INTRODUÇÃO A SISTEMAS DE BANCO DE DADOS - JOGO RPG

Integrantes:

- Gabriel Fagundes Mesquita Sousa
- Gilmar Silva de Medeiros Filho
- Hugo Dias Pontello
- Melissa de Carvalho Vila Real
- Samuel de Oliveira Vanoni

➢ **DESCRIÇÃO DO PROJETO:**

A ideia do projeto é criar um sistema de gerenciamento de dados para um jogo RPG, onde serão armazenadas as informações de personagens, classes, itens e missões para que o jogo possa funcionar corretamente além de dar a opção de melhorias como adição de novos itens, missões ou personagens.

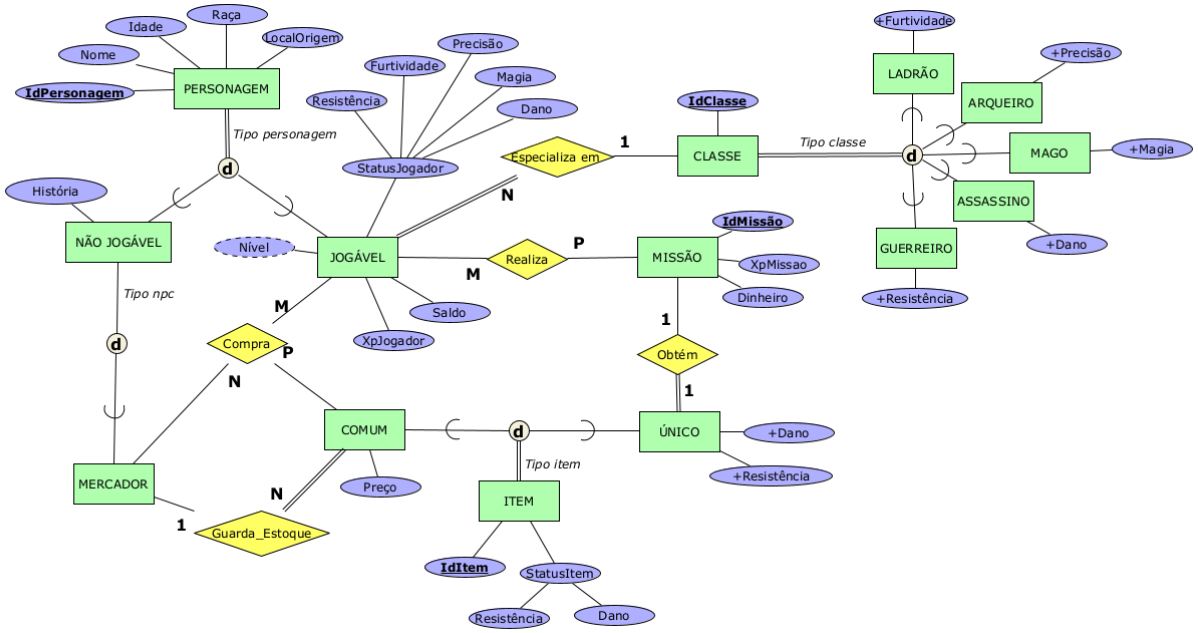
Um **PERSONAGEM** possui como atributos um identificador (único), nome, idade, raça e local de origem. Esse personagem precisa ser obrigatoriamente um **JOGÁVEL** ou **NÃO JOGÁVEL**. O **NÃO JOGÁVEL** possui como atributo uma história que ele contará para o personagem jogável quando se encontrarem. Além disso, o **NÃO JOGÁVEL** pode ser um **MERCADOR** que possui uma relação para guardar um estoque de itens **COMUM,** cujos quais podem ser vendidos para o personagem **JOGÁVEL**.

