

# SQL

PostgreSQL

## I – Criação de Tabelas

Disciplina: Banco de Dados e suas  
Aplicações

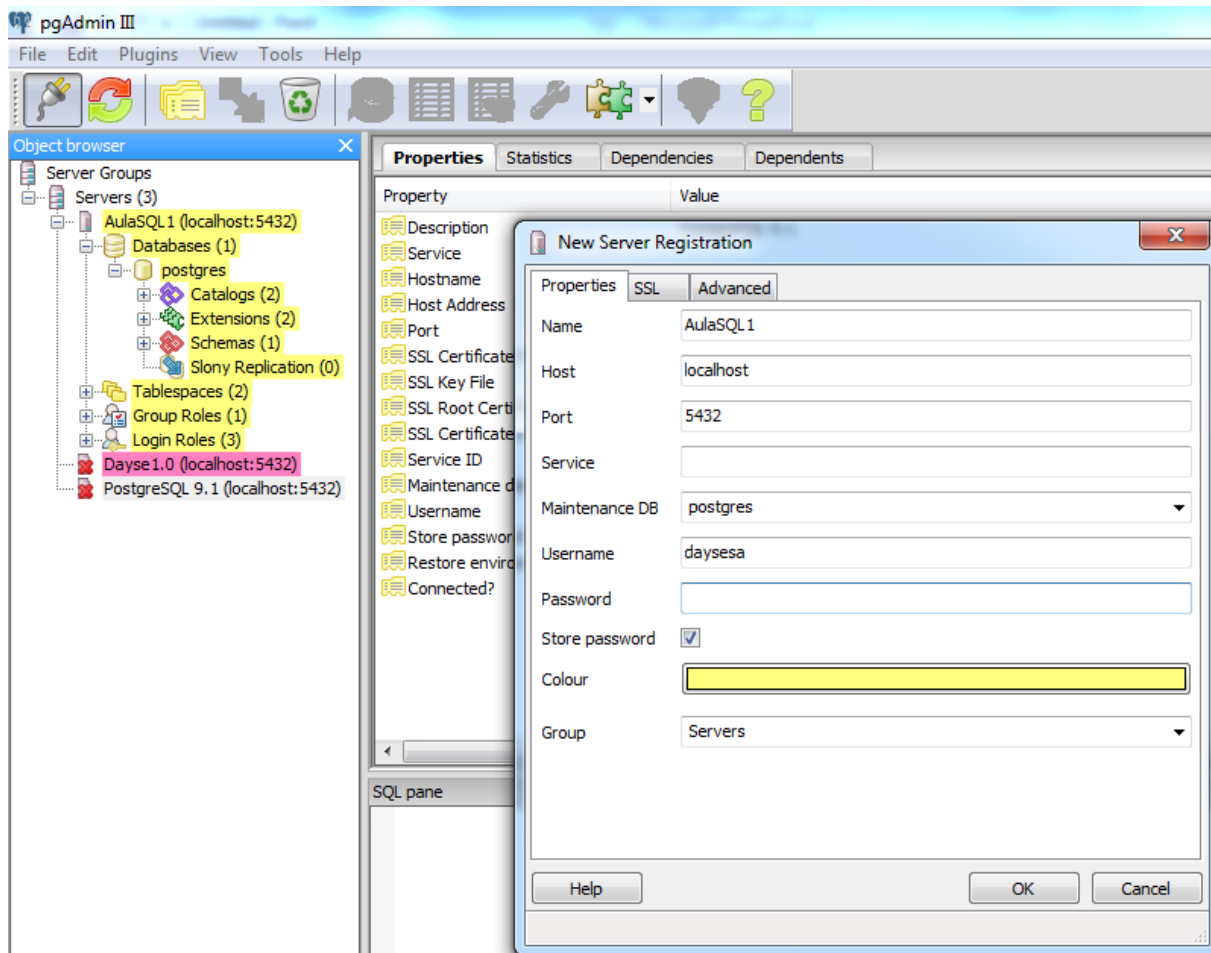
Professor: Eduardo Hruschka

Estagiária PAE: Dayse de Almeida

# Composição da SQL

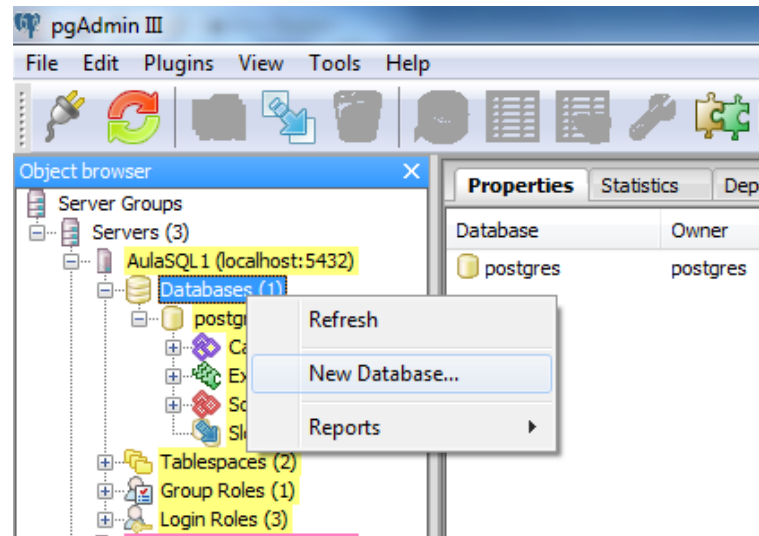
- Linguagem de Definição de Dados (DDL):
  - Comandos para definir, modificar e remover **tabelas**;
  - Além de criar e remover índices e visões.
- Linguagem de Manipulação de Dados (DML):
  - Comandos para criar, consultar, atualizar e remover **tuplas**.

# Adicionar conexão com o servidor



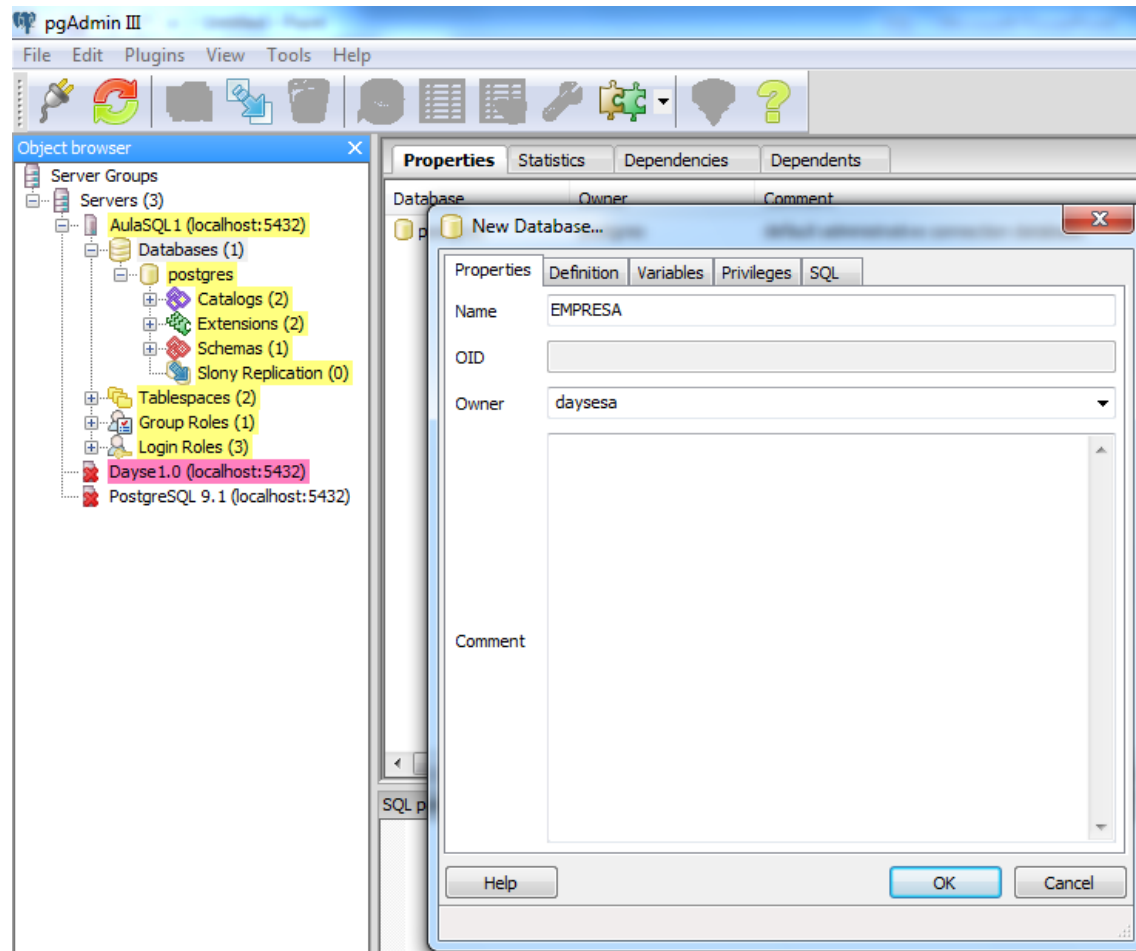
# Create database

- Opção 1:



