



SQL

Criação de Objetos



SQL

- SQL é uma linguagem declarativa utilizada para armazenar, manipular e recuperar dados em um banco de dados relacional
- Declarativa indica que o programador diz *o que quer* e não *como quer*.
- Em 1986, ANSI e ISO oficialmente adotaram o padrão oficial para SQL.
Versões 1989, 1992, 1996, 1999, 2003, 2006, 2008, 2011 e a mais recente 2016.
- As implementações SQL não são totalmente compatíveis entre os SGBDs
 - Sintaxe de data e hora
 - Concatenação de strings
 - Tratamento de NULLs
 - Nomes dos tipos de dados (e.g., int, integer, numeric, ...)
 - Entre outros

SQL

- SQL é dividida em três partes conforme a função:
 - DCL - Data Control Language
 - Todos os comandos relativos a permissões e controles de acesso
 - DDL - Data Definition Language
 - Todos os comandos relativos à criação de objetos (tabelas, bancos, visões, ...)
 - DML - Data Manipulation Language
 - Todos os comandos relativos à manipulação e atualização dos dados armazenados no banco de dados

SQL

- SQL é dividida em três partes conforme a função:
 - DCL - Data Control Language
 - Todos os comandos relativos a permissões e controles de acesso
 - **DDL - Data Definition Language**
 - **Todos os comandos relativos à criação de objetos (tabelas, bancos, visões, ...)**
 - DML - Data Manipulation Language
 - Todos os comandos relativos à manipulação e atualização dos dados armazenados no banco de dados

DDL - Data Definition Language

- Vamos estudar um subgrupo da DDL:
 - Criação, alteração e exclusão de tabelas
 - Criação e exclusão de banco de dados
- Lembrando que a sintaxe será a do PostgreSQL
 - Provavelmente, compatível com outros SGBDs mas deve ser verificado antes de rodar um script PostgreSQL em outro SGBD, por exemplo, MySQL.

DDL - Data Definition Language

- Comando para criar/excluir um BD é simples:

```
create database nomedobanco;
```

- Obs.: Procure dar nomes sem usar letras maiúsculas, só minúsculas.

```
drop database nomedobanco;
```

- Cuidado: **excluindo um BD todos os objetos deles são excluídos também.**
- Exemplo:

```
create database academico;  
create database clinicamedica;  
drop database academico;
```

DDL - Data Definition Language

- Criar um usuário

```
create user nomeusuario password 'password';  
drop user nomeusuario password 'password';
```

- Exemplo:

```
create user deniod password 'deniodpostgres';  
drop user deniod;
```

Obs.: Se o usuário `deniod` tiver objetos associados a ele, ele não poderá ser excluído

DDL - Data Definition Language

- Depois veremos como executar esses comandos no PostgreSQL
- Usaremos o aplicativo `bash psql`

DDL - Data Definition Language

- Criação e exclusão de tabelas
- Exclusão por ser mais simples: `drop table nometabela;`
- Se a tabela tiver dependências (e.g., FK associadas), ela não poderá ser excluída

Exemplo:

```
drop table cliente;
```

DDL - Data Definition Language

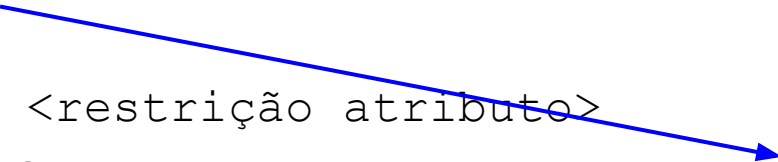
- Criação

```
create table nometabela(  
    atrib1 tipo <restrição atributo>,  
    :  
    atribn tipo <restrição atributo>  
    <restrição de PK>  
    <restrição de FK>  
);
```

DDL - Data Definition Language

- Criação

```
create table nometabela(  
    atrib1 tipo <restrição atributo>,  
    :  
    atribn tipo <restrição atributo>  
    <restrição de PK>  
    <restrição de FK>  
);
```



Pode ser:

- text: texto livre
- varchar(n): sequência de n caracteres (n>0)
- integer: 4 bytes inteiros
- numeric(tam, decimal): números com no máximo *tam* dígitos e *decimal* casas decimais.
- date: permite inserir valores no formato dada