Já o personagem **JOGÁVEL** possui como atributos nível, experiência (xp), saldo e status que são divididos em: resistência, furtividade, precisão, magia e dano. A experiência (xp) é a quantidade de pontos que o **JOGÁVEL** acumula durante o jogo ao realizar missões e isso reflete no nível (derivado) que consequentemente aumenta seus status. O personagem **JOGÁVEL** pode comprar itens do **MERCADOR**. Esse tipo de personagem precisa obrigatoriamente se especializar em uma **CLASSE**, que possui como atributo um identificador e obrigatoriamente deve ser um **LADRÃO**, **ARQUEIRO**, **MAGO**, **ASSASSINO** ou **GUERREIRO**. Cada uma dessas entidades da **CLASSE** dá ao personagem **JOGÁVEL** um aumento em algum status: o **LADRÃO** dá mais furtividade, o **ARQUEIRO** mais precisão, o **MAGO** mais magia, o **ASSASSINO** mais dano e o **GUERREIRO** mais resistência.

O personagem **JOGÁVEL** também pode realizar várias **MISSÕES** que têm como atributos um identificador (único), a experiência e a quantia de dinheiro que dará para o Jogável ao ser concluída. Uma **MISSÃO** também pode se relacionar com um único item, isto é, dará ao **JOGÁVEL** um item **ÚNICO** ao ser concluída.

Os itens mencionados anteriormente são classificados pela entidade **ITEM** que tem como atributo um identificador e status que são divididos em: resistência e dano. Um **ITEM** deve obrigatoriamente ser **COMUM** ou **ÚNICO**, onde o item **COMUM** possui como atributo um preço a ser vendido pelo **MERCADOR** para o **JOGÁVEL**; e o item **ÚNICO** possui como atributos uma bonificação nos atributos de resistência e dano, além de obrigatoriamente se relacionar com apenas uma **MISSÃO**.

➢ **DIAGRAMA ER:**



➢ **DICIONÁRIO DE DADOS DO MODELO ER:**

Tipo Entidade	PERSONAGEM		
Descrição	Conjunto de todos os personagens que fazem parte do jogo de RPG		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
IdPersonagem	Identificação do personagem	Inteiro	N
Nome	Nome do personagem	Texto(40)	N
Idade	Idade do personagem	Inteiro(3) Positivo	N
Raça	Qual a raça do personagem	Texto(20)	N
LocalOrigem	Local de origem do personagem	Texto(20)	S

Tipo Entidade	PERSONAGEM -> NÃO JOGÁVEL		
Descrição	Faz parte do jogo apenas como auxiliador no desenvolvimento da história, não está disponível para jogar		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
História	Guarda as vivências do personagem não jogável	Texto(200)	N

Tipo Entidade	PERSONAGEM -> NÃO JOGÁVEL -> MERCADOR		
Descrição	Um personagem não jogável responsável pela negociação de itens e recursos ou um narrador da história		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**

Tipo Entidade	PERSONAGEM -> JOGÁVEL		
Descrição	Uma subclasse de personagem que identifica os personagens jogáveis, ou seja, aqueles que são controlados pelos jogadores		
ATRIBUTOS			
Saldo	Saldo que o personagem jogável possui para a compra de itens	Decimal(8,2) Positivo	N
XpJogador	Experiência do personagem jogável	Inteiro(5) Positivo	N
Nível	Atributo que mostra o nível do personagem jogável, é derivado, pois através de XpJogador é possível definir o nível do personagem	Inteiro(2) Positivo	N
Resistência	Resistência do personagem jogável ao receber dano	Inteiro(2) Positivo	N
Furtividade	Capacidade do personagem jogável não ser detectado por outros personagens	Inteiro(2) Positivo	N
Precisão	Taxa de acerto do personagem jogável ao realizar um ataque	Inteiro(2) Positivo	N
Magia	Quantidade de magia do personagem jogável	Inteiro(2) Positivo	N
Dano	Quantidade de dano físico do personagem jogável	Inteiro(2) Positivo	N

Tipo Entidade	CLASSE		
Descrição	Confere aumento de determinado status de acordo com a especialidade selecionada pelo jogador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
IdClasse	Identificação da classe	Inteiro	N

Tipo Entidade	CLASSE -> LADRÃO		
Descrição	Disponibiliza mais furtividade ao jogador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
+Furtividade	Aumento no atributo furtividade	Inteiro(2) Positivo	N

