BANCO DE DADOS

AULA 4

EXPLICADORES.NET

@explicadoresnet

GRUPOS

Uma característica muito importante do SQL é o poder de agrupar linhas com base em valores de determinadas colunas. Dessa forma, não estaremos trabalhando na pesquisa em todas as linhas da tabela, como fizemos anteriormente, mas sim em grupos menores. Para isso, utilizamos as funções de grupo já mostradas, com a cláusula GRUPO BY no comando SELECT. A cláusula GRUPO BY deve vir antes da cláusula ORDER BY e depois do WHERE (se houver necessidade de utilizá-los).

```
SELECT CODVENDA, COUNT(*) FROM PEDIDO
GRUPO BY CODVENDA;
```

```
SELECT CODCLIENTE, SUM(TOTALPEDIDO) FROM PEDIDO
GROUP BY CODCLIENTE;
```

```
SELECT CODCLIENTE, COUNT(*) FROM PEDIDO
WHERE PRECOVENDA < 15
GROUP BY CODCLIENTE;
```

```
SELECT CODCLIENTE, COUNT(*) FROM PEDIDO
GROUP BY CODCLIENTE
HAVING PRECOVENDA < 15;
```

– Leia com atenção.

A tabela "Produtos" possui as seguintes colunas: codigo_produto, nome_produto, descricao_produto, preco, peso e categoria.

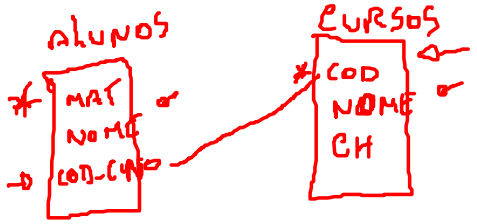
Assinale a alternativa que retorna quantos produtos existem em cada categoria de produto.

Obs.: A consulta na tabela "Produtos" deve retornar um resultado que mostre a existência de:
um produto na categoria eletrodomésticos; dois produtos na categoria informática e dois produtos na categoria de esportes.

Categoria	Função utilizada pelo candidato
Eletrodomésticos	1
Informática	2
Esportes	2

a) SELECT categoria, COUNT(*) FROM produtos GROUP BY categoria.
b) SELECT categoria, DISTINCT(*) FROM produtos GROUP BY categoria HAVING COUNT(*) >1.
c) SELECT categoria, SUM(*) FROM produtos GROUP BY categoria HAVING COUNT(*) >1.
d) SELECT categoria, COUNT(*) FROM produtos GROUP BY categoria HAVING COUNT(*) >1.

APELIDOS




```
SELECT a.nome, c.nome
FROM alunos AS a, cursos AS c
WHERE a.codCurso = c.cod;
```

```
SELECT a.nome, c.nome
FROM alunos a, cursos c
WHERE a.codCurso = c.cod;
```

```
SELECT ALUNOS.NOME, CURSOS.NOME
FROM ALUNOS, CURSOS
WHERE ALUNOS.COD_CURSO = CURSOS.COD;
```


01 URIAS 2 SAD 6

ALUNOS

MAT	NOME	COD_CURSO
01	URIAS	2
02	BIA	3
03	LUIZA	4
04	CARLA	

COUNT(*) = 4

INNER JOIN

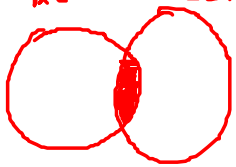
INTERSEÇÃO

```
SELECT *
FROM alunos AS a
INNER JOIN cursos AS c
ON a.codCurso = c.cod;
```