Existem vários outros mas vamos utilizar, por enquanto, apenas os apresentados

DDL - Data Definition Language

- Criação
- Exemplo sem restrições de nulidade e referência

```
create table funcionario(  
    matric varchar(10),  
    nome varchar(30),  
    ender varchar(50),  
    dtnasc date,  
    salario numeric(10,2)  
);
```

-> **matric** é uma sequência de no máximo 10 caracteres

-> no máximo 30 caracteres

-> endereço pode ter até 50 caracteres

-> aceita apenas data (o formato é configurável)

-> aceita no máximo o seguinte valor 99.999.999,99

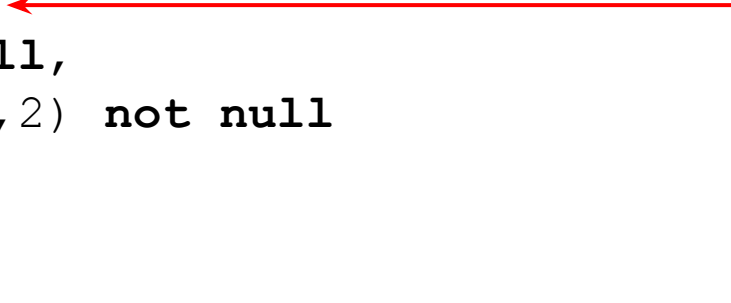
Nesta definição, todos os atributos são considerados opcionais.

Em PostgreSQL, se não for colocado explicitamente como obrigatório, o atributo se torna opcional por padrão

DDL - Data Definition Language

- Acrescentando restrições de obrigatoriedade. Vamos supor que ender é opcional, os outros atributos obrigatórios

```
create table funcionario(  
    matric varchar(10) not null,  
    nome varchar(30) not null,  
    ender varchar(50),  
    dtnasc date not null,  
    salario numeric(10,2) not null  
);
```



Opcionalmente, pode-se colocar apenas NULL

DDL - Data Definition Language

- Agora vamos definir e criar a chave primária (PK)

```
create table funcionario(  
    matric varchar(10) not null,  
    nome varchar(30) not null,  
    ender varchar(50),  
    dtnasc date not null,  
    salario numeric(10,2) not null,  
    constraint pk_funcionario primary key (matric)  
);
```

Atributo que a chave primária da tabela. Veja que o atributo deve ser criado normalmente e depois definido como PK

Nome da restrição. Importante dar um nome que indique de qual tabela é a PK. Geralmente, coloca-se **pk_nometabela**

DDL - Data Definition Language

- Vamos aumentar um pouco o esquema do banco e criar uma tabela departamento e associar o funcionário ao departamento que ele trabalha

```
create table depto(  
    coddep integer not null,  
    nomedep varchar(30),  
    constraint pk_depto primary key  
(coddep)  
);
```

DDL - Data Definition Language

- Vamos aumentar um pouco o esquema do banco e criar uma tabela departamento e associar o funcionário ao departamento que ele trabalha

```
create table depto(  
    coddep integer not null,  
    nomedep varchar(30),  
    constraint pk_depto primary key  
(coddep)  
);  
  
create table funcionario(  
    matric varchar(10) not null,  
    nome varchar(30) not null,  
    ender varchar(50),  
    dtnasc date not null,  
    coddep integer not null,  
    salario numeric(10,2) not null,  
    constraint pk_funcionario primary key (matric)  
    constraint fk_depto_func foreign key ( coddep) references depto(coddep) );
```

Cria o atributo normalmente
(o tipo tem que ser o mesmo)

Define a FK. Indicando o
nome do atributo

E a tabela e atributos
referenciados

Exercício Resolvido

- Estudo de caso (da Atividade da aula sobre o modelo relacional)
- Esquema do banco de dados (nossa notação) - coloquei o atributo `obs` em paciente para termos um atributo opcional

```
paciente(cpf, nome, dtnasc, ncid, uf, obs)  
especialidade(codesp, descr)  
medico(crm, nome, dtnasc, ncid, uf, code(especialidade))  
consulta(crm(medico), cpf(paciente), datac, horac)
```

Exercício Resolvido

- Primeiro, vamos criar paciente:

```
--paciente(cpf, nome, dtnasc, ncid, uf, obs)  
--  
create table paciente (  
    cpf varchar(11) not null,  
    nome varchar(30) not null,  
    dtnasc date not null,  
    ncid varchar(20) not null,  
    uf varchar(2) not null,  
    obs text,  
    constraint pk_paciente primary key (cpf)  
);
```