Tipo Entidade	CLASSE -> ARQUEIRO		
Descrição	Disponibiliza mais precisão ao jogador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
+Precisão	Aumento no atributo precisão	Inteiro(2) Positivo	N

Tipo Entidade	CLASSE -> MAGO		
Descrição	Disponibiliza mais magia ao jogador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
+Magia	Aumento do atributo magia	Inteiro(2) Positivo	N

Tipo Entidade	CLASSE -> ASSASSINO		
Descrição	Disponibiliza mais dano ao jogador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
+Dano	Aumento no atributo dano	Inteiro(2) Positivo	N

Tipo Entidade	CLASSE -> GUERREIRO		
Descrição	Disponibiliza mais resistência ao jogador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
+Resistência	Aumento no atributo resistência	Inteiro(2) Positivo	N

Tipo Entidade	ITEM		
Descrição	Pode ser comprado ou ganho pelo jogador. Fornece aumento de dano e/ou durabilidade		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
IdItem	Cadastro do item	Inteiro	N
ResistênciaItem	Resistência do item	Inteiro(2) Positivo	N
DanoItem	Dano do item	Inteiro(2) Positivo	N

Tipo Entidade	ITEM -> ÚNICO		
Descrição	Item fornecido ao jogador apenas sob conclusão de missão		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
+DanoItem	Atribui mais dano comparado aos itens comuns	Inteiro(2) Positivo	N
+ResistênciaItem	Atribui mais resistência comparado aos itens comuns	Inteiro(2) Positivo	N

Tipo Entidade	ITEM -> COMUM		
Descrição	Item possível de comprar pelo jogador através do mercador		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
Preço	Preço que o mercador vende para o jogável	Real(6,2) Positivo	N

Tipo Entidade	MISSÃO		
Descrição	Tarefas que podem ser realizadas pelos jogadores, se realizadas, geram bonificações poderosas		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**
IdMissão	Identificação da missão	Inteiro	N
XpMissão	Experiência obtida pelo jogável ao realizar uma missão	Inteiro(3) Positivo	N
Dinheiro	Dinheiro obtido pelo jogável ao realizar uma missão	Real(6,2) Positivo	N

Tipo Relacionamento	Especializa em		
Descrição	Indica em qual classe o personagem jogável se especializou		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**