CE P CP A

COUNT(COD_CURSO) = 3

ALUNOS CURSOS



CURSOS

COD	NOME	CH
1	ENGSIN	5
2	SAD	6
3	SEG	8
4	BCT	9
5	SEL	10

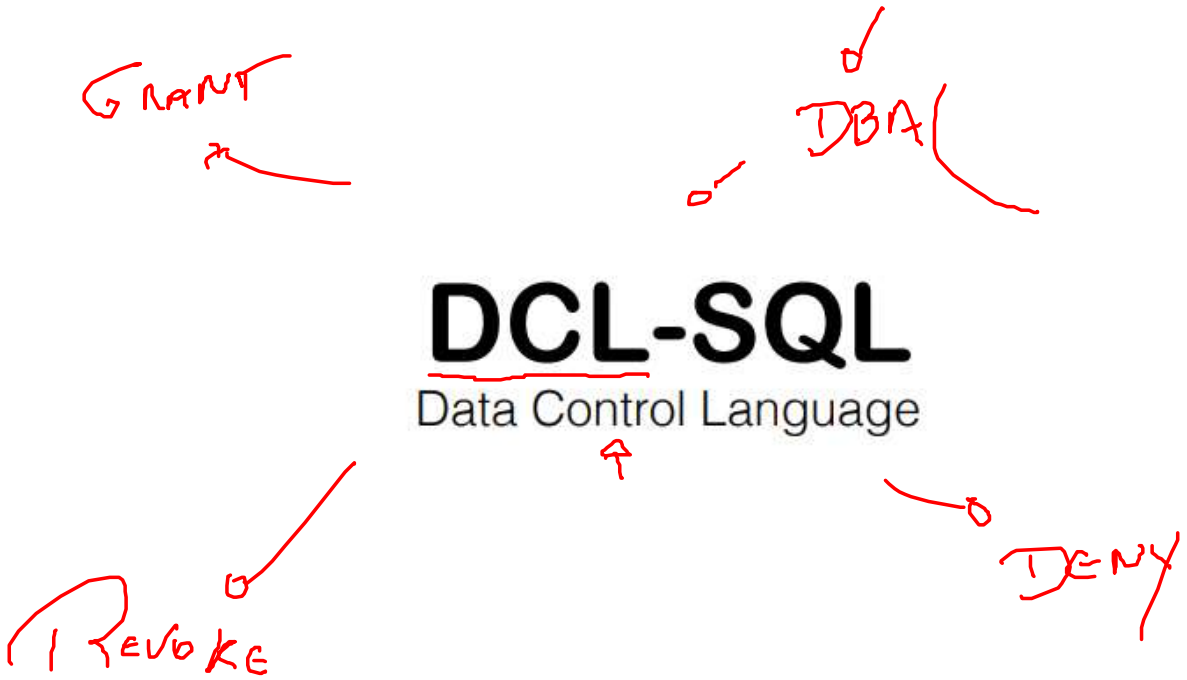
GRUPOS

```
SELECT c.nome, COUNT(a.mat)
FROM alunos a
INNER JOIN cursos c
ON a.codCurso = c.cod
GROUP BY c.nome;
```

NOME	MAT
SIN	10
SAD	20

```
SELECT a.nome
FROM alunos a
INNER JOIN cursos c
ON a.codCurso = c.cod
GROUP BY a.nome
HAVING COUNT(c.cod) > 2;
```

MOSTRA O NOME DOS ALUNOS QUE ESTÁ MATRICULADO EM MAIS DE 2 CURSOS!


GRANT
DBA
DCL-SQL
Data Control Language
REVOKE
DENY

GRANT

Privilégios para trabalhar com dados:	
Privilégio	*Descrição*
INSERT	Inserir dados em uma tabela
UPDATE	Atualizar dados em uma tabela
DELETE	Excluir dados de uma tabela
EXECUTE	Executar funções ou procedimentos armazenados
SELECT	Efetuar consultas em uma tabela

DDL

Privilégios para modificar a estrutura do banco de dados:

Privilégio	*Descrição*
CREATE	Criar tabela ou banco de dados
ALTER	Modificar uma tabela
DROP	Excluir uma tabela ou um banco de dados
CREATE VIEWS	Criar exibições
TRIGGER	Criar ou excluir um trigger em uma tabela (FUNÇÃO)
INDEX	Criar ou excluir um índice
CREATE ROUTINE	Criar uma função ou um procedimento armazenado
ALTER ROUTINE	Alterar ou excluir uma função ou procedimento armazenado

Privilégios Administrativos	
Privilégio	*Descrição*
CREATE USER	Criar contas de usuários
SHOW DATABASES	Ver os nomes dos bancos de dados no servidor
SHUTDOWN	Desligar o servidor
RELOAD	Recarregar as tabelas que armazenam os privilégios dos usuários dos bancos de dados. Assim elas são atualizadas se tiverem sido modificadas.
Outros privilégios	
ALL	Todos os privilégios disponíveis em um determinado nível, exceto GRANT OPTION
GRANT OPTION	Permite dar privilégios a outros usuários
USAGE	Não altera privilégios; usado para tarefas administrativas na conta do usuário.

Níveis dos privilégios

No MySQL os privilégios são atribuídos em quatro níveis diferentes:

- **Global** – O usuário tem acesso a todas as tabelas de todos os bancos de dados
- **Database** – Esse privilégio dá ao usuário acesso a todas as tabelas de um banco de dados específico
- **Table** – O usuário tem aceso a todas as colunas de uma tabela específica em um banco de dados
- **Column** – O usuário possui acesso apenas a colunas especificadas em uma determinada tabela.

