SQL

STRUCTURED QUERY LANGUAGE
LINGUAGEM DE CONSULTA ESTRUTURA

QUERIES

AGENDA

- INTRODUÇÃO
- EXPRESSÕES DA TABELA
  - FROM*
  - JOINS
- ALIAS
- SUB CONSULTAS
- WHERE
- GROUP BY
- HAVING
- DISTINCT
- ORDER BY
- LIMIT

TABELAS EXEMPLO

```
CREATE TABLE departamentos(
    id_departamento INT PRIMARY KEY,
    nome VARCHAR(60),
    responsavel VARCHAR(60)
);

CREATE TABLE grupos(
    id_grupo serial,
    nome VARCHAR(60),
    id_departamento INT REFERENCES departamentos
);
```

INTRODUÇÃO

- Além de armazenar os dados no banco de dados de forma satisfatória, adequada e íntegra, é necessário recuperá-los para consumo do usuário.
- Queries ou consultas são comandos utilizados para fazer essa recuperação.
- Obedecem a estrutura do banco de dados, ou melhor, é necessária conhecer a estrutura do BD para poder recuperar os dados de forma correta e útil.
- Diversos recursos podem ser utilizados aqui

INTRODUÇÃO

- O comando para executar uma *query* é o SELECT.
- Sintaxe geral é:

```
SELECT <campos> FROM <tabela> [condições, ordem, agrupamento]
```

- Alguns exemplos gerais:

```
SELECT * FROM departamentos;
```

```
SELECT 2 + 2;
```

```
SELECT random();
```

EXPRESSÕES DE TABELA

- O comando para executar uma *query* é o SELECT.
- Sintaxe geral é:

```
SELECT <campos> FROM <tabela> [condições, ordem, agrupamento]
```

- Alguns exemplos gerais:

```
SELECT * FROM departamentos;
```

```
SELECT 2 + 2;
```

```
SELECT random();
```

CLÁUSULA FROM

- Lugar onde são indicadas uma ou mais tabelas de onde será buscados os dados, separadas por “,”
- Pode ser usado:
  - um nome de uma tabela (com schema ou não)
  - Sub Consulta
  - Uma construção de JOIN
  - Ou uma combinação complexa de todos eles
- O resultado é uma tabela virtual que pode ser modificada pelas cláusulas WHERE, GROUP BY ou HAVING

SINTAXE: “T1 **tipo join** T2 **conidção join**”

TIPOS DE JOIN - CROSS JOIN

- Equivalente ao INNER JOIN
- Para cada linha de T1, terá uma linha de T2 que satisfaça a condição
- retorna somente os registros que tenham "junção" entre as tabelas, ou seja, que satisfaça a condição definida.

… **FROM** T1 **CROSS JOIN** T2 ou

… **FROM** T1 **INNER JOIN** T2 **ON** CONDIÇÃO ou

… **FROM** T1, T2 **WHERE** condição

OBS: se não for definida a condição, para cada linha de T1 serão retornadas todas as linhas de T2, com um total de linhas T1 * T2.

CROSS JOIN EXEMPLOS

```
>Usando “,”:
SELECT *
FROM departamentos,grupos
WHERE departamentos.id_departamento = grupos.id_departamento

>Usando CROSS JOIN:
SELECT *
FROM departamentos
CROSS JOIN grupos
WHERE departamentos.id_departamento = grupos.id_departamento

>Usando INNER JOIN:
SELECT *
FROM departamentos
INNER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

JOINS QUALIFICADOS

```
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 ON boolean_expression
T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING ( join column list )
T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2
```

- INNER ou OUTER JOIN. Opção usar essas palavras.
- INNER É O PADRÃO.
- LEFT, RIGTH, FULL são OUTER.
- A condição é especificada na cláusula ON ou USING ou esta implicita no NATURAL JOIN

JOINS QUALIFICADOS - INNER JOIN

- Para cada linha em T1, será retornada a linha em T2 correspondente conforme a condição definida em ON.
- Exemplo:

```
SELECT *
FROM departamentos
INNER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

```
SELECT *
FROM departamentos
JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

JOINS QUALIFICADOS - LEFT OUTER JOIN

- Primeiro, um INNER JOIN é performado internamente. Então, para cada linha de T1 que não tenha linha em T2 que satisfaça a condição, é retornado NULL para T2 :

```
SELECT *
FROM departamentos
LEFT OUTER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

Não retornará duplicada a coluna da condição:

```
SELECT *
FROM departamentos
LEFT JOIN grupos USING (id_departamento)
```

JOINS QUALIFICADOS - RIGTH OUTER JOIN

- Inverso do LEFT. Primeiro, um INNER JOIN é performado internamente. Então, para cada linha de T2 que não tenha linha em T1 que satisfaça a condição, é retornado NULL para T1 :

```
SELECT *
FROM departamentos
RIGHT OUTER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

Não retornará duplicada a coluna da condição:

```
SELECT *
FROM departamentos
RIGHT JOIN grupos USING (id_departamento)
```

JOINS QUALIFICADOS - FULL OUTER JOIN

- Junção do LEFT com RIGHT. Primeiro, um INNER JOIN é performado internamente. Então, para cada linha de T2 que não tenha linha em T1 que satisfaça a condição, é retornado NULL para T1. Depois para cad linha em T1 que não tenha linha que satisfaça a condição em T2, é retornado NULL para T2.

```
SELECT *
FROM departamentos
FULL OUTER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

Não retornará duplicada a coluna da condição:

```
SELECT *
FROM departamentos
FULL JOIN grupos USING (id_departamento)
```

CLÁUSULA ON

- É onde se define o tipo de condição do JOIN.
- É uma expressão lógica, com resultado BOOLEANO
- A mesma condição definida em ON pode ser usada na cláusula WHERE

```
SELECT *
FROM departamentos
INNER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento
```

```
SELECT *
FROM departamentos, grupos
WHERE departamentos.id_departamento = grupos.id_departamento
```

CLÁUSULA ON

- Também pode contem outras condições não relacionadas diretamente com o JOIN.
- O resultado será diferente de usar tal condição na cláusula WHERE.
- Isso acontece por o ON é aplicado antes do JOIN e WHERE, após

```
SELECT *
FROM departamentos
INNER JOIN grupos ON departamentos.id_departamento =
grupos.id_departamento and departamento.nome = 'Eletrodomésticos'
```

CLÁUSULA ON

- Outro exemplo:

```
=> SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num AND t2.value = 'xxx';
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
   2 | b    |     |
   3 | c    |     |
(3 rows)
```

```
=> SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num WHERE t2.value = 'xxx';
 num | name | num | value
-----+------+-----+-------
   1 | a    |   1 | xxx
(1 row)
```

CLÁUSULA USING

- Também é onde se define a condição de JOIN
- É uma abreviação da expressão lógica, que pode ser usanda quando os campos em ambas as tabelas tem o mesmo nome
- Tem a vantagem de suprimir colunas redundantes

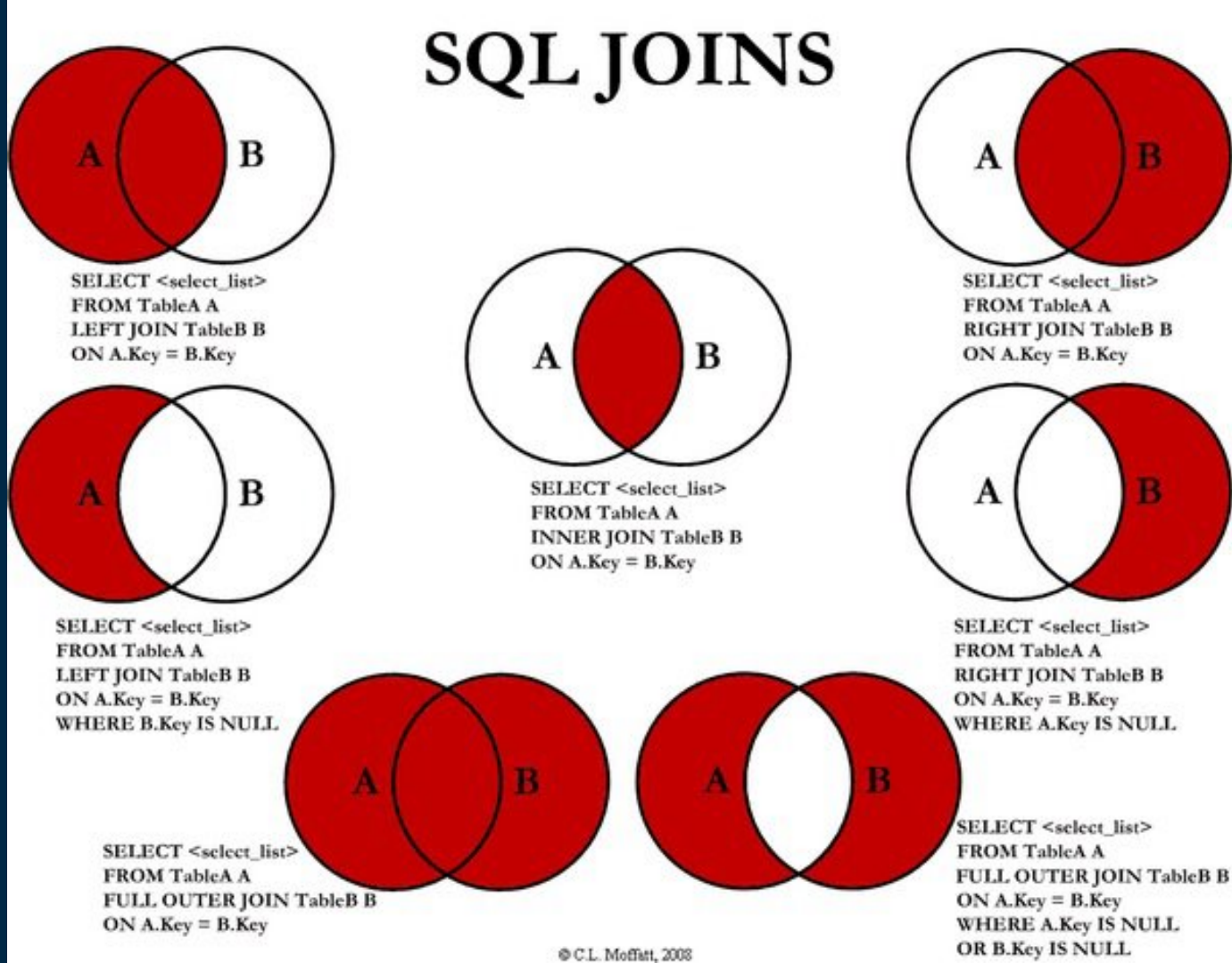
```
SELECT *
FROM departamentos
INNER JOIN grupos USING (id_departamento)
```

CLÁUSULA NATURAL

- Também é onde se define a condição de JOIN
- É uma abreviação do USING
- OBS: JOIN será feito usando todas as colunas com mesmo nome em ambas as tabelas.
- Precisar ter mais cuidado com mudanças na estrutura do banco

```
SELECT *
FROM departamentos
NATURAL INNER JOIN grupos
```

TIPOS DE JOINS E EXEMPLOS:



ALIAS

- Apelido que se pode dar para uma tabela ou sub consulta na cláusulo FROM
- Exemplo:

```
SELECT *
FROM departamentos d
INNER JOIN grupos g ON d.id_departamento = g.id_departamento
```

SUB CONSULTAS

- Representa uma nova fonte de dados na cláula from.
- Sempre deve ter um ALIAS
- Exemplo:

```
SELECT *
FROM (SELECT * FROM departamento) as d
```

WHERE

- Condições a serem aplicadas ao conjunto de dados obtidos pela cláusula FROM para filtrar os resultados
- Sempre deve ter sem uma expressão lógica, ou seja, resultar em um booleano
- Exemplos:

```
SELECT ... FROM fdt WHERE c1 > 5

SELECT ... FROM fdt WHERE c1 IN (1, 2, 3)

SELECT ... FROM fdt WHERE c1 IN (SELECT c1 FROM t2)

SELECT ... FROM fdt WHERE c1 IN (SELECT c3 FROM t2 WHERE c2 = fdt.c1 + 10)

SELECT ... FROM fdt WHERE c1 BETWEEN (SELECT c3 FROM t2 WHERE c2 = fdt.c1 + 10) AND 100

SELECT ... FROM fdt WHERE EXISTS (SELECT c1 FROM t2 WHERE c2 > fdt.c1)
```

GROUP BY

- Após o WHERE, é possível agregar os resultados usando cláusula GROUP BY (ou DISTINCT no SELECT)
- Pode excluir os valores duplicados ou ser usado em conjunto com funções de agregação, como SUM(), MAX(), MIN(), AVG(), COUNT()
- Exemplos:

```
SELECT id_departamento FROM grupos GROUP BY
id_departamento
```

```
SELECT id_departamento, Count(*) FROM grupos GROUP BY
id_departamento
```

HAVING

- Novo filtro que pode ser aplicado após a agregação pelo GROUP BY
- Condição deve considerar a agregação realizada

```
SELECT id_departamento, Count(*)
FROM grupos
GROUP BY id_departamento
HAVING count(*) > 1
```

DISTINCT

- Elimina valores duplicados do atributo
- Função similar a GROUP BY, porém não precisa utilizá-lo
- Executado após o Select para eliminar as linhas duplicadas
- Não faz parte do padrão SQL, é uma facilidade em relação ao GROUP BY

```
SELECT DISTINCT id_departamento
FROM grupos
```

ORDER BY - ORDEM/CALSSIFICAÇÃO

- Após a queri ser construída, é possível ordenar o resultado a ser retornado/exibido
- A ordem pode ser por um campo ou expressão
- Pede-se utilizar o nome do campo ou o número da coluna na consulta
- Ordem pode ser crescente ou decrescente
- pode ser aplicada a resultados de UNIONS, INTERSECT OU EXCEPT

```
SELECT *
FROM grupos
ORDER BY nome
```

```
SELECT id_grupo, nome, id_departamento
FROM grupos
ORDER BY 3
```

LIMIT

- limita a quantidade de resultados
- Importante sempre usar junto um ORDER BY para ordenar os dados

```
SELECT id_grupos
FROM grupos
ORDER BY id_grupos
LIMIT 3
```

REFERÊNCIAS

Elmasi E.; navathe, S. B. Sistemas de Banco de dados. São paulo: Pearson. 2018. Disponível em: https://plataforma.bvirtual.com.br/Acervo/Publicacao/168492 (Biblioteca Pearson Unipar - acesso pelo aluno on-line, menu a direita).

POSTGRESQL. Documentação oficial. QUERIES https://www.postgresql.org/docs/13/queries-table-expressions.html

Postgres Tutorial. https://www.postgresqltutorial.com/postgresql-select/