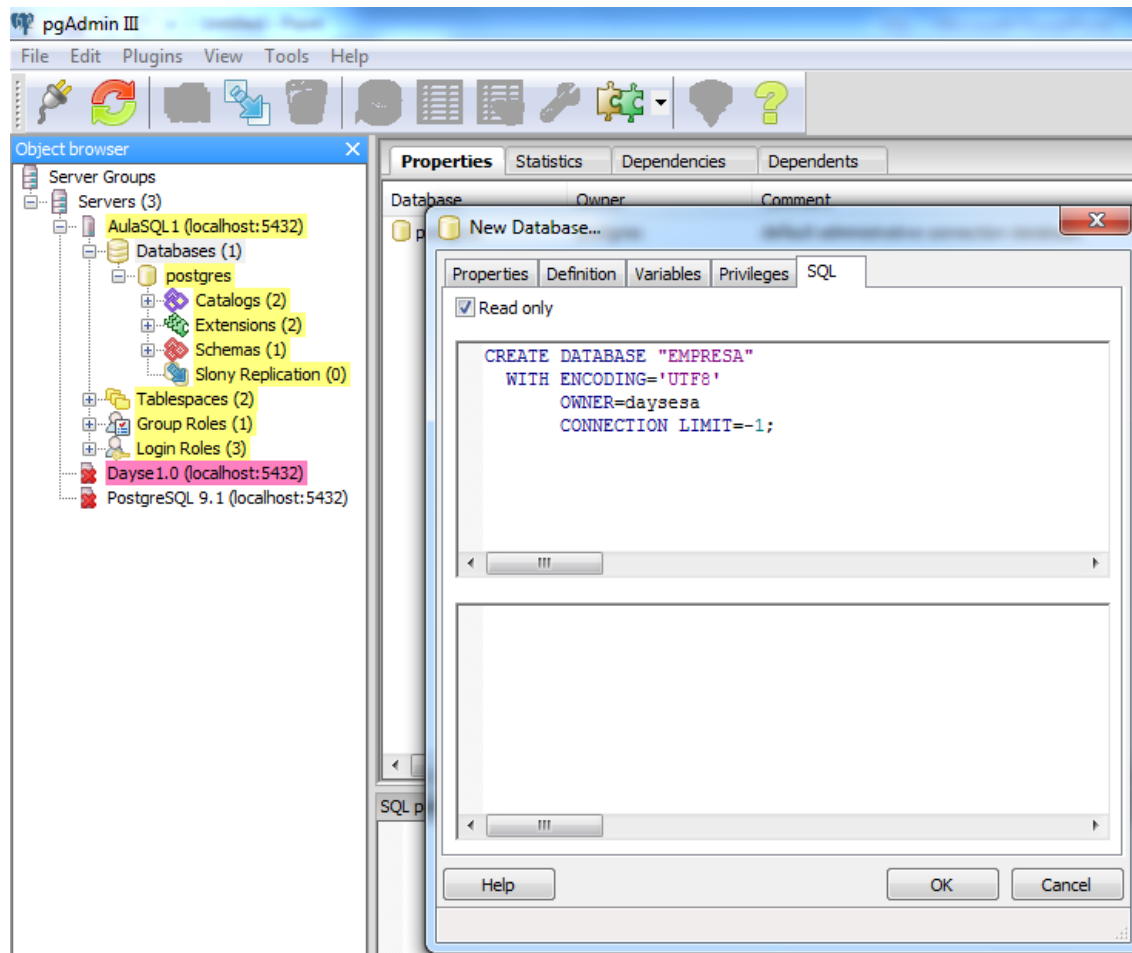
# Create database

- Opção 1:



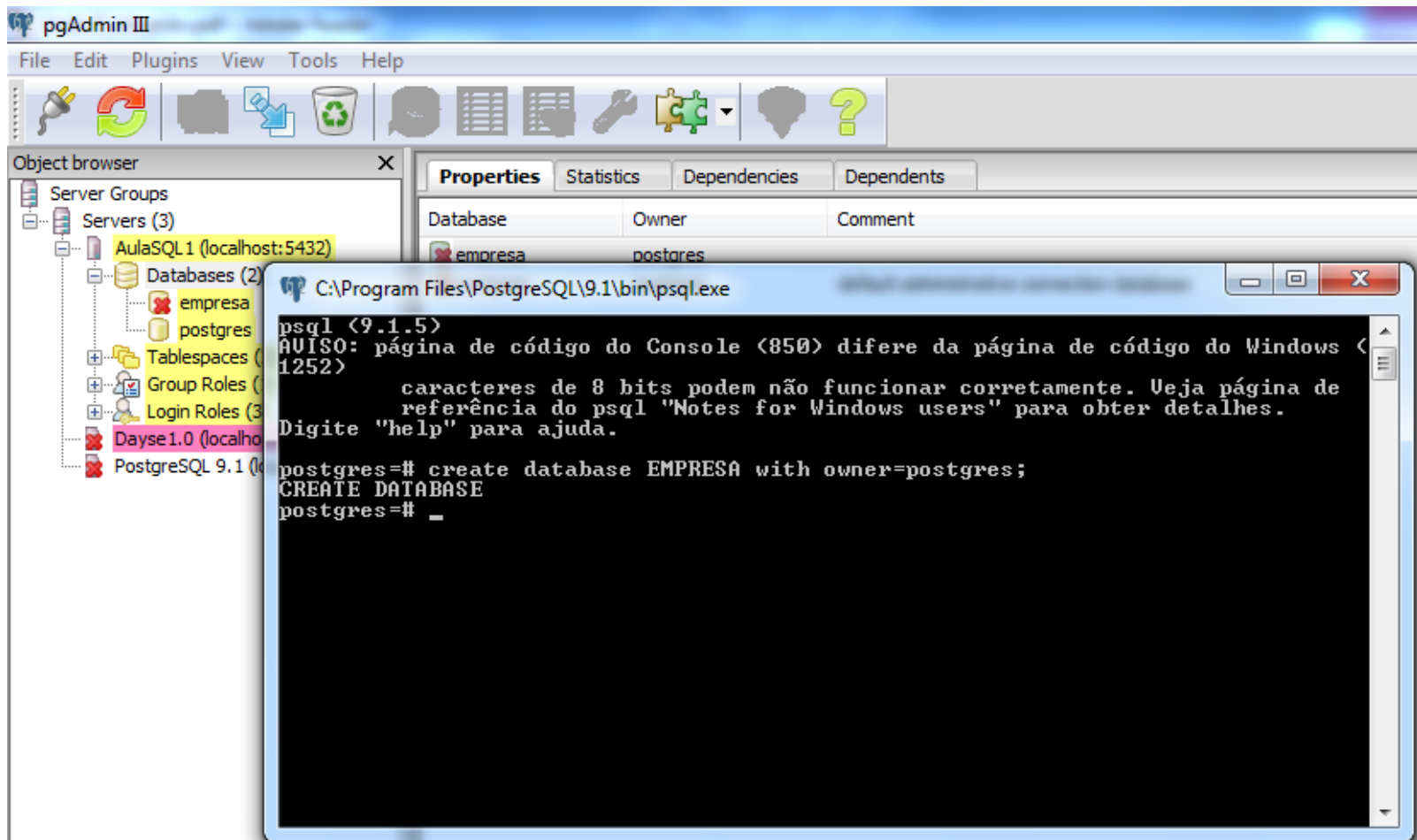
# Create database

- Opção 1:



# Create database

- Opção 2:



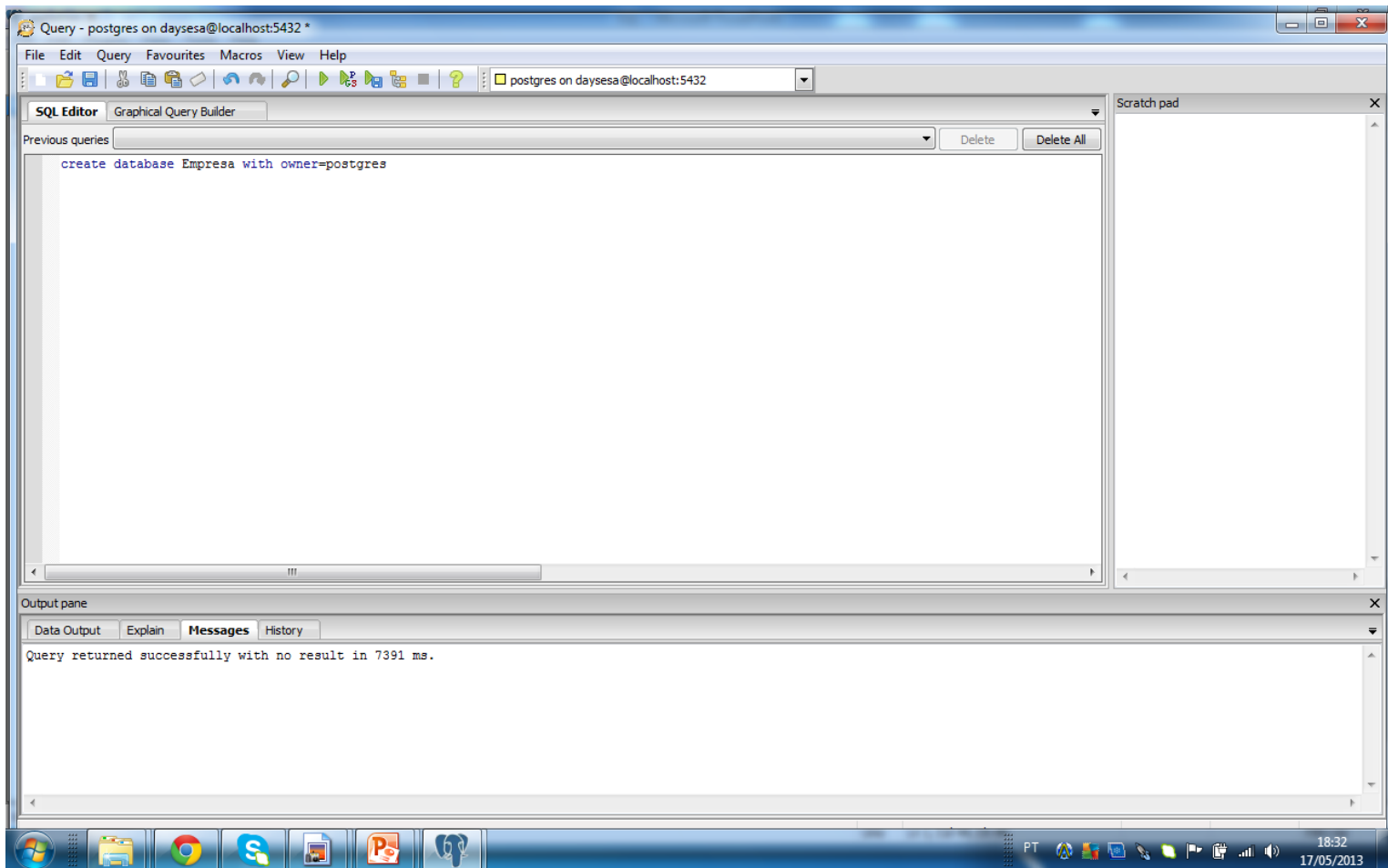
The screenshot shows the pgAdmin III interface. In the Object browser, the 'empresa' database is selected. The Properties tab is active, showing the database name 'empresa' and the owner 'postgres'. A terminal window titled 'C:\Program Files\PostgreSQL\9.1\bin\psql.exe' is overlaid on the interface, showing the following command and output:

```
psql (9.1.5)
AVISO: página de código do Console (850) difere da página de código do Windows (
1252)
      caracteres de 8 bits podem não funcionar corretamente. Veja página de
referência do psql "Notes for Windows users" para obter detalhes.
Digite "help" para ajuda.

postgres=# create database EMPRESA with owner=postgres;
CREATE DATABASE
postgres=# _
```

# Create database

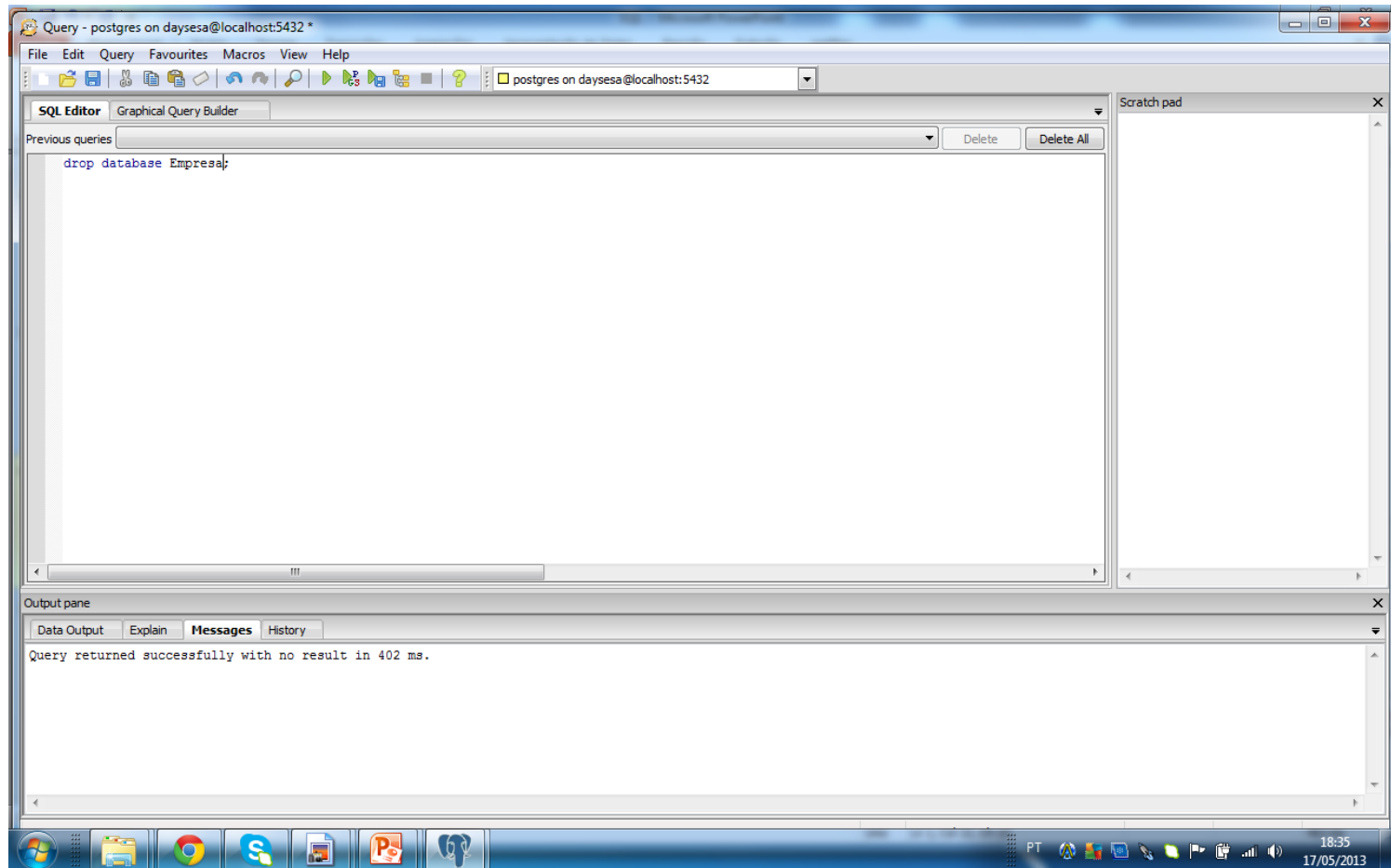
- Opção 3:





# Drop database

- drop database Empresa;



# Create database

- Criar um banco de dados para outro usuário:  
`create database teste1 owner daysesa;`
- Excluir:  
`drop database teste1;`
- Obs.: login roles.

