTORS
DROP TABLE films, distributors;

SQL - DDL

Download da Base Pagila

- http://pgfoundry.org/frs/?group_id=1000150

Importando dados Pagila:

- unzip pagila-0.10.1.zip

\i pagila-schema.sql

\i pagila-insert-data.sql

SQL - SELECT

```
SELECT
   campos
FROM
   tabela
{INNER JOIN}
{LEFT JOIN}
WHERE
GROUP BY
HAVING
```

SQL - SELECT

```
SELECT
  campos
FROM
  tabela
{INNER JOIN}
{LEFT JOIN}
WHERE
ORDER BY
LIMIT - OFFSET
```

SQL - SELECT

Select simples:

```
SELECT * FROM city;
```

```
 city_id |          city           | country_id |     last_update
---------+-------------------------+------------+---------------------
       1 | A Corua (La Corua)      |         87 | 2006-02-15 09:45:25
       2 | Abha                    |         82 | 2006-02-15 09:45:25
       3 | Abu Dhabi               |        101 | 2006-02-15 09:45:25
       4 | Acua                    |         60 | 2006-02-15 09:45:25
       5 | Adana                   |         97 | 2006-02-15 09:45:25
       6 | Addis Abeba             |         31 | 2006-02-15 09:45:25
       7 | Aden                    |        107 | 2006-02-15 09:45:25
       8 | Adoni                   |         44 | 2006-02-15 09:45:25
```

SQL - SELECT

Select especificando colunas:

```
SELECT city_id, city FROM city;
```

```
 city_id |          city
---------+--------------------------
       1 | A Corua (La Corua)
       2 | Abha
       3 | Abu Dhabi
       4 | Acua
       5 | Adana
       6 | Addis Abeba
       7 | Aden
       8 | Adoni
       9 | Ahmadnagar
      10 | Akishima
```

SQL - SELECT

Select com expressões:

```
SELECT customer_id, amount * 0.30
as lucro FROM payment;
```

```
 customer_id | lucro
-------------+--------
         269 | 0.5970
         269 | 0.2970
         269 | 2.0970
         269 | 0.2970
         269 | 1.4970
         269 | 0.8970
         270 | 0.5970
```

SQL - SELECT

Select eliminando duplicados com **DISTINCT:**

```
SELECT uf FROM cidade;

Uf
---
SP
SP
RJ
RJ
```

```
SELECT DISTINCT uf FROM cidades;

Uf
----
SP
RJ
```

SQL - SELECT

Select com expressões:

```
SELECT customer_id, amount * 0.30
as lucro FROM payment;
```

```
 customer_id | lucro
-------------+--------
         269 | 0.5970
         269 | 0.2970
         269 | 2.0970
         269 | 0.2970
         269 | 1.4970
         269 | 0.8970
         270 | 0.5970
```

SQL - SELECT

Select limitando os resultados:

Seleciona 10 linhas da tabela a partir da primeira linha.

```
SELECT * FROM cidade LIMIT 10;
```

Seleciona 10 linhas da tabela a partir da décima primeira linha, ou seja de 11 a 20.

```
SELECT * FROM cidade LIMIT 10 OFFSET 10;
```

SQL - SELECT

Select utilizando clausula **WHERE:**

```
SELECT
    film_id,
    category_id
FROM
    film_category
WHERE
    category_id = 9;
```

```
 film_id | category_id
---------+-------------
       6 |           9
      11 |           9
      15 |           9
      16 |           9
      47 |           9
      52 |           9
     100 |           9
```

SQL - SELECT

Select utilizando operador **like/ilike:**

```
SELECT
   city_id,
   city
FROM
   city
WHERE
   city like 'S%';
```

```
 city_id |            city
---------+-----------------------------
     439 | Saarbrcken
     440 | Sagamihara
     441 | Saint Louis
     442 | Saint-Denis
     443 | Sal
     444 | Salala
     445 | Salamanca
```

Obs.: ilike para operações case insensitve.