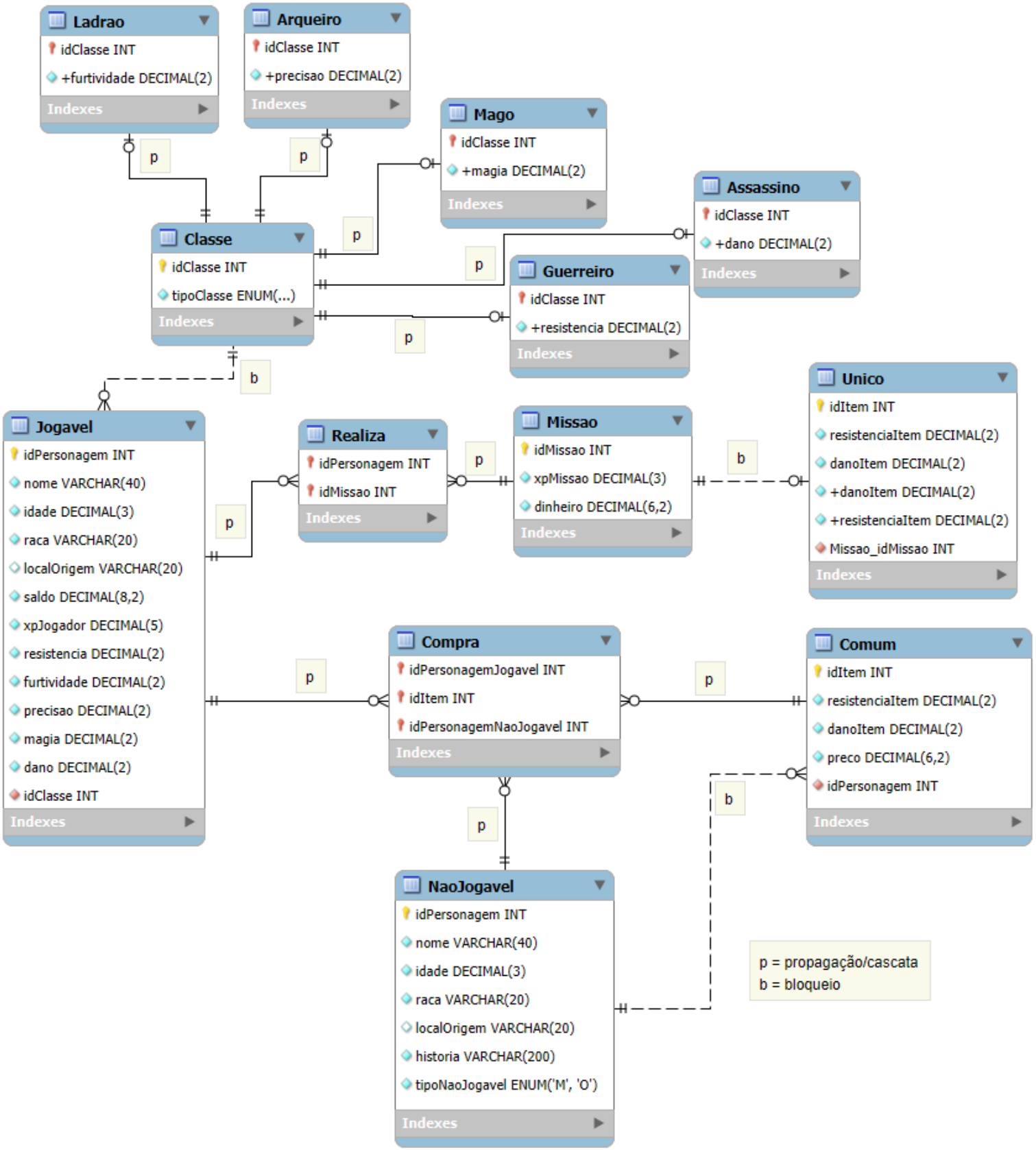
Tipo Relacionamento	Realiza		
Descrição	Indica as missões que o personagem jogável realiza		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**

Tipo Relacionamento	Obtém		
Descrição	Indica que ao completar uma missão ela poderá disponibilizar para o personagem jogável um item único. Ou que um item único deve ser obtido ao realizar uma missão		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**

Tipo Relacionamento	Compra		
Descrição	Indica que personagens jogáveis podem adquirir itens comuns através da compra com mercadores		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**

Tipo Relacionamento	Guarda_Estoque		
Descrição	Estabelece que um item comum deve fazer fazer parte do estoque de um mercador e que um mercador pode guardar vários itens comuns no seu estoque		
ATRIBUTOS			
Nome	**Descrição**	**Domínio**	**Permite nulo ? (S/N)**

➢ **DIAGRAMA RELACIONAL**



➢ **DICIONÁRIO DE DADOS DO MODELO RELACIONAL:**

Tabela	Personagem -> Jogavel
Descrição	Conjunto de todos os personagens jogáveis que fazem parte do RPG
Atributos	
Nome	**Descrição**
idPersonagem	Identificação do personagem (PK)
nome	Nome do personagem
idade	Idade do personagem
raca	Raça do personagem
localOrigem	Local de origem do personagem
saldo	Saldo que o personagem jogável possui para a compra de itens
xpJogador	Experiência do personagem jogável
resistencia	Resistência do personagem jogável ao receber dano
furtividade	Capacidade do personagem jogável de não ser detectado por outros personagens
precisao	Taxa de acerto do personagem jogável ao realizar um ataque
magia	Quantidade de magia do personagem jogável
dano	Quantidade de dano físico do personagem jogável
idClasse	Identificação da classe (FK)