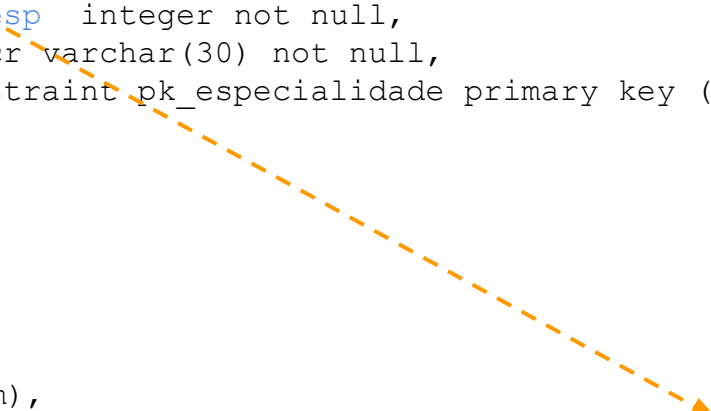
Exercício Resolvido

- Agora especialidade e médico:

```
--especialidade(codesp, descr)
--medico(crm, nome, dtnasc, ncid, uf, code(especialidade))

create table especialidade (
    codesp integer not null,
    descr varchar(30) not null,
    constraint pk_especialidade primary key (codesp));

create table medico (
    crm integer not null,
    nome varchar(30) not null,
    dtnasc date not null,
    ncid varchar(20) not null,
    uf varchar(2) not null,
    code integer not null,
    constraint pk_medico primary key (crm),
    constraint fk_medico_espec foreign key (code) references especialidade(codesp));
```



Exercício Resolvido

- Finalmente, consulta a com mais referências e chave primária composta

```
-- consulta(crm(medico), cpf(paciente), datac, horac)  
--  
create table consulta (  
    crm integer not null,  
    cpf varchar(11) not null,  
    datac date not null,  
    horac timestamp not null,  
    constraint pk_consulta primary key (crm,cpf,datac,horac),  
    constraint fk_consulta_medico foreign key (crm) references medico(crm),  
    constraint fk_consulta_paciente foreign key (cpf) references paciente(cpf)  
);
```

PSQL

- PSQL é um utilitário do PostgreSQL para gerenciar os banco de dados
- O psql é instalado automaticamente quando o PostgreSQL é instalado
 - Verifiquem na web instruções para instalar o PostgreSQL
- Se o SGBD estiver rodando, o psql pode ser executado
- Abaixo um exemplo de como executar. O **-d** indica o nome de banco para acessar **-U** indica qual o usuário que se conecta e **-h** o servidor (conexão local digita-se **localhost**)

```
[home@ddurte]psql -d postgres -U postgres -h localhost
```

```
psql (12.2)
```

```
Type "help" for help
```

esperando digitar um novo comando

```
postgres=#
```

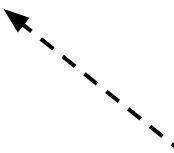
nome do banco
conectado

conectado como superusuário (o usuário postgres é root)

PSQL

```
[home@ddurte]psql -d postgres -U postgres -h localhost  
psql (12.2)  
Type "help" for help
```

```
postgres=#create database clinica;  
CREATE DATABASE  
postgres=#\c clinica  
You are now connected to database "clinica" as user "postgres".  
clinica=#\c postgres  
You are now connected to database "postgres" as user "postgres".  
postgres=#create user eu password 'eu123';  
CREATE ROLE  
postgres=#\c clinica eu  
You are now connected to database "clinica" as user "eu".  
clinica=>
```



veja que agora aparece > ao invés de #, ou seja,
o usuário **eu** não é um superusuário

DDL - Create Table

- Vamos começar resolvendo o Home Work 2 (conforme aula virtual dia 06/04)
- Abaixo segue a solução e em seguida vamos para a criação das tabelas:

```
curso(codc, nome)
```

```
aluno(matr, nome, cpf, ender, email, codc(curso))
```

```
ccr(codccr, sigla, nome, cred, codc(curso))
```