SQL - SELECT

Select utilizando operador **BETWEEN:**

```
SELECT
    city_id,
    city
FROM
    city
WHERE
    city_id BETWEEN 1 AND 10;
```

```
 city_id |        city
---------+--------------------
       1 | A Corua (La Corua)
       2 | Abha
       3 | Abu Dhabi
       4 | Acua
       5 | Adana
       6 | Addis Abeba
       7 | Aden
       8 | Adoni
       9 | Ahmadnagar
      10 | Akishima
```

SQL - SELECT

Select utilizando operador **IN:**

```
SELECT
   city_id,
   city
FROM
   city
WHERE
   city_id IN(2,4,6);
```

```
 city_id |        city
---------+--------------------
       2 | Abha
       4 | Acua
       6 | Addis Abeba
```

SQL - SELECT

Select utilizando operador **IS/NULL/NOT:**

```
SELECT city_id, city FROM city WHERE city IS NULL;
```

```
SELECT city_id, city FROM city WHERE city IS NOT NULL;
```

SQL - SELECT

Select utilizando clausula **ORDER BY:**

Ordenação pela coluna.

```
SELECT
    city_id,
    city
FROM
    city
ORDER BY
    City ASC;
```

```
 city_id |        city
---------+--------------------
       1 | A Corua (La Corua)
       2 | Abha
       3 | Abu Dhabi
       4 | Acua
       5 | Adana
       6 | Addis Abeba
       7 | Aden
       8 | Adoni
       9 | Ahmadnagar
      10 | Akishima
```

- **ASC**: Ascendente
- **DESC**: Descendente

Associando Dados

Utilizando uma base de dados bem modelada, é comum a necessidade de associar dados entre várias tabelas. Essa associação foi definida através do uso das chaves primárias e estrangeiras.

No SQL esse processo se chama JOIN.

A função do JOIN é agrupar um ou mais conjuntos de dados que, agrupados, formam um novo conjunto, que pode conter informações das tabelas que foram associadas.

O produto cartesiano, gerado entre as combinações das tabelas pode ser filtrado através do WHERE.

Associando Dados

INNER JOIN: Cria um subconjunto de um produto cartesiano entre conjuntos de dados. Este requer clausulas condicionais para especificar um critério no qual dois registros são ligados.

```
SELECT * FROM film INNER JOIN language USING (language_id);
```

```
language_id          | 1
film_id              | 1
title                | ACADEMY DINOSAUR
description          | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
release_year         | 2006
original_language_id |
rental_duration      | 6
rental_rate          | 0.99
length               | 86
replacement_cost     | 20.99
rating               | PG
last_update          | 2007-09-10 17:46:03.905795
special_features     | {"Deleted Scenes","Behind the Scenes"}
fulltext             | 'academi':1 'battl':15 'canadian':20 'dinosaur':2 'drama':5 'epic':4 'feminist':8 'mad':11 'must':14 'rocki':21 'scientist':12 'teacher':17
name                 | English
last_update          | 2006-02-15 10:02:19
```

Associando Dados

INNER JOIN:

```
SELECT * FROM film INNER JOIN language ON language.language_id =
film.language_id;
```

```
film_id              | 1
title                | ACADEMY DINOSAUR
description          | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
release_year         | 2006
language_id          | 1
original_language_id |
rental_duration      | 6
rental_rate          | 0.99
length               | 86
replacement_cost     | 20.99
rating               | PG
last_update          | 2007-09-10 17:46:03.905795
special_features     | {"Deleted Scenes","Behind the Scenes"}
fulltext             | 'academi':1 'battl':15 'canadian':20 'dinosaur':2 'drama':5 'epic':4 'feminist':8 'mad':11 'must':14 'rocki':21 'scientist':12 'teacher':17
language_id          | 1
name                 | English
last_update          | 2006-02-15 10:02:19
```

SQL - Operadores

Operadores de Texto

- Relacionais

  =(igual), != ou <>(diferente), >= (maior ou igual), <= (menor ou igual).

- Concatenação:

  || -> select first_name || ' - ' || last_name from actor;

Operadores Matemáticos

- +, -, /, *
- % Resto de divisão

SQL – Conversão de Tipos

Executa uma operação numérica e converte o resultado para varchar.

