

Banco de Dados

SQL – Conceitos e Principais Comandos

Prof. Dr. Ives Renê V. Pola

ivesr@utfpr.edu.br

Departamento Acadêmico de Informática – DAINF
UTFPR – Pato Branco DAINF
UTFPR
Pato Branco - PR

Uma apresentação dos comandos mais importantes da Linguagem SQL.



Introdução a SQL

- A Linguagem SQL – “*Structured Query Language*” foi desenvolvida pelos pesquisadores Donald D. Chamberlin and Raymond F. Boyce a partir de 1972 no Laboratório de Pesquisa da IBM em San Jose, logo depois da introdução do modelo relacional por Edgar F. Codd em 1970.
- Inicialmente chamada “SEQUEL”, foi criada para ser a linguagem de consulta do SGBD Relacional “System R”, então em desenvolvimento no Laboratório. Logo foi renomeada para SQL (“*Structured Query Language*”), por questões de patente.
- Por sua simplicidade e grande poder de consulta, SQL é atualmente o padrão industrial em linguagens de consultas a banco de dados, dominando mais de 95% do mercado de sistemas de gerenciamento de bases de dados.

Introdução

SQL pode ser dividida em 3 categorias

- ❶ Linguagem de Definição de Dados - DDL
 - Criação de estruturas como tabelas e seus atributos.
- ❷ Linguagem de Manipulação de Dados - DML
 - Recuperação e atualização dos dados.
- ❸ Linguagem de Controle de Dados - DCL
 - Permissões de acesso aos dados e transações.

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação

Sintaxe geral de um Comando CREATE TABLE

```
CREATE TABLE <nome da tabela> (  
    <definição de Coluna>,...  
    <Restrições de Integridade>,...  
);
```

onde <definição de Coluna> pode ser:

```
<nome atr> <tipo de dado>  
    [NULL | NOT NULL]  
    [USER | DEFAULT <value>  
        | COMPUTED BY <expresion>  
    ]
```

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação – Exemplo

Exemplo: Projeto Lógico: Uma tabela é descrita indicando seus atributos (com as respectivas restrições de integridade):

Aluno (RA, Nome, Cidade, Idade)

```
CREATE TABLE Aluno (  
    RA      decimal(7) NOT NULL,  
    Nome    varchar(60) NOT NULL,  
    Cidade  char(25),  
    Idade   decimal(3) NOT NULL  
);
```

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação – Exemplo

Exemplo: Criar uma tabela com Atributos DEFAULT

Professor (Nome, Nivel)

```
CREATE TABLE professor (  
    nome          varchar(60) not null,  
    nivel         char(4)      not null default 'MS-3'  
);
```

Para verificar a tabela criada (PostgreSQL):

```
select *  
from information_schema.columns  
where table_name = 'professor';
```

DDL – Comando CREATE TABLE

Tipos de Dados Principais

tipo de dado pode ser:

```
SMALLINT | INTEGER | FLOAT | DOUBLE PRECISION  
| {DECIMAL | NUMERIC}[( precision [, scale])]  
| DATE | TIME | TIMESTAMP  
| {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR[(int)]}  
| CLOB | BLOB - Oracle  
| bytea - PostgreSQL
```

DDL – Comando CREATE TABLE

Tipos de Dados do PostgreSQL

<https://www.postgresql.org/docs/9.6/static/datatype.html>

Name	Aliases	Description
bigint	int8	signed eight-byte integer
bigserial	serial8	autoincrementing eight-byte integer
bit [(n)]		fixed-length bit string
bit varying [(n)]	varbit	variable-length bit string
boolean	bool	logical Boolean (true/false)
box		rectangular box on a plane
bytea		binary data ("byte array")
character [(n)]	char [(n)]	fixed-length character string
character varying [(n)]	varchar [(n)]	variable-length character string
cidr		IPv4 or IPv6 network address
circle		circle on a plane
date		calendar date (year, month, day)
double precision	float8	double precision floating-point number (8 bytes)
inet		IPv4 or IPv6 host address
integer	int, int4	signed four-byte integer
interval [fields] [(p)]		time span
json		textual JSON data
jsonb		binary JSON data, decomposed
line		infinite line on a plane

DDL – Comando CREATE TABLE

Restrições de Integridade

Restrições de Integridade podem ser:

```
CHECK  
NOT NULL  
UNIQUE  
PRIMARY KEY  
FOREIGN KEY  
EXCLUSION (PostgreSQL)
```

Existem duas maneiras de aplicá-las:

- 1 Restrição de atributo
- 2 Restrição de tabela

DDL – Comando CREATE TABLE

Restrições de Integridade como Declaração de Restrições

Restrições de Integridade são tratadas em SQL como Restrições (CONSTRAINT). Elas podem ser restrições de Atributo ou de Tabela.

- Restrições de atributos (ou de colunas) são declaradas para cada atributo:

```
<nome atr> <tipo de dado> CONSTRAINT <nome Constraint>
    {PRIMARY KEY | UNIQUE | FOREIGN KEY ...
    | CHECK ...}
```

- Restrições de tabela são declaradas separadamente, depois que todos os atributos necessários tenham sido declarados:

```
[CONSTRAINT <nome Constraint>
    PRIMARY KEY(ATR,...) | UNIQUE(ATR,...) |
    FOREIGN KEY ... | CHECK ...]
```

- Ambas podem ser avaliadas de imediato, ou postergadas:

```
[DEFERRABLE | NOT DEFERRABLE]
[INITIALLY DEFERRED | INITIALLY IMMEDIATE]
```

DDL – Comando CREATE TABLE

Restrições Postergadas

- Por padrão, CONSTRAINTS são validadas (aplicadas e verificadas) no momento em que uma instrução DML é executada no Banco de Dados.
 - sempre que uma instrução violar uma CONSTRAINT ela irá gerar um erro "imediatamente", ou seja, a instrução será validada sempre no momento de sua execução.
- É possível adiar a validação da CONSTRAINT para o momento do término da transação (commit).
- São úteis para:
 - Atualização de 2 tabelas que possuem relacionamento. Ex. Incluir uma tupla nova na tabela filha que contém a chave estrangeira antes da tabela mãe (que contém a primária).
 - Inserção de itens em massa.