# Create table

```
create table primeira_tabela(  
primeiro_campo text,  
segundo_campo integer);
```

- Drop table:

```
drop table primeira_tabela;
```

# Create table

- Valor *default* para campos:
  - Ao definir um valor default para um campo, ao ser cadastrado o registro e este campo não for informado, o valor default é assumido.

```
create table produtos (  
produto_no integer,  
descricao text,  
preco numeric default 9.99  
);
```

- insert into produtos(produto\_no, descricao, preco) values (45, 'qquer', 32);
- insert into produtos (produto\_no, descricao) values (45, 'qquer');

# Create table

- *Ckeck:*
  - Ao criar uma tabela podemos prever que o banco exija que o valor de um campo satisfaça uma expressão.

```
create table produtos2 (  
produto_no integer,  
descricao text,  
preco numeric check (preco > 0)  
);
```

- insert into produtos2 (produto\_no, descricao, preco) values (45, 'qquer', 0);
  - ERRO: novo registro da relação "produtos3" viola restrição de verificação "produtos2\_preco\_check"

# Create table

- Dar nome à restrição *check*:
  - Isso ajuda a tornar mais amigável as mensagens de erro.

```
CREATE TABLE produtos3 (  
produto_no integer,  
descricao text,  
preco numeric CONSTRAINT preco_positivo CHECK (preco > 0)  
);
```

- insert into produtos3 (produto\_no, descricao, preco) values (45, 'qquer', 0);
  - ERRO: novo registro da relação "produtos4" viola restrição de verificação "preco\_positivo"

# Create table

- Dar nome à restrição *check*:
  - Isso ajuda a tornar mais amigável as mensagens de erro.

```
CREATE TABLE produtos4 (  
    produto_no integer,  
    descricao text,  
    desconto numeric CHECK (desconto > 0 AND desconto < 0.10),  
    preco numeric CONSTRAINT preco_positivo CHECK (preco > 0),  
    CHECK (preco > desconto)  
);
```

- insert into produtos4(produto\_no, descricao, desconto, preco) values (45, 'qquer', 15, 100);
- insert into produtos4(produto\_no, descricao, desconto, preco) values (45, 'qquer', 0.5, 0.4);

# Create table

- Restrição *NOT NULL*:
  - Obriga o preenchimento de um campo.
  - Obs.: até um espaço em branco atende a esta restrição.

```
CREATE TABLE produtos5 (  
cod_prod integer NOT NULL CHECK (cod_prod > 0),  
nome text NOT NULL,  
preco numeric  
);
```

- insert into produtos5 (cod\_prod, nome, preco) values (-1, 'produtoX', 32);
  - ERRO: novo registro da relação "produtos6" viola restrição de verificação "produtos5\_cod\_prod\_check"
- insert into produtos5 (nome, preco) values ('produtoX', 32);
  - ERRO: valor nulo na coluna "cod\_prod" viola a restrição não-nula



# Create table

- Restrição *Unique*:
  - Valores exclusivos para cada campo em todos os registros;
  - Obs.: nulos não são checados. UNIQUE não aceita valores repetidos, mas aceita vários nulos (já que estes não são checados).

```
CREATE TABLE produtos6 (  
cod_prod integer UNIQUE,  
nome text,  
preco numeric  
);
```

```
CREATE TABLE produtos7(  
cod_prod integer,  
nome text,  
preco numeric,  
UNIQUE (cod_prod)  
);
```

- insert into produtos7(cod\_prod, nome, preco)values (45, 'produtoX', 34);
- insert into produtos7(cod\_prod, nome, preco)values (45, 'produtoY', 23);

# Create table

- Restrição *Unique*:

```
CREATE TABLE exemplo (  
  a integer,  
  b integer,  
  c integer,  
  UNIQUE (a, c)  
);
```

```
CREATE TABLE produtos8(  
  cod_prod integer CONSTRAINT unq_cod_prod UNIQUE,  
  nome text,  
  preco numeric  
);
```

# Exercício

- Criar a tabela produtos9 com os seguintes atributos: cod\_prod, nome e preco. O atributo cod\_prod deve ser inteiro, único, não nulo e positivo. O atributo nome é do tipo texto e o atributo preco é do tipo numérico.

# Exercício – Reposta

```
CREATE TABLE produtos9 (  
    cod_prod integer UNIQUE NOT NULL  
    CHECK(cod_prod > 0),  
    nome text,  
    preco numeric  
);
```

# Create table

- Chaves Primárias:
  - A chave primária de uma tabela é formada internamente pela combinação das restrições *UNIQUE* e *NOT NULL*;
  - Uma tabela pode ter no máximo uma chave primária;
  - A teoria de bancos de dados relacional dita que toda tabela deve ter uma chave primária;
  - O PostgreSQL não obriga que uma tabela tenha chave primária, mas é recomendável seguir, a não ser que esteja criando uma tabela para importar dados de outra que contenha registros duplicados para tratamento futuro, por exemplo.

# Create table

- Chaves Primárias (*Primary Key*):

```
CREATE TABLE produtos10 (  
  cod_prod integer UNIQUE NOT NULL,  
  nome text,  
  preco numeric  
);
```

```
CREATE TABLE produtos11 (  
  cod_prod integer PRIMARY KEY,  
  nome text,  
  preco numeric  
);
```

- Se mais que um atributo forma a chave primária:

```
CREATE TABLE exemplo (  
  a integer,  
  b integer,  
  c integer,  
  PRIMARY KEY (a, c)  
);
```

# Create table

- Chave Estrangeira (*Foreign Key*):
  - Criadas com o objetivo de relacionar duas tabelas, mantendo a integridade referencial entre ambas.
  - Especifica que o valor da coluna (ou grupo de colunas) deve corresponder a algum valor existente em um registro da outra tabela.
  - Na tabela estrangeira deve existir somente registros que tenham um registro relacionado na tabela principal.
  - Deve-se garantir que não se remova um registro na tabela principal que tenha registros relacionados na estrangeira.

# Create table

- Chave Estrangeira (*Foreign Key*):

- Tabela primária:

```
CREATE TABLE produtos11 (  
cod_prod integer PRIMARY KEY,  
nome text,  
preco numeric  
);
```

```
CREATE TABLE pedidos (  
cod_pedido integer PRIMARY KEY,  
cod_prod integer,  
quantidade integer,  
CONSTRAINT pedidos_fk FOREIGN KEY (cod_prod) REFERENCES produtos11 (cod_prod)  
);
```

\*



# Create table

- Chave Estrangeira (*Foreign Key*):

```
CREATE TABLE t0 (  
  a integer,  
  b integer,  
  c integer,  
  PRIMARY KEY(a, b)  
);
```

```
CREATE TABLE t1 (  
  a integer PRIMARY KEY,  
  b integer,  
  c integer,  
  d integer,  
  FOREIGN KEY (c, d) REFERENCES t0 (a, b)  
);
```

# Create table

- Simulando Enum:

```
CREATE TABLE pessoa(  
codigo int PRIMARY KEY,  
cor_favorita varchar(255) NOT NULL,  
check (cor_favorita IN ('vermelha', 'verde', 'azul'))  
);
```

- INSERT INTO pessoa (codigo, cor\_favorita) values (1, 'vermelha');
- INSERT INTO pessoa (codigo, cor\_favorita) values (2, 'amarela');

# Create table

- Herança:  
Pode-se criar uma tabela que herda todos os campos de outra tabela existente.

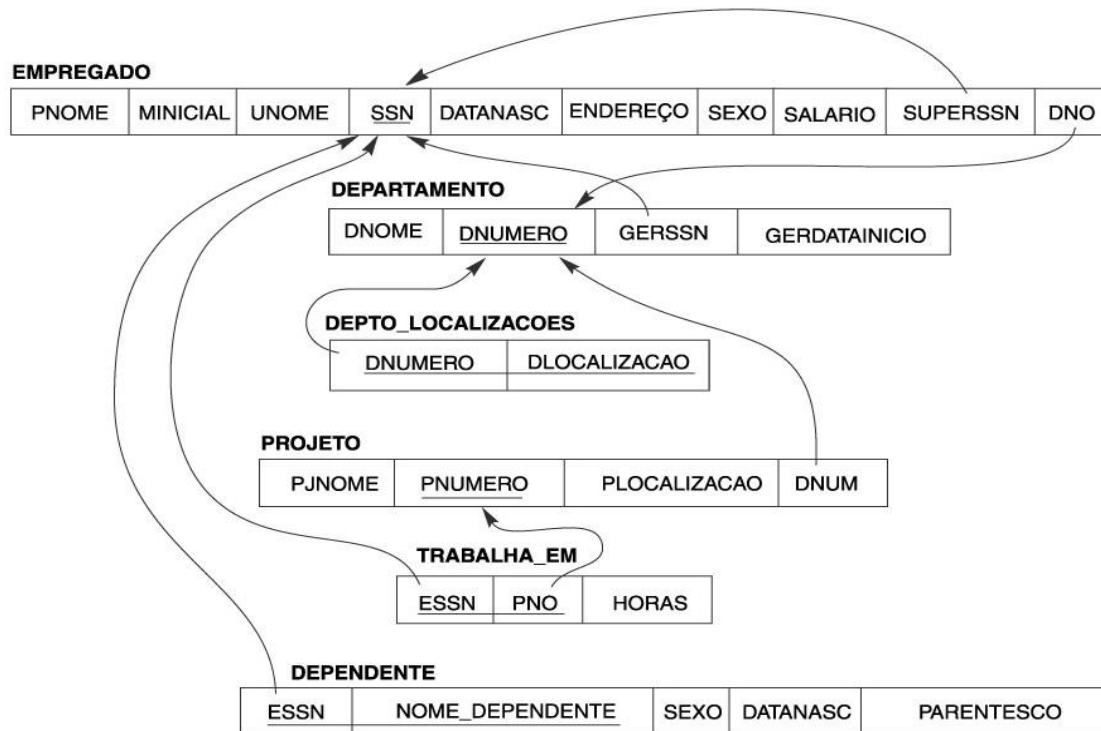
```
CREATE TABLE cidades (  
nome text,  
populacao float,  
altitude int  
);
```

```
CREATE TABLE capitais (  
estado char(2)  
) INHERITS (cidades);
```

- Assim, capitais passa a ter também todos os campos da tabela cidades.