- `SELECT (1+1)::varchar;`

Concatena duas strings e realiza a conversão para numérico e realiza uma operação matemática.

- `SELECT (('1'||'2')::integer)+1;`

SQL – Funções de Agregação

As funções de agregação são utilizadas para gerar valores agrupados, ou seja, utiliza um mecanismo para integrar as linhas e retornar um valor relacionado as linhas selecionadas.

- SUM(coluna): efetua soma dos valores da coluna
- AVG(coluna) calcula a médio dos valores de uma coluna
- COUNT(coluna): conta o número de valores não nulos
- MAX(coluna): busca o maior valor da coluna
- MIN(coluna): busca o menor valor da coluna

SQL – Funções de Agregação

Qual a quantidade de registros na tabela actor?

- `SELECT count(*) FROM actor;`

Qual o total de locações na tabela payment?

- `SELECT SUM(amount) FROM payment;`

SQL – A Cláusula GROUP BY

GROUP BY: Responsável por agrupar linhas que tenham o mesmo valor, reduzindo um conjunto de linhas de um grupo que represente todas as linhas, eliminando redundâncias e aplicando funções de agregações.

```
SELECT
    colunas
FROM
    tabela
WHERE
    condição
GROUP BY colunas;
```

SQL – A Cláusula GROUP BY

GROUP BY: Total gasto por cada cliente em locações.

```
SELECT
    customer_id,
    sum(amount)
FROM
    payment
GROUP BY
    customer_id;
```

SQL – A Cláusula HAVING

HAVING: Deve ser utilizado sempre após a clausula GROUP BY, e serve para eliminar resultados de forma similar a clausula WHERE, mas baseado no resultado do agrupamento.

```
SELECT
    customer_id, sum(amount)
FROM
    payment
GROUP BY
    customer_id
HAVING
    sum(amount) > 150;
```

Apresenta apenas os resultados onde a soma é superior 150.

SQL – SUB-SELECT

O uso de sub-selects pode ser apropriado para restringir resultados em uma outra consulta, também pode ser aplicado como especificação de espécie de tabela temporária ou mesmo de colunas temporárias.

```
SELECT coluna1, (SELECT ) as coluna2

FROM tabela1, (SELECT ) as tabela2

WHERE condicao (SELECT )
```

SQL – SUB-SELECT

1 - Lista apenas os clientes que tiveram média de pagamento de 5.

```
SELECT customer_id, AVG(amount) FROM payment GROUP BY customer_id HAVING AVG(amount) > 5 ORDER BY customer_id;
```

2 – Seleciona apenas a coluna customer_id do agrupamento de média de pagamento.

```
SELECT customer_id FROM (SELECT customer_id, AVG(amount) FROM payment GROUP BY customer_id HAVING AVG(amount) > 5 ORDER BY customer_id) as lista
```

SQL – SUB-SELECT

3 – Apresenta o nome completo dos clientes que tiveram média de pagamento superior a 5 ordenado por first_name.

```
SELECT first_name, last_name FROM customer
WHERE customer_id IN (SELECT customer_id FROM
(SELECT customer_id, AVG(amount) FROM payment
GROUP BY customer_id HAVING AVG(amount) > 5)
as lista) ORDER BY first_name;
```

SQL – INSERT

INSERT – Realiza a inserção de registros.

- Ex1.:

```
INSERT INTO cidade (nome, uf) VALUES('IRATI', 'PR');
```

- Ex2.: Especifica estado como NULL.

```
INSERT INTO cidade (nome, uf) VALUES('IRATI', null);
```

SQL – UPDATE

UPDATE – Atualiza a informação de registros.

- Ex1.:

```
UPDATE cidade SET nome = 'IRATI' WHERE codigo = 3;
```

- Ex2.:

```
UPDATE cidade SET nome = 'IRATI', uf = 'PR' WHERE
codigo = 3;
```

SQL – DELETE

DELETE – Exclui registros.

- Ex1.:

```
DELETE FROM cidade WHERE codigo = 1;
```

- Ex2.:

```
DELETE FROM cidade WHERE codigo >= 10;
```