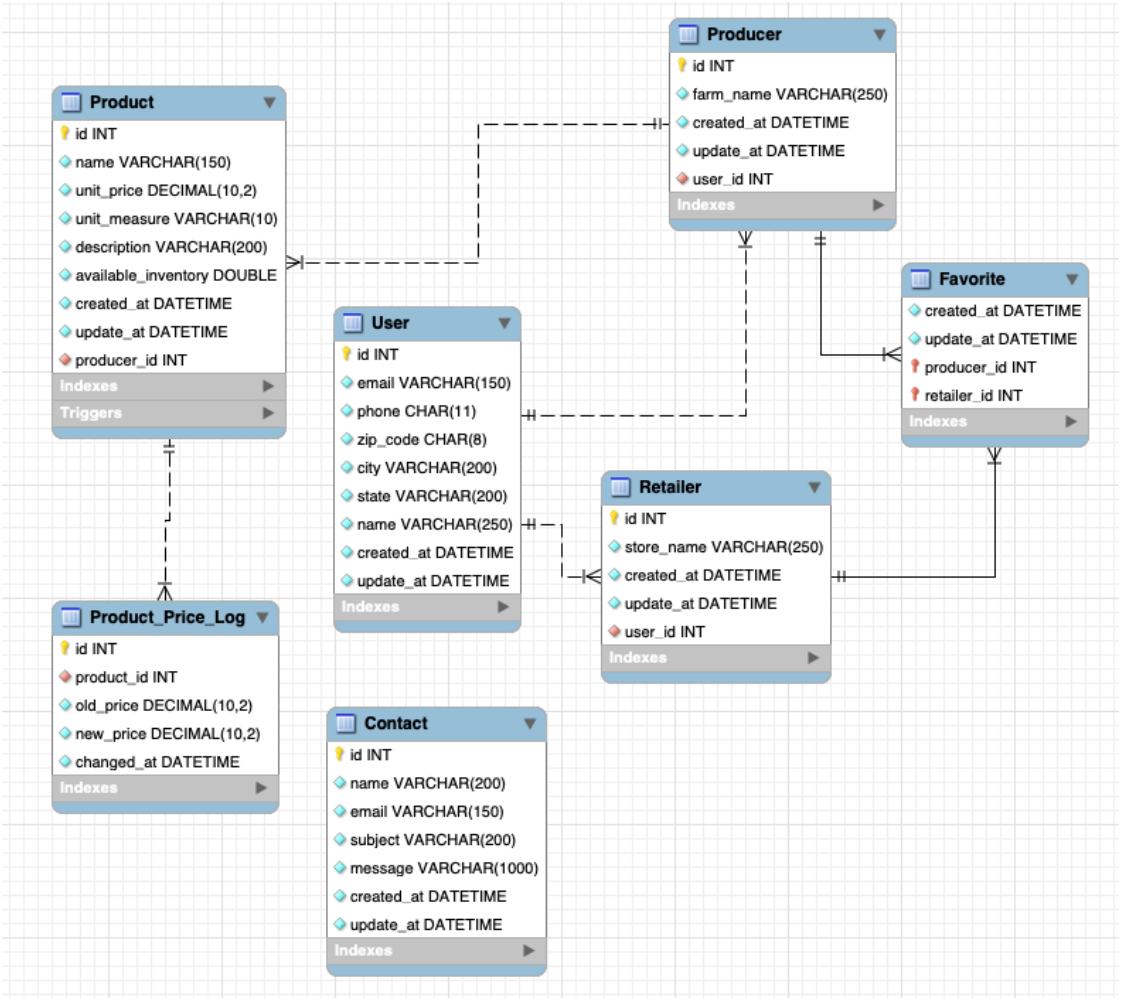
Documentação Técnica
Projeto de Extensão (Agrolink)
Banco de Dados 2

Descrição do Projeto:
A AgroLink é uma startup emergente no setor de agrotech, cujo objetivo é revolucionar a conexão entre produtores rurais e varejistas por meio de uma plataforma digital inovadora. A solução não apenas facilita a comunicação direta entre as duas pontas da cadeia de suprimentos, mas também gera oportunidades de venda para produtores e otimiza o processo de busca e comparação de produtos para os varejistas.

