Documentação sobre a atividade

PRIMEIRA VIEW: TABELA SALES

1.1 - na tabela sales foi criada uma view para a visualização das colunas sales_total, sales-date, who_bought onde o id funcionário é maior que 4, para uma melhor visualização dessas colunas

Criação da tabela

```
CREATE TABLE SALES(
SALE_ID INT PRIMARY KEY AUTO_INCREMENT,
SALES_DATE DATE,
WHO_BOUGHT VARCHAR (50),
SALES_TOTAL DECIMAL,
worker_id int,
foreign key (worker_id) references FUNCIONARIOS (ID_FUNCIONARIO)
);
```

Criação da view

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `factory`.`view_sales_informations` AS
    SELECT
        `factory`.`sales`.`SALES_DATE` AS `sales_date`,
        `factory`.`sales`.`SALES_TOTAL` AS `sales_total`,
        `factory`.`sales`.`WHO_BOUGHT` AS `who_bought`
    FROM
        `factory`.`sales`
    WHERE
        (`factory`.`sales`.`SALE_ID` > 4)
```

SEGUNDA VIEW: TABELA SUPPLIERS

1.2 - na tabela suppliers foi criada uma view para a visualização das colunas namee e address em que o id do supplier é menor ou igual a 5, isto facilita a visualização das colunas endereço do supplier e nome do supplier.

Criação da tabela

```
CREATE TABLE SUPPLIERS (
SUPPLIERS_ID INT PRIMARY KEY AUTO_INCREMENT,
NAMEE VARCHAR(50),
TELEPHONE CHAR (14),
ADDRESS VARCHAR(50),
TYPE_OF_MATERIALS VARCHAR(50)
);
```

Criação da view

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `factory`.visualização_suppliers AS
    SELECT
        factory.suppliers.NAMEE AS namee,
        factory.suppliers.ADDRESS AS address
    FROM
        factory.suppliers
    WHERE
        (factory.suppliers.SUPPLIERS_ID <= 5)
```

TERCEIRA VIEW: TABELA CLIENTS

1.3 - na tabela clients foi feito uma view das colunas clients_id e clients_name, porém só vai ser mostrado os dados dos clientes que tem um id maior que 3, está view facilitará na visualização das colunas id do cliente e nome do cliente.

Criação da tabela

```
CREATE TABLE CLIENTS (
CLIENTS_ID INT PRIMARY KEY AUTO_INCREMENT,
CLIENTS_NAME text,
CONTACTS text
);
```

Criação da view

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `factory`.visualização_clientes AS
    SELECT
        factory.clients.CLIENTS_ID AS clients_id,
        factory.clients.CLIENTS_NAME AS clients_name
    FROM
        factory.clients
    WHERE
        (factory.clients.CLIENTS_ID > 3)
```

QUARTA VIEW: TABELA PRODUCTS

1.4 - na tabela products foi criada uma view para a visualização das colunas name_products e price, porém só será mostrada as informações se o id do produto for maior ou igual a 3, está view facilita na visualização das colunas nome do produto e preço.

Criação da tabela

```
CREATE TABLE PRODUCTS (
ID_PRODUCTS INT PRIMARY KEY AUTO_INCREMENT,
NAME_PRODUCTS VARCHAR (50),
DESCRIPTION_PRODUCT TEXT,
PRICE DECIMAL (6,2),
AVAILABLE_IN_STOCK INT,
SIZE VARCHAR(50),
VERSIONS VARCHAR(50),
SUPPLIERS_ID INT,
FOREIGN KEY ( SUPPLIERS_ID ) REFERENCES SUPPLIERS (SUPPLIERS_ID),
foreign key ( suppliers_id) references suppliers (suppliers_id) on update cascade
);
```

Criação da view

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `factory`.visualização_produtos AS
    SELECT
        factory.products.NAME_PRODUCTS AS name_products,
        factory.products.PRICE AS price
    FROM
        factory.products
```

```
WHERE
    (factory.products.ID_PRODUCTS >= 3)
```

STORED PROCEDURES

PRIMEIRO STORED PROCEDURES 👍 :

1.1 - Este stored procedure foi criado para que haja inserção de dados na tabela funcionarios, poderá ser inseridos as seguintes informações sobre os funcionários nome_funcionario, cargo_funcionaio, data_contratação, id_funcionario, salario e fabrica.

Tabela

```
CREATE TABLE FUNCIONARIOS (
ID_FUNCIONARIO INT PRIMARY KEY AUTO_INCREMENT,
NOME_FUNCIONARIO VARCHAR(50),
CARGO varchar(50),
DATA_DE_CONTRATAÇÃO DATE,
SALARIO DECIMAL (6,2),
FABRICA VARCHAR (50)
);
```

Primeiro Stored Procedure

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Manipulação_Funcionarios`()
BEGIN

insert into funcionarios (id_funcionario, nome_funcionario, cargo,data_contratação) values
(default,nome_funcionario,cargo,data_contratação);

END
```

Tabela

```
CREATE TABLE PRODUCTS (
ID_PRODUCTS INT PRIMARY KEY AUTO_INCREMENT,
NAME_PRODUCTS VARCHAR (50),
DESCRIPTION_PRODUCT TEXT,
PRICE DECIMAL (6,2),
AVAILABLE_IN_STOCK INT,
SIZE VARCHAR(50),
VERSIONS VARCHAR(50),
SUPPLIERS_ID INT,
FOREIGN KEY ( SUPPLIERS_ID ) REFERENCES SUPPLIERS (SUPPLIERS_ID),
foreign key ( suppliers_id) references suppliers (suppliers_id) on update cascade
);
```

SEGUNDO STORED PROCEDURES 👍 :

1.2 - Este stored procedures tem a função de inserção dos seguintes dados das seguintes colunas : id_products, name_products, description_product, price,size na tabela products, este stored procedure facilitará na inserção de dados.

Segundo Stored Procedure

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `Inserção_de_Produtos`()
BEGIN

insert into products (id_products,name_products,description_product,price,size) values
(default,name_products,description_product,price,size);

END
```

TRIGGERS

Esta trigger ira deletar os suppliers_id da tabela products quando este mesmo dado da tabela suppliers for deletado, o supplier_id é uma chave estrangeira na tabela suppliers

Código:

```
delimiter $$
create trigger adicionar_supplier
before delete on suppliers
for each row
begin
delete from products where suppliers_id = OLD.suppliers_id;

end $$
```