# Exercício

- Criar relações com atributos, chaves primárias e chaves estrangeiras para:



# Exercício - Create schema

- `CREATE SCHEMA Esquema2;`
- `CREATE TABLE ESQUEMA2.TESTE(A1 int, A2 int);`

# Exercício – Resposta

```
CREATE TABLE EMPREGADO(  
    PNOOME varchar(255) NOT NULL,  
    MINICIAL char(1),  
    UNOME varchar(255) NOT NULL,  
    SSN integer PRIMARY KEY,  
    DATANASC date,  
    ENDERECO varchar(255),  
    SEXO char(2),  
    CHECK (SEXO IN ('F', 'M')),  
    SALARIO numeric NOT NULL CHECK (SALARIO > 0),  
    SUPERSSN integer,  
    CONSTRAINT supervisor FOREIGN KEY (SUPERSSN) REFERENCES EMPREGADO (SSN),  
    DNO integer CHECK (DNO > 0),  
    FOREIGN KEY (DNO) REFERENCES DEPARTAMENTO (DNUMERO)  
);
```

- ERRO: relação "departamento" não existe

# Exercício – Resposta

```
CREATE TABLE EMPREGADO(  
    PNAME varchar(255) NOT NULL,  
    MINICIAL char(1),  
    UNAME varchar(255) NOT NULL,  
    SSN integer PRIMARY KEY,  
    DATANASC date,  
    ENDERECO varchar(255),  
    SEXO char(2),  
    CHECK (SEXO IN ('F', 'M')),  
    SALARIO numeric NOT NULL CHECK (SALARIO > 0),  
    SUPERSSN integer,  
    CONSTRAINT supervisor FOREIGN KEY (SUPERSSN) REFERENCES EMPREGADO  
    (SSN),  
    DNO integer CHECK (DNO > 0)  
);
```

# Exercício – Resposta

```
CREATE TABLE DEPARTAMENTO(  
  DNOME varchar(255) NOT NULL,  
  DNUMERO integer PRIMARY KEY CHECK (DNUMERO > 0),  
  GERSSN integer,  
  CONSTRAINT gerente FOREIGN KEY (GERSSN) REFERENCES  
  EMPREGADO (SSN),  
  GERDATAINICIO date  
);
```



# Exercício – Resposta

- ALTER TABLE:

ALTER TABLE EMPREGADO

ADD CONSTRAINT numerodepto FOREIGN KEY (DNO)  
REFERENCES DEPARTAMENTO (DNUMERO)

;

# ALTER TABLE

- Adicionar atributo:

```
ALTER TABLE EMPREGADO  
ADD novoAtributo integer;
```

- Remover atributo:

```
ALTER TABLE EMPREGADO  
DROP COLUMN novoAtributo;
```

- Alterar o nome do atributo/coluna:

```
ALTER TABLE EMPREGADO  
CHANGE atributo novoAtributo integer;
```

- Alterar o tipo do atributo:

```
ALTER TABLE EMPREGADO  
MODIFY novoAtributo varchar(10);
```

# Exercício – Resposta

```
CREATE TABLE DEPTO_LOCALIZACOES(  
    DNUMERO integer,  
    DLOCALIZACAO varchar(255),  
    PRIMARY KEY(DNUMERO, DLOCALIZACAO),  
    CONSTRAINT numerodepartamento FOREIGN KEY  
    (DNUMERO) REFERENCES DEPARTAMENTO (DNUMERO)  
);
```

# Exercício – Resposta

```
CREATE TABLE PROJETO(  
    PJNOME varchar(255),  
    PNUMERO integer PRIMARY KEY,  
    PLOCALIZACAO varchar(255),  
    DNUM integer,  
    CONSTRAINT numerodepartamento FOREIGN KEY  
    (DNUM) REFERENCES DEPARTAMENTO (DNUMERO)  
);
```

# Exercício – Resposta

```
CREATE TABLE TRABALHA_EM(  
    ESN integer,  
    PNO integer,  
    PRIMARY KEY (ESN, PNO),  
    HORAS real,  
    CONSTRAINT numeroprojeto FOREIGN KEY (PNO)  
    REFERENCES PROJETO (PNUMERO)  
);
```

# Exercício – Resposta

```
CREATE TABLE DEPENDENTE(  
    ESN integer,  
    NOME_DEPENDENTE varchar(255),  
    SEXO char(1) CHECK (SEXO IN ('F', 'M')),  
    DATANASC date,  
    PARENTESCO varchar(255),  
    PRIMARY KEY (ESN, NOME_DEPENDENTE),  
    CONSTRAINT SSNempregado FOREIGN KEY (ESN)  
    REFERENCES EMPREGADO (SSN)  
);
```

# Tipos numéricos

Name	Storage Size	Description	Range
<code>smallint</code>	2 bytes	small-range integer	-32768 to +32767
<code>integer</code>	4 bytes	usual choice for integer	-2147483648 to +2147483647
<code>bigint</code>	8 bytes	large-range integer	-9223372036854775808 to 9223372036854775807
<code>decimal</code>	variable	user-specified precision, exact	no limit
<code>numeric</code>	variable	user-specified precision, exact	no limit
<code>real</code>	4 bytes	variable-precision, inexact	6 decimal digits precision
<code>double precision</code>	8 bytes	variable-precision, inexact	15 decimal digits precision
<code>serial</code>	4 bytes	autoincrementing integer	1 to 2147483647
<code>bigserial</code>	8 bytes	large autoincrementing integer	1 to 9223372036854775807

# Tipos para cadeias de caracteres

<b>Nome</b>	<b>Descrição</b>
<i>character varying(n), varchar(n)</i>	comprimento variável com limite
<i>character(n), char(n)</i>	comprimento fixo, completado com brancos
<i>text</i>	comprimento variável não limitado



# Tipos para data e hora

Nome	Tamanho de Armazenamento	Descrição	Menor valor	Maior valor	Resolução
<i>timestamp [ (p) ] [ without time zone ]</i>	8 bytes	tanto data quanto hora	4713 AC	5874897 DC	1 microssegundo / 14 dígitos
<i>timestamp [ (p) ] with time zone</i>	8 bytes	tanto data quanto hora, com zona horária	4713 AC	5874897 DC	1 microssegundo / 14 dígitos
<i>interval [ (p) ]</i>	12 bytes	intervalo de tempo	-178000000 anos	178000000 anos	1 microssegundo / 14 dígitos
<i>date</i>	4 bytes	somente data	4713 AC	32767 DC	1 dia
<i>time [ (p) ] [ without time zone ]</i>	8 bytes	somente a hora do dia	00:00:00.00	23:59:59.99	1 microssegundo / 14 dígitos
<i>time [ (p) ] with time zone</i>	12 bytes	somente a hora do dia, com zona horária	00:00:00.00+12	23:59:59.99-12	1 microssegundo / 14 dígitos

- Os tipos *time*, *timestamp*, e *interval* aceitam um valor opcional de precisão *p*, que especifica o número de dígitos fracionários mantidos no campo de segundos. Por padrão não existe limite explícito para a precisão. O intervalo permitido para *p* é de 0 a 6 para os tipos *timestamp* e *interval*.

# SQL

## II – Inserção e Atualização

Disciplina: Banco de Dados e suas  
Aplicações

Professor: Eduardo Hruschka

Estagiária PAE: Dayse de Almeida

# DML

- INSERT INTO ...
  - insere dados em uma tabela.
- UPDATE ... SET ... WHERE ...
  - altera dados específicos de uma tabela.

# Insert

**INSERT INTO** nome\_tabela

**VALUES** (V1, V2, Vn);

– Ordem dos atributos deve ser mantida.

**INSERT INTO** nome\_tabela (A1, A2, An)

**VALUES** (V1, V2, Vn);

– Ordem dos atributos não precisa ser mantida.