Objetivo do Banco de Dados:
Esta documentação descreve a estrutura, os componentes e a lógica de negócios do banco de dados Agrolink. O objetivo deste banco de dados é gerenciar uma plataforma que conecta produtores (agricultores, fazendas) a varejistas (lojas, mercados), facilitando a visualização e o gerenciamento de produtos, inventário e relacionamentos comerciais (como favoritos).

Participantes:
- Lucas Henrique Melo Sena - CP3032094
- Gustava Pereira Gonçalves - CP3032221

Modelagem de Dados
Diagrama Entidade-Relacionamento



Estrutura do Banco de Dados:

O banco de dados é composto por sete tabelas principais, conforme definido em criacao_banco.sql e visualizado em Esquema.pdf.

Script de Criação:

```sql
DROP DATABASE IF EXISTS agrolink;
CREATE DATABASE agrolink;
USE agrolink;

CREATE TABLE User (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    email VARCHAR(150) NOT NULL UNIQUE,
    phone CHAR(11) NOT NULL UNIQUE,
    zip_code CHAR(8) NOT NULL,
    city VARCHAR(200) NOT NULL,
    state VARCHAR(200) NOT NULL,
    name VARCHAR(250) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    update_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Retailer (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    store_name VARCHAR(250) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    update_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

    user_id INT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(id)
);
```

```sql
CREATE TABLE Producer (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    farm_name VARCHAR(250) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    update_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

    user_id INT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(id)
);

CREATE TABLE Favorite (
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    update_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

    producer_id INT NOT NULL,
    FOREIGN KEY (producer_id) REFERENCES Producer (id),
    retailer_id INT NOT NULL,
    FOREIGN KEY (retailer_id) REFERENCES Retailer (id),

    PRIMARY KEY (producer_id, retailer_id)
);

CREATE TABLE Product (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(150) NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    unit_measure VARCHAR(10) NOT NULL,
    description VARCHAR(200) NOT NULL,
    available_inventory DOUBLE NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    update_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

    producer_id INT NOT NULL,
    FOREIGN KEY (producer_id) REFERENCES Producer(id)
);
```

```sql
CREATE TABLE Contact (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(200) NOT NULL,
    email VARCHAR(150) NOT NULL,
    subject VARCHAR(200) NOT NULL,
    message VARCHAR(1000) NOT NULL,
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    update_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Product_Price_Log (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    product_id INT NOT NULL,
    old_price DECIMAL(10, 2) NOT NULL,
    new_price DECIMAL(10, 2) NOT NULL,
    changed_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (product_id) REFERENCES Product(id) ON DELETE CASCADE
);
```

Tabela: User

- Descrição: Tabela central que armazena informações básicas de todos os usuários da plataforma. Ela serve como base para os perfis de Producer e Retailer.
- Campos Principais:
    - id (PK): Identificador único do usuário.
    - email (UNIQUE): Email de login, não pode ser repetido.
    - phone (UNIQUE): Telefone, não pode ser repetido.
    - name, zip_code, city, state: Informações de perfil e localização.

Tabela: Producer

- Descrição: Representa o perfil de um produtor (fazenda) na plataforma.
- Campos Principais:
    - id (PK): Identificador único do produtor.
    - farm_name: O nome da fazenda ou propriedade.
    - user_id (FK): Chave estrangeira que referencia User.id, ligando o perfil do produtor a uma conta de usuário.

Tabela: Retailer

- Descrição: Representa o perfil de um varejista (loja, mercado) na plataforma.
- Campos Principais:
    - id (PK): Identificador único do varejista.
    - store_name: O nome do estabelecimento comercial.

    - user_id (FK): Chave estrangeira que referencia User.id, ligando o perfil do varejista a uma conta de usuário.

Tabela: Product

- Descrição: Armazena todos os produtos cadastrados pelos produtores.
- Campos Principais:
    - id (PK): Identificador único do produto.
    - name: Nome do produto (ex: 'Laranja Pera').
    - unit_price: Preço por unidade de medida.
    - unit_measure: Unidade de medida (ex: 'kg', 'un', 'sc').
    - available_inventory: Quantidade atual em estoque.
    - producer_id (FK): Chave estrangeira que referencia Producer.id, indicando qual produtor cadastrou o item.