DDL – Comando CREATE TABLE

Restrições Postergadas

```
CREATE TABLE DEPARTAMENTO (  
  ID INTEGER PRIMARY KEY,  
  NOME VARCHAR(15)  
);
```

```
CREATE TABLE EMPREGADO (  
  ID INTEGER PRIMARY KEY,  
  NOME VARCHAR(30),  
  DEPARTAMENTO_ID INTEGER,  
  CONSTRAINT EMP_DEPTO_FK FOREIGN KEY (DEPARTAMENTO_ID)  
    REFERENCES DEPARTAMENTO (ID)  
    DEFERRABLE INITIALLY DEFERRED  
);
```

DDL – Comando CREATE TABLE

Restrições Postergadas

A qualquer momento é possível alterar o “comportamento” DEFERRABLE da CONSTRAINT:

- a Se for desejado alterar para validar imediatamente (sem postergação):
`SET CONSTRAINTS EMP_DEPTO_FK IMMEDIATE;`
- b Se for desejado alterar para validar somente no final da transação (com postergação):
`SET CONSTRAINTS EMP_DEPTO_FK DEFERRED;`

DDL – Comando CREATE TABLE

Restrições Postergadas

Agora os comandos a seguir **não gerarão erros**:

```
begin transaction;
```

```
INSERT INTO EMPREGADO (ID, NOME, DEPARTAMENTO_ID)  
VALUES (103, 'Zezinho', 23);
```

```
INSERT INTO DEPARTAMENTO (ID, NOME)  
VALUES (23, 'TI');
```

```
COMMIT;
```

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação – Exemplo

Disciplina (Sigla, Nome, NCréditos)

Turma (Código, NNalunos, Sigla, Número, NomeProf, Horário)

```
CREATE TABLE turma (
  codigo      char(7)      PRIMARY KEY,
  nnalunos    decimal(2) NOT NULL,
  sigla       decimal(4) NOT NULL,
  numero      decimal(3) NOT NULL,
  nomeprof    varchar(60),
  horário     time,
  FOREIGN KEY sigladaturma (sigla)
    REFERENCES disciplina (sigla)
    ON DELETE CASCADE ON UPDATE CASCADE,
  UNIQUE siglanumero  (sigla, numero),
  CHECK limitedevagas (nnalunos<50)
);
```

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação – Exemplo

Restrições extendidas – PostgreSQL

É possível designar restrições adicionais para representar diferentes tipos de comparações. Por exemplo, considere um atributo que representa um círculo, não queremos que seja inserido outro círculo que sobreponha algum já inserido na tabela:

```
CREATE TABLE circulo (  
    c    circle,  
    EXCLUDE USING gist (c WITH &&)  
);
```


DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação – Exemplo

- Veja que as seguintes tuplas vão gerar erro, onde o segundo círculo inserido causa violação da restrição, pois intersecta o primeiro círculo já inserido.

```
insert into circulo values ( circle '((1,1),2)' );  
insert into circulo values ( circle '((2,2),5)' );
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplo

```
CREATE TABLE cliente (  
    codCli integer primary key,  
    nome varchar(40),  
    info jsonb  
);
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplo INSERT

```
insert into cliente
values (1, 'Joao', '{
    "endereco":{
        "rua": "Rua X",
        "numero": 12,
        "bairro": "Primavera"},
    "telefones":{
        "residencial": "32255678",
        "celular": "99346222"}
}');
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplo CONSULTAS

- Existem dois operadores para acessar campos no formato JSON
- -> recupera o campo no formato JSON (pode ser aninhado)
- ->> recupera o campo no formato texto (não pode ser aninhado)
- Exemplos (buscando valores via texto)
- info->'endereco' , info->'telefones'
- info->'endereco'->>'rua', info->'telefones'->>'residencial'

DDL – Comando CREATE TABLE

Dados JSON – Exemplo CONSULTAS

- Existem operadores adicionais para procurar valores
- `#>'{a,b}'` procura a sequência de campos pelas chaves a,b.
 - (`#>>` para retornar text)
 - Exemplo:

```
select info #> '{endereco,rua}'  
from cliente
```

- `@>` procura por campos contidos
 - Exemplo:

```
Select nome  
FROM cliente
```

```
WHERE info->'endereco' @> '{"bairro": "Primavera"}'::jsonb;
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplos operadores

- Operador de existência: ?
- Retorna verdadeiro se existe a chave existente no nível superior comparado
- Exemplos:

```
select * from cliente c
```

```
where info ? 'endereco'
```

```
where info->'endereco' ? 'rua'
```

```
where info ? 'telefones'
```

```
where info->'telefones' ? 'celular'
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplos operadores

- Consulta se existem **todas** as chaves no nível superior

```
select *  
from cliente c  
where info->'endereco' ?& array['rua','numero','bairro']
```

- Consulta se existem **alguma** das chaves no nível superior

```
select *  
from cliente c  
where info->'telefones' ?| array['celular','residencial']
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplos operadores

- Transformar JSON em ROW

```
SELECT c.nome, d.*  
FROM   cliente c,  
       jsonb_to_record(c.info->'endereco') AS  
       d(bairro text, rua text, numero int);
```

```
SELECT c.nome, d.*  
FROM   cliente c,  
       jsonb_to_record(c.info->'telefones') AS  
       d(residencial text, celular text);
```


DDL – Comando CREATE TABLE

Dados JSON – Exemplo CONSULTAS

- Como identificar o tipo de dado de uma chave JSON

```
select jsonb_typeof(c.info->'endereco'->'rua')  
from cliente c
```

```
select jsonb_typeof(c.info->'endereco'->'numero')  
from cliente c
```

DDL – Comando CREATE TABLE

Dados JSON – Exemplo CONSULTAS

- transformar cada tupla de uma tabela em JSON separados

```
select row_to_json(c) from cliente c
```

- transformar todas tuplas de uma tabela em um JSON (com as vírgulas)

```
Select array_to_json(array_agg(cliente), FALSE)  
AS ok_json FROM cliente;
```

- Visualizar o JSON de um modo mais amigável

```
select nome, jsonb_pretty(c.info) as Informacoes  
from cliente
```

DDL – Comando ALTER TABLE – Padrão ISO

Modifica tabelas já definidas

Sintaxe do Comando ALTER TABLE - PostgreSQL

```
ALTER TABLE <nome da tabela>
  ADD [COLUMN] <definição da Coluna>
  ADD <restrição de integridade> ex: Chave primária,
                                   Candidata, Estrangeira.

ALTER [COLUMN] <nome da coluna> TYPE <novo tipo>
ALTER <nome da coluna> {SET|DROP} DEFAULT <expressão>
ALTER <nome da coluna> {SET|DROP} NOT NULL

DROP [COLUMN] <nome da coluna>
DROP CONSTRAINT <nome da restrição> [RESTRICT | CASCADE]

RENAME TO <novo nome tabela>
RENAME [COLUMN] <nome da coluna> TO <novo nome>
RENAME CONSTRAINT <nome da restrição> TO <novo nome>
```

DDL – Comando ALTER TABLE

Modifica tabelas já definidas

onde <definição de coluna> pode ser:

```
<Nome Atributo> <Tipo de Dado>  
{SET|DROP} [NULL] | [DEFAULT default-value ]
```

DDL – Comando ALTER TABLE

Modificar tabelas já definidas – Exemplos

```
ALTER TABLE professor
```

```
    ADD COLUMN corcabelos CHAR(25) DEFAULT 'Branco';
```

```
ALTER TABLE aluno ADD COLUMN altura INT DEFAULT NULL;
```

```
ALTER TABLE aluno DROP COLUMN altura;
```

```
ALTER TABLE professor ALTER COLUMN corcabelos TYPE char(30);
```

```
ALTER TABLE aluno ADD COLUMN monitoradiscip char(7);
```

```
ALTER TABLE aluno ADD CONSTRAINT monitor_discip FOREIGN KEY  
    (monitoradiscip) REFERENCES disciplina (sigla)  
    ON UPDATE CASCADE ON DELETE SET NULL;
```

```
ALTER TABLE aluno DROP COLUMN MonitoraDiscip;
```

DDL – Comando DROP TABLE

Remove completamente uma tabela e sua definição

Sintaxe do Comando DROP TABLE

```
DROP TABLE [IF EXISTS] <nome da tabela> [, ...]  
[CASCADE | RESTRICT];
```

- **CASCADE**: Todas as visões que referenciam o atributo são removidas e remove as restrições do tipo FOREIGN KEY das tabelas que a referenciam.
- **RESTRICT**: A tabela só é removida se não houver nenhum objeto que dependa dela.