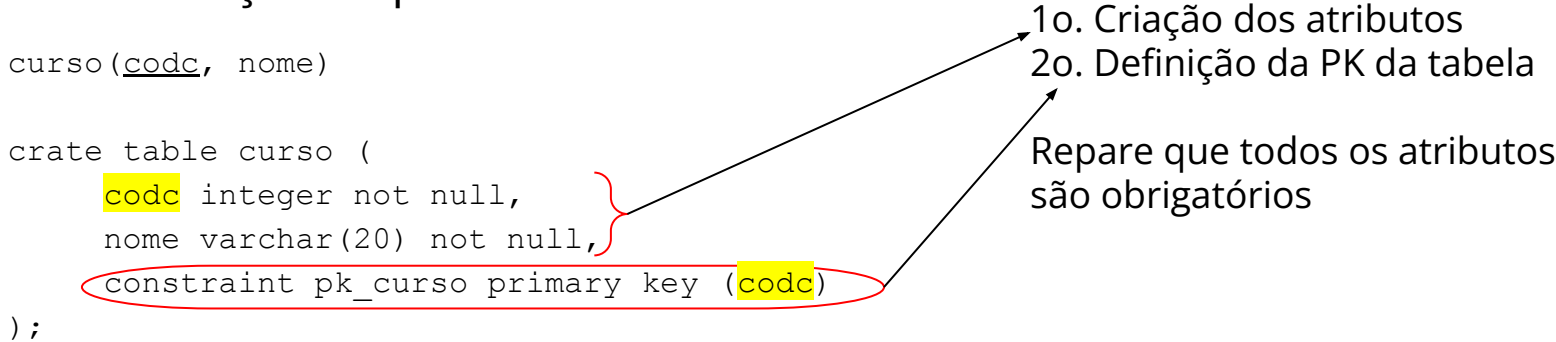
```
historico(matr(aluno), codccr(ccr), semestre, ppres, media)
```

DDL - Create Table

- Começamos pela tabela curso

curso(codc, nome)

```
create table curso (  
  codc integer not null,  
  nome varchar(20) not null,  
  constraint pk_curso primary key (codc)  
);
```



- 1o. Criação dos atributos
- 2o. Definição da PK da tabela

Repare que todos os atributos
são obrigatórios

DDL - Create Table

- Agora a tabela CCR (observe que ela tem uma FK para curso)

```
ccr(codccr, sigla, nome, cred, codc(curso))
```

```
create table ccr(
```

```
    codccr integer not null,  
    sigla varchar(5) not null,  
    nome varchar(20) not null,  
    cred integer not null,  
    codc integer not null,  
    constraint pk_ccr primary key (codccr),  
    constraint fk_ccr_curso foreign key (codc)
```

```
);
```

```
create table curso (
```

```
    codc integer not null,  
    nome varchar(20) not null,  
    constraint pk_curso primary key (codc)
```

```
);
```

references curso(**codc**)

Atributos são criados antes de definir as chaves

As vezes, é uma boa prática o nome do atributo FK ser o mesmo da tabela referenciada (como é esse caso)

DDL - Create Table

- Próximo passo, tabela aluno (ela também tem um FK para curso)

```
aluno(matr, nome, cpf, ender, email, codc(curso))
```

```
create table aluno (  
    matr integer not null,  
    nome varchar(30) not null,  
    cpf varchar(11) not null,  
    ender varchar(50) not null,  
    email varchar(30) not null,  
    codc integer not null,  
    constraint pk_aluno primary key (matr),  
    constraint fk_aluno_curso foreign key (codc)  
);
```

```
create table curso (  
    codc integer not null,  
    nome varchar(20) not null,  
    constraint pk_curso primary key (codc)  
);
```

references curso(codc)

No encontro de segunda, eu defini **cpf** como chave (não primária). O comando foi `constraint uk_aluno_cpf unique (cpf)` ← só inserir na criação da tabela