Armazenando informações sobre privilégios

O MySQL utiliza tabelas especiais denominada **grant tables** para armazenar informações sobre os privilégios dos usuários, em um banco de dados interno de nome **mysql**. A tabela a seguir detalha essas tabelas de privilégios:

Tabela	Descrição
user	Armazena nomes e senhas de todos os usuários do servidor. Também armazena os privilégios globais que são aplicados a todos os bancos de dados do servidor.
db	Armazena privilégios dos bancos de dados
tables_priv	Armazena privilégios das tabelas
columns_priv	Armazena privilégios de colunas
procs_priv	Armazena privilégios de acesso a funções e stored procedures (procedimentos armazenados).

Usando a declaração GRANT para atribuir privilégios

Sintaxe:

```
GRANT lista_privilégios
ON [nome_banco.]tabela
TO usuário1 [IDENTIFIED BY 'senha1'],
usuário2 [IDENTIFIED BY 'senha2'] ...
[WITH GRANT OPTION]
```

opcional

```
GRANT USAGE
ON *.*
TO julia@localhost IDENTIFIED BY '1234';
```

Garantir acesso a um usuário de nome **julia**, sem privilégios:

```
GRANT ALL
ON *.*
TO alexandre IDENTIFIED BY '1234'
WITH GRANT OPTION
```

Dar privilégios globais a um usuário de nome alexandre:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON db_biblioteca.*
TO ana@localhost;
```

Dar privilégios específicos para execução de comandos DML em todas as tabelas do banco db_biblioteca ao usuário ana:

```
GRANT ALL
ON db_biblioteca.*
TO ana@localhost;
```

Dar todos os privilégios no banco db_biblioteca à usuária ana:

```
GRANT SELECT, INSERT, UPDATE
ON db_biblioteca.tbl_autores
TO julia@localhost;
```

Garantir privilégios de inserção e atualização de registros e efetuar consultas na a tabela **tbl_autores** do banco de dados **db_biblioteca** ao usuário julia:

```
GRANT SELECT (nome_autor, sobrenome_autor), UPDATE (nome_autor)
ON db_biblioteca.tbl_autores
TO fabio@localhost;
```

Garantir o privilégio de consultar nomes e sobrenomes e alterar somente nomes dos autores (coluna nome-autor) da tabela tbl_autores do banco db_biblioteca ao usuário fabio:

```
SHOW GRANTS FOR fabio@localhost;
```

Visualizar permissões do usuário Fabio

REVOKE

Revogando privilégios com a declaração REVOKE

Podemos revogar (retirar) privilégios dos usuários usando a declaração **REVOKE**.

Sintaxe:

```
REVOKE lista_privilégios
ON objeto
FROM usuário1, usuário2, ...;
```


```
REVOKE DELETE
ON db_biblioteca.*
FROM ana@localhost;
```

revogar o privilégio de exclusão de dados no banco db_biblioteca à usuária ana:

```
REVOKE UPDATE (nome_autor)
ON db_biblioteca.tbl_autores
FROM fabio@localhost;
```

Retirando o privilégio de atualização da coluna nome_autor do banco db_biblioteca, na tabela de autores, do usuário fabio:

```
REVOKE ALL, GRANT OPTION
FROM alexandre, ana@localhost;
```

Remover todos os privilégios em todos os bancos de dados dos usuários alexandre e ana:

O comando é usado para impedir explicitamente que um usuário receba uma permissão específica.

DENY

```
DENY ALL ON alunos TO gerente;

DENY DELETE ON alunos TO secretaria;

DENY SELECT ON alunos TO diretor;
```

81 – Ao acessar uma base de dados MySQL, usando o comando *GRANT*, os privilégios são dados por meio de palavras reservadas, entre as quais podemos citar:

a) FILE, SHUTDOWN, UPDATE, INDEX, CREATE.
b) FILE, INSERT, PROCESS, CREATE, SORT.
c) FILE, SHUTDOWN, UPDATE, INDEX, MAIL.
d) INDEX, DOWNLOAD,UPDATE, DROP, CREATE.

Data warehouse

Surgido em **1980**, um **armazém de dados** é um sistema utilizado para armazenar informações sobre **atividades de uma organização** em bancos de dados de forma consolidada. Esses dados vêm dos **sistemas transacionais** (OLTP).

Com isso, é possível gerar **relatórios**, análises dos dados e tomar **decisões estratégicas** importantes.

Dados em um armazém **não podem ser voláteis** e geralmente **imutáveis** (somente leitura), salvo quando necessário fazer correções.

Data warehouse

Para explorar um data warehouse, usamos uma **ferramenta OLAP** (Online Analytical Processing) ou **Processo Analítico em Tempo Real**.

As ferramentas data warehouse fazem parte do núcleo do mercado de **Business Intelligence** (BI).

Deve-se considerar também um **hardware** potente e **usuários** preparados para esse modelo.

Prospecção de dados ou mineração de dados (também conhecida pelo termo inglês data mining) é o processo de explorar dados à procura de padrões consistentes, como regras de associação ou sequências temporais, para detectar relacionamentos sistemáticos entre variáveis, detectando assim novos subconjuntos de dados.

TÉCNICA DE EXPLORAÇÃO DE DADOS

Data mart (repositório de dados) é sub-conjunto de dados de um Data warehouse (ou DW, armazém de dados). Geralmente são dados referentes a um assunto em especial (ex: Vendas, Estoque, Controladoria) ou diferentes níveis de sumarização (ex: Vendas Anual, Vendas Mensal, Vendas 5 anos), que focalizam uma ou mais áreas específicas.