Exemplo:

```
DROP TABLE aluno;
```

DDL – Comando CREATE DOMAIN

Cria um tipo de dado definido pelo usuário

Sintaxe:

```
CREATE DOMAIN <Nome do Domínio> [AS] <Tipo de Dado>  
    [DEFAULT <expressão>]  
    [<restrição> [ ... ]]
```

onde <restrição>:

```
[CONSTRAINT <Nome_restrição>]  
{NOT NULL | NULL | CHECK (<expressão>) }
```

DDL – Comando CREATE DOMAIN

Cria um tipo de dado definido pelo usuário

Alguns operadores (existem muitos outros em cada SGBD)

```
<expressão> = {  
    VALUE <operador> <val>  
    | VALUE [NOT] BETWEEN <val> AND <val>  
    | VALUE [NOT] LIKE <val> [ESCAPE <val>]  
    | VALUE [NOT] IN ( <val> [, <val> ...])  
    | VALUE IS [NOT] NULL  
}
```

```
<operador> = {= | < | > | <= | >= | !< | !> | <> | !=}
```


DDL – Comando CREATE DOMAIN

Criar tipos de dado definidos pelo usuário – Exemplos

Exemplos:

```
CREATE DOMAIN DNome_Pessoa CHAR (40) NULL;
```

```
CREATE DOMAIN DCodigo INT NOT NULL;
```

```
CREATE DOMAIN DIdade INT  
CHECK (VALUE BETWEEN 1 AND 120);
```

```
CREATE TABLE Pessoa{  
    Nome DNome,  
    Codigo DCodigo,  
    Idade DIdade);
```

DDL – Comando ALTER DOMAIN

Altera um tipo de dado já definido pelo usuário

Sintaxe:

```
ALTER DOMAIN <Nome>  
    {SET DEFAULT expression | DROP DEFAULT}
```

```
ALTER DOMAIN <Nome>  
    {SET | DROP} NOT NULL
```

```
ALTER DOMAIN <Nome>  
    ADD <domain_constraint>
```

```
ALTER DOMAIN <Nome>  
    DROP CONSTRAINT <constraint_name> [RESTRICT | CASCADE]
```

DDL – Comando DROP DOMAIN

Elimina um tipo de dado já definido pelo usuário

Sintaxe:

```
DROP DOMAIN [IF EXISTS] <Nome do Domínio>[, ...]  
[ CASCADE | RESTRICT ]
```

👉 CASCADE: Elimina todas as colunas que dependem do domínio.

👉 RESTRICT: Não remove o domínio se existir qualquer objeto dependente deste domínio.

👉 [IF EXISTS]: Caso o domínio não existe, não emite um erro.

Exemplo:

```
DROP DOMAIN Nome_Pessoa;
```

Linguagem de Manipulação de Dados - DML

- A categoria de Manipulação de Dados tem quatro comandos:

- 1 **INSERT**

Insere tuplas nas tabelas;

- 2 **UPDATE**

Atualiza dados de tuplas existentes;

- 3 **DELETE**

Elimina tuplas com base no critério de busca;

- 4 **SELECT**

Realiza consultas nos dados existentes;

DML – Comando SELECT

Realiza as consultas em uma base de dados

Sintaxe geral do comando SELECT

```
SELECT [ALL | DISTINCT] <lista de atributos>  
FROM <lista de Tabelas>  
[WHERE <condições>]  
[GROUP BY <lista de atributos>  
  [HAVING <condição>]]  
[ORDER BY <Lista de atributos> [ASC|DESC], ...] ;
```

- Somente as cláusulas **SELECT** e **FROM** são obrigatórias.

DML – Comando SELECT

Parte básica: SELECT a FROM t

Um comando SELECT precisa indicar pelo menos os atributos que serão recuperados, de pelo menos uma tabela:

```
SELECT <lista de atributos>  
FROM <tabela>;
```

Por exemplo:

```
SELECT Nome, RA  
FROM Aluno;
```

- Cada atributo da lista é separado por vírgula;
- Nomes dos atributos e tabelas não são sensíveis à caixa da letra (maiúsculas ou minúsculas).

NOME	RA
Carlos	1234
Celso	2345
Cicero	3456
Carlitos	4567
Catarina	5678
...	...

DML – Comando SELECT

Parte básica: SELECT a FROM t

- Se a lista de atributos não contiver uma chave, a resposta pode ter tuplas repetidas.
- A eliminação de repetições pode ser solicitada com a diretriz **DISTINCT**:

```
SELECT DISTINCT Nome, Cidade  
FROM Aluno;
```

DML – Comando SELECT

Parte básica: SELECT a FROM t

- Nomes de atributos e de tabelas podem ter um *alias*:

```
SELECT Nome, RA as RegistroAcademico
FROM Aluno as A;
```

- **PostgreSQL** obrigava o **AS** na lista de atributos, **Oracle** não permite o **AS** na cláusula **FROM**.
- O *alias* pode ser colocado entre " " para que se respeite a caixa do texto ou para usar separadores:

```
SELECT Nome as "Primeiro Nome", RA as "Reg. Acad."
FROM Aluno A;
```

Primeiro Nome	Reg. Acad.
Carlos Silva	1234
Celso	2345
Cicero	3456
Carlitos	4567
Catarina	5678
...	...

DML – Comando SELECT

Parte básica: SELECT a FROM t

- Os atributos podem ser qualificados pela tabela a que pertencem
(Útil quando se envolvem várias tabelas que podem ter nomes de atributos repetidos)

```
SELECT Aluno.Nome, RA  
FROM Aluno;
```

- Sempre que se usa um *alias* numa tabela, a qualificação deve passar a ser feita com ele

```
SELECT A.Nome, Aluno.RA, Idade  
FROM Aluno A;
```

DML – Comando SELECT

Parte básica: SELECT a FROM t

- Quando se quer obter todos os atributos de uma tabela, usa-se * em lugar da lista de atributos

```
SELECT *  
FROM Aluno;
```

– todos os atributos

```
SELECT Aluno.*  
FROM Aluno;
```

– todos os atributos da relação ALUNO

```
SELECT A.*  
FROM Aluno A;
```

– todos os atributos da relação ALUNO

- Usar * facilita escrever comandos quando se está testando comandos, mas não é uma boa prática para programação – se a tabela for atualizada incluindo ou renomeando atributos, um comando programado pode passar a dar erro.

DML – Comando SELECT

Parte básica: SELECT a FROM t

A lista de atributos pode conter:

- O nome de atributos: `SELECT Idade From Aluno;`
- Operações entre atributos;

```
SELECT Nome, Idade as Anos, Idade*12 as Meses  
FROM Aluno A;
```

- Funções `SELECT upper(Nome) FROM Aluno;`
- Expressões CASE;
- Subselects.

DML – Comando SELECT

Expressões CASE

Sintaxe de uma expressão CASE

```
CASE <Expressão>  
    WHEN <Valor> THEN <resultado>  
    [WHEN...]  
    [ELSE <resultado>]  
END
```

ou

```
CASE  
    WHEN <Condição> THEN <resultado>  
    [WHEN...]  
    [ELSE <resultado>]  
END
```

DML – Comando SELECT

Expressões CASE

Por exemplo: (Expressão:)

```
SELECT Nome, CASE Cidade WHEN 'Curitiba' THEN 'Capital'
                ELSE 'Interior'
                END AS 'Região'
FROM Aluno A;
```

ou (Condição:)


```
SELECT Nome, CASE WHEN Idade < 18 THEN 'Adolescente'
                  WHEN Idade BETWEEN 18 AND 24 THEN 'Jovem'
                  ELSE 'Adulto'
                  END AS 'Faixa Etária'
FROM Aluno A;
```

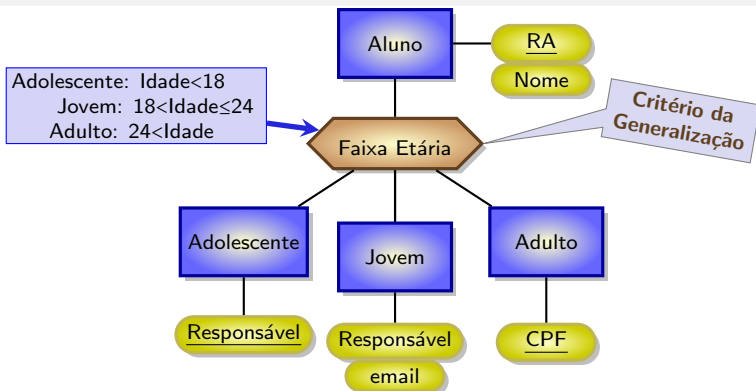
Se não for especificado **ELSE**, valores que não atendam a nenhuma condição **WHEN** assumem **null**.

DML – Comando SELECT

Expressões CASE

- Expressões **CASE** são especialmente interessantes para definir atributos calculados que atuam como identificadores de classes:

 Atributos Critério para a Abstração de Generalização, com predicado definido por regra.



DML – Comando SELECT

Expressões CASE

```
SELECT nome, RA, CASE WHEN Idade < 18 THEN 'Adolescente'
                      WHEN Idade BETWEEN 18 AND 24 THEN 'Jovem'
                      ELSE 'Adulto'
                      END AS "Faixa Etária"
FROM Aluno A;
```

nome	RA	Faixa Etária
Carlos	21	Jovem
Celso	22	Jovem
Cicero	22	Jovem
Carlitos	21	Jovem
Catarina	23	Jovem
Cibele	21	Jovem
Corina	25	Adulto
Celina	27	Adulto
Celia	20	Adolescente
Cesar	21	Jovem
Denise	35	Adulto
Durval		Adulto

DML – Comando SELECT

Condições de comparação na Cláusula WHERE

- Seja uma consulta sobre a seguinte relação:

Aluno (RA, Nome, Sobrenome, Idade, Cidade, Estado)

Consulta:

Encontre os alunos com idade entre 20 e 25 anos com sobrenome contendo 'Silva', da cidade de Pato Branco do estado do PR.

```
SELECT nome, sobrenome  
FROM aluno  
WHERE idade BETWEEN 20 AND 25 AND  
       lower(sobrenome) LIKE '%silva%' AND  
       (Cidade, Estado) = ('Pato Branco', 'PR');
```


DML – Comando SELECT

Condições de comparação

Uma condição de comparação do tipo `<atr> <operator> <val>` pode ser generalizada em SQL para indicar uma comparação entre elementos de tipo **ROW**.

- Um elemento de tipo **ROW** é indicado em SQL como uma sequência de atributos entre parênteses. por exemplo: `(Nome, Idade, Cidade)`.
- Uma comparação do tipo `<atr> <operator> <val>` é equivalente a uma comparação de uma **ROW** de grau 1 (um atributo apenas):

`(<atr>) <operator> (<val>)`

Portanto, o comando:

```
SELECT *
  FROM Aluno
 WHERE Nome='Jose da Silva';
```

é equivalente ao comando:

```
SELECT *
  FROM Aluno
 WHERE (Nome)=('Jose da Silva');
```

DML – Comando SELECT

Condições de comparação

O comando:

```
SELECT *  
FROM Aluno  
WHERE (Nome, NomeMae) = ('Jose da Silva', 'Maria da Silva');
```

é equivalente ao comando:

```
SELECT *  
FROM Aluno  
WHERE Nome = 'Jose da Silva'  
AND NomeMae = 'Maria da Silva';
```

DML – Comando SELECT

Condições de comparação

Além disso, a comparação de atributos textuais respeita a ordem lexicográfica:

```
SELECT *  
  FROM Aluno  
 WHERE (Nome, NomeMae) < ('Jose da Silva', 'Maria da Silva');
```

é equivalente ao comando:

```
SELECT *  
  FROM Aluno  
 WHERE Nome < 'Jose da Silva'  
        OR (Nome = 'Jose da Silva'  
            AND NomeMae < 'Maria da Silva');
```

DML – Comando SELECT

FROM diversas tabelas

Note-se que:

- Para operar N tabelas por junção, deve haver $N - 1$ condições de junção.
 - Se houver menos, será aplicado o produto cartesiano.
 - Pode existir qualquer quantidade de operadores de comparação.

DML – Comando SELECT

FROM diversas tabelas — Exemplo 2

Por exemplo: Listar o horário e o número de alunos atendidos pela monitoria da disciplina de 'Banco de dados' dada pelo aluno 'Zico':

```
SELECT Monitorar.horario, Turma.nnalunos
FROM Aluno, Monitorar, Turma, Disciplina
WHERE Aluno.ra=Monitorar.raaluno AND
       Monitorar.codigoturma=Turma.codigo AND
       Turma.sigla=Disciplina.sigla AND
       Disciplina.nome = 'Banco de Dados' AND
       Aluno.Nome='Zico';
```

Aluno (RA, Nome, Cidade, Idade);
 Monitorar (NomeProf, RAAluno, CódigoTurma, Horário);
 Turma (Código, NNalunos, Sigla, NomeProf, Horário, DataOfere);
 Disciplina (Sigla, Nome, NCréditos);

DML – Comando SELECT

Correspondência entre os operadores de junção com a sintaxe do SQL

- Na teoria existem três operadores de Junção Interna:
 - θ -junção,
 - equi-junção, e
 - junção natural.
- Esses operadores são associados à seguinte sintaxe em SQL:
 - Junções em que o operador de comparação é diferente de '=' são θ -junções;
 - Junções que o operador de comparação na cláusula **WHERE** é '=' ou a construção **ON** na cláusula **FROM** são equi-junções.
 - Junções expressas com a construção **USING** ou **NATURAL JOIN** na cláusula **FROM** são Junções Naturais.

DML – Comando SELECT

Correspondência entre os operadores de junção com a sintaxe do SQL

Exemplo: Suponha que existam as seguintes relações na base de dados:

$R=\{A, B\}$ $S=\{A, C\}$

Então a resposta de:

```
SELECT *  
FROM R JOIN S ON R.A=S.A;
```

ou de

```
SELECT *  
FROM R, S  
WHERE R.A=S.A;
```

tem o esquema:

$Result=\{R.A, R.B, S.A, S.C\}$

Já a resposta de:

```
SELECT *  
FROM R JOIN S USING (A);
```

ou de

```
SELECT *  
FROM R NATURAL JOIN S;
```

tem o esquema:

$Result=\{A, R.B, S.C\}$

DML – Comando SELECT

1-Sub-selects na cláusula FROM

- O resultado de um comando **SELECT** é sempre uma tabela, portanto pode ser usado como uma tabela da cláusula **FROM**, tal como se fosse uma tabela-base.
- Para isso, o subcomando **SELECT** deve ser colocado entre parênteses e sempre deve ter um *alias*;
- Comandos *Sub-select* são úteis especialmente quando a sub-expressão contém operadores de agregação e/ou agrupamento.
- Por exemplo:
Listar as notas em que o aluno 'Zico' foi aprovado:

```
SELECT Aprov.Sigla, Aprov.Nota  
FROM Aluno AS A JOIN (  
    SELECT * FROM Matricula  
    WHERE Nota>=5.0) AS Aprov  
ON A.ra=Aprov.raaluno  
WHERE A.nome='Zico';
```


DML – Comando SELECT

2-Sub-selects como valor de tupla

- Se o resultado de um sub-comando **SELECT** for uma tabela com exatamente uma tupla (ou nenhuma tupla), essa tupla pode ser usada para comparar as tuplas da tabela de consulta.
- Se a tabela resultado de um sub-comando **SELECT** tiver somente um atributo, o parêntese da sintaxe da tupla pode ser omitido
- Por exemplo:
Listar as disciplinas em que o aluno 'Zico' se matriculou:

```
SELECT sigla  
FROM Matricula  
WHERE raaluno=(  
    SELECT ra FROM Aluno  
    WHERE nome='Zico');
```

DML – Comando SELECT

2-Sub-selects como valor de tupla

- Outro exemplo:

Listar as disciplinas em que o aluno 'Zico' se matriculou no ano de 2012:

```
SELECT Sigla
  FROM Matricula
 WHERE (raaluno, ano)=(
        SELECT ra, 2012 AS ano FROM Aluno
        WHERE Nome='Zico');
```

- Note que se a sub-consulta retornar mais do que uma tupla, será gerado um erro durante a execução.
- Se a sub-consulta retornar nenhuma tupla, será considerada a tupla nula (todos os seus atributos têm valor nulo).

DML – Comando SELECT

3-Sub-selects como expressões de tabelas

- Em uma expressão de tabela, um sub-comando **SELECT** é aplicado para cada tupla da relação “externa”.
- A tupla passa para o resultado se o resultado dessa aplicação é **TRUE**.
- Existem os seguintes operadores de expressão de tabelas:
 - **EXISTS / NOT EXISTS** (*< subconsulta >*)
 - (*< tupla >*) **IN / NOT IN** (*< subconsulta >*)
 - (*< tupla >*) **θ ANY / SOME** (*< subconsulta >*)
 - (*< tupla >*) **θ ALL** (*< subconsulta >*)
- Os atributos da tabela “externa” podem ser referenciados na sub-consulta, mas não vice-versa.

Terminologia: **Consulta Correlacionada**

Quando os atributos da tabela “externa” são referenciados na sub-consulta, diz-se que a sub-consulta é correlacionada.

DML – Comando SELECT

3-Sub-selects como expressões de tabelas — Exemplos

- Por exemplo, a seguinte sub-consulta é correlacionada:
Listar os alunos matriculados:

```
SELECT Nome, RA
FROM Aluno
WHERE EXISTS(
    SELECT 'SIM' FROM Matricula
    WHERE Alunos.RA = Matric.RA);
```

- A seguinte sub-consulta é não-correlacionada:
Listar os alunos aprovados em ao menos uma disciplina:

```
SELECT Nome, RA
FROM Aluno
WHERE RA IN (
    SELECT RA FROM Matricula
    WHERE Nota>=5.0);
```

DML – Comando SELECT

3-Sub-selects como expressões de tabelas — Exemplos

- Listar os alunos mais velhos que algum professor:

```
SELECT Nome, RA  
FROM Aluno  
WHERE Idade > ANY (  
    SELECT Idade FROM Professor);
```

- Listar os alunos mais velhos do que qualquer professor:

```
SELECT Nome, RA  
FROM Aluno  
WHERE Idade > ALL (  
    SELECT Idade FROM Professor);
```

DML – Comando SELECT

As Cláusula GROUP BY e HAVING

- A cláusula **GROUP BY** agrupa todas as tuplas da (única) relação resultante das cláusulas **FROM** e **WHERE** e permite calcular atributos agregados sobre cada grupo.

Sintaxe da cláusula GROUP BY

```
SELECT <lista de atributos>...  
    FROM <Lista de tabelas>  
    WHERE <condições>  
    [GROUP BY <Atributo1>[, Atributo2, ...]  
    [HAVING <Condições>]  
    ]
```

- A <lista de atributos> somente pode conter atributos que estão listados na cláusula **GROUP BY** e funções de agregação (ou expressões constantes);
- As condições da cláusula **HAVING** devem ser sobre os atributos agrupados.

DML – Comando SELECT

As Funções de Agregação – Exemplo

Listar quantos alunos existem de cada cidade, qual a menor e maior idade dentre eles e qual a sua média de idade:

```
SELECT Cidade,  
       Count(*), Min(Idade), Max(Idade), Avg(Idade)  
FROM Aluno  
WHERE Cidade IS NOT NULL  
GROUP BY Cidade;
```

👉 Se a cláusula **WHERE** for omitida e houver alguma tupla com o valor de Cidade nulo, haverá uma linha para indicar isso.

Listar quantos alunos existem de cada cidade e cada idade:

```
SELECT Cidade, Idade, Count(*)  
FROM Aluno  
GROUP BY Cidade, Idade;
```

DML – Comando SELECT

As Funções de Agregação – Exemplo

Listar a menor e a maior idade dentre os alunos de cada cidade, das cidades que têm mais de um aluno:

```
SELECT Cidade, Min(Idade), Max(Idade)
FROM Alunos
WHERE Cidade IS NOT NULL
GROUP BY Cidade
HAVING Count(Cidade)>1;
```

Listar a menor e a maior idade dentre os alunos de cada cidade, das cidades que têm mais de um aluno, mas somente as cidades que tenham pelo menos um aluno com idade menor que 18 anos:

```
SELECT Cidade, Min(Idade), Max(Idade)
FROM Alunos
WHERE Cidade IS NOT NULL
GROUP BY Cidade
HAVING Count(Cidade)>1 AND Min(Idade)<18;
```


DML – Comando SELECT

Otimização GROUP BY

- Evite colunas GROUP BY desnecessárias.
 - Quanto mais colunas na lista da cláusula GROUP BY você adicionar, mais o processo de agrupar linhas torna-se dispendioso.
- Se a sua consulta possui poucas colunas de agregação, mas muitas colunas agrupadas não agregadas
 - Reconstrua sua consulta utilizando subqueries. Isso resultará em menos trabalho para agrupar na consulta.

DML – Comando SELECT

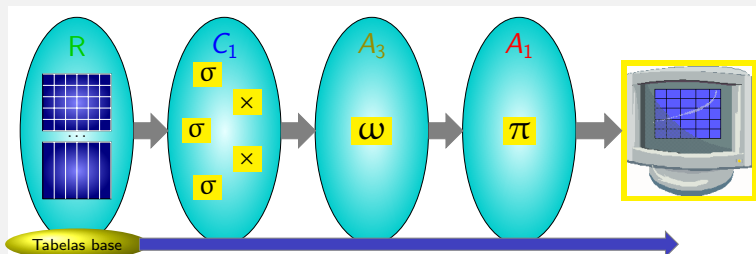
Otimização Junções

- Quando Existirem muitas junções, cujas tabelas são muito grandes. Pode-se reduzir a carga de I/O criando-se **tabelas temporárias**:
`CREATE TEMPORARY TABLE resultado(...);`
- A tabela temporária deve ser populada com parte do que se deseja da consulta original, unindo colunas que estavam em tabelas distintas.
- Você pode então fazer uma junção com tabelas temporárias para produzir um resultado final.
- A tabela temporária é eliminada no final da transação.

DML – Comando SELECT – GROUP BY

Execução de consultas sem usar a cláusula GROUP BY

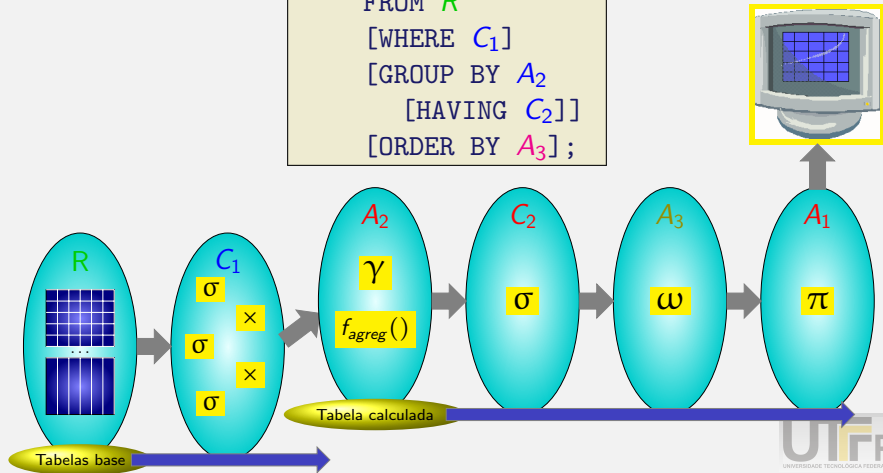
```
SELECT  $A_1$ 
FROM  $R$ 
[WHERE  $C_1$ ]
[ORDER BY  $A_3$ ];
```



DML – Comando SELECT – GROUP BY

Execução de consultas usando a cláusula GROUP BY

```
SELECT  $A_1$ 
FROM  $R$ 
[WHERE  $C_1$ ]
[GROUP BY  $A_2$ 
[HAVING  $C_2$ ]]
[ORDER BY  $A_3$ ];
```



DML – Comando SELECT

Cláusula LIMIT

- A cláusula **LIMIT** permite obter apenas uma parte das tuplas recuperadas pela consulta.

Sintaxe da cláusula LIMIT

```
SELECT <lista de atributos>...  
  FROM <table expression> ...  
  [ORDER BY ... ]  
  [LIMIT {valor1 | ALL}] [OFFSET valor2];
```

- Quando **LIMIT valor1** é dado, esse é o número máximo de tuplas retornado (pode ser menos se não houver esse número para retornar);
- Indicar **LIMIT ALL** é o mesmo que omitir **valor1**;
- Quando **OFFSET valor2** é dado, esse número de tuplas é pulado antes de começar a retornar tuplas;
- É importante usar a cláusula **ORDER BY** nos comandos que têm a cláusula **LIMIT**, para que exista uma ordem única de escolha das tuplas a retornar.

Tratamento de Valores Nulos

Valores Nulos e a “Lógica” de três valores

- Atributos que não têm valores atribuídos são ditos estarem **nulos** (**null**). Atributos de qualquer tipo podem estar nulos.
- Nulo não é um valor, é estado.
- Qualquer comparação entre os valores de dois atributos sempre retorna **Verdade** ou **Falso**. Se um dos atributos estiver nulo, a resposta é desconhecida, e isso é indicado em SQL como **desconhecido** (**unknown**).
- Existem dois predicados de comparação com nulo em SQL: **IS NULL** e **IS NOT NULL**, para testar se algum atributo é ou não **nulo**.
- Todos os operadores lógicos, aritméticos e todos os operadores de comparação são estendidos para contemplar o estado **nulo**.
- Os operadores de seleção na cláusula **WHERE** retornam apenas as tuplas que resultam **Verdade**.

Tratamento de Valores Nulos

Funções específicas para tratar valores nulos

Existem diversas funções específicas para tratar valores nulos, dentre elas:

- `COALESCE(value [, ...])`
- `NULLIF(valor1, valor2)`

Tratamento de Valores Nulos

Função COALESCE

```
COALESCE(value [, ...])
```

- Essa função retorna o primeiro valor da lista de argumentos que é não nulo.
- Se todos os argumentos forem nulos, então retorna nulo.
- Os argumentos são avaliados em sequência, e aqueles à direita do primeiro não nulo não são avaliados.

Por exemplo, suponha que existe a relação:

```
FonePessoa (RG, Resid, Celular, Comerc, Pais, Esposa)
```

Pode-se pedir um telefone individual e um de emergência assim:

```
SELECT RG,  
       COALESCE (Resid, Celular, Comerc) AS Individual,  
       COALESCE (Pais, Esposa, 'não tem') AS Emergência  
FROM FonePessoa  
WHERE RG='1234';
```


Tratamento de Valores Nulos

Função NULLIF

```
NULLIF(valor1, valor2)
```

- Essa função retorna nulo se **valor1** é igual a **valor2**, caso contrário retorna **valor1**.
- Ela pode ser vista como uma função que faz o inverso de **COALESCE**.

Por exemplo, suponha que existe a relação:

```
FonePessoa (RG, Resid, Celular, Comercial, Pais, Esposa)
```

Pode-se pedir o telefone residencial e comercial sem repetir o telefone assim:

```
SELECT Resid, NULLIF (Comercial, Resid)
FROM FonePessoa
WHERE RG='1234';
```

O catálogo

- Um SGBD em geral pode gerenciar mais do que uma base de dados.
- Cada base de dados tem suas próprias tabelas, índices, usuários, etc.
- Sempre que se faz uma conexão ao SGBD, existe uma Base *default*.
- Para acessar um objeto fora da base *default*, deve-se nomear explicitamente a base.
- Por exemplo, se a base *default* for **AnoLetivo**, para acessar uma tabela na base **Histórica** é necessário nomeá-la:

```
SELECT A.Nome, B.Nome  
      FROM Aluno AS A,  
           Historica.Aluno AS B,  
           ...
```

A tabela A é a tabela **Aluno** da base **AnoLetivo**, e a tabela B é a tabela **Aluno** da base **Historica**.

DML – Comando SELECT

Meta-modelo Relacional

Terminologia: **Meta-Modelo**

Um modelo capaz de modelar a si mesmo é chamado um Meta-Modelo.

- O Modelo Relacional é um Meta Modelo
- Um SGBD implementa o meta-modelo relacional com Tabelas de Tabelas, Tabelas de Atributos, etc.
- Essas tabelas são ditas “**do sistema**” e são mantidas em uma base especial, chamada **catálogo**
- O Catálogo inclui todas as tabelas gerenciadas, em qualquer base.

DML – Comando SELECT

Meta-modelo Relacional – **Postgres**

No gerenciador *Postgres*, o catálogo é mantido em um esquema separado, chamado *Information_Schema*. Os nomes dos objetos do sistema começam com 'pg_' ou 'sql_'.

Por exemplo: Listar todas as tabelas do usuário

```
SELECT table_name, table_type, table_catalog
FROM information_schema.tables
WHERE table_type='BASE TABLE' and
      table_name not like 'pg_%' and
      table_name not like 'sql_%';
```

DML – Comando SELECT

Meta-modelo Relacional – **Postgres**

Exemplo 4: Listar as colunas de todas as tabelas do usuário:

```
SELECT Cols.table_name, Cols.column_name,  
       Cols.ordinal_position  
FROM information_schema.columns Cols,  
     information_schema.tables Tabs  
WHERE Tabs.table_type='BASE TABLE' AND  
       Tabs.table_name=Cols.table_name AND  
       Tabs.table_name not like 'pg_%' AND  
       Tabs.table_name not like 'sql_%';
```

DML – Comando INSERT INTO

Insere tuplas em uma Relação

Sintaxe:

- Formato 1: Insere uma tupla de cada vez.

```
INSERT INTO <Tabela> [( <Atributo>, ... )]  
VALUES ( expression | DEFAULT, ... );
```

- Formato 2: Insere múltiplas tuplas a partir de uma tabela.

```
INSERT INTO <Tabela> [( <Atributo>, ... )]  
<Comando SELECT>;
```

DML – Comando INSERT INTO - DML

Exemplos

- Formato 1: Insere uma tupla de cada vez.

```
insert into Professor values ('Antonio','5656','MS-3',33);
```

```
insert into Professor ( Nome, Grau, NNfuncional)  
values ('Antoninho', 'MS-3', '5757');
```

- Formato 2: Insere múltiplas tuplas a partir de uma tabela.

```
INSERT INTO pessoa ( Nome, Idade )  
SELECT nome, idade from Aluno;
```

```
INSERT INTO pessoa ( Nome, Idade )  
SELECT nome, idade from Professor;
```

```
INSERT INTO Patobranquense  
SELECT *  
FROM aluno  
WHERE cidade = 'Pato Branco';
```

DML – Comando UPDATE

Altera o valor de atributos de tuplas de uma relação

Sintaxe geral do comando UPDATE

```
UPDATE <tabela>  
    SET <Atributo> = <expressão>, ...  
    [WHERE <Condição>]  
    ;
```

`<expressão> = <Atributo>|<constante>|<expr>|NULL`

Onde `<expr>` é qualquer comando `SELECT` que resulte em apenas uma tupla e uma coluna.

DML – Comando UPDATE- DML

Exemplos

Aumentar em uma unidade a idade de todos os alunos.

```
UPDATE Aluno  
SET Idade=Idade+1;
```

Contar quantas **matrículas** numa existem numa determinada **turma** na relação de **Matrículas** para atualizar a relação de **turmas**.

```
UPDATE Turma  
SET NNAunos=(  
    SELECT count(*)  
    FROM Matricula  
    WHERE codigoTurma=101)  
WHERE Codigo=101;
```

Note-se que a cláusula WHERE deve selecionar apenas as tuplas da **Turma** com **código=101**.

DML – Comando UPDATE- DML

Exemplos

Atualizar todas as tuplas da relação **Turma**, contando quantas **matrículas** existem em cada **turma** na relação de **Matrículas**.

```
UPDATE Turma
  SET NNAunos=(
    SELECT count(*)
      FROM Matricula
     WHERE Matricula.codigoTurma=Turma.Codigo
  )
;
```

Note-se que com a omissão da cláusula **WHERE** do comando **UPDATE**, todas as tuplas da relação **Turma** são atualizadas.

DML – Comando UPDATE- DML

Exemplos

Assuma que existem as seguintes relações:

```
Matricula={CodigoTurma, RA, Prova1, Prova2, NotaFinal}
```

```
Prova={CodigoTurma, RA, Nota}
```

Atualizar a relação de Matrículas para inserir as notas da Prova2 indicadas na relação Prova e calcular a NotaFinal correspondente:

```
UPDATE Matricula M
  SET (Prova2, NotaFinal)=
      (SELECT P.Nota, (P.nota+M.Prova1)/2.0
       FROM Prova P
       WHERE P.codigoTurma=M.codigoTurma AND
            P.RA=M.RA)
```

DML – Comando DELETE FROM

Remove tuplas de uma relação

Sintaxe geral do comando DELETE

```
DELETE [FROM] <tabela>  
    [WHERE <Condição>]  
;
```

Exemplos:

Apagar todas as tuplas do **aluno** cujo **NUSP** vale 1234:

```
DELETE FROM Aluno  
WHERE RA=1234;
```

Remover todos os **Alunos** em que o atributo **Cidade** tem o valor indicado:

```
DELETE FROM Aluno  
    WHERE Cidade = 'Mirim-Guaçu';
```

Apagar todas as tuplas da relação **Aluno**.

```
DELETE FROM Aluno;
```

Operadores de comparação

- Os tipos de dados básicos de SQL envolvem números, cadeias de caracteres e datas.
- Todos eles podem ser comparados pelos tipos básicos de operadores:

<	menor que
>	maior que
<=	menor ou igual que
>=	maior ou igual que
=	igual
<> (ISO) <>, != (PostgreSQL) <>, !=, ^= (Oracle)	diferente

- pelo operador de faixa (*range*): `atr [NOT] BETWEEN x AND y`
que é equivalente a `atr >= x AND a <= y`.
- e pelo operador de padrão (*match*): `atr [NOT] LIKE x`
que é verdade se *atr* segue o padrão *x*. '%' em *x* bate com qualquer cadeia de zero ou mais caracteres. '_' em *x* bate com qualquer caractere.

Operadores de comparação – comparação com nulo

- Lembre-se que nulo não é valor: é estado, é ausência de valor.
- Para comparar com nulo, é necessário usar:

```
<expr> IS NULL
<expr> IS NOT NULL
```

- Se a expressão for uma tupla, então
 - **IS NULL** retorna verdade quando a tupla é nula ou todos os seus atributos são nulos;
 - **IS NOT NULL** retorna verdade quando a tupla não é nula e todos os seus atributos são não nulos;
 - portanto, **IS NULL** e **IS NOT NULL** nem sempre tem valores negados.
- Comparadores tradicionais retornam verdade, falso ou **Nulo** (“unknown”) quando algum lado é **Nulo**.

Já:

```
<expr> IS DISTINCT FROM <expr>
```



```
<expr> IS NOT DISTINCT FROM <expr>
```

 não retornam nulo.

Para duas expressões não nulas, **IS DISTINCT FROM** é equivalente a **<>**.

Se apenas uma expressão é nula ele retorna **Verdade** e se ambas são nulas ele retorna **Falso**. **IS NOT DISTINCT FROM** faz o oposto.

Uso de Funções

Funções usadas em comandos da DML

Funções podem ser utilizadas em geral, em qualquer lugar onde um `<atributo>` pode ser utilizado. Por exemplo, nas cláusulas `SELECT` e `WHERE` do comando `SELECT`, etc

Existem funções para todos os tipos de dados da linguagem, como por exemplo:

- 1 Números
- 2 Cadeias de caracteres
- 3 Datas
- 4 e funções especiais para Agregações.

Uso de Funções

Funções sobre tipos de dados: Funções Matemáticas – **PostgreSQL**

Função	Descrição	Exemplo	Resultado
<code>abs(x)</code>	valor absoluto	<code>abs(-5.2)</code>	5.2
<code>cbrt(dp)</code>	Raiz Cúbica	<code>cbrt(27.0)</code>	3
<code>ceil(dp)</code>	Próximo inteiro \geq parâmetro	<code>ceil(4.8)</code>	5
<code>degrees(dp)</code>	radianos para graus	<code>degrees(0.5)</code>	28.6479
<code>div(y, x)</code>	quociente inteiro de y/x	<code>div(9/4)</code>	2
<code>exp(dp)</code>	exponencial	<code>exp(1.0)</code>	2.7183
<code>floor(dp)</code>	Próximo inteiro \leq parâmetro	<code>floor(4.8)</code>	4
<code>ln(dp)</code>	logaritmo natural	<code>ln(2.0)</code>	0.6931
<code>log(dp)</code>	logaritmo na base 10	<code>log(100)</code>	2
<code>log(b,x)</code>	logaritmo de x na base b	<code>log(2.0, 64.0)</code>	6.0

Uso de Funções

Funções sobre tipos de dados: Funções Matemáticas – **PostgreSQL**

Função	Descrição	Exemplo	Resultado
<code>mod(y,x)</code>	resto de y/x	<code>mod(9,4)</code>	1
<code>pi()</code>	número pi	<code>pi()</code>	3.1415
<code>power(a, b)</code>	a^b	<code>power(9,3)</code>	729
<code>radians(dp)</code>	graus para radianos	<code>radians(45)</code>	0.7854
<code>round(dp)</code>	arredonda próx inteiro	<code>round(42.4)</code>	42
<code>round(v, s)</code>	arredonda v em s casas	<code>round(2.454, 2)</code>	2.46
<code>sign(dp)</code>	sinal do parâmetro	<code>sign(-2.4)</code>	-1
<code>sqrt(dp)</code>	raiz quadrada	<code>sqrt(64.0)</code>	8.0
<code>trunc(dp)</code>	trunca o valor	<code>trunc(6.456)</code>	6

Uso de Funções

Funções sobre tipos de dados: Funções Matemáticas – **PostgreSQL**

Função	Descrição	Exemplo	Resultado
random()	valor aleatório entre 0 e 1		
setseed(dp)	altera a semente		
cos(x)	cosseno	cos(radians(60))	0.5
acos	cosseno inverso	degrees(acos(0.5))	60
sin(x)	seno	sin(radians(30))	0.5
asin(x)	seno inverso	degrees(asin(0.5))	30
tan(x)	tangente	tan(radians(45))	1
atan(x)	tangente inversa	degrees(atan(1))	45
cot(x)	cotangente	cot(10)	1.5423

Uso de Funções

Funções sobre tipos de dados: Funções para Strings – **PostgreSQL**

Função	Descrição	Exemplo	Resultado
<code>string string</code>	concatena strings	<code>'Post' 'greSQL'</code>	PostgreSQL
<code>string numeric</code>	concatena strings com números		
<code>bit_length</code>	número de bits na string	<code>bit_length('SGBD')</code>	32
<code>char_length(string)</code>	número de caracteres da string	<code>char_length('SGDB')</code>	4
<code>lower(string)</code>	converte a string para letras minúsculas	<code>lower('SISTEMA')</code>	sistema
<code>upper(string)</code>	converte a string para letras maiúsculas	<code>upper('sistema')</code>	SISTEMA
<code>atan(x)</code>	tangente inversa	<code>degrees(atan(1))</code>	45

Uso de Funções

Funções sobre tipos de dados: Strings

- O comando TRIM tem a forma:
`trim([leading | trailing | both] [caracteres] from string1)`
- Ele serve para remover os caracteres do início, do fim ou ambos (default) da string1, de acordo com a lista passada de caracteres.

Exemplo: `trim('xy' from 'xyxySGBDyyy')` = SGDB

- O comando LIKE procura padrões entre as strings. Por exemplo

<code>'abc' LIKE 'abc'</code>	<code>true</code>
<code>'abc' LIKE 'a%'</code>	<code>true</code>
<code>'abc' LIKE '_b_'</code>	<code>true</code>
<code>'abc' LIKE 'c'</code>	<code>false</code>

Uso de Funções

Funções sobre tipos de dados: Funções para Datas – **PostgreSQL**

Função	Descrição	Exemplo	Resultado
<code>now()</code>	data e horário atual		
<code>current_date</code>	data atual		
<code>current_time</code>	horário atual		
<code>timeofday()</code>	data e horário atual (texto)		
<code>makedate(ano, mes, dia)</code>	cria uma data	<code>make_date(2017, 3, 20)</code>	2017-03-20
<code>extract(field from timestamp)</code>	extraí um campo da data	<code>extract(hour from now())</code>	21
<code>age(timestamp)</code>	Calcula a idade	<code>age(timestamp '1990-03-01')</code>	26 years 11 mons 14 days

Uso de Funções

Funções sobre tipos de dados: Miscelânea

- Mudanças do tipo dos dados podem ser feitas através do comando

`CAST(expression AS DataType)` que define um CAST.

- Um CAST serve para especificar como realizar uma conversão entre dois tipos.
- Por exemplo, o comando

```
SELECT CAST(42 AS float8);
```

converte um número inteiro para o tipo float8.

Roteiro

1 Introdução

2 DDL

3 DML

- Comando SELECT
 - SELECT a FROM t
 - SELECT a FROM t WHERE c
 - Agrupamentos e Agregações
 - Ordem de execução dos comandos
 - SELECT ... LIMIT
 - Tratamento de Valores Nulos
 - O catálogo
- Comando INSERT INTO
- Comando UPDATE
- Comando DELETE

Banco de Dados

SQL – Conceitos e Principais Comandos

Prof. Dr. Ives Renê V. Pola

ivesr@utfpr.edu.br

Departamento Acadêmico de Informática – DAINF

UTFPR – Pato Branco DAINF

UTFPR

Pato Branco - PR

FIM