DDL - Create Table

- Finalmente a tabela Historico (ela tem a particularidade de dois atributos serem PK e FK ao mesmo tempo)

historico(matr(aluno), codccr(ccr), semestre, ppres,media)

```
create table historico (  
    matr integer not null,  
    codccr integer not null,  
    semestre varchar(4) not null,  
    ppres integer not null,  
    media numeric (4,2) not null,  
    constraint pk_historico primary key (matr,codccr, semestre),  
    constraint fk_hist_aluno foreign key (matr) references aluno(matr),  
    constraint fk_hist_ccr foreign key (codccr) references ccr(codccr)  
);
```

```
create table aluno (  
    matr integer not null,  
    nome varchar(30) not null,  
    cpf varchar(11) not null,  
    ender varchar(50) not null,  
    email varchar(30) not null,  
    codc integer not null,  
    constraint pk_aluno primary key (matr),  
    constraint fk_aluno_curso foreign key (codc) references curso(codc)  
);
```

```
create table ccr(  
    codccr integer not null,  
    sigla varchar(5) not null,  
    nome varchar(20) not null,  
    cred integer not null,  
    codc integer not null,  
    constraint pk_ccr primary key (codccr),  
    constraint fk_ccr_curso foreign key (codc) references curso(codc)  
);
```

DML - Insert

- Para finalizar, vamos inserir algumas tuplas nas tabelas:

Sintaxe: `insert into nomeTabela (atributos) values (valores a serem inseridos);`

--cursos:

-- uma tupla por vez

```
insert into curso (codc,nome) values (1,'Ciência da Computação');
```

-- se deu certo o PSQL retorna

```
INSERT 0 1
```

-- várias tuplas

```
insert into curso (codc,nome) values (2,'Administração'), (3,'Enfermagem'), (4,'Filosofia');
```

```
INSERT 0 3
```

--ccrs:

```
insert into ccr (codccr,sigla,nome,cred,codc) values (1,'BD I','Banco de Dados I',4,1),  
            (2,'BD II','Banco de Dados II',4,1), (3,'TGA I','Teoria Geral I',2,2),  
            (4,'CH I','Corpo Humano I',3,3), (5,'EP I','Epistemologia',2,4);
```

DML - Insert

- Para finalizar, vamos inserir algumas tuplas nas tabelas:

--alunos:

```
insert into aluno (matr,nome,cpf,ender, email,codc) values
    (201111,'Mister Eleven','11111111','Eleven Street','eleven@uffs',1),
    (201101,'Sra Um','01010101','Rua Um','um@uffs',1),
    (201103,'Tre','03030303','Tre Gatan','tre@uffs',2),
    (201108,'Huit','08080808','Rue Huit','huit@uffs',3);
```

-- se deu certo o PSQL retorna

--historico:

```
insert into historico (matr,codccr,semestre,ppres,media) values
    (201101,1,'19-2',78,5.4),
    (201111,1,'19-2',81,7.6),
    (201111,3,'19-2',95,8.6),
    (201101,1,'20-1',NULL,NULL);
```

DML - Select

- Para verificar os conteúdos das tabelas:

```
-- Comando select: sintaxe select listaAtributos from nomeTabela;  
select * from cursor;  
  codc |   nome  
-----+-----  
      1 | Computação  
      2 | Administração  
      3 | Enfermagem  
      4 | Filosofia  
(4 rows)  
  
select * from ccr;  
select * from aluno;  
select * from historico;
```

DML - Update

- O comando update atualiza tupla(s) de uma tabela.
 - `update nomeTabela set coluna=valor [where condição]`
 - A cláusula `where` é opcional mas geralmente é usada em quase 100% dos updates. Por quê? Se não utilizarmos a cláusula `where` TODAS as tuplas da tabela terão a `coluna` atualizada.

-- Atualizar o curso do aluno 201103 de 2 para 3

-- Consulta na tabela aluno para 'chechar'

academico=> select * from aluno where matr=201103;

matr	nome	cpf	ender	email	codc
201103	Tre	33333	Tre Gatan	tre@uffs	2

(1 row)

academico=>update aluno set codc=3 where matr=201103;

UPDATE 1

-- agora a tupla acima possui 3 no atributo codc e não mais 2.

DML - Update

```
-- Atualizar a sigla do CCR Teoria Geral I de TGA I para TG I (código 3)
academico=> update ccr set sigla='TG I' where codccr=3;
UPDATE 1
-- dica: quando apenas uma tupla da tabela será atualizada, geralmente, na cláusula
-- where a condição envolve a chave primária (no caso da tabela ccr, o atributo codccr)
-- CUIDADO, se o comando fosse (sem a cláusula where)
academico=> update ccr set sigla='TG I';
UPDATE 5
-- veja agora atualizou 5 tuplas (ou seja, todas da tabela)
-- Agora, todos os ccr têm a sigla TG I - uma pequena catástrofe :)
```