Tabela	Personagem -> NaoJogavel
Descrição	Faz parte do jogo apenas como auxiliador no desenvolvimento da história, não está disponível para jogar
Atributos	
Nome	**Descrição**
idPersonagem	Identificação do personagem (PK)
nome	Nome do personagem
idade	Idade do personagem
raca	Raça do personagem
localOrigem	Local de origem do personagem
historia	Guarda as vivências do personagem não jogável
tipoNaoJogavel	Caracteriza o tipo de personagem não jogável entre “mercador”(‘M’) ou “outro”(‘O’)

Tabela	Classe
Descrição	Confere status de acordo com a especialidade selecionada pelo jogador
Atributos	
Nome	**Descrição**
IdClasse	Identificação da classe (PK)
tipoClasse	Identifica o tipo da classe

Tabela	Classe -> Ladrao
Descrição	Disponibiliza furtividade
Atributos	
Nome	**Descrição**
idClasse	Identificação da classe (FK)
+furtividade	Aumento no atributo furtividade

Tabela	Classe -> Arqueiro
Descrição	Disponibiliza precisão
Atributos	
Nome	**Descrição**
idClasse	Identificação da classe (FK)
+precisao	Aumento no atributo precisão

Tabela	Classe -> Mago
Descrição	Disponibiliza magia
Atributos	
Nome	**Descrição**
idClasse	Identificação da classe (FK)
+magia	Aumento no atributo magia

Tabela	Classe -> Assassino
Descrição	Disponibiliza dano
Atributos	
Nome	**Descrição**
idClasse	Identificação da classe (FK)
+dano	Aumento no atributo dano

Tabela	Classe -> Guerreiro
Descrição	Disponibiliza resistência
Atributos	
Nome	**Descrição**
idClasse	Identificação da classe (FK)
+resistencia	Aumento no atributo resistência

Tabela	Missao
Descrição	Tarefas que podem ser realizadas pelos personagens jogáveis e que se realizadas geram bonificações poderosas
Atributos	
Nome	**Descrição**
idMissao	Identificação da missão (PK)
XpMissao	Experiência obtida pelo jogável ao realizar uma missão
dinheiro	Dinheiro obtido pelo jogável ao realizar uma missão

Tabela	Realiza
Descrição	Indica as missões que o personagem jogável realiza
Atributos	
Nome	**Descrição**
IdPersonagem	Identificação do personagem (FK)
IdMissao	Identificação da missão (FK)

Tabela	Item -> Unico
Descrição	Item fornecido ao jogador apenas sob conclusão de missão
Atributos	
Nome	**Descrição**
IdItem	Cadastro do item (PK)
resistenciaItem	Resistência do item
danoItem	Dano do item
+danoItem	Atribui mais dano comparado aos itens comuns
+resistenciaItem	Atribui mais resistência comparado aos itens comuns
idMissao	Identificação da missão (FK)

Tabela	Compra
Descrição	Indica que personagens jogáveis podem adquirir itens comuns através da compra com mercadores
Atributos	
Nome	**Descrição**
idPersonagemJogavel	Identificação do personagem jogável (FK)
IdItem	Identificação do item (FK)
idPersonagemNaoJogavel	Identificação do personagem não jogável (FK)

Tabela	Item -> Comum
Descrição	Item adquirido pelo jogador por compra
Atributos	
Nome	**Descrição**
IdItem	Identificação do item (PK)
resistenciaItem	Resistência do item
danoItem	Dano do item
preco	Valor do item que será negociado pelo mercador.
idPersonagem	Identificação do personagem (FK)