```
SELECT NOME, IDADE
  FROM TABLA
WHERE SEXO = "MASCULINO";
```

DATA MART

EXPLORADOS

Stored Procedure, que traduzido significa Procedimento Armazenado, é uma conjunto de comandos em **SQL** que podem ser executados de uma só vez, como em uma função. Ele armazena tarefas repetitivas e aceita parâmetros de entrada para que a tarefa seja efetuada de acordo com a necessidade individual.

Este é um exemplo de um stored procedure que executa uma consulta utlizando um filtro por descrição, em uma tabela específica de nosso banco de dados.

```
1  USE BancoDados
2  GO
3  CREATE PROCEDURE Busca --- Declarando o nome da procedure
4  @CampoBusca VARCHAR (20) --- Declarando variável (note que utilizamos o @ antes do
5  AS
6  SELECT Codigo, Descrição --- Consulta
7  FROM NomeTabela
8  WHERE Descricao = @CampoBusca --- Utilizando variável como filtro para a consulta
```

```
1  EXECUTE Busca 'DEVMEDIA'
```


Triggers são pequenas rotinas programadas no banco de dados, semelhantes às conhecidas Stored Procedures. Porém, como o nome indica (trigger = gatilho), uma Trigger pode ser disparada em resposta a um determinado evento. G

Onde se tem os seguintes parâmetros:

- nome: nome do gatilho, segue as mesmas regras de nomeação dos demais objetos do banco.

- momento: quando o gatilho será executado. Os valores válidos são BEFORE (antes) e AFTER (depois).

- evento: evento que vai disparar o gatilho. Os valores possíveis são INSERT, UPDATE e DELETE. Vale salientar que os comandos LOAD DATA e REPLACE também disparam os eventos de inserção e exclusão de registros, com isso, os gatilhos também são executados.

- tabela: nome da tabela a qual o gatilho está associado.