DML - Update

```
academico=>select * from historico;
```

matr	codccr	semestre	ppres	media
201101	1	19-2	78	5.4
201111	1	19-2	81	7.6
201111	3	19-2	95	8.6
201101	1	20-1		

(4 rows)

-- Agora vamos atualizar a tabela historico, acrescentando 2 percentuais na presença
-- de todos os alunos que cursaram algum ccr em 19-2 (ou seja 78 vai p/ 80 e assim por diante)

```
academico=>update historico set ppres=ppres+2 where semestre='19-2';
```

UPDATE 3

-- 3 tuplas foram atualizadas pois existem 3 tuplas com semestre igual a 19-2

DML - Update

-- nova consulta apresenta

academico=>select * from historico;

matr	codccr	semestre	ppres	media
201101	1	20-1		
201101	1	19-2	80	5.4
201111	1	19-2	83	7.6
201111	3	19-2	97	8.6

(4 rows)

-- os valores de **ppres** form atualizados como 78--> 80, 81--> 83 e 95--> 97 (ou seja ppres+2)

DML - Update

```
academico=>select * from historico;
```

matr	codccr	semestre	ppres	media
201101	1	20-1		
201101	1	19-2	80	5.4
201111	1	19-2	83	7.6
201111	3	19-2	97	8.6

(4 rows)

-- AVISO: perceba que na primeira tupla os valores de ppre e media são nulos (vazios)

-- NULL significa nada então não podemos fazer nenhum tipo de condição com ele. Por exemplo:

```
academico=>update historico set media=7 where media=NULL;
```

UPDATE 0

-- Nenhuma foi tupla foi atualizada pois não podemos fazer comparações com NULL :(

-- Solução? Claro que tem: o SQL tem a comparação **is null** e **is not null**.

DML - Update

```
academico=>update historico set media=7 where media is null;
```

```
UPDATE 1
```

```
-- Agora sim! :)
```

```
-- CUIDADO: o where do comando acima é PERIGOSO pois passaria TODAS as médias sem
```

```
-- valor para 7! Lembrem, sempre tentem envolver a PK na condição da cláusula where.
```

DML - Delete

-- O comando para excluir tuplas de uma tabela é:

-- delete from nometabela [where condição]

-- CUIDADO: O where é opcional mas o não uso é PERIGOSO

-- Se rodarmos o comando:

academico=>delete from historico;

DELETE 4

-- APAGAREMOS TODAS AS TUPLAS da tabela

-- Isso me faz lembrar de um video. Conta a estorinha de um delete :)

<https://www.youtube.com/watch?v=uKMpJyeLysE&fbclid=IwAR2Fojgh-Rm3Fy6WfYFbb7RiLiG04Yh4Mr51XnWgYiJwnlvxxQrx2auVxhc>

DML - Delete

```
-- Mais do que no update, procure colocar a PK como condição na cláusula where
-- Isso garante que apenas uma tupla seria excluída.
-- Mas por exemplo, se quiséssemos apagar todo o histórico do semestre 19-2
academico=>delete from historico where semestre='19-2';
DELETE 3
-- Existiam 3 tuplas com semestre='19-2'
-- DICA: Antes de executar, execute um select equivalente
-- Por exemplo, no delete acima, poderíamos executar um select para ver as tuplas que seriam
-- afetadas pelo comando, antes de apagá-las
academico=>select * from historico where semestre='19-2';
  matr  | codccr | semestre | ppres | media
-----+-----+-----+-----+-----
 201101 |      1 | 19-2    |     80 |    5.4
 201111 |      1 | 19-2    |     83 |    7.6
 201111 |      3 | 19-2    |     97 |    8.6
(3 rows)
```

DML - Select

Todos os atributos.
De qual tabela?
Quais tuplas? Todas (sem where)

-- O comando select é o mais utilizado do grupo DML e o mais poderoso.
-- Ele sozinho é quase um programa completo, daí a complexidade do mesmo.
-- Sintaxe: select **listaAtributos** from **tabelas** where **condição**.
-- **listAtributos** indica quais atributos das **tabelas** envolvidas na consulta serão apresentados para os usuários (* indica todos).
-- **tabelas** indica de onde os dados serão extraídos para apresentar para o usuário.
-- **condição** indica as tuplas das **tabelas** que deverão retornar da consulta
-- Formato mais simples (já visto):

academico=>select * from aluno;

matr	nome	cpf	ender	email	codc
201111	Mister Eleven	11111111	Eleven Street	eleven@uffs	1
201101	Sra Um	01010101	Rua Um	um@uffs	1
201103	Tre	03030303	Tre Gatan	tre@uffs	2
201108	Huit	08080808	Rue Huit	huit@uffs	3