➢ **CÓDIGO SQL:**

a) Criação de todas as tabelas e de todas as restrições de integridade. Todas as restrições de chave (`PRIMARY KEY`) e de integridade referencial (`FOREIGN KEY`) devem ser criadas. Além disso, crie pelo menos um exemplo com cada uma das restrições `UNIQUE` e `DEFAULT`.

```
-- -----------------------------------------------------
-- Schema projetoISBD
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `projetoISBD` DEFAULT CHARACTER SET utf8 ;
USE `projetoISBD` ;

-- -----------------------------------------------------
-- Table `projetoISBD`.`Classe`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Classe` (
  `idClasse` INT NOT NULL AUTO_INCREMENT,
  `tipoClasse` ENUM('LAD', 'ARQ', 'MAG', 'ASS', 'GUE') NOT NULL,
  PRIMARY KEY (`idClasse`));


-- -----------------------------------------------------
-- Table `projetoISBD`.`Jogavel`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Jogavel` (
  `idPersonagem` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(40) UNIQUE NOT NULL,
  `idade` DECIMAL(3) NOT NULL,
  `raca` VARCHAR(20) NOT NULL,
  `localOrigem` VARCHAR(20) NULL,
  `saldo` DECIMAL(8,2) DEFAULT 0.00 NOT NULL,
  `xpJogador` DECIMAL(5) DEFAULT 0.00 NOT NULL,
  `resistencia` DECIMAL(2) NOT NULL,
  `furtividade` DECIMAL(2) NOT NULL,
  `precisao` DECIMAL(2) NOT NULL,
  `magia` DECIMAL(2) NOT NULL,
  `dano` DECIMAL(2) NOT NULL,
  `idClasse` INT NOT NULL,
  PRIMARY KEY (`idPersonagem`),
  CONSTRAINT `fk_Jogavel_Classe1`
    FOREIGN KEY (`idClasse`)
    REFERENCES `projetoISBD`.`Classe` (`idClasse`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION);
```

```sql
-- -----------------------------------------------------
-- Table `projetoISBD`.`NaoJogavel`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`NaoJogavel` (
  `idPersonagem` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(40) UNIQUE NOT NULL,
  `idade` DECIMAL(3) NOT NULL,
  `raca` VARCHAR(20) NOT NULL,
  `localOrigem` VARCHAR(20) NULL,
  `historia` VARCHAR(200) NOT NULL,
  `tipoNaoJogavel` ENUM('M', 'O') NOT NULL,
  PRIMARY KEY (`idPersonagem`));


-- -----------------------------------------------------
-- Table `projetoISBD`.`Missao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Missao` (
  `idMissao` INT NOT NULL AUTO_INCREMENT,
  `xpMissao` DECIMAL(3) NOT NULL,
  `dinheiro` DECIMAL(6,2) DEFAULT 0.00 NOT NULL, -- DECIMAL(6,2) permite
valores até 9999.99
  PRIMARY KEY (`idMissao`));


-- -----------------------------------------------------
-- Table `projetoISBD`.`Realiza`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Realiza` (
  `idPersonagem` INT NOT NULL,
  `idMissao` INT NOT NULL,
  PRIMARY KEY (`idPersonagem`, `idMissao`),
  CONSTRAINT `fk_Jogavel_has_Missao_Jogavel`
    FOREIGN KEY (`idPersonagem`)
    REFERENCES `projetoISBD`.`Jogavel` (`idPersonagem`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Jogavel_has_Missao_Missao1`
    FOREIGN KEY (`idMissao`)
    REFERENCES `projetoISBD`.`Missao` (`idMissao`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Unico`
```

```sql
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Unico` (
  `idItem` INT NOT NULL AUTO_INCREMENT,
  `resistencia` DECIMAL(2) NOT NULL,
  `dano` DECIMAL(2) NOT NULL,
  `+dano` DECIMAL(2) NOT NULL,