# Update

```
UPDATE nome_tabela  
    SET coluna = <valor>  
    WHERE predicado;
```

- Cláusula WHERE
  - É opcional.

# Insert

```
INSERT INTO EMPREGADO (PNOME, MINICIAL,  
UNOME, SSN, DATANASC, ENDERECO, SEXO,  
SALARIO, SUPERSSN, DNO) values ('John', 'B',  
'Smith', 123456789, '09/01/1965', '731 Fondren,  
Houston, Tx', 'M', 30000, null, null);
```

```
INSERT INTO EMPREGADO (PNOME, MINICIAL,  
UNOME, SSN, DATANASC, ENDERECO, SEXO,  
SALARIO, SUPERSSN) values ('Franklin', 'T', 'Wong',  
333445555, '08/12/1955', '638 Voss, Houston, Tx',  
'M', 40000, null, null);
```

# Update

```
UPDATE EMPREGADO  
  SET SUPERSSN= 333445555  
  WHERE SSN = 123456789;
```

# Exercício

- Inserir as seguintes tuplas:

EMPREGADO	PNOME	MINICIAL	UNOME	SSN	DATANASC	ENDERECO	SEXO	SALARIO	SUPERSSN	DNO
John	B		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	<i>null</i>	1

DEPT	LOCALIZACOES	DNUMERO	DLOCALIZACAO
		1	Houston
		4	Stafford
		5	Bellaire
		5	Sugarland
			Houston

DEPARTAMENTO	DNOME	DNUMERO	GERSSN	GERDATAINICIO
	Pesquisa	5	333445555	1988-05-22
	Administração	4	987654321	1995-01-01
	Sede administrativa	1	888665555	1981-06-19

TRABALHA_EM	ESSN	PNO	HORAS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	<i>null</i>

PROJETO	PJNOME	PNUMERO	PLOCALIZACAO	DNUM
	ProdutoX	1	Bellaire	5
	ProdutoY	2	Sugarland	5
	ProdutoZ	3	Houston	5
	Automatização	10	Stafford	4
	Reorganização	20	Houston	1
	Novos Benefícios	30	Stafford	4

DEPENDENTE	ESSN	NOME_DEPENDENTE	SEXO	DATANASC	PARENTESCO
	333445555	Alice	F	1986-04-05	FILHA
	333445555	Theodore	M	1983-10-25	FILHO
	333445555	Joy	F	1958-05-03	CÔNJUGE
	987654321	Abner	M	1942-02-28	CÔNJUGE
	123456789	Michael	M	1988-01-04	FILHO
	123456789	Alice	F	1988-12-30	FILHA
	123456789	Elizabeth	F	1967-05-05	CÔNJUGE



# Exercício

- Inserir/modificar as seguintes tuplas:
  - (a) Inserir < 'Robert', 'F', 'Scott', '943775543', '21-JUN-42', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1 > em EMPREGADO.
  - (b) Inserir < 'ProductA', 4, 'Bellaire', 2 > em PROJETO.
  - (c) Inserir < 'Production', 4, '943775543', '01-OCT-88' > em DEPARTAMENTO.
  - (d) Inserir < '677678989', null, '40.0' > em TRABALHA\_EM.
  - (e) Inserir < '453453453', 'John', M, '12-DEC-60', 'Cônjuge' > em DEPENDENTE.
  - (j) Modificar o valor do atributo SUPERSSN da tupla de EMPREGADO com SSN='999887777' para '943775543'.
  - (k) Modificar o valor do atributo HORAS da tupla de TRABALHA EM com ESSN='999887777' e PNO= 10 para '5.0'.

# SQL

## III – Consulta

Disciplina: Banco de Dados e suas  
Aplicações

Professor: Eduardo Hruschka  
Estagiária PAE: Dayse de Almeida

# Select

- `SELECT ... FROM ... WHERE ...`
  - Lista atributos de uma ou mais tabelas de acordo com alguma condição.

# Select

```
SELECT <lista de atributos e funções>  
FROM <lista de tabelas>  
[ WHERE predicado ]  
[ GROUP BY <atributos de agrupamento> ]  
[ HAVING <condição para agrupamento> ]  
[ ORDER BY <lista de atributos> ] ;
```

- Cláusula SELECT
  - Lista os atributos e/ou as funções a serem exibidos no resultado da consulta.
- Cláusula FROM
  - Especifica as relações a serem examinadas na avaliação da consulta.
- Cláusula WHERE
  - Especifica as condições para a seleção das **tuplas** no resultado da consulta;
    - As condições devem ser definidas sobre os atributos das relações que aparecem na cláusula FROM.
  - Pode ser omitida.

# Select

```
SELECT datanasc, endereco  
FROM esquema2."EMPREGADO"  
WHERE Pnome='Alicia' AND Unome='Zelaya';
```

```
SELECT *  
FROM esquema2."EMPREGADO";
```

# Select

- Ordem de apresentação dos atributos:
  - SELECT
- Duas ou mais tuplas podem possuir valores idênticos de atributos;
  - Eliminação de tuplas duplicadas;
    - SELECT DISTINCT
- Cláusula ORDER BY
  - Ordem de apresentação dos dados;
  - Ordem ascendente (**ASC**) ou descendente (**DESC**).

# Select

- Operadores:
  - Conjunção de condições: AND
  - Disjunção de condições: OR
  - Negação de condições: NOT
  - =, <>, >, <, >=, <=
  - Entre dois valores: BETWEEN... AND
  - Compara cadeias de caracteres: LIKE ou NOT LIKE
    - % (porcentagem): substitui qualquer *string*
    - \_ (underscore): substitui qualquer *caractere*
      - WHERE Pnome LIKE 'Jo%'
        - » qualquer string que se inicie com 'Jo'
      - WHERE Pnome LIKE 'Jo\_'
        - » qualquer string de 3 caracteres que se inicie com 'Jo'

# Select

```
SELECT Pnome, SSN  
FROM esquema2."EMPREGADO"  
WHERE Pnome LIKE 'J%';
```

```
SELECT Pnome, SSN  
FROM esquema2."EMPREGADO"  
WHERE Pnome LIKE 'Jo__';
```



# Select

```
SELECT Pnome, SSN  
FROM esquema2."EMPREGADO"  
WHERE Pnome LIKE 'J%' AND  
Sexo = 'F' AND  
Dno = 5;
```

# Operações sobre conjuntos

- Operações sobre conjuntos:
  - União: UNION
  - Intersecção: INTERSECT
  - Diferença: EXCEPT

# Union

- Exemplo:
  - Liste os nomes dos projetos dos departamentos 4 e 5.

```
SELECT PJnome
FROM esquema2."PROJETO"
WHERE Dnum = 4
UNION
SELECT PJnome
FROM esquema2."PROJETO"
WHERE Dnum = 5;
```

# Intersect

```
– INSERT INTO esquema2."DEPENDENTE" values  
(333445555, 'Alicia', 'F', '2013-05-27', 'Neta');
```

- Exemplo:

- Liste os nomes dos dependentes que tem nome igual a de algum empregado.

```
SELECT Nome_dependente  
FROM esquema2."DEPENDENTE"  
INTERSECT  
SELECT Pnome  
FROM esquema2."EMPREGADO"
```

# Except

- Exemplo:
  - Liste os nomes dos empregados que não têm dependentes.

```
SELECT Pnome
FROM esquema2."EMPREGADO"
EXCEPT
SELECT Pnome
FROM esquema2."EMPREGADO",
esquema2."DEPENDENTE"
WHERE esquema2."EMPREGADO".SSN =
esquema2."DEPENDENTE".ESSN;
```

# Exercício

- Liste os nomes dos empregados que não têm filhos (filho *e/ou* filha).

# Exercício - Resposta

- Liste os nomes dos empregados que não têm filhos (filho e/ou filha).

```
SELECT Pnome
FROM esquema2."EMPREGADO"
EXCEPT
SELECT Pnome
FROM esquema2."EMPREGADO",
esquema2."DEPENDENTE"
WHERE esquema2."EMPREGADO".SSN =
esquema2."DEPENDENTE".ESSN AND
esquema2."DEPENDENTE".parentesco LIKE 'Filh_';
```

# Operações sobre conjuntos

- UNION ( $R \cup S$ )
  - Une todas as linhas selecionadas por duas consultas, *eliminando* as linhas duplicadas;
  - Gera uma relação que contém todas as tuplas pertencentes a R, a S, ou a ambas R e S.
- UNION ALL
  - Une todas as linhas selecionadas por duas consultas, *inclusive* as linhas duplicadas.
- INTERSECT ( $R \cap S$ )
  - Gera uma relação que contém todas as tuplas pertencentes tanto a R quanto a S.
- EXCEPT ( $R - S$ )
  - Gera uma relação que contém todas as tuplas pertencentes a R que não pertencem a S.