(4 rows)

DML - Select

-- Retorna as matrículas dos alunos que obtiveram média maior que 6 em qualquer CCR e
-- em qualquer semestre
-- Primeiro: identificar onde os dados necessários se encontram. Pelo conhecimento do BD
-- sabemos que podemos obter a matrícula e a média da tabela histórico.
-- Esses atributos são chamados de matr e media.

```
academico=>select matr from historico where media > 6;
```

<resultado omitido>

-- Retornar todos os dados do histórico dos alunos que ainda não tiveram a média lançada
-- Estudo: todos os atributos (podemos utilizar o *), do histórico (utilizaremos a tabela
-- historico) e não tem média lançada (o atributo media tem que ser nulo)

```
academico=>select * from historico where media is null;
```

matr	codccr	semestre	ppres	media
201101	1	20-1		

(1 row)

-- perceba que media=null não funciona temos que utilizar **is null** ou **is not null**

DML - Select

```
academico=>select * from historico where media is null;
```

matr	codccr	semestre	ppres	media
201101	1	20-1		

(1 row)

-- Esta consulta não precisaria retornar a média pois já é sabida que ela é nula
-- Também, o ppres não é necessário já que não temos a média. Assim, poderíamos retornar
-- apenas os dados existentes para as tuplas com média nula

```
academico=>select matr,codccr,semestre from historico where media is null;
```

matr	codccr	semestre
201101	1	20-1

(1 row)

-- DICA: só utilize o * quando deve se retornar TODOS os atributos (para evitar o uso
-- desnecessário de memória do computador)

DML - Select

```
-- Para finalizar as consultas simples:
-- Retornar a sigla e o nome de todos os CCRS do curso 2
-- Estudo: onde estes dados estão armazenados? Na tabela CCR temos todos eles: sigla, nome e
-- código do curso. Os atributos que armazenam esses dados são: sigla, nome e codc
-- Lembrando que não são TODOS os CCRS, apenas aqueles do curso 2 (temos que utilizar where)
academico=>select sigla, nome from ccr where codc=2;
  sigla |      nome
-----+-----
  TGA I | Teoria Geral I
(1 row)
```

DML - Select

-- Vamos alterar a consulta para que traga todos os ccr de todos os cursos:

academico=>select sigla, nome, codc from ccr;

sigla	nome	codc
BD I	Banco de Dados I	1
BD II	Banco de Dados II	1
TGA I	Teoria Geral I	2
CH I	Corpo Humano I	3
EP I	Epistemologia	4

(5 rows)

Teríamos que olhar o **codc** da tabela **ccr** e comparar com **codc** da tabela **curso** para saber os nomes dos cursos.

-- Problema: eu não lembro o código de cada curso.

-- Teria que fazer outra consulta:

academico=# select * from curso;

codc	nome
1	Ciência da Computação
2	Administração
3	Enfermagem
4	Filosofia

(4 rows)

DML - Select

```
-- Quando uma consulta envolve mais de uma tabela. No nosso exemplo tínhamos que retornar
-- os dados do CCR mais o nome do curso de cada CCR. Ou seja, os dados dos CCRs estão na
-- tabela ccr mas o nome do curso se encontra na tabela curso. A tabela ccr só tem o código
-- A boa notícia é que o código do curso da tabela ccr é uma chave estrangeira para o código
-- da tabela curso.
-- Em situações como essa utilizamos uma técnica chamada JOIN
-- Regra 1: quando os dados de uma consulta estão armazenados em mais de uma tabela
--         temos que utilizar a técnica de join
-- Regra 2: as tabelas do join têm que ter algo em comum (neste caso chave estrangeira)
--         pois é necessário fazer uma condição de ''amarração'' (condição de join)
-- Regra 3: join em select é um operador binário. Só podemos utilizar 2 tabelas por join.
--         É igual a adição na matemática. Se quisermos somar 3 números, primeiro somamos
--         2 e o resultado somamos ao terceiro. Por exemplo, que vamos somar 3, 4 e 5
--         temos que fazer 3+4 e depois 5, ficando assim 3+4+5, não dá para fazer
--         3 4 5+, pois o + é binário.
```