Tabela: Favorite

- Descrição: Tabela de junção (N:M) que implementa a funcionalidade de "favoritos". Ela rastreia quais varejistas marcaram quais produtores como favoritos.
- Campos Principais:
    - producer_id (PK, FK): Referencia Producer.id.
    - retailer_id (PK, FK): Referencia Retailer.id.

Tabela: Contact

- Descrição: Armazena as mensagens enviadas através de um formulário de contato do site, como dúvidas, sugestões ou problemas técnicos.
- Campos Principais: name, email, subject, message.

Tabela: Product_Price_Log

- Descrição: Tabela de auditoria dedicada a rastrear todas as alterações de preço dos produtos.
- Campos Principais:
    - product_id (FK): Referencia Product.id.
    - old_price: O preço anterior à atualização.
    - new_price: O novo preço definido na atualização.
    - changed_at: Data e hora da alteração.

Consultas:

1.1 Utiliza um JOIN triplo (envolvendo Producer, Favorite e User) para listar todos os produtores (nome da fazenda, nome do produtor e localização) que foram favoritados por um varejista específico (neste caso, retailer_id = 1).

```sql
SELECT
    P.id,
    P.farm_name,
    U.name AS nome_produtor,
    U.city,
    U.state
FROM
    Producer P
JOIN
    Favorite F ON P.id = F.producer_id
JOIN
    User U ON P.user_id = U.id
WHERE
    F.retailer_id = 1;
```

1.2 Retorna as informações de contato detalhadas (nome, email, telefone, localização) e o nome da fazenda de um produtor específico (P.id = 1), unindo as tabelas User e Producer.

```sql
SELECT
    U.name,
    U.email,
    U.phone,
    U.city,
    U.state,
    P.farm_name
FROM
    User U
JOIN
    Producer P ON U.id = P.user_id
WHERE
    P.id = 1;
```

2. Junta Producer, User e Product. Filtra apenas produtos que têm estoque > 0. Agrupa os resultados por produtor. Filtra os grupos para mostrar apenas os produtores que têm mais de 2 produtos com estoque disponível.

```sql
SELECT
    P.farm_name,
    U.name AS nome_produtor,
    COUNT(Prod.id) AS total_produtos_disponiveis
FROM
    Producer P
JOIN
    User U ON P.user_id = U.id
JOIN
    Product Prod ON P.id = Prod.producer_id
WHERE
    Prod.available_inventory > 0
GROUP BY
    P.id, P.farm_name, U.name
HAVING
    total_produtos_disponiveis > 2;
```

3. Filtra produtores de acordo com produto: Junta Producer e User. Filtra os produtores (P.id) que estão na lista de producer_id retornada pela subconsulta (produtos com nome 'Tomate').

```sql
SELECT
    P.id,
    P.farm_name,
    U.name
FROM
    Producer P
JOIN
    User U ON P.user_id = U.id
WHERE
    P.id IN (
        SELECT DISTINCT producer_id
        FROM Product
        WHERE name LIKE '%Tomate%'
    );
```

4. Busca o valor total em estoque de cada produtor: Acessa Product. Agrupa por producer_id. Calcula a SOMA(unit_price * available_inventory) para cada grupo. Ordena o resultado pelo valor total em ordem decrescente.

```sql
SELECT
    producer_id,
    SUM(unit_price * available_inventory) AS valor_total_estoque
FROM
    Product
GROUP BY
    producer_id
ORDER BY
    valor_total_estoque DESC;
```

5. Retorna a quantidade de produtos que foram cadastrados no dia atual: Acessa Product. Conta os produtos onde o created_at está entre a meia-noite de hoje e a meia-noite de amanhã.

```sql
SELECT
    COUNT(id) AS novos_produtos_hoje
FROM
    Product
WHERE
    created_at >= CURDATE()
    AND created_at < (CURDATE() + INTERVAL 1 DAY);
```

Procedures e Functions:

1. Tenta inserir a relação retailer_id e producer_id na tabela Favorite. Se a chave já existir, apenas atualiza o registro. Confirma a operação ou reverte em caso de erro.

```sql
DELIMITER //
CREATE PROCEDURE Proc_Favoritar_Produtor(
    IN p_retailer_id INT,
    IN p_producer_id INT
)
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
    END;

  START TRANSACTION;

    INSERT INTO Favorite (retailer_id, producer_id)
    VALUES (p_retailer_id, p_producer_id)
    ON DUPLICATE KEY UPDATE producer_id = p_producer_id;

    COMMIT;
END //
DELIMITER ;

CALL Proc_Favoritar_Produtor(1, 2);
```