  `+resistencia` DECIMAL(2) NOT NULL,
  `Missao_idMissao` INT NOT NULL,
  PRIMARY KEY (`idItem`),
  CONSTRAINT `fk_Unico_Missao1`
    FOREIGN KEY (`Missao_idMissao`)
    REFERENCES `projetoISBD`.`Missao` (`idMissao`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Comum`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Comum` (
  `idItem` INT NOT NULL AUTO_INCREMENT,
  `resistencia` DECIMAL(2) NOT NULL,
  `dano` DECIMAL(2) NOT NULL,
  `preco` DECIMAL(6,2) NOT NULL, -- Alterado para permitir valores até
9999.99
  `idPersonagem` INT NOT NULL,
  PRIMARY KEY (`idItem`),
  CONSTRAINT `fk_Comum_NaoJogavel1`
    FOREIGN KEY (`idPersonagem`)
    REFERENCES `projetoISBD`.`NaoJogavel` (`idPersonagem`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Compra`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Compra` (
  `idPersonagemJogavel` INT NOT NULL,
  `idItem` INT NOT NULL,
  `idPersonagemNaoJogavel` INT NOT NULL,
  PRIMARY KEY (`idPersonagemJogavel`, `idItem`,
`idPersonagemNaoJogavel`),
  CONSTRAINT `fk_jogavel`
    FOREIGN KEY (`idPersonagemJogavel`)
    REFERENCES `projetoISBD`.`Jogavel` (`idPersonagem`)
```

```
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_comum`
    FOREIGN KEY (`idItem`)
    REFERENCES `projetoISBD`.`Comum` (`idItem`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Compra_NaoJogavel1`
    FOREIGN KEY (`idPersonagemNaoJogavel`)
    REFERENCES `projetoISBD`.`NaoJogavel` (`idPersonagem`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Ladrao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Ladrao` (
  `idClasse` INT NOT NULL,
  `+furtividade` DECIMAL(2) NOT NULL,
  PRIMARY KEY (`idClasse`),
  CONSTRAINT `fk_Ladrao_Classe1`
    FOREIGN KEY (`idClasse`)
    REFERENCES `projetoISBD`.`Classe` (`idClasse`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Arqueiro`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Arqueiro` (
  `idClasse` INT NOT NULL,
  `+precisao` DECIMAL(2) NOT NULL,
  PRIMARY KEY (`idClasse`),
  CONSTRAINT `fk_Arqueiro_Classe1`
    FOREIGN KEY (`idClasse`)
    REFERENCES `projetoISBD`.`Classe` (`idClasse`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Mago`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Mago` (
```

```
  `idClasse` INT NOT NULL,
  `+magia` DECIMAL(2) NOT NULL,
  PRIMARY KEY (`idClasse`),
  CONSTRAINT `fk_Mago_Classe1`
    FOREIGN KEY (`idClasse`)
    REFERENCES `projetoISBD`.`Classe` (`idClasse`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Assassino`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Assassino` (
  `idClasse` INT NOT NULL,
  `+dano` DECIMAL(2) NOT NULL,
  PRIMARY KEY (`idClasse`),
  CONSTRAINT `fk_Assassino_Classe1`
    FOREIGN KEY (`idClasse`)
    REFERENCES `projetoISBD`.`Classe` (`idClasse`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);


-- -----------------------------------------------------
-- Table `projetoISBD`.`Guerreiro`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `projetoISBD`.`Guerreiro` (
  `idClasse` INT NOT NULL,
  `+resistencia` DECIMAL(2) NOT NULL,
  PRIMARY KEY (`idClasse`),
  CONSTRAINT `fk_Guerreiro_Classe1`
    FOREIGN KEY (`idClasse`)
    REFERENCES `projetoISBD`.`Classe` (`idClasse`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION);
```