# Subconsultas aninhadas

- Subconsulta
  - Expressão `SELECT ... FROM ... WHERE ...` aninhada dentro de outra consulta.
- Aplicações mais comuns:
  - Testes para membros de conjuntos;
  - Comparações de conjuntos;
  - Cardinalidade de conjuntos.
- `IN`
  - Testa se um atributo ou uma lista de atributos é membro do conjunto.
- `NOT IN`
  - Verifica a ausência de um membro em um conjunto.
- Conjunto:
  - Coleção de valores produzidos por uma cláusula `SELECT ... FROM ... WHERE ...`

# Subconsultas aninhadas

- Exemplo:
  - Liste os nomes dos dependentes que tem nome igual a de algum empregado.

```
SELECT Nome_dependente
FROM esquema2."DEPENDENTE"
WHERE Nome_dependente IN
    (SELECT Pnome FROM esquema2."EMPREGADO")
```

# Comparação de conjuntos

- SOME
  - ... WHERE salario > SOME (lista)
    - A condição é verdadeira quando *salario* for maior que algum dos resultados presentes na lista (resultado de uma consulta).

# Comparação de conjuntos

- Exemplo:
  - Liste os nomes dos empregados que têm salário superior a algum empregado do departamento 4.

```
SELECT Pnome  
FROM esquema2."EMPREGADO"  
WHERE salario > SOME  
(SELECT salario  
FROM esquema2."EMPREGADO"  
WHERE Dno = 4);
```

# Comparação de conjuntos

- ALL
  - ... WHERE salario > ALL (lista)
    - A condição é verdadeira quando *salario* for maior que todos os resultados presentes na lista (resultado de uma consulta).

# Comparação de conjuntos

- Exemplo:
  - Liste os nomes dos empregados que têm salário superior ao salário de todos os empregados do departamento 4.

```
SELECT Pnome  
FROM esquema2."EMPREGADO"  
WHERE salario > ALL  
(SELECT salario  
FROM esquema2."EMPREGADO"  
WHERE Dno = 4);
```

# Cardinalidade de conjuntos

- EXISTS

- ... WHERE EXISTS (lista)

- A condição é verdadeira quando a lista (resultado de uma consulta) não for vazia.

- NOT EXISTS

- ... WHERE NOT EXISTS (lista)

- A condição é verdadeira quando a lista for vazia.

# Cardinalidade de conjuntos

- Exemplo:
  - Liste os nomes dos dependentes que tem nome igual a de algum empregado.

```
SELECT Nome_dependente
FROM esquema2."DEPENDENTE"
WHERE EXISTS
    (SELECT Pnome
     FROM esquema2."EMPREGADO"
     WHERE esquema2."DEPENDENTE".Nome_dependente
     = esquema2."EMPREGADO".Pnome)
```



# Junção

- Idéia:
  - Concatenar tuplas relacionadas de duas relações em tuplas únicas.
- Passos:
  - Formar um produto cartesiano das relações;
  - Fazer uma seleção forçando igualdade sobre os atributos que aparecem nas relações.

# Junção

- Usar SELECT e WHERE
  - Atributos com mesmo nome são especificados usando nomes de tabelas e atributos (nome\_tabela.nome\_atributo).
- Cláusula FROM
  - Possui mais do que uma tabela.
- Cláusula WHERE
  - Inclui as condições de junção.

# Junção

1. Nome do empregado juntamente com nome do seu departamento:

```
SELECT Pnome, Dnome  
FROM esquema2."EMPREGADO", esquema2."DEPARTAMENTO"  
WHERE esquema2."EMPREGADO".Dno =  
       esquema2."DEPARTAMENTO".Dnumero;
```

2. Nome do empregado, nome do seu departamento e a localização do departamento:

```
SELECT Pnome, Dnome, Dlocalizacao  
FROM esquema2."EMPREGADO", esquema2."DEPARTAMENTO",  
       esquema2."DEPTO_LOCALIZACOES"  
WHERE esquema2."EMPREGADO".Dno =  
       esquema2."DEPARTAMENTO".Dnumero AND  
       esquema2."DEPTO_LOCALIZACOES".Dnumero =  
       esquema2."EMPREGADO".Dno;
```

# Junção

3. Nome do empregado, nome do seu departamento e seus dependentes:

```
SELECT Pnome, Dnome, Nome_dependente
FROM esquema2."EMPREGADO",
     esquema2."DEPARTAMENTO",
     esquema2."DEPENDENTE"
WHERE esquema2."EMPREGADO".Dno =
       esquema2."DEPARTAMENTO".Dnumero AND
       esquema2."DEPENDENTE".ESSN=
       esquema2."EMPREGADO".SSN;
```

# Join

1. Nome do empregado juntamente com o nome do seu departamento:

```
SELECT Pnome, Dnome
FROM esquema2."EMPREGADO", esquema2."DEPARTAMENTO"
WHERE esquema2."EMPREGADO".Dno =
      esquema2."DEPARTAMENTO".Dnumero;
```

```
SELECT Pnome, Dnome
FROM esquema2."EMPREGADO" JOIN
      esquema2."DEPARTAMENTO"
ON esquema2."EMPREGADO".Dno =
      esquema2."DEPARTAMENTO".Dnumero;
```

# Join

3. Nome do empregado, seu departamento e seus dependentes:

```
SELECT Pnome, Dnome, Nome_dependente
FROM esquema2."EMPREGADO"
  JOIN esquema2."DEPARTAMENTO"
    ON esquema2."EMPREGADO".Dno =
      esquema2."DEPARTAMENTO".Dnumero
  JOIN esquema2."DEPENDENTE"
    ON esquema2."DEPENDENTE".ESSN =
      esquema2."EMPREGADO".SSN;
```

# As

- Renomear:
  - Atributos
    - Deve aparecer na cláusula SELECT;
    - Útil para a visualização das respostas na tela.
  - Relações
    - Deve aparecer na cláusula FROM;
    - Útil quando a mesma relação é utilizada mais do que uma vez na mesma consulta.
- Sintaxe
  - nome\_antigo **AS** nome\_novo

# As

- Nome do empregado, nome do seu departamento e seus dependentes:

```
SELECT Pnome AS nome_empregado,  
       Dnome AS nome_departamento,  
       Nome_dependente AS dependente  
FROM esquema2."EMPREGADO" AS emp,  
     esquema2."DEPARTAMENTO" AS depto,  
     esquema2."DEPENDENTE" AS dep  
WHERE emp.Dno = depto.Dnumero AND dep.ESSN=  
emp.SSN;
```



# Order by

- Ordena as tuplas resultantes de uma consulta:
  - **ASC**: ordem ascendente (padrão);
  - **DESC**: ordem descendente.
- Ordenação pode ser especificada em vários atributos:
  - Ordenação referente ao primeiro atributo é prioritária;
  - Se houver valores repetidos, então é utilizada a ordenação referente ao segundo atributo, e assim por diante.

# Exercício

- Liste os dados dos empregados e seus departamentos. Ordene o resultado pelo nome do departamento.

# Exercício - Resposta

- Liste os dados dos empregados e seus departamentos. Ordene o resultado pelo nome do departamento.

```
SELECT *  
FROM esquema2."EMPREGADO",  
esquema2."DEPARTAMENTO"  
WHERE esquema2."EMPREGADO".Dno =  
esquema2."DEPARTAMENTO".Dnumero  
ORDER BY Dnome;
```

# Order by

- Se houver valores repetidos, então é utilizada a ordenação referente ao segundo atributo, e assim por diante.

```
SELECT Pnome, Minicial, Unome, Dnome  
FROM esquema2."EMPREGADO", esquema2. "DEPARTAMENTO"  
WHERE esquema2."EMPREGADO".Dno =  
esquema2."DEPARTAMENTO".Dnumero  
ORDER BY Dnome;
```

```
SELECT Pnome, Minicial, Unome, Dnome  
FROM esquema2."EMPREGADO", esquema2."DEPARTAMENTO"  
WHERE esquema2."EMPREGADO".Dno =  
esquema2."DEPARTAMENTO".Dnumero  
ORDER BY Dnome, Pnome;
```

# Funções de agregação

- Recebem uma coleção de valores como entrada;
- Retornam um único valor como saída.
  - Média: AVG( )
  - Mínimo: MIN( )
  - Máximo: MAX( )
  - Total: SUM( )
  - Contagem: COUNT( )
  -
- Obs.:
  - DISTINCT: não considera valores duplicados
  - ALL: inclui valores duplicados

# Exercício

- Qual a média dos salários dos empregados?
- Qual a soma dos salários dos empregados?
- Qual é o salário mais baixo dos salários dos empregados?
- Qual é o salário mais alto dos salários dos empregados?

# Exercício - Resposta

- Qual a média dos salários dos empregados?  
`SELECT AVG(salario)`  
`FROM esquema2."EMPREGADO"`
- Qual a soma dos salários dos empregados?
- Qual é o salário mais baixo dos salários dos empregados?
- Qual é o salário mais alto dos salários dos empregados?

# Exercício - Resposta

- Qual a média dos salários dos empregados?
- Qual a soma dos salários dos empregados?  
`SELECT SUM(salario)`  
`FROM esquema2."EMPREGADO"`
- Qual é o salário mais baixo dos salários dos empregados?
- Qual é o salário mais alto dos salários dos empregados?



# Exercício - Resposta

- Qual a média dos salários dos empregados?
- Qual a soma dos salários dos empregados?
- Qual é o salário mais baixo dos salários dos empregados?  
`SELECT MIN(salario)`  
`FROM esquema2."EMPREGADO"`
- Qual é o salário mais alto dos salários dos empregados?

# Exercício - Resposta

- Qual a média dos salários dos empregados?
- Qual a soma dos salários dos empregados?
- Qual é o salário mais baixo dos salários dos empregados?
- Qual é o salário mais alto dos salários dos empregados?

```
SELECT MAX(salario)  
FROM esquema2."EMPREGADO"
```

# Exercício

- Quantos supervisores existem na relação EMPREGADO?

# Exercício - Resposta

- Quantos supervisores existem na relação EMPREGADO?

```
SELECT COUNT(SUPERSSN)
```

```
FROM esquema2."EMPREGADO"
```

# Exercício

- Quantos supervisores existem na relação EMPREGADO?

```
SELECT COUNT(SUPERSSN)
```

```
FROM esquema2."EMPREGADO"
```

- Quantos supervisores diferentes existem na relação EMPREGADO?

# Exercício - Resposta

- Quantos supervisores existem na relação EMPREGADO?

```
SELECT COUNT(SUPERSSN)  
FROM esquema2."EMPREGADO"
```

- Quantos supervisores existem na relação EMPREGADO?

```
SELECT COUNT(DISTINCT SUPERSSN)  
FROM esquema2."EMPREGADO"
```

# Group by

- Permite aplicar uma função de agregação não somente a um conjunto de tuplas, mas a um grupo de um conjunto de tuplas;
- Grupo de um conjunto de tuplas:
  - Conjunto de tuplas que possuem o mesmo valor para os atributos de agrupamento.

# Group by

- Qual o maior salário, o menor salário e a média de salários na relação EMPREGADO por supervisor?

```
SELECT MIN(salario), MAX(salario), AVG(salario)
FROM esquema2."EMPREGADO"
GROUP BY SUPERSSN;
```

```
SELECT SUPERSSN, MIN(salario), MAX(salario),
AVG(salario)
FROM esquema2."EMPREGADO"
GROUP BY SUPERSSN;
```



# Having

- Especifica uma condição de seleção para grupos;
- Recupera os valores para as funções somente para aqueles grupos que satisfazem à condição imposta na cláusula HAVING.

# Having

- Qual o maior salário, o menor salário e a média de salários na relação EMPREGADO por supervisor, para médias salariais superiores a 30000?

```
SELECT MIN(salario), MAX(salario), AVG(salario)
FROM esquema2."EMPREGADO"
GROUP BY SUPERSSN
HAVING AVG(salario) > 30000
```

```
SELECT SUPERSSN, MIN(salario), MAX(salario), AVG(salario)
FROM esquema2."EMPREGADO"
GROUP BY SUPERSSN
HAVING AVG(salario) > 30000
```

# SQL

## IV – Remoção

Disciplina: Banco de Dados e suas  
Aplicações

Professor: Eduardo Hruschka  
Estagiária PAE: Dayse de Almeida

# DML

- DELETE FROM ... WHERE ...
  - Remove dados de tabelas já existentes.

# Delete

**DELETE FROM** nome\_tabela  
**WHERE** predicado;

- Cláusula WHERE
  - É opcional:
    - Todas as tuplas da tabela são eliminadas;
    - A tabela continua a existir.

# Exercícios

1. Liste os nomes dos empregados e os projetos para os quais cada empregado trabalha. Ordene o resultado pelo nome do projeto, em ordem ascendente. Renomeie as colunas exibidas para Nome do Empregado e Nome do Projeto.
2. Resolva o exercício anterior usando a cláusula JOIN.
3. Liste os nomes dos departamentos e seus respectivos projetos. Ordene o resultado pelo nome do departamento, em ordem ascendente. Renomeie as colunas exibidas para Nome do Departamento e Nome do Projeto.
4. Resolva o exercício anterior usando a cláusula JOIN.
5. Liste os nomes dos departamentos que possuem projetos usando a cláusula IN.

# Exercícios

6. Selecione os nomes e salários dos empregados que ganham entre 40000 e 50000.
7. Resolva o exercício anterior usando a cláusula BETWEEN... AND.
8. Selecione os nomes do empregados que tem dependentes, usando a cláusula INTERSECT.
9. Selecione os nomes do empregados que tem dependentes, usando apenas a cláusula IN.
10. Selecione os nomes do empregados que tem dependentes, usando apenas a cláusula EXISTS.

# Exercícios

11. Liste a quantidade de empregados cadastrados.
12. Liste a quantidade de empregados por supervisor, bem como o ssn do supervisor.
13. Liste a quantidade de empregados por supervisor e o ssn do supervisor, de forma que apenas os supervisores com mais de dois empregados supervisionados sejam listados.
14. Liste os números dos departamentos, bem como seus gastos totais com salários de empregados, para os casos em que os gastos são superiores à média dos gastos (com salário de todos os empregados). Ordene o resultado de forma descendente pelo gasto.



# Exercício – Delete

15. Remover as seguintes tuplas no banco de dados EMPRESA:

- (f) Remover as tuplas com ESSN= '333445555' de TRABALHA\_EM.
- (g) Remover de EMPREGADO a tupla com SSN= '987654321'.
- (h) Remover de PROJETO a tupla com PJNOME= 'ProdutoX'.

# Extras

- Retornar bancos, dono, codificação, comentários e tablespace:

```
SELECT pdb.datname AS banco,  
       pu.username AS dono,  
       pg_encoding_to_char(encoding) AS codificacao,  
       (SELECT description FROM pg_description pd WHERE  
        pdb.oid=pd.objoid) AS comentario,  
       (SELECT spcname FROM pg_catalog.pg_tablespace pt WHERE  
        pt.oid=pdb.dattablespace) AS tablespace
```

```
FROM pg_database pdb, pg_user pu  
WHERE pdb.datdba = pu.usesysid ORDER BY pdb.datname
```

# Extras

- Selecionar colunas e tabelas:

```
SELECT
c.relname,
a.attname AS "Column",
pg_catalog.format_type(a.atttypid, a.atttypmod) AS "Datatype"

FROM pg_catalog.pg_attribute a
INNER JOIN pg_stat_user_tables c on a.attrelid = c.relid
WHERE
a.attnum > 0 AND
NOT a.attisdropped
ORDER BY c.relname, a.attname
```

# Extras

- Selecionar colunas e tabelas:

```
SELECT
c.relname AS tabela,
a.attname AS coluna,
d.adsrc AS default,
pg_catalog.format_type(a.atttypid, a.atttypmod) AS tipo

FROM pg_catalog.pg_attribute a
INNER JOIN pg_stat_user_tables c
ON a.attrelid = c.relid
LEFT JOIN pg_catalog.pg_attrdef d
ON d.adrelid = c.relid and a.attnum = d.adnum
WHERE a.attnum > 0 AND NOT a.attisdropped
ORDER BY c.relname, a.attname
```

# Extras

- Retornar tabelas e esquemas do banco de dados atual:

```
SELECT n.nspname AS esquema, c.relname AS tabela
FROM pg_namespace n, pg_class c
WHERE n.oid = c.relnamespace
      AND c.relkind = 'r'    -- no indices
      AND n.nspname NOT LIKE 'pg\_%' -- no catalogs
      AND n.nspname != 'information_schema' -- no
information_schema
ORDER BY nspname, relname
```

# Extras

- Retornar tabelas do banco de dados e esquema atual:

```
SELECT schemaname AS esquema, tablename AS  
tabela, tableowner AS dono  
FROM pg_catalog.pg_tables  
WHERE schemaname NOT IN ('pg_catalog',  
'information_schema', 'pg_toast')  
ORDER BY schemaname, tablename
```