DML - Select (Join)

-- o join, como esperado, é feito dentro do **from** pois envolve tabelas.
-- Formato: from **tabela1 apelido1** join **tabela2 apelido2** on **condição join**
-- No nosso exemplo: retornar a sigla e o nome do CCR além do nome do curso que o CCR está associado. Se verificarmos a definição das tabelas ccr e curso temos:
-- curso(codc, nome) e ccr(codccr, sigla, nome, cred, codc(curso))
-- Veja que ccr tem o atributo codc que é uma FK que aponta para curso, ou seja,
-- podemos fazer o join

```
academico=>select c.nome, cc.sigla, cc.ome
```

```
academico->from ccr cc join curso c on cc.codc=c.codc;
```

Condição para fazer o match entre as duas tabelas

nome	sigla	nome
Ciência da Computação	BD I	Banco de Dados I
Ciência da Computação	BD II	Banco de Dados II
Administração	TGA I	Teoria Geral I
Enfermagem	CH I	Corpo Humano I
Filosofia	EP I	Epistemologia

(5 rows)

DML - Select (Join)

-- Suponha que você queira uma consulta assim:

nome	nome	semestre	media
Sra Um	Banco de Dados I	20-1	
Sra Um	Banco de Dados I	19-2	5.4
Mister Eleven	Banco de Dados I	19-2	7.6
Mister Eleven	Teoria Geral I	19-2	8.6

--

-- Essa consulta tem dados de 3 tabelas: o nome do aluno está na tabela aluno, o nome do CCR está na tabela CCR e o semestre a média do CCR cursado está na tabela histórico.
-- Temos que colocar no from aluno, ccr e histórico. Nós só podemos fazer isso se existe uma ligação entre essas tabelas. Se observamos a tabela historico, ela tem chave estrangeira para curso e ccr. Então OK, a consulta é possível de ser realizada.

DML - Select (Join)

```
-- Vamos por parte. Primeiro vamos utilizar apenas a tabela historico
academico=>select h.matr, h.codccr, h.semestre, h.media from historico h;
```

matr	codccr	semestre	media
------	--------	----------	-------

-----+-----+-----+-----

201101		1		20-1	
--------	--	---	--	------	--

201101		1		19-2		5.4
--------	--	---	--	------	--	-----

201111		1		19-2		7.6
--------	--	---	--	------	--	-----

201111		3		19-2		8.6
--------	--	---	--	------	--	-----

```
-- Perceba que só com o histórico, não conseguimos retornar os nomes dos alunos e dos ccrs
```

```
-- Vamos adicionar a tabela aluno para retornar o nome ao invés da matrícula
```

```
academico=>select a.nome, h.codccr, h.semestre, h.media from historico h join aluno a
academico->                                     on h.matr=a.matr;
```

nome	codccr	semestre	media
------	--------	----------	-------

-----+-----+-----+-----

Sra Um		1		20-1	
--------	--	---	--	------	--

Sra Um		1		19-2		5.4
--------	--	---	--	------	--	-----

Mister Eleven		1		19-2		7.6
---------------	--	---	--	------	--	-----

Mister Eleven		3		19-2		8.6
---------------	--	---	--	------	--	-----

DML - Select (Join)

```
academico=>select a.nome, h.codccr, h.semestre, h.media from historico h  
academico->                                     join aluno a on h.matr=a.matr;
```

nome	codccr	semestre	media
Sra Um	1	20-1	
Sra Um	1	19-2	5.4
Mister Eleven	1	19-2	7.6
Mister Eleven	3	19-2	8.6

-- O que falta agora é substituir o código do ccr pelo seu nome (vamos colocar a sigla).
-- Para colocar a sigla, temos acrescentar a tabela ccr que armazena a sigla

```
academico=>select a.nome, cc.sigla, h.semestre, h.media from historico h join aluno a on  
academico-> h.matr=a.matr join ccr cc on h.codccr=cc.codccr;
```

nome	sigla	semestre	media
Sra Um	BD I	20-1	
Sra Um	BD I	19-2	5.4
Mister Eleven	BD I	19-2	7.6
Mister Eleven	TGA I	19-2	8.6