2. Recebe um preço. Utiliza uma estrutura condicional para classificar o preço em "Barato", "Médio", "Caro" ou "Indefinido". Retorna a string correspondente à faixa de preço.

```sql
DELIMITER //
CREATE FUNCTION Fn_Get_Price_Range(
    p_price DECIMAL(10, 2)
)
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
    DECLARE v_range VARCHAR(20);

    IF p_price IS NULL THEN
        SET v_range = 'Indefinido';
    ELSEIF p_price <= 5.00 THEN
        SET v_range = 'Barato';
    ELSEIF p_price <= 15.00 THEN
        SET v_range = 'Médio';
    ELSE
        SET v_range = 'Caro';
    END IF;

    RETURN v_range;
END //
DELIMITER ;

SELECT
    name,
    unit_price,
    Fn_Get_Price_Range(unit_price) AS faixa_preco
FROM
    Product;
```

3. Recebe um p_producer_id. Cria uma tabela temporária (Temp_Relatorio_Produtos) para armazenar os dados. Insere os produtos filtrados pelo p_producer_id nessa tabela. Por fim, exibe todos os dados da tabela temporária.

```sql
DELIMITER //
CREATE PROCEDURE Proc_Relatorio_Produtos_Produtor(
    IN p_producer_id INT
)
BEGIN
    DROP TEMPORARY TABLE IF EXISTS Temp_Relatorio_Produtos;

    CREATE TEMPORARY TABLE Temp_Relatorio_Produtos (
        nome_produto VARCHAR(150),
        estoque DOUBLE,
        preco DECIMAL(10, 2)
    );

    INSERT INTO Temp_Relatorio_Produtos (nome_produto, estoque, p
    SELECT
        name,
        available_inventory,
        unit_price
    FROM
        Product
    WHERE
        producer_id = p_producer_id;

    SELECT * FROM Temp_Relatorio_Produtos;
END //
DELIMITER ;

CALL Proc_Relatorio_Produtos_Produtor(1);
```

4. Recebe um p_product_id e uma p_quantidade_mudanca. Busca o estoque atual, calcula o novo estoque. Se o novo estoque for maior ou igual a zero, realiza a atualização. Caso contrário, dispara um erro e impede a atualização.

```sql
DELIMITER //
CREATE PROCEDURE Proc_Atualizar_Estoque(
    IN p_product_id INT,
    IN p_quantidade_mudanca DOUBLE
)
BEGIN
    DECLARE v_estoque_atual DOUBLE;
    DECLARE v_estoque_novo DOUBLE;

    SELECT available_inventory INTO v_estoque_atual
    FROM Product
    WHERE id = p_product_id;

    SET v_estoque_novo = v_estoque_atual + p_quantidade_mudanca;

    IF v_estoque_novo >= 0 THEN
        UPDATE Product
        SET available_inventory = v_estoque_novo
        WHERE id = p_product_id;
    ELSE
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Mudança no estoqu
    END IF;
END //
DELIMITER ;

CALL Proc_Atualizar_Estoque(1, 49.0);
CALL Proc_Atualizar_Estoque(1, -550.0);
```

Transactions:

01. Tenta executar duas inserções: um User e um Retailer. Como a segunda inserção pode falhar a depender do user_id fornecido, o manipulador de erros é acionado, desfazendo a primeira inserção e garantindo que nenhum dado seja salvo.

```
DELIMITER //
CREATE PROCEDURE Proc_Create_Retailer()
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Erro detectado! Transação revertida.' AS Resultad
    END;

    START TRANSACTION;

    INSERT INTO User (email, phone, zip_code, city, state, name)
    VALUES ('usuario_teste@example.com', '11988887777', '12345678

    INSERT INTO Retailer (store_name, user_id)
    VALUES ('Loja Inválida', 999999);

    COMMIT;
    SELECT 'Transação concluída com sucesso.' AS Resultado;
END //

DELIMITER ;

CALL Proc_Create_Retailer();
SELECT * FROM User WHERE email = 'usuario_teste@example.com';
```

Triggers:

01. Verifica se o estoque do produto a ser excluído é maior que 0. Se for, dispara um erro e bloqueia a exclusão.

```
DELIMITER //
CREATE TRIGGER Trg_Prevent_Product_Deletion
BEFORE DELETE ON Product
FOR EACH ROW
BEGIN
    IF OLD.available_inventory > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Não é permitido excluir um produt
    END IF;
END //
DELIMITER ;

SELECT name, available_inventory FROM Product WHERE id = 1;
DELETE FROM Product WHERE id = 1;
```

```
✖ 43  18:09:02  DELETE FROM Product WHERE id = 1  Error Code: 1644. Não é permitido excluir um produto que ainda possui estoque disponível.
```

02. Verifica se o unit_price antigo é diferente do novo. Se for, insere um registro na tabela Product_Price_Log com os detalhes da mudança.

```
DELIMITER //
CREATE TRIGGER Trg_Audit_Product_Price_Change
AFTER UPDATE ON Product
FOR EACH ROW
BEGIN
    IF OLD.unit_price <> NEW.unit_price THEN
        INSERT INTO Product_Price_Log
            (product_id, old_price, new_price, changed_at)
        VALUES
            (NEW.id, OLD.unit_price, NEW.unit_price, NOW());
    END IF;
END //
DELIMITER ;

SELECT name, unit_price FROM Product WHERE id = 1;
UPDATE Product SET unit_price = 5.20 WHERE id = 1;
SELECT * FROM Product_Price_Log;
```

id	product_...	old_price	new_price	changed_at
1	1	4.50	4.75	2025-11-09 21:03:00
2	2	10.00	9.80	2025-11-09 21:03:00
3	1	4.75	5.00	2025-11-09 21:03:00
4	3	1.99	2.25	2025-11-09 21:03:00
5	2	9.80	10.50	2025-11-09 21:03:00
6	1	4.50	5.20	2025-11-09 21:10:27

Product_Price_Log 3

Views:

01. Junta Producer, User e faz LEFT JOIN com Product. Agrupa por produtor. Calcula a contagem de produtos, o valor total do estoque e usa uma subconsulta na tabela Favorite para contar o total de favoritos de cada produtor.

```
CREATE VIEW Vw_Relatorio_Produtores AS
SELECT
    P.id AS id_produtor,
    P.farm_name AS nome_fazenda,
    U.state AS estado,

    COUNT(DISTINCT Prod.id) AS total_produtos_ofertados,
    COALESCE(SUM(Prod.unit_price * Prod.available_inventory), 0) AS valor_total_e
stoque,
    (SELECT COUNT(*) FROM Favorite F WHERE F.producer_id = P.id) AS total_favorit
os
FROM
    Producer P
JOIN
    User U ON P.user_id = U.id
LEFT JOIN
    Product Prod ON P.id = Prod.producer_id
GROUP BY
    P.id, P.farm_name, U.state;

SELECT * FROM Vw_Relatorio_Produtores
ORDER BY valor_total_estoque DESC;
```

02. Junta as tabelas Product, Producer e User. Cria uma visualização única que combina todos os detalhes de um produto com as informações completas do seu produtor (nome da fazenda, nome do usuário, localização).

```
CREATE VIEW Vw_Full_Product_Details AS
SELECT
    Prod.id AS product_id,
    Prod.name AS product_name,
    Prod.unit_price,
    Prod.unit_measure,
    Prod.available_inventory,
    Prod.description,
    P.id AS producer_id,
    P.farm_name,
    U.name AS producer_name,
    U.city,
    U.state
FROM
    Product Prod
JOIN
    Producer P ON Prod.producer_id = P.id
JOIN
    User U ON P.user_id = U.id;
```

03. Demonstra o uso da View anterior. A consulta filtra a Vw_Full_Product_Details (como se fosse uma tabela) para retornar apenas os registros de um producer_id específico.

```sql
SELECT * FROM Vw_Full_Product_Details
WHERE producer_id = 12;
```

04. Acessa a tabela User. Seleciona um subconjunto de colunas para criar uma visualização que oculta dados sensíveis (como email, telefone e cep).

```sql
CREATE VIEW Vw_Public_User_Directory AS
SELECT
    id,
    name,
    city,
    state,
    created_at
FROM
    User;
```