b) Exemplos de `ALTER TABLE` (pelo menos 3 exemplos, envolvendo alterações diversas) e `DROP TABLE`. Crie uma tabela extra que não existe no seu trabalho, somente para exemplificar, e a apague no final com o `DROP TABLE`.

```sql
-- Exemplo 1: Adicionando uma nova coluna
ALTER TABLE Jogavel ADD COLUMN nivel INT DEFAULT 1;

-- Exemplo 2: Modificando o tipo de uma coluna
ALTER TABLE Jogavel MODIFY COLUMN nivel DECIMAL(3,2);

-- Exemplo 3: Removendo uma coluna
ALTER TABLE Jogavel DROP COLUMN nivel;

-- Criando uma tabela extra para exemplo
CREATE TABLE ExemploExtra (
      IdExemplo INT PRIMARY KEY,
      descricao VARCHAR(100)
);

-- Excluindo a tabela extra
DROP TABLE ExemploExtra;
```

c) Exemplos de inserções de dados em cada uma das tabelas (pelo menos 5 em cada tabela). Para testar bem o trabalho, recomenda-se inserir dezenas de registros em cada tabela.

```
-- Inserindo classes base
INSERT INTO Classe (tipoClasse) VALUES
('GUE'), -- Guerreiro
('ARQ'), -- Arqueiro
('MAG'), -- Mago
('ASS'), -- Assassino
('LAD'); -- Ladrão


-- Especializações
INSERT INTO Guerreiro (idClasse, `+resistencia`) VALUES (1, 5);
INSERT INTO Arqueiro (idClasse, `+precisao`) VALUES (2, 6);
INSERT INTO Mago (idClasse, `+magia`) VALUES (3, 7);
INSERT INTO Assassino (idClasse, `+dano`) VALUES (4, 8);
INSERT INTO Ladrao (idClasse, `+furtividade`) VALUES (5, 6);


-- EXEMPLOS DE INSERÇÃO
INSERT INTO Jogavel (nome, idade, raca, localOrigem, saldo, xpJogador,
resistencia, furtividade, precisao, magia, dano, idClasse) VALUES
('Aragorn', 87, 'Dúnadan', 'Vales do Anduin', 500.00, 200, 9, 4, 8, 2, 7,
1),      -- Guerreiro (idClasse 1)
('Legolas', 293, 'Elfo', 'Floresta das Trevas', 300.00, 180, 5, 7, 10, 3,
6, 2),   -- Arqueiro (idClasse 2)
('Arya Stark', 18, 'Humana', 'Winterfell', 150.00, 150, 6, 10, 9, 1, 8,
4),        -- Assassino (idClasse 4)
('Artorias', 200, 'Cavaleiro', 'Abismo', 0.00, 400, 10, 2, 8, 3, 9, 1),
-- Guerreiro
('Lautrec', 38, 'Assassino', 'Carim', 300.00, 200, 5, 9, 7, 1, 8, 4),
-- Assassino
('Radahn', 150, 'Demigodo', 'Caelid', 5000.00, 500, 10, 2, 9, 4, 10, 1),
-- Guerreiro
('Ranni', 120, 'Feiticeira', 'Lunaris', 2000.00, 300, 3, 8, 7, 10, 4, 3),
-- Mago (idClasse 3)
('Blaidd', 90, 'Meio-Lobo', 'Lands Between', 800.00, 250, 8, 6, 8, 2, 7,
1),       -- Guerreiro
('Boromir', 40, 'Humano', 'Gondor', 200.00, 150, 8, 3, 7, 1, 6, 5),
-- Ladrão (idClasse 5)
('Faramir', 35, 'Humano', 'Gondor', 180.00, 130, 6, 5, 9, 2, 5, 2),
-- Arqueiro
('Galadriel', 300, 'Elfo', 'Lothlórien', 500.00, 400, 4, 7, 8, 9, 3, 3),
-- Mago
```

```
('Geralt', 98, 'Bruxo', 'Kaer Morhen', 800.00, 300, 8, 7, 9, 6, 9, 4);
-- Assassino


-- NPCs (NaoJogavel):
INSERT INTO NaoJogavel (nome, idade, raca, localOrigem, historia,
tipoNaoJogavel) VALUES
('Corvo de Três Olhos', 100, 'Corvo Místico', 'Westeros', 'Guardião da
memória do reino', 'M'),  -- Mercador (M)
('Ferreiro Tobho', 60, 'Humano', 'Kings Landing', 'Forja armas com aço
valiriano', 'O'),         -- Outro (O)
('Andre of Astora', 200, 'Gigante', 'Astora', 'Ferreiro lendário de
Lordran', 'O'),
('Quelana', 143, 'Filha do Caos', 'Izalith', 'Mestra do de fogo', 'M'),
('Siegmeyer', 50, 'Humano', 'Catarina', 'Cavaleiro da cebola', 'O'),
('Quelaag', 500, 'Filha do Caos', 'Izalith', 'Irmã de Quelana', 'M'),
('Miriel', 500, 'Tartaruga', 'Ter. Intermédias', 'Sacerdote das Vows',
'M'),
('Thops', 30, 'Humano', 'Raya Lucaria', 'Estudioso de barