BIG DATA

LABDATA

FUNDAÇÃO
INSTITUTO DE
ADMINISTRAÇÃO

LABDATA

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

APLICAÇÕES DE BIG DATA COM HADOOP

Disciplina: Aplicações de Big Data com Hadoop

Tema da Aula: Banco de Dados e SQL

Prof. Bruno Paulinelli

Junho de 2016

Currículo

Formação

- Engenharia de Computação – USJT
- Mestrando estatística – USP

Experiência

- Professor dos cursos de MBA e extensão Big Data na FIA
- Pesquisador do laboratório de análise de dados LABDATA-FIA
- Professor no curso Inteligência na Gestão de Dados na POLI PECE
- Arquiteto de Soluções Big Data no Itaú Unibanco
- Implementação do Hadoop no Itaú Unibanco e pesquisas de novas tecnologias para Big Data (2013-2014)
- Analista de Business Intelligence no Itaú Unibanco (2007-2013)
- Analista de Business Intelligence na Totvs/Microsiga (2004-2007)

LABDATA

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

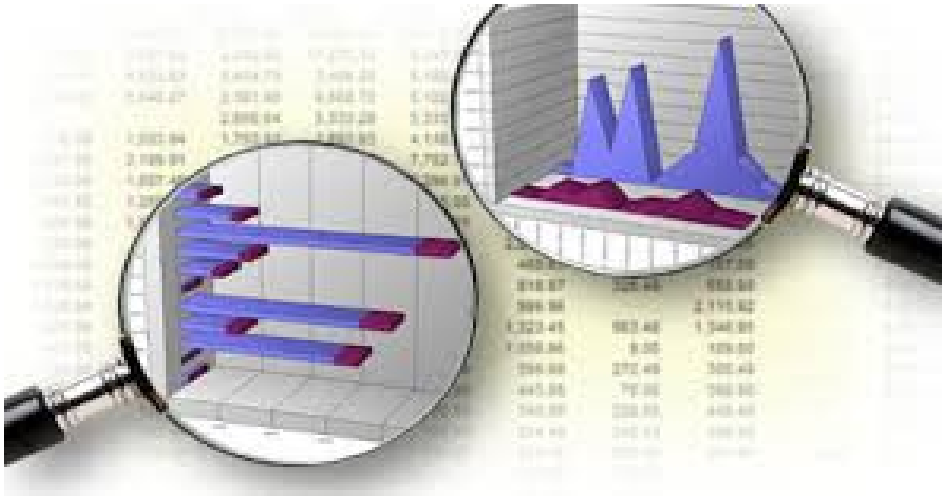
Agenda:

- Banco de Dados – Introdução
  - SGBD
  - O que é SQL?
  - Linguagem de Definição de Dados
  - Linguagem de Manipulação de Dados
  - Linguagem de Consulta de Dados
    - Cláusulas
    - Operadores Lógicos
    - Operadores relacionais
    - Ordenação
    - Junção
    - Agregação

Banco de Dados – Introdução

Bancos de dados, ou bases de dados, são coleções de dados brutos e informações, que se relacionam de forma que crie um sentido gerando conhecimento.

Banco de Dados – Introdução



Dados brutos: Sequência de símbolos quantificáveis ou qualificáveis.

Ex.: Fluxo de caixa e cadastro de clientes

Informações: Dados organizados, e que relacionados geram sentido.

Ex.: Relatório de Vendas

Conhecimentos: Conjunto de informações relacionadas ou agrupadas, que expliquem determinados eventos, auxiliando na tomada de decisões, criando, entendendo, manipulando e inteferindo em novos eventos.

Ex.: Modelos preditivos de risco, modelo de CRM para ofertar produtos.

SGBD

Um Sistema de Gerenciamento de Banco de Dados (SGBD) – do inglês Data Base Management System (DBMS) – é o conjunto de programas de computador responsáveis pelo gerenciamento de base de dados.

Principais atribuições de um SGBD:

- Distrubuir e gravar os dados em disco
- Decidir melhor plano de execução
- Controle de acesso
- Controle de redundâncias
- Backup para tolerência a falhas
- Estatísticas dos dados mais acessados
- Relacionar dados aos metadados
- Controle de integridade
- Compartilhamento dos dados
- Interface para programas e usuários

SGBD

Mais utilizados no mercado:

Banco de dados relacionais

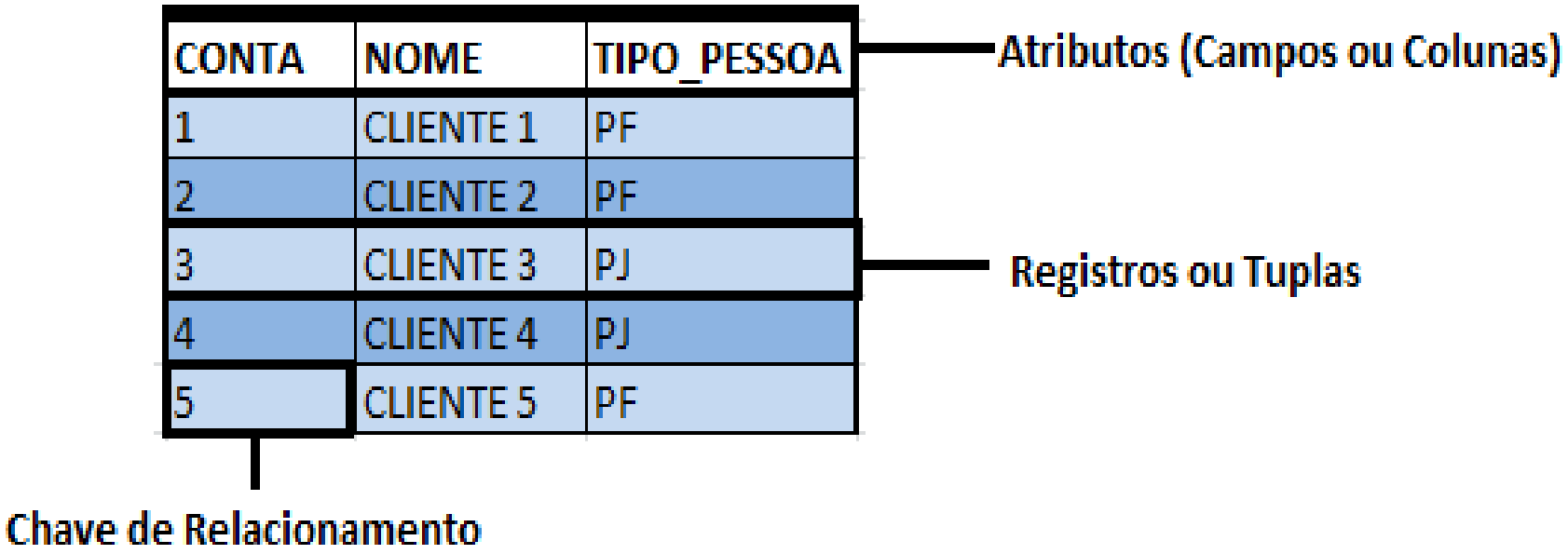
É um modelo de dados, utilizados pelo SGBD, que se baseia no princípio em que todos os dados estão armazenados em formatos de tabelas.

O modelo baseia-se em dois conceitos: entidade e relação.

Entidade ou **tabela**, é uma estrutura de armazenamento de dados organizada por linhas e colunas.

Relação determina o modo como cada registro de cada tabela se associa a registros de outras tabelas.

Tabela - Estrutura



O que é SQL?

Structured Query Language, ou Linguagem de Consulta Estruturada, é a linguagem de pesquisa declarativa padrão para banco de dados relacional.

A linguagem é um grande padrão de banco de dados. Isto decorre da sua simplicidade e facilidade de uso.

O SQL foi desenvolvido originalmente no início dos anos 70 nos laboratórios da IBM, e devido o surgimento de vários fabricantes de SGBD, foi necessário criar um padrão, esta tarefa foi realizada pela American National Standards Institute (ANSI) em 1986 e ISO em 1987.

DDL – Linguagem de Definição de Dados

É um conjunto de comandos da linguagem SQL usada para a definição das estruturas de dados, fornecendo as instruções que permitem a criação, modificação e remoção das tabelas, assim como criação de índices.

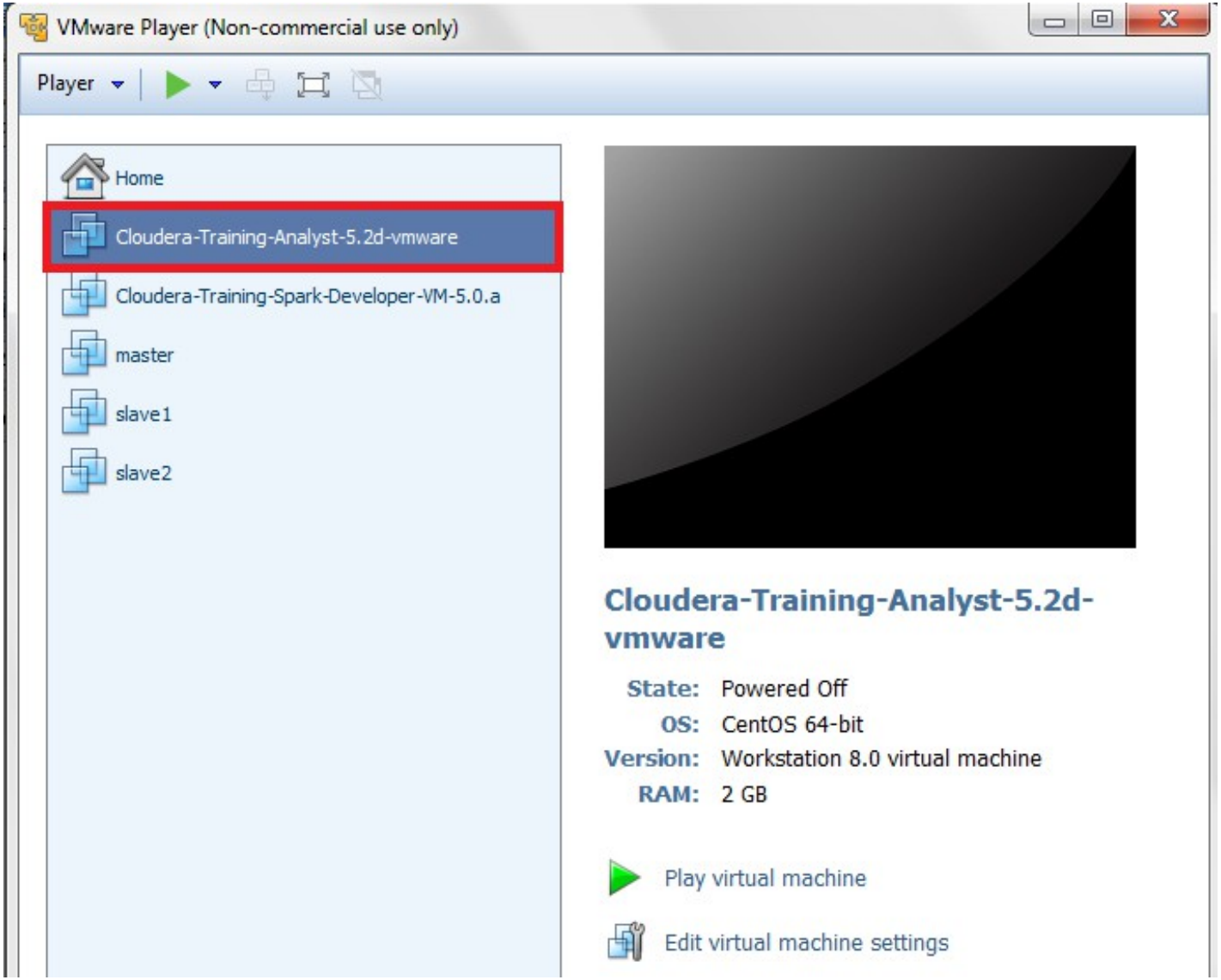
Metadados:

Uma vez submetido a instrução, são armezanados as informações em um dicionário de dados, que contém todas as informações da estrutura de armazenamento dos dados, ou seja, metadados ou metainformação é os dados sobre outros dados.

TABELA - CADASTRO		
CONTA	NOME	TIPO_PESSOA
1	CLIENTE 1	PF
2	CLIENTE 2	PF
3	CLIENTE 3	PJ
4	CLIENTE 4	PJ
5	CLIENTE 5	PF

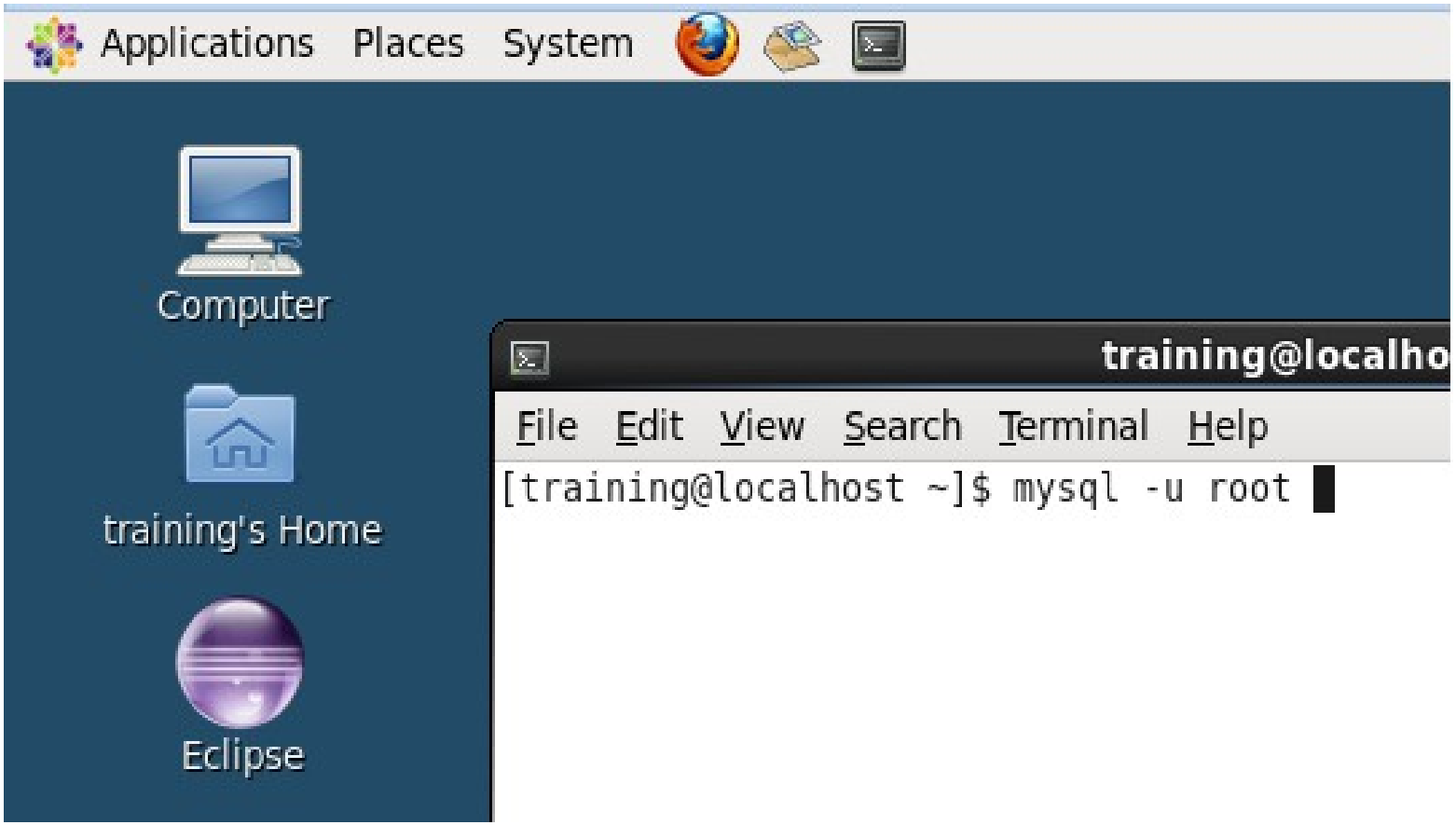
METADADOS			
TABELA	CAMPO	TIPO	CHAVE
CADASTRO	CONTA	INT	YES
CADASTRO	NOME	CHAR(50)	NO
CADASTRO	TIPO_PESSOA	CHAR(2)	NO

Iniciar a VM


VMware Player (Non-commercial use only)
Player
Home
Cloudera-Training-Analyst-5.2d-vmware
Cloudera-Training-Spark-Developer-VM-5.0.a
master
slave1
slave2
Cloudera-Training-Analyst-5.2d-vmware
State: Powered Off
OS: CentOS 64-bit
Version: Workstation 8.0 virtual machine
RAM: 2 GB
Play virtual machine
Edit virtual machine settings

MySQL

Abrir o terminal e conectar no mysql:



DDL – Linguagem de Definição de Dados

Comando **CREATE**

Permite criar um objeto como base de dados, tabelas, views e índices.

Criaremos uma base de dados e uma tabela.

```
CREATE DATABASE cliente;

show databases;

use cliente;

CREATE TABLE    cadastro (
                conta int NOT NULL,
                nome char(50),
                tipo_pessoa char(2),
                PRIMARY KEY(conta));

show tables;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cliente            |
| dualcore           |
| hue                |
| metastore          |
| movielens          |
| mysql              |
| test               |
+--------------------+
```

```
+-------------------+
| Tables_in_cliente |
+-------------------+
| cadastro          |
+-------------------+
```

DDL – Linguagem de Definição de Dados

Comando **ALTER**

Altera a estrutura do objeto.

Vamos criar uma segunda tabela para ser alterada.

```
CREATE TABLE    tabela(
                campo1 int,
                campo2 char(10));

DESCRIBE tabela;

ALTER TABLE tabela ADD campo3 int;

DESC  tabela;
```

DDL – Linguagem de Definição de Dados

Comando **ALTER**

Altera a estrutura do objeto.

Vamos criar uma segunda tabela para ser alterada.

```
+--------+----------+------+-----+---------+-------+
| Field  | Type     | Null | Key | Default | Extra |
+--------+----------+------+-----+---------+-------+
| campo1 | int(11)  | YES  |     | NULL    |       |
| campo2 | char(10) | YES  |     | NULL    |       |
+--------+----------+------+-----+---------+-------+
```

```
+--------+----------+------+-----+---------+-------+
| Field  | Type     | Null | Key | Default | Extra |
+--------+----------+------+-----+---------+-------+
| campo1 | int(11)  | YES  |     | NULL    |       |
| campo2 | char(10) | YES  |     | NULL    |       |
| campo3 | int(11)  | YES  |     | NULL    |       |
+--------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

DDL – Linguagem de Definição de Dados

Mais propriedades do comando **ALTER**

Cláusulas:

ADD – Adiciona um campo ou chave

ADD PRIMARY KEY – Adiciona uma chave primária

DROP – Remove um campo

MODIFY – Modifica as propriedades de um campo

DDL – Linguagem de Definição de Dados

Comando **DROP**

Exclui o objeto.

```
show tables;
```

```
+--------------------+
| Tables_in_cliente  |
+--------------------+
| cadastro           |
| tabela             |
+--------------------+
```

```
DROP TABLE     tabela;
```

```
show tables;
```

```
+--------------------+
| Tables_in_cliente  |
+--------------------+
| cadastro           |
+--------------------+
```

DML – Linguagem de Manipulação de Dados

Comando **INSERT**

Insere registros na tabela.

Estrutura: **INSERT INTO** tabela **VALUES** valor

```
INSERT INTO cadastro VALUES (1,'cliente_1','PF');
INSERT INTO cadastro VALUES (2,'cliente_2','PF');
INSERT INTO cadastro VALUES (3,'cliente_3','PJ');
INSERT INTO cadastro VALUES (4,'cliente_4','PJ');
INSERT INTO cadastro VALUES (5,'cliente_5','PF');
INSERT INTO cadastro VALUES (6,'cliente_6','PF');
INSERT INTO cadastro VALUES (7,'cliente_7','PJ');
```

A inserção pode ser feita tambem a partir dos dados de uma outra tabela.

Ex.: **INSERT INTO** tabela1 **SELECT * FROM** tabela2;

DML – Linguagem de Manipulação de Dados

Comando **UPDATE**

Altera os dados.

Estrutura: **UPDATE** tabela **SET** campo=novo_valor **WHERE** condição

```
SELECT * FROM cadastro;

UPDATE cadastro SET tipo_pessoa='PF'
WHERE conta=7;

SELECT * FROM cadastro;
```

```
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     1 | cliente_1 | PF          |
|     2 | cliente_2 | PF          |
|     3 | cliente_3 | PJ          |
|     4 | cliente_4 | PJ          |
|     5 | cliente_5 | PF          |
|     6 | cliente_6 | PF          |
|     7 | cliente_7 | PJ          |
+-------+-----------+-------------+
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     1 | cliente_1 | PF          |
|     2 | cliente_2 | PF          |
|     3 | cliente_3 | PJ          |
|     4 | cliente_4 | PJ          |
|     5 | cliente_5 | PF          |
|     6 | cliente_6 | PF          |
|     7 | cliente_7 | PF          |
+-------+-----------+-------------+
```

DML – Linguagem de Manipulação de Dados

Comando **DELETE**

Exclui os dados.

Estrutura: **DELETE FROM** tabela **WHERE** condição

```
DELETE FROM cadastro WHERE conta=7;
```

```
SELECT * FROM cadastro;
```

```
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     1 | cliente_1 | PF          |
|     2 | cliente_2 | PF          |
|     3 | cliente_3 | PJ          |
|     4 | cliente_4 | PJ          |
|     5 | cliente_5 | PF          |
|     6 | cliente_6 | PF          |
+-------+-----------+-------------+
```

DQL – Linguagem de Consulta de Dados

Comando **SELECT**

Seleciona os dados, efetua uma consulta(query).

Cláusula	Descrição
FROM	Tabela que se vai selecionar os registros.
WHERE	Condições que devem reunir os registros, filtro.
GROUP BY	Agrupamento por chave (um ou mais campos).
HAVING	Condições, filtros por grupo.
ORDER BY	Ordenação por chave (um ou mais campos).
DISTINCT	Utilizada para selecionar dados sem repetição.
UNION	combina os resultados de duas ou mais consultas em uma única tabela.

DQL – Linguagem de Consulta de Dados

Comando **SELECT**

Estrutura: **SELECT** campo **FROM** tabela

```
SELECT conta, nome, tipo_pessoa FROM cadastro;
```

- Podemos utilizar o * quando quisermos selecionar todos os campos da tabela.

```
SELECT * FROM cadastro;
```

DQL – Linguagem de Consulta de Dados

Cláusula **WHERE**

Estrutura: **SELECT** campo **FROM** tabela **WHERE** condição

```
SELECT * FROM cadastro WHERE tipo_pessoa = 'PJ';
```

```
+-------+----------+-------------+
| conta | nome     | tipo_pessoa |
+-------+----------+-------------+
|     3 | cliente_3 | PJ          |
|     4 | cliente_4 | PJ          |
+-------+----------+-------------+
```

DQL - Linguagem de Consulta de Dados

Cláusula **DISTINCT**

Apresenta somente os dados distintos, ou seja, não permite que retorne a mesma informação mais de uma vez.

```
SELECT distinct(tipo_pessoa) FROM cadastro;
```

```
+-------------+
| tipo_pessoa |
+-------------+
| PF          |
| PJ          |
+-------------+
```

DQL - Linguagem de Consulta de Dados

Cláusula **ORDER BY**

Ordena de forma crescente utilizando o complemento **ASC** ou decrescente quando complementado com **DESC**.

```
SELECT * FROM cadastro ORDER BY conta ASC;

SELECT * FROM cadastro ORDER BY conta DESC;
```

Observações:

- Por padrão, o resultado de um ORDER BY é crescente, ou seja, colocar ou não o complemento ASC, o resultado será o mesmo.

- No lugar no nome do campo após o ORDER BY, também podemos colocar o número da posição dele na tabela, por exemplo ORDER BY 1, ordenará pelo primeiro campo da tabela.

DQL - Linguagem de Consulta de Dados

Cláusula **UNION**

Combina os **valores diferentes** do resultado de duas ou mais querys.

```
CREATE TABLE    cadastro2 (
                conta int,
                nome char(50),
                tipo_pessoa char(2));
```

```
INSERT INTO cadastro2 VALUES (6,'cliente_6','PF');
INSERT INTO cadastro2 VALUES (7,'cliente_7','PJ');
```

```
SELECT * FROM cadastro
UNION
SELECT * FROM cadastro2;
```

conta	nome	tipo_pessoa
1	cliente_1	PF
2	cliente_2	PF
3	cliente_3	PJ
4	cliente_4	PJ
5	cliente_5	PF
6	cliente_6	PF
7	cliente_7	PJ

DQL - Linguagem de Consulta de Dados

Cláusula **UNION ALL**

Combina **todos os valores** do resultado de duas ou mais querys.

```
SELECT * FROM cadastro

UNION ALL

SELECT * FROM cadastro2;
```

```
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     1 | cliente_1 | PF          |
|     2 | cliente_2 | PF          |
|     3 | cliente_3 | PJ          |
|     4 | cliente_4 | PJ          |
|     5 | cliente_5 | PF          |
|     6 | cliente_6 | PF          |
|     6 | cliente_6 | PF          |
|     7 | cliente_7 | PJ          |
+-------+-----------+-------------+
```

DQL - Linguagem de Consulta de Dados

Cláusula **UNION** com estruturas de tabelas diferentes

É necessário que as tabelas ou resultado da query tenham o mesmo número de campos, mesmos tipos e ordem de seleção.

```
ALTER TABLE    cadastro2 MODIFY conta char(5);
ALTER TABLE    cadastro2 ADD COLUMN salario decimal(10,2);
UPDATE cadastro2 SET salario=1000 WHERE conta=6;
UPDATE cadastro2 SET salario=2000 WHERE conta=7;
```

```
SELECT cast(conta as char(5)) conta, nome, tipo_pessoa, 0 as salario FROM cadastro

UNION

SELECT conta, nome, tipo_pessoa, salario FROM cadastro2;
```

DQL - Linguagem de Consulta de Dados

Cláusula **UNION** com estruturas de tabelas diferentes

É necessário que as tabelas ou resultado da query tenham o mesmo número de campos, mesmos tipos e ordem de seleção.

```
SELECT cast(conta as char(5)) conta, nome, tipo_pessoa, 0 as salario FROM cadastro

UNION

SELECT conta, nome, tipo_pessoa, salario FROM cadastro2;
```

conta	nome	tipo_pessoa	salario
1	cliente_1	PF	0.00
2	cliente_2	PF	0.00
3	cliente_3	PJ	0.00
4	cliente_4	PJ	0.00
5	cliente_5	PF	0.00
6	cliente_6	PF	0.00
6	cliente_6	PF	1000.00
7	cliente_7	PJ	2000.00

DQL – Linguagem de Consulta de Dados

Operadores Lógicos

Operador	Descrição
AND	E lógico
OR	OU lógico
NOT	Negação lógica

DQL – Linguagem de Consulta de Dados

Operadores Relacionais

Operador	Descrição
<	Menor
>	Maior
<=	Menor ou igual
>=	Maior ou igual
=	Igual
<>	Diferente
!=	Diferente
between	Intervalo
like	Busca parcial de string
in	Contém em uma lista

DQL – Linguagem de Consulta de Dados

Operadores Relacionais

Exemplos:

--Intervalo (**BETWEEN**)

```
SELECT * FROM cadastro
WHERE conta BETWEEN 2 AND 4;
```

```
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     2 | cliente_2 | PF          |
|     3 | cliente_3 | PJ          |
|     4 | cliente_4 | PJ          |
+-------+-----------+-------------+
```

--Busca parcial de string (**LIKE**)

```
SELECT * FROM cadastro
WHERE tipo_pessoa LIKE '%F';
```

```
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     1 | cliente_1 | PF          |
|     2 | cliente_2 | PF          |
|     5 | cliente_5 | PF          |
|     6 | cliente_6 | PF          |
+-------+-----------+-------------+
```

--Lista (**IN**)

```
SELECT * FROM cadastro
WHERE conta IN (1,3,5);
```

```
+-------+-----------+-------------+
| conta | nome      | tipo_pessoa |
+-------+-----------+-------------+
|     1 | cliente_1 | PF          |
|     3 | cliente_3 | PJ          |
|     5 | cliente_5 | PF          |
+-------+-----------+-------------+
```

DQL - Linguagem de Consulta de Dados

Cláusula **GROUP BY**

É utilizada para agrupamentos de chaves (campo ou conjunto de campos), que contém os mesmos valores.

```
SELECT tipo_pessoa FROM cadastro GROUP BY tipo_pessoa;
```

```
+-------------+
| tipo_pessoa |
+-------------+
| PF          |
| PJ          |
+-------------+
```

Neste exemplo, o resultado será o mesmo que um **DISTINCT**(tipo_pessoa), mas a cláusula **GROUP BY** permite a utilização de funções de agregação, que veremos no próximo slide.

DQL - Linguagem de Consulta de Dados

Cláusula **GROUP BY**, com funções de agregação.

As funções de agregação permitem tratar e retornar um resultado de vários registros de um campo.

Função	Descrição
COUNT	Contagem de linhas ou campo com valor
MAX	Valor máximo do campo
MIN	Valor mínimo do campo
SUM	Somatório do campo
AVG	Valor da média do campo

Estas funções podem ser utlizadas sem GROUP BY, ou seja, permite retornar um resultado com base na tabela inteira.

DQL - Linguagem de Consulta de Dados

Cláusula **GROUP BY**, com funções de agregação.

Função **COUNT**

Exemplo de contagem de registros por tipo de pessoa.

```
SELECT   tipo_pessoa,
         COUNT(*) as quantidade
FROM cadastro
GROUP BY tipo_pessoa;
```

```
+-------------+------------+
| tipo_pessoa | quantidade |
+-------------+------------+
| PF          |          4 |
| PJ          |          2 |
+-------------+------------+
```

Esta função quando chamada com COUNT(*), retorna a quantidade total de registros, e quando submetida com COUNT(tipo_pessoa), retornará somente a quantidade de registros do campo tipo_pessoa preenchido, ou seja, não nulos.

DQL - Linguagem de Consulta de Dados

Cláusula **GROUP BY**, com funções de agregação.

Funções **COUNT**, **SUM**, **MAX**, **MIN**, **AVG**

```
CREATE TABLE    fluxo_conta (
                conta int,
                tipo_lancamento char(1),
                lancamento decimal(10,2));

INSERT INTO fluxo_conta VALUES (1,'C',1000);
INSERT INTO fluxo_conta VALUES (1,'D',500);
INSERT INTO fluxo_conta VALUES (1,'D',300);
INSERT INTO fluxo_conta VALUES (2,'C',2000);
INSERT INTO fluxo_conta VALUES (2,'D',800);
INSERT INTO fluxo_conta VALUES (2,'D',1500);
INSERT INTO fluxo_conta VALUES (8,'C',0);

SELECT * FROM fluxo_conta;
```

conta	tipo_lancamento	lancamento
1	C	1000.00
1	D	500.00
1	D	300.00
2	C	2000.00
2	D	800.00
2	D	1500.00
8	C	0.00

DQL - Linguagem de Consulta de Dados

Cláusula **GROUP BY**, com funções de agregação.

Funções **COUNT**, **SUM**, **MAX**, **MIN**, **AVG**

```
SELECT conta,
        COUNT(lancamento) quantidade_debito,
        SUM(lancamento) total_debito,
        MAX(lancamento) maior_debito,
        MIN(lancamento) menor_debito,
        AVG(lancamento) media_debito
FROM fluxo_conta
WHERE tipo_lancamento='D'
GROUP BY conta;
```

conta	quantidade_debito	total_debito	maior_debito	menor_debito	media_debito
1	2	800.00	500.00	300.00	400.000000
2	2	2300.00	1500.00	800.00	1150.000000

DQL - Linguagem de Consulta de Dados

Cláusula **HAVING**

Filtro por resultado do agrupamento.

```
SELECT   tipo_pessoa,
         COUNT(*) as quantidade
FROM cadastro
GROUP BY tipo_pessoa
           HAVING COUNT(*)>2;
```

```
+-------------+------------+
| tipo_pessoa | quantidade |
+-------------+------------+
| PF          |          4 |
+-------------+------------+
```

DQL - Linguagem de Consulta de Dados

Expressão **CASE**

É utilizada para avaliar uma ou mais condições, retornando apenas um resultado.

```
SELECT conta,
       lancamento,
       tipo_lancamento,
       CASE WHEN tipo_lancamento='D' THEN lancamento*-1
            ELSE lancamento
       END as lancamento_sinal
FROM fluxo_conta ;
```

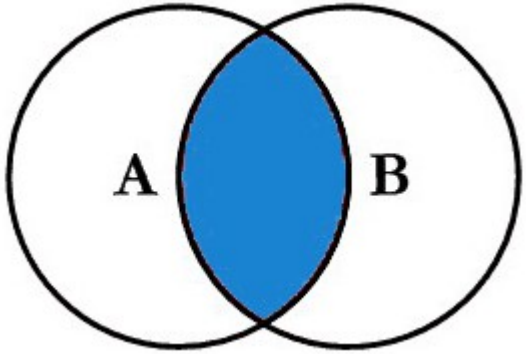
conta	lancamento	tipo_lancamento	lancamento_sinal
1	1000.00	C	1000.00
1	500.00	D	-500.00
1	300.00	D	-300.00
2	2000.00	C	2000.00
2	800.00	D	-800.00
2	1500.00	D	-1500.00
8	0.00	C	0.00

DQL - Linguagem de Consulta de Dados

Cláusula **INNER JOIN** – Junção Interna

Junção dos registros entre duas tabelas, desde que as chaves de cruzamento existam em ambas.

```
SELECT  *
FROM cadastro a
INNER JOIN fluxo_conta b
ON a.conta=b.conta;
```



conta	nome	tipo_pessoa	conta	tipo_lancamento	lancamento
1	cliente_1	PF	1	C	1000.00
1	cliente_1	PF	1	D	500.00
1	cliente_1	PF	1	D	300.00
2	cliente_2	PF	2	C	2000.00
2	cliente_2	PF	2	D	800.00
2	cliente_2	PF	2	D	1500.00

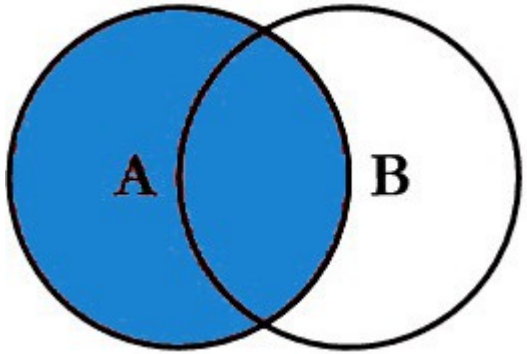
Na falta de chave de cruzamento no ON, ou existência de chave duplicada na tabela, isto implicará em um produto cartesiano.

DQL - Linguagem de Consulta de Dados

Cláusula **LEFT OUTER JOIN** – Junção externa

Junção dos registros entre duas tabelas, partindo da tabela da esquerda, ou seja, o resultado sempre retornará todos os registros da tabela da esquerda (a primeira) mesmo que não existam chaves correspondentes na tabela da direita(a segunda).

```
SELECT  *
FROM cadastro a
LEFT OUTER JOIN fluxo_conta b
ON a.conta=b.conta;
```



conta	nome	tipo_pessoa	conta	tipo_lancamento	lancamento
1	cliente_1	PF	1	C	1000.00
1	cliente_1	PF	1	D	500.00
1	cliente_1	PF	1	D	300.00
2	cliente_2	PF	2	C	2000.00
2	cliente_2	PF	2	D	800.00
2	cliente_2	PF	2	D	1500.00
3	cliente_3	PJ	NULL	NULL	NULL
4	cliente_4	PJ	NULL	NULL	NULL
5	cliente_5	PF	NULL	NULL	NULL
6	cliente_6	PF	NULL	NULL	NULL

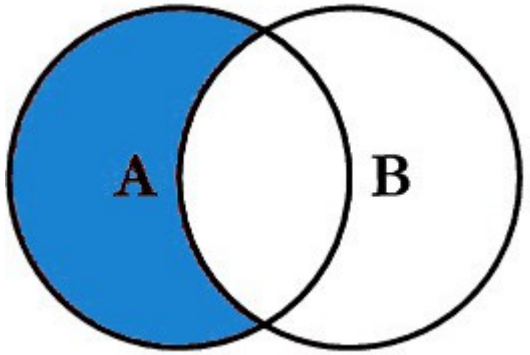
Dependendo do banco de dados, é permitido utilizar a sintaxe sem o OUTER.

DQL - Linguagem de Consulta de Dados

Cláusula **LEFT OUTER JOIN** – Junção externa

Junção dos registros entre duas tabelas, partindo da tabela da esquerda, ou seja, o resultado sempre retornará todos os registros da tabela da esquerda (a primeira) mesmo que não existam chaves correspondentes na tabela da direita(a segunda), **desconsiderando os registros correspondentes em ambas**.

```
SELECT  *
FROM cadastro a
LEFT OUTER JOIN fluxo_conta b
ON a.conta=b.conta
WHERE b.conta IS NULL;
```



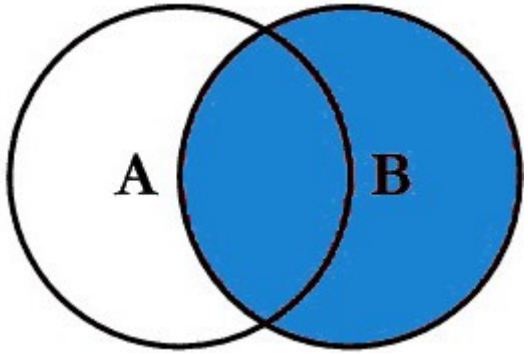
conta	nome	tipo_pessoa	conta	tipo_lancamento	lancamento
3	cliente_3	PJ	NULL	NULL	NULL
4	cliente_4	PJ	NULL	NULL	NULL
5	cliente_5	PF	NULL	NULL	NULL
6	cliente_6	PF	NULL	NULL	NULL

DQL - Linguagem de Consulta de Dados

Cláusula **RIGHT OUTER JOIN** – Junção externa

É o inverso do **LEFT OUTER JOIN**, ou seja, o resultado sempre retornará todos os registros da tabela da direita (a segunda) mesmo que não existam chaves correspondentes na tabela da esquerda(a primeira).

```
SELECT   *
FROM cadastro a
RIGHT OUTER JOIN fluxo_conta b
ON a.conta=b.conta;
```



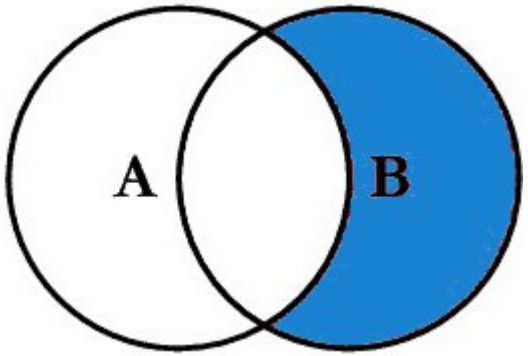
conta	nome	tipo_pessoa	conta	tipo_lancamento	lancamento
1	cliente_1	PF	1	C	1000.00
1	cliente_1	PF	1	D	500.00
1	cliente_1	PF	1	D	300.00
2	cliente_2	PF	2	C	2000.00
2	cliente_2	PF	2	D	800.00
2	cliente_2	PF	2	D	1500.00
NULL	NULL	NULL	8	C	0.00

DQL - Linguagem de Consulta de Dados

Cláusula **RIGHT OUTER JOIN** – Junção externa

É o inverso do **LEFT OUTER JOIN**, ou seja, o resultado sempre retornará todos os registros da tabela da direita (a segunda) mesmo que não existam chaves correspondentes na tabela da esquerda(a primeira), **desconsiderando os registros correspondentes em ambas**.

```
SELECT  *
FROM cadastro a
RIGHT OUTER JOIN fluxo_conta b
ON a.conta=b.conta
WHERE a.conta IS NULL;
```




```
+-------+------+-------------+-------+-----------------+------------+
| conta | nome | tipo_pessoa | conta | tipo_lancamento | lancamento |
+-------+------+-------------+-------+-----------------+------------+
|  NULL | NULL | NULL        |     8 | C               |       0.00 |
+-------+------+-------------+-------+-----------------+------------+
```

DQL - Linguagem de Consulta de Dados

Cláusula **FULL OUTER JOIN**

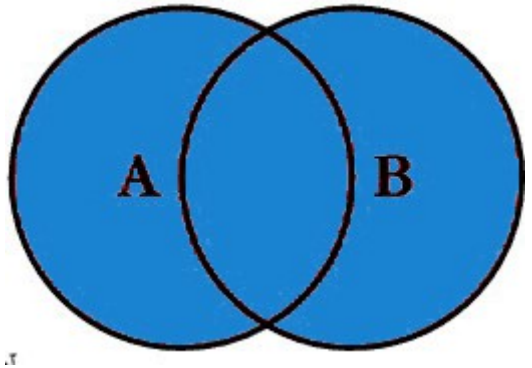
O resultado sempre retornará todos os registros de todas as tabelas mesmo que não existam chaves correspondentes entre elas.

OBS.: Como não existe o comando **FULL OUTER JOIN** no mysql, a maneira de simular este tipo de junção é:

```
SELECT   *
FROM cadastro a
LEFT OUTER JOIN fluxo_conta b
ON a.conta=b.conta

UNION ALL

SELECT   *
FROM cadastro a
RIGHT OUTER JOIN fluxo_conta b
ON a.conta=b.conta;
```



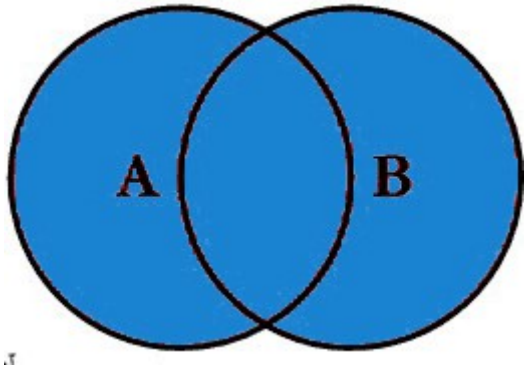
DQL - Linguagem de Consulta de Dados

Cláusula **FULL OUTER JOIN**

O resultado sempre retornará todos os registros de todas as tabelas mesmo que não existam chaves correspondentes entre elas.

OBS.: Como não existe o comando **FULL OUTER JOIN** no mysql, a maneira de simular este tipo de junção é:

conta	nome	tipo_pessoa	conta	tipo_lancamento	lancamento
1	cliente_1	PF	1	C	1000.00
1	cliente_1	PF	1	D	500.00
1	cliente_1	PF	1	D	300.00
2	cliente_2	PF	2	C	2000.00
2	cliente_2	PF	2	D	800.00
2	cliente_2	PF	2	D	1500.00
3	cliente_3	PJ	NULL	NULL	NULL
4	cliente_4	PJ	NULL	NULL	NULL
5	cliente_5	PF	NULL	NULL	NULL
6	cliente_6	PF	NULL	NULL	NULL
1	cliente_1	PF	1	C	1000.00
1	cliente_1	PF	1	D	500.00
1	cliente_1	PF	1	D	300.00
2	cliente_2	PF	2	C	2000.00
2	cliente_2	PF	2	D	800.00
2	cliente_2	PF	2	D	1500.00
NULL	NULL	NULL	8	C	0.00



DQL - Linguagem de Consulta de Dados

Cláusula **FULL OUTER JOIN**

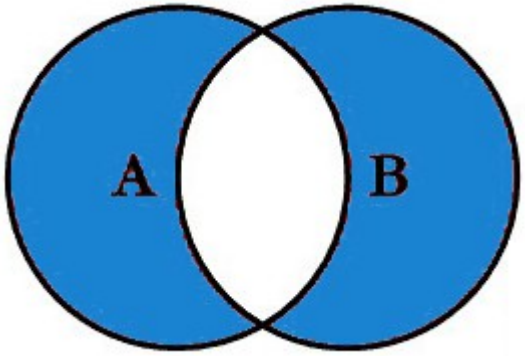
O resultado sempre retornará todos os registros de todas as tabelas mesmo que não existam chaves correspondentes entre elas, **desconsiderando os registros correspondentes em ambas**.

OBS.: Como não existe o comando **FULL OUTER JOIN** no mysql, a maneira de simular este tipo de junção é:

```
SELECT   *
FROM cadastro a
LEFT OUTER JOIN fluxo_conta b
ON a.conta=b.conta
WHERE b.conta IS NULL

UNION ALL

SELECT   *
FROM cadastro a
RIGHT OUTER JOIN fluxo_conta b
ON a.conta=b.conta
WHERE a.conta IS NULL;
```



conta	nome	tipo_pessoa	conta	tipo_lancamento	lancamento
3	cliente_3	PJ	NULL	NULL	NULL
4	cliente_4	PJ	NULL	NULL	NULL
5	cliente_5	PF	NULL	NULL	NULL
6	cliente_6	PF	NULL	NULL	NULL
NULL	NULL	NULL	8	C	0.00

DQL - Linguagem de Consulta de Dados

Expressão **COALESCE**

É um atalho sintático da função **CASE** na substituição de valores o quando o resultado é nulo.

```
SELECT   a.conta,
         CASE WHEN b.lancamento IS NULL THEN 'Sem Lancamento'
              ELSE b.lancamento
         END as lancamento
FROM cadastro a
LEFT OUTER JOIN cliente.fluxo_conta b
ON a.conta=b.conta ;
```

```
SELECT   a.conta,
         COALESCE(b.lancamento,'Sem Lancamento') lancamento
FROM cliente.cadastro a
LEFT OUTER JOIN cliente.fluxo_conta b
ON a.conta=b.conta ;
```

conta	lancamento
1	1000.00
1	500.00
1	300.00
2	2000.00
2	800.00
2	1500.00
3	Sem Lancamento
4	Sem Lancamento
5	Sem Lancamento
6	Sem Lancamento

DQL - Linguagem de Consulta de Dados

Expressão **COALESCE**

Avaliando os argumentos na ordem e retorna o valor da primeira expressão que não é avaliada como NULL.

```
CREATE TABLE    teste(
                campo1 int,
                campo2 int);

INSERT INTO teste VALUES (1,2);
INSERT INTO teste VALUES (NULL,2);
INSERT INTO teste VALUES (NULL,NULL);
```

```
+--------+--------+---------------+
| campo1 | campo2 | campo         |
+--------+--------+---------------+
|      1 |      2 | 1             |
|   NULL |      2 | 2             |
|   NULL |   NULL | Sem Resultado |
+--------+--------+---------------+
```

```
SELECT  campo1,
        campo2,
        COALESCE(campo1,campo2,'Sem Resultado') campo
FROM teste;
```

DQL - Linguagem de Consulta de Dados

Subquery

Ou subconsulta, é uma consulta aninhada nas instruções **SELECT, INSERT, UPDATE e DELETE.**

```
INSERT INTO cadastro2 SELECT   *, 0 as salario FROM cadastro;

SELECT * FROM cadastro;
SELECT * FROM cadastro2;

DELETE FROM cliente.cadastro2
WHERE conta IN (SELECT conta
                 FROM cadastro
                );

SELECT * FROM cadastro2;
```

DQL - Linguagem de Consulta de Dados

Subquery

```
SELECT a.conta, SUM(a.lancamento_sinal) saldo
FROM (SELECT conta,
             CASE WHEN tipo_lancamento='D' THEN lancamento*-1
                  ELSE lancamento
             END as lancamento_sinal
        FROM fluxo_conta
      ) a
GROUP BY conta;
```

conta	saldo
1	200.00
2	-300.00
8	0.00

Exercícios

Exercício 1:

- Crie uma tabela “tb_exercicio” com um campo “cpo” tipo int
- Insira os registros com os valores 1, 2 , 2, 3, 4, 4, 5
- Desenvolva uma query que retorne somente os registros duplicados

cpo	quantidade
2	2
4	2

- Desenvolva uma query que retorne somente os registros sem duplicidade

cpo	quantidade
1	1
3	1
5	1

Exercícios

Exercício 2:

Desenvolva uma query na tabela fluxo_conta que retorne:

- O numero da conta

- O total de lançamentos agrupados por conta

- Inclua os sinais no valor do lançamento (positivo ou negativo), sumarizando por conta calculando o saldo

- Mensagem da situação do saldo

conta	total_lancamento	saldo	mensagem
1	3	200.00	Saldo Positivo
2	3	-300.00	Saldo Negativo
8	1	0.00	Sem Saldo

Prof. Bruno Paulinelli

https://br.linkedin.com/pub/bruno-paulinelli/bb/218/154

bruno.paulinelli@gmail.com

b.paulinelli