```
CREATE TRIGGER Tgr_ItensVenda_Insert AFTER INSERT
ON ItensVenda
FOR EACH ROW
BEGIN UPDATE Produtos SET Estoque = Estoque - NEW.Quantidade
WHERE Referencia = NEW.Produto; END$
```


TCL (Comando de Transação) COMMIT – Esse comando serve para confirmar uma ação dentro de uma banco de dados, por exemplo:

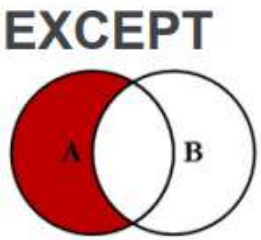
SAVEPOINT – Esse comando tem uma função parecida com a função "Ponto de Restauração" do Windows.

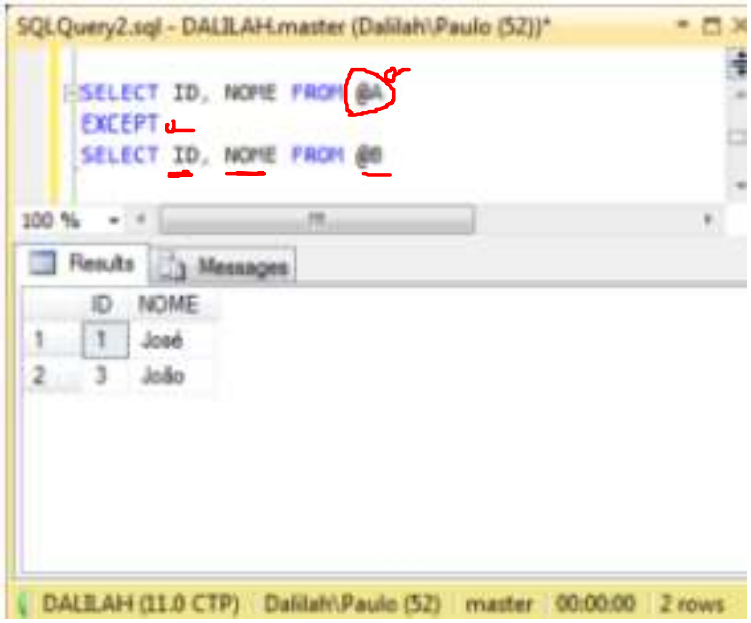
ROLLBACK – Esse comando é utilizado para desfazer as alteração até um ponto de restauração definido por SAVEPOINT.

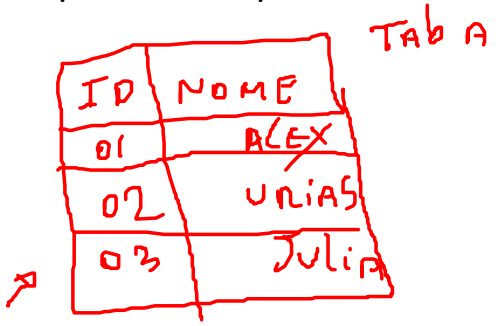
```
BEGIN TRANSACTION
INSERT INTO tabela_teste VALUES (1);
SAVEPOINT meu_savepoint;
INSERT INTO tabela_teste VALUES (2);
ROLLBACK TO SAVEPOINT meu_savepoint;
INSERT INTO tabela_teste VALUES (3);
INSERT INTO tabela_teste VALUES (4);
COMMIT;
END;
```

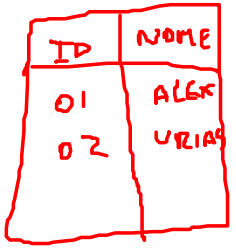

ALGUMAS OPERAÇÕES EM COM REGISTROS

EXCEPT (EXCETO) retorna linhas distintas da consulta de entrada à esquerda que não são produzidas pela consulta de entrada à direita.



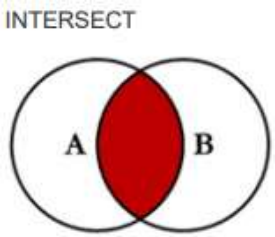


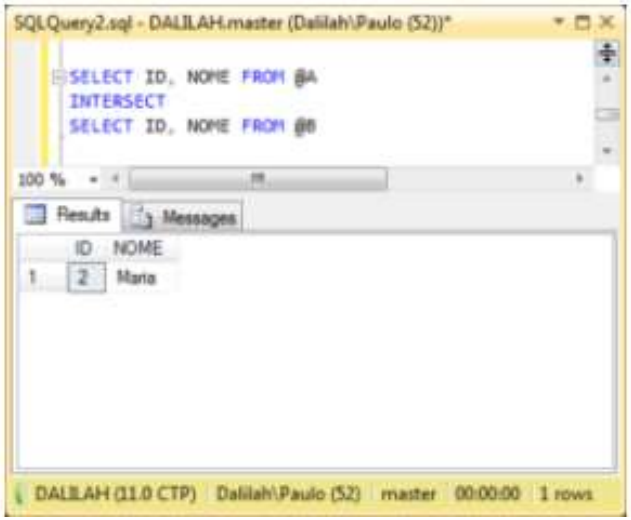




ALGUMAS OPERAÇÕES EM COM REGISTROS

INTERSECT retorna linhas distintas que são produzidas pelo operador das consultas de entrada à esquerda e à direita.





Função ISNULL

No sistema MySQL, a função **ISNULL( )** é utilizada para testar se uma expressão é NULL. Se a expressão for NULL, esta função apresenta o valor 1. Caso contrário, esta função apresenta o valor 0.
Por exemplo,

```
SELECT ID FROM A
WHERE NOME IS NULL;
```

03

01 e 02

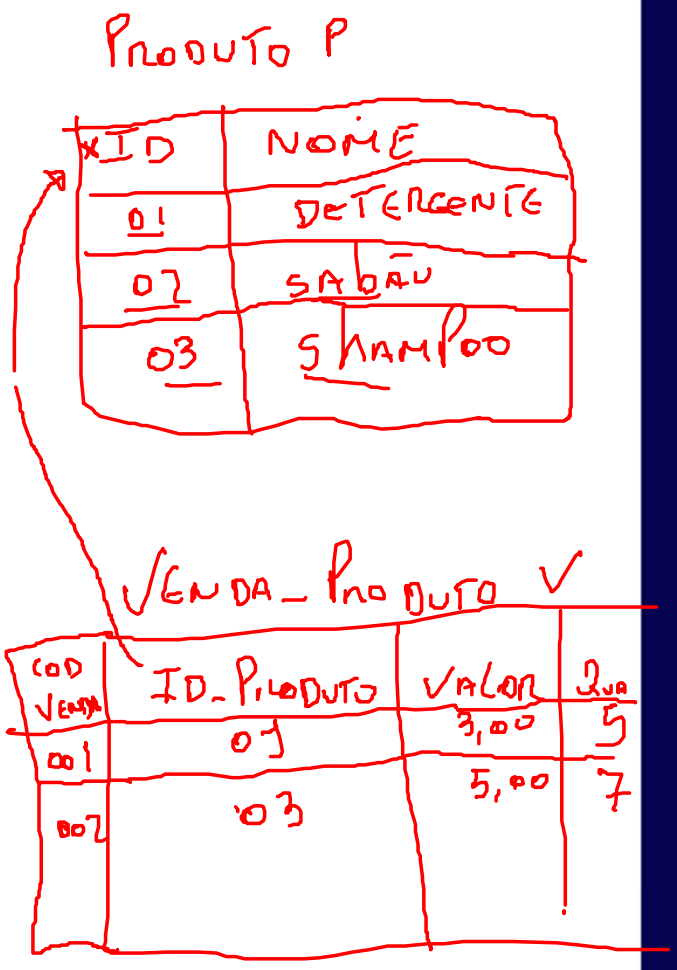
A

ID	NOME
01	ALEX
02	URIAS
03	

A **cláusula EXISTS** faz uma verificação se existe algum resultado para a subquery informada.

```
SELECT
        [ coluna1, coluna2, ... | * ]
        FROM
        [ tabela1, tabela2, ... ]
        WHERE
        EXISTS (
        SELECT
        [ coluna1, coluna2, ... | * ]
        FROM
        [ tabela1, tabela2, ... ]
        WHERE [ condicao ]
        )
```


```sql
SELECT
        P.ID,
        P.nome
        FROM
        produto P
        WHERE
    NOT EXISTS ( 01, 03
        SELECT
        V.ID_PRODUTO
        FROM
        venda_produto V
        WHERE
        V.ID_PRODUTO = P.ID
        )
```



```sql
SELECT
        P.ID,
        P.nome
        FROM
        produto P
        WHERE
        NOT EXISTS (
        SELECT
        V.ID_PRODUTO
        FROM
        venda_produto V
        WHERE
        V.ID_PRODUTO = P.ID
        )
```


– No trecho de código abaixo, identifique a linha com erro de sintaxe e marque a alternativa que contém a correção dessa linha.

```
CREATE TABLE prova ( cod_prova smallint NOT NULL,
nome_prova varchar NOT NULL,
nota real NOT NULL );
```

a) Na quarta linha, a sintaxe correta é: nota real NOT NULL,
b) Na terceira linha, a sintaxe correta é: nome_prova varchar(60) NOT NULL
c) Na segunda linha, a sintaxe correta é: cod_prova smallint NOT NULL
d) Na primeira linha, a sintaxe correta é: CREAT TABLE {};

2 – Considere a tabela chamada “clientes”, com a estrutura e os dados a seguir:

codigo_cliente	nome_cliente
1	Davi
2	Carlos
3	Ana
4	Bruna

Assinale o comando SQL utilizado para listar os clientes, ordenando os nomes dos clientes de forma decrescente.

a) SELECT codigo_cliente, nome_cliente FROM clientes ORDER BY nome _cliente DESC;
b) SELECT codigo_cliente, nome_cliente FROM clientes ORDER BY codigo_cliente ASC;
c) SELECT codigo_cliente, nome_cliente FROM clientes ORDER BY nome cliente DESC
d) SELECT codigo_cliente, nome_cliente FROM clientes ORDER BY codigo_cliente ASC;

Analise a tabela SQL abaixo chamada “escola”

codigo_aluno	nome_aluno
1100	Ana
1200	Edilene
1300	Maria
1400	Aline
1500	Ariana

Considerando o comando SQL abaixo, assinale a alternativa que apresenta todos os registros encontrados.

SELECT nome_aluno FROM escola WHERE nome_aluno LIKE ‘%a’;

a) “Ana”, “Edilene”, “Maria”, “Aline”, “Ariana”
b) “Ana”, “Maria”, “Ariana”
c) “Edilene”, “Aline”
d) “Ana”, “Ariana”

– Supondo-se que existe um usuário chamado “username” em um banco de dados MySQL, assinale a alternativa que representa o comando usado por este usuário para acessar o banco de dados.

a) mysql -n username -u
b) mysql -u username -p
c) mysql -n username -p
d) mysql -u username -l

– Assinale a alternativa que mostra a opção que ordena os dados da coluna “nome_produto” da tabela “produtos” de forma decrescente num banco de dados.

a) SELECT codigo_produto, nome_produto FROM produtos ORDER BY nome_produto;
b) SELECT codigo_produto, nome_produto FROM produtos ORDER BY nome_produto ASC;
c) SELECT codigo_produto, nome_produto FROM produtos ORDER BY nome_produto DESC;
d) SELECT codigo_produto, nome_produto FROM produtos ORDER BY nome_produto DECRESC;

– Assinale a alternativa que contém o operador LIKE retornando a letra “E” na segunda posição da coluna.

a) LIKE ‘e*%’
b) LIKE ‘*E’
c) LIKE ‘_E%’
d) LIKE ‘E*%’

MYSQL + PHP

EXPLICADORES.NET

@explicadoresnet

MYSQL

NOME

Conexão

```
$link = mysql_connect($servidor, $usuario, $senha)
or die ("Houve uma falha");

$link = mysql_pconnect($servidor, $usuario, $senha);
```

POSTGRESQL

```
pg_connect (string_conexão)

$conexao = pg_connect ("dbname=bdteste port 5432
user=postgres password=postgres");
```

A porta padrão para conexão no PostgreSQL é a 5432 (mas pode ser alterada somente no momento da instalação.)

```
pg_close ($conexao);
```


Abrindo o Banco

```
mysql_select_db("banco", $link);
```

Executando Queries

```
$busca = mysql_query("SELECT * FROM tabela");
```


CONECTA_MYSQL.INC

```php
<?php

$conexao = mysql_connect ("localhost", "juan" , "teste");

mysql_select_db ("bdteste");

?>
```

```php
<?php

    include "conecta_mysql.inc";

    $resultado = mysql_query ("SELECT * FROM produtos");

    mysql_close($conexao);

?>
```

```php
<?php

$str_conexao = "dbname=bdteste port=5432 user=postgres password=postgres";

  If(!($conexao=pg_connect ($str_conexao))) {

      echo "Não foi possível estabelecer uma conexão com o banco de dados.";

exit;

}

?>
```


```php
<?php
    include "conecta_pg.inc";
    $sql = "SELECT * FROM produtos";
    $resultado = pg_query ($conexao, $sql);
    pg_close ($conexao);
?>
```

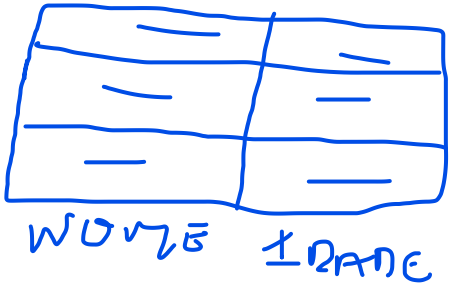

$X = SELECT NOME, IDADE FROM ALUNO WHERE UF="RJ";

SELECT NOME FROM ALUNO WHERE UF="SP";

10

5

MySQL	PostgreSQL
Abrir conexão	
mysql_connect ou mysqli_connect	pg_connect
Fechar conexão	
mysql_close	pg_close
Especificar banco de dados	
mysql_select_db ou mysqli_select_db	Não possui esta função.



Para tratar as informações retornadas pelos comandos SQL que executamos, existe uma série de funções que podemos utilizar, segue abaixo as principais funções (Única diferença é a nomenclatura para o MySQL e o PostgreSQL.

MySQL	PostgreSQL	Descrição
mysql_affected_rows utilizada nas operações: **(INSERT – UPDATE – DELETE)**	pg_affected_rows	Retorna o número de linhas afetadas.
mysql_fetch_array	pg_fetch_array	Armazena a linha atual do resultado em um array associativo.
mysql_fetch_object	pg_fetch_object	Retorna uma linha como um objeto.
mysql_fetch_row	pg_fetch_row	Armazena a linha atual do resultado **em um array.**
mysql_result	pg_result	Retorna uma coluna do resultado.
mysql_result_num_rows utilizada em consultas: **(SELECT)**	pg_result_num_rows	Retorna o número de linhas.
mysql_num_fields	pg_num_fields	Retorna o número de colunas.
mysql_field_name	pg_field_name	Retorna o nome de uma coluna.

➢ PostgreSQL

↪ Está disponível sob a flexível BSD
↪ É mais robusto e possui mais recursos
↪ É mais qualificado / eficiente
↪ Possui sofisticado mecanismo de bloqueio (MVCC)
↪ Suporta tamanhos ilimitados de linhas
↪ BD de até 16TB
↪ Aceita vários tipos de subconsulta
↪ Conta com um mecanismo failsafe (segurança contra falhas.)

↪ Utilitário postgreSQL = psql

↪ COMANDO para CRIAR um BD: createdb ou psql (utilizando o comando CREATE DATABASE)

Ex: $ createdb bdteste

↪ postmaster : gerenciador de conexões (Deve estar rodando para que o bd aceite as conexões solicitadas.)

↪ COMANDO para ACESSAR um BD:
psql (É um gerenciador que acompanha o PostgreSQL.)

Ex: $ psql nomedobd

↪ Recorremos ao psql para saber quais são os bancos de dados que já foram criados e quais as tabelas e os índices existentes em cada um.

➢ \ ? > Obtém acesso a todas opções oferecidas pelo gerenciador. ➢

\dt > Visualizar as tabelas existentes. ➢

\di > Visualiza os índices. ➢

\h > (HELP) = ajuda / obtém ajuda na sintaxe de comandos SQL.

Opções:

- \d > Mostra o nome de todas as tabelas, os índices e as sequências existentes no banco de dados.

COMANDO para VISUALIZAR A ESTRUTURA da tabela:

↪ \d COMANDO para VISUALIZAR A TABELA DOS PRODUTOS:

↪ \d produtos

mysql post

desc alunos;

CRIANDO SEQUÊNCIA

mat int serial

↪ Sequência: É o recurso que utilizamos para fazer uma numeração automática de determinado campo de uma tabela.

↪ A maneira mais fácil no PostgreSQL de criar uma sequência é criando um campo do tipo SERIAL em uma tabela EX da maneira mais fácil:

↪Outra maneira é utilizando o comando CREATE SEQUENCE. Ex da criação de uma sequência chamada seq_teste: CREATE SEQUENCE seq_teste;

↪ Após a criação da sequência, podemos utilizar as funções abaixo para poder administrá-la:

- NEXTVAL > Retorna o próximo valor da sequência e incrementa o contador.

EX: SELECT NEXTVAL('seq_teste');

- CURRVAL > Retorna o valor atual da sequência;

EX: SELECT CURRVAL('seq_teste');

- SETVAL > Altera o valor do contador da sequência.

EX: SELECT SETVAL ('seq_teste' , 5);

– Quais são as funções utilizadas para executar comandos SQL no MySQL e no PostgreSQL, respectivamente?

a) mysql_query e pg_query
b) pg_query e mysql_query
c) mysql_base e pg_base
d) mysql_db e pg_db

8 – Em se tratando do utilitário psql, para a criação de um banco de dados no PostgreSQL, devemos recorrer a esse utilitário para saber quais são os bancos de dados que já foram criados e quais são as tabelas e os índices existentes em cada um. Considerando essa proposição, relacione as colunas de acordo com cada definição e assinale a alternativa correta.

1 – \?
2 – \h
3 – \di
4 – \dt

( ) Para visualizar os índices.
( ) Para obter ajuda na sintaxe de comandos SQL.
( ) Para visualizar o nome das tabelas existentes.
( ) Para obter acesso a todas as opções oferecidas pelo gerenciador.

a) 3 – 1 – 4 – 2
b) 2 – 1 – 4 – 3
c) 3 – 2 – 4 – 1
d) 4 – 1 – 2 – 3

– A fim de tratar as informações retomadas por comandos SQL, o PHP apresenta uma série de funções a serem utilizadas. Dadas as funções PHP existentes para tratar informações vindas de bancos de dados MySQL, relacione a coluna da esquerda com a da direita e, em seguida, assinale a alternativa correta.

1 – mysql_affected_rows
2 – mysql_fetch_object
3 – mysql_fetch_array
4 – mysql_num_fields
5 – mysql_num_rows

(3) Armazena a linha atual do resultado em um array associativo.
(1) Retoma o número de linhas afetadas por uma operação.
(4) Retoma o número de colunas de uma consulta.
(5) Retoma o número de linhas de uma consulta.
(2) Retoma uma linha como um objeto.

a) 5 – 1 – 4 – 2 – 3
b) 3 – 1 – 4 – 5 – 2
c) 3 – 4 – 1 – 5 – 2
d) 5 – 4 – 1 – 2 – 3

– Em um programa PHP, para fazer a conexão com um banco de dados PostgreSQL, qual função é utilizada ?

a) pg_connect
b) pg_insert
c) pg_close
d) connect

Assinale a alternativa que contém o comando em PHP utilizado para selecionar o banco de dados ativo, depois de aberta a conexão com o MySQL.

a) mysql_query("select * from nome_do_banco",$id_da_conexao);
b) mysql_connect("host", "usuario", "senha");
c) mysql_select_db("nome_do_banco");
d) mysql_close($id_da_conexao);

– Em um script PHP que acessa um banco de dados MySQL, o programador armazenou o resultado de uma consulta SQL em uma variável utilizando a seguinte linha: $resultado = mysql_query("SELECT * FROM usuarios"); Caso o programador deseje saber a quantidade de linhas retornadas pelo banco de dados, deverá usar a função

a) mysql_affected_rows($resultado)
b) mysql_num_rows($resultado)
c) mysql_num_fields($resultado)
d) mysql_fetch_row($resultado)

– Assinale a alternativa que mostra o recurso disponível em SQL para realizar a numeração automática em determinado campo de uma tabela.

a) LIMIT
b) SEQUENCE
c) MAX
d) SUM