

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 é uma linguagem de consulta sofisticada, que vem evoluindo continuamente desde sua criação, embora mantendo um nível de padronização muito alto.
- Entre seus principais atrativos está a pequena quantidade de comandos extremamente poderosos
 - atendendo ao paradigma Relacional, ou seja, o programador expressa em SQL “o que” ele quer recuperar, não “como” deve ser recuperado.
- SQL é padronizado pelo “*American National Standard Institute*” (ANSI) e pela “*International Standard Organization*” (ISO).
- Sua última versão integral foi aprovada pela ISO em 2001, tendo sido designada SQL:2001.
- Extensões e correções de pouco impacto foram publicadas em 2003, 2006, 2008 e 2011.

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.

Linguagem de Definição de Dados - DDL

- Elementos fundamentais da linguagem:
 - DATABASE
 - USER
 - ROLE
 - RULE
 - SCHEMA
 - TABLESPACE
 - TABLE
 - INDEX
 - DOMAIN
 - FUNCTION
 - SEQUENCE
 - TRIGGER
 - VIEW
- Todos os elementos podem ser criados (CREATE), corrigidos (ALTER) e removidos (DROP).

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação

Sintaxe geral de um Comando CREATE TABLE

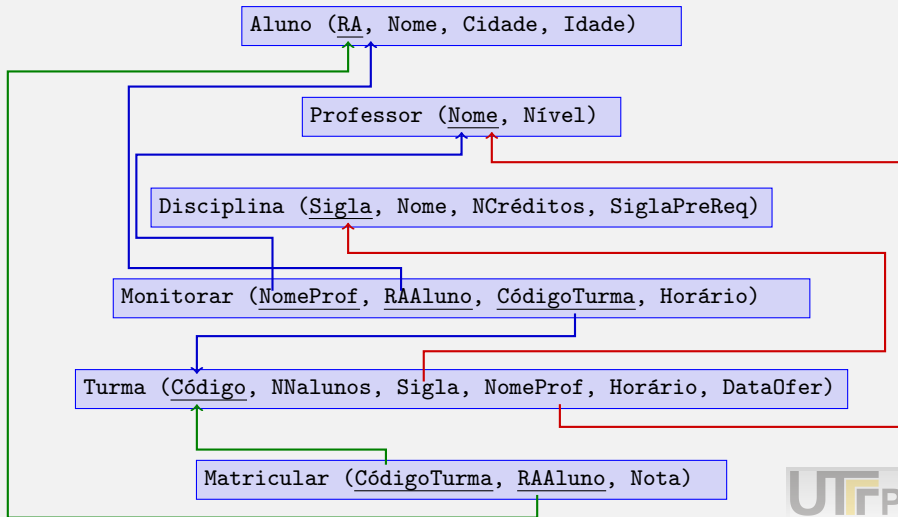
```
CREATE TABLE <nome da tabela> (  
    <definição de Coluna>,...  
    <Restrições de Integridade>,...  
);
```

onde <definição de Coluna> pode ser:

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

DDL – Comando CREATE TABLE

Esquema da Aplicação – Recordando



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

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 ...]
```

DDL – Comando CREATE TABLE

Criar uma Tabela no Esquema da Aplicação – Exemplo

Exemplo: Criar uma tabela com as Restrições de Integridade, como **restrição de atributo**

Aluno (RA, Nome, Cidade, Idade)

```
CREATE TABLE Aluno (  
    RA      decimal(7)  PRIMARY KEY,  
    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 as Restrições de Integridade, como **restrição de tabela**

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,  
    CONSTRAINT codigo_aluno PRIMARY KEY (ra)  
);
```

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,
  CONSTRAINT sigladaturma FOREIGN KEY (sigla)
    REFERENCES disciplina (sigla)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT siglanumero UNIQUE (sigla, numero),
  CONSTRAINT limitedevagas CHECK (nnalunos<50)
);
```


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

Criar tipos de dado definidos pelo usuário – Exemplos

Comando CREATE DOMAIN:

```
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);
```

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 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 Matricular  
    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 NNAalunos=(
    SELECT count(*)
      FROM Matricular
     WHERE Matricular.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 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;
```

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** obriga 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;
```

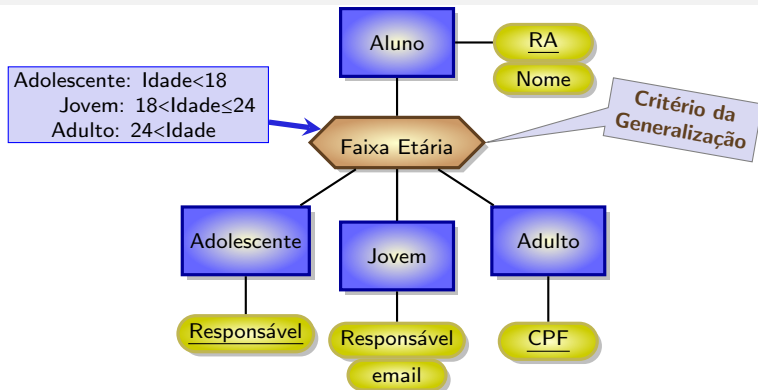
Se não for especificado **ELSE**, valores que não atendam a nenhuma condição **WHEN** assumem **null**.

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

nome	Idade	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	15	Adolescente
Cesar	21	Jovem
Denise	35	Adulto
Durval		Adulto

DML – Comando SELECT

Parte básica: `SELECT a FROM t WHERE c`

Note-se que num comando `SELECT`:

- Sempre é preciso indicar ao menos uma tabela, pois é de onde se busca os dados;
- A lista de atributos corresponde a um operador de projeção da álgebra relacional:

`SELECT a FROM R_1` é equivalente a $\pi_{\{a\}} R_1$.

- Para indicar um operador de seleção, é usada a cláusula `WHERE`, onde as condições se aplicam ou a operadores de seleção ou a junções internas:

```
SELECT <lista de atributos>
      FROM <tabela>;
      [WHERE <condições>];
```

- Cada <condição> `c` da cláusula `WHERE` gera um operador algébrico, que pode ser $\sigma_{(c)} R_1$ ou $R_1 \bowtie_c R_2$, dependendo da condição.

DML – Comando SELECT

Parte básica: SELECT a FROM t WHERE c

- Se o atributo a_1 for de uma tabela R_1 e a comparação é com uma constante cte ou outro atributo a_2 da mesma tabela, então a condição corresponde a uma seleção: $\sigma_{(a_1 \theta cte)} R_1$ ou $\sigma_{(a_1 \theta a_2)} R_1$.
- Se o atributo a_1 for de uma tabela R_1 e o outro atributo a_2 for de uma outra tabela R_2 , então a condição corresponde a uma junção:

$$\begin{matrix} (R_1.a_1 \theta R_2.a_2) \\ R_1 \bowtie R_2 \end{matrix}.$$

DML – Comando SELECT

Condições de comparação na Cláusula WHERE

Condições de comparação na Cláusula WHERE

```
<condição> = {  
<atr> <operator> <val>  
| <atr> [NOT] BETWEEN <val1> AND <val2>  
| <atr> [NOT] LIKE <val1> [ESCAPE <val2>]  
| <atr> IS [NOT] NULL  
| <atr> [NOT] IN ( <val1> [, <valn> ] | <subconsulta> )  
| EXISTS ( <select expr> )  
| <atr> { $\theta$  {ALL|SOME|ANY} ( <subconsulta> )  
| <condição> OR <condição>  
| <condição> AND <condição>  
}
```


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

FROM diversas tabelas

- A cláusula **FROM** permite indicar mais de uma tabela para recuperar dados;
- Como o resultado do comando é sempre exatamente **uma** tabela, elas têm que ser operadas por Junção $R_1 \bowtie_c R_2$ ou Produto Cartesiano

$$R_1 \times R_2.$$



Veja que a junção requer a comparação $c = (R_1.a_1 \theta R_2.a_2)$ de um (ou mais) atributo a_1 da relação R_1 com um (ou mais) atributo a_2 da relação R_2 . Caso essa condição não esteja expressa, é realizado o produto cartesiano.

- A condição c pode ser expressa na cláusula **WHERE** ou na cláusula **FROM**.

DML – Comando SELECT

FROM diversas tabelas — Exemplo 1

Listar as notas em que o aluno 'Zico' foi aprovado:

```
SELECT M.sigla, nota
FROM Aluno A, Matricula M
WHERE A.ra=M.raaluno AND
      M.nota>=5.0 AND
      A.Nome='Zico';
```

que é equivalente a:

$$\sigma(\text{Aluno.nome}='Zico' \wedge \text{Matricular.Nota} \geq 5.0) \text{ Aluno } \bowtie \text{ RA=RAAluno } \text{Matricular}$$

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

Junção na Cláusula WHERE

- Junções podem ser expressas na cláusula **WHERE** ou na cláusula **FROM**.
- Junções são expressas na cláusula **WHERE** como comparações entre atributos das duas relações R_1 e R_2 envolvidas na operação:
 $R_1.a_1 \theta R_2.a_1 \wedge R_1.a_2 \theta R_2.a_2 \wedge \dots R_1.a_j \theta R_2.a_j$, sendo que as relações são indicadas na cláusula **FROM** separadas por vírgulas.
- Na cláusula **WHERE** somente podem ser expressas equi-junções e θ -junções.

Exemplo: Para cada aluno, listar os códigos de todas as disciplinas em que eles se matricularam (equi-junção):

```
SELECT A.nome, M.sigla  
FROM Aluno A, Matricular M  
WHERE A.ra = M.raaluno;
```

DML – Comando SELECT

Exemplo de OUTER JOIN

Exemplo de **junção natural**:

Listar os nomes de todas as disciplinas e o código de suas turmas criadas:

```
SELECT D.nome, T.codigo  
FROM Disciplina D JOIN Turma T USING sigla;
```

Exemplo de **junção externa**:

Listar todas as disciplinas, com seus respectivos pré-requisitos.

```
SELECT D.sigla, D.nome, Pre.sigla  
FROM Discip D LEFT OUTER JOIN Discip Pre  
ON D.siglaprereq=Pre.sigla;
```

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 turmas e as notas em que o aluno 'Zico' foi aprovado:

```
SELECT Aprov.codigoturma, Aprov.Nota  
FROM Aluno AS A JOIN (  
    SELECT * FROM Matricular  
    WHERE Nota>=6.0) 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 siglas das disciplinas em que o aluno 'Zico' se matriculou:

```
SELECT T.sigla
      FROM Matricular as M join
Turma as T on M.codigoturma =
T.codigo
      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 2017:

```
SELECT M.codigoturma, T.sigla
FROM Matricular AS M join turma AS T on
M.codigoturma=T.codigo
WHERE M.raaluno=(
    SELECT ra FROM Aluno
    WHERE Nome='Zico') and extract(year from
T.dataofer) = 2017;
```

- 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 — Exemplos

- A seguinte sub-consulta é correlacionada:

Listar os alunos matriculados:

```
SELECT nome, ra
FROM aluno
WHERE EXISTS(
    SELECT 'SIM' FROM Matricular
    WHERE aluno.ra = matricular.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 Matricular
    WHERE Nota >= 6.0);
```

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

Terminologia: Funções de Agregação

Funções de agregação recebem como argumento um ou mais atributos e retornam um valor que sumariza todos os valores que esse(s) atributo(s) assume(m) em todas as tuplas relação.

- Se a função de agregação está no escopo de um comando que não tem uma cláusula **GROUP BY**, então ela age sobre a relação inteira e o resultado tem exatamente uma tupla, caso contrário, ela age sobre cada grupo gerado pela cláusula **GROUP BY** e o resultado tem uma tupla para cada grupo.
- O padrão SQL define uma coleção de funções de agregação básicas, e cada gerenciador pode acrescentar as que interessar.

DML – Comando SELECT

As Funções de Agregação

Funções de Agregação podem ser utilizadas apenas na lista de atributos do comando **SELECT** ou nas cláusulas **HAVING** e **ORDER BY**.

As Funções de Agregação mais comuns são:

- `AVG(<Param agreg>)` – Retorna a média
- `COUNT(*)` ou `COUNT(Param agreg)` – Retorna o número de tuplas.
- `MAX(<Param agreg>)` – Retorna o maior valor encontrado
- `MIN(<Param agreg>)` – Retorna o menor valor encontrado
- `SUM(<Param agreg>)` – Retorna a soma dos valores



Funções que recebem `<Param agreg>` não consideram as tuplas onde esse valor é nulo.

Onde: `<Param agreg> = {DISTINCT <atributos> | <atributos>}`

DML – Comando SELECT

As Funções de Agregação – Exemplo

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

```
SELECT Count(*), Min(Idade), Max(Idade), Avg(Idade)
FROM Aluno;
```

Listar quantos alunos não têm cidade indicada:

```
SELECT Count(*)
FROM Aluno
WHERE Idade IS NULL;
```


👉 Veja que, por não haver comando **GROUP BY**, o resultado sempre tem uma única tupla.

DML – Comando SELECT

As Funções de Agregação – Exemplo

Listar quantos alunos existem, quantos têm idade indicada, e quantas idades distintas existem:

```
select count(*), count(idade), count(distinct idade)
FROM Aluno;
```

 Lembrar que funções que recebem <Param agreg> não consideram as tuplas onde esse valor é nulo.

Mostrar a média de idade, não considerando e considerando valores `null`:

```
SELECT AVG(Idade), AVG(COALESCE(Idade,0))
FROM Aluno;
```

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 Aluno
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 Aluno
WHERE Cidade IS NOT NULL
GROUP BY Cidade
HAVING Count(Cidade)>1 AND Min(Idade)<18;
```

DML – Comando SELECT – ORDER BY

Execução de consultas sem usar a cláusula GROUP BY – Exemplo

Exemplo: Listar os alunos, disciplinas e notas tiradas em turmas que tenham registrados mais de 2 alunos

```
SELECT A.Nome, T.Sigla AS Disciplinas, M.Nota  
FROM Aluno A, Turma T, Matricular M  
WHERE A.RA=M.RAAluno AND  
      M.CodigoTurma=T.Codigo AND  
      T.NNAAlunos>2
```

DML – Comando SELECT – ORDER BY

Execução de consultas sem usar a cláusula GROUP BY – Exemplo

Exemplo: Listar a média das notas das disciplinas em turmas com mais de 1 aluno matriculados.

```
SELECT T.Sigla, AVG(M.Nota)
FROM Turma T, Matricular M
WHERE M.CodigoTurma=T.Codigo
GROUP BY T.sigla
HAVING Count(T.sigla)>1;
```

DML – Comando SELECT

Operadores de conjunto – Exemplo

Exemplo: Listar Todos os Alunos e Professores com respectivas idades, ordenados pela idade. Quando existirem idades iguais, colocar primeiro os professores, depois os alunos, em qualquer ordem dentro de cada grupo:

```
SELECT Nome, idade, '2-Aluno' AS Tipo FROM Aluno
      UNION
SELECT Nome, idade, '1-Professor' AS Tipo FROM Professor
      ORDER BY Idade, Tipo;
```

DML – Comando SELECT

Cláusula ORDER BY

Listar todos os alunos em ordem alfabética de seu nome, separados pela cidade. Se houver mais de um aluno com o mesmo nome da mesma cidade, ordenar pela ordem descendente de idade. Listar primeiro os alunos com cidade desconhecida:

```
SELECT RA, Nome, Cidade  
FROM Aluno  
ORDER BY Cidade NULLS FIRST, Nome, Idade DESC;
```


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.

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
abs(x)	valor absoluto	abs(-5.2)	5.2
cbrt(dp)	Raiz Cúbica	cbrt(27.0)	3
ceil(dp)	Próximo inteiro \geq parâmetro	ceil(4.8)	5
degrees(dp)	radianos para graus	degrees(0.5)	28.6479
div(y, x)	quociente inteiro de y/x	div(9/4)	2
exp(dp)	exponencial	exp(1.0)	2.7183
floor(dp)	Próximo inteiro \leq parâmetro	floor(4.8)	4
ln(dp)	logaritmo natural	ln(2.0)	0.6931
log(dp)	logaritmo na base 10	log(100)	2
log(b,x)	logaritmo de x na base b	log(2.0, 64.0)	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
string string	concatena strings	'Post' 'greSQL'	PostgreSQL
string numeric	concatena strings com números		
bit_length	número de bits na string	bit_length('SGBD')	32
char_length(string)	número de caracteres da string	char_length('SGDB')	4
lower(string)	converte a string para letras minúsculas	lower('SISTEMA')	sistema
upper(string)	converte a string para letras maiúsculas	upper('sistema')	SISTEMA
atan(x)	tangente inversa	degrees(atan(1))	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

Roteiro

1 Introdução

2 DDL

3 DML

- Comando SELECT

- SELECT a FROM t
- SELECT a FROM t WHERE c
- Agrupamentos e Agregações
- Operadores de Conjuntos: UNION, INTERSECT, EXCEPT
- SELECT ... ORDER BY
- SELECT ... LIMIT

- Operadores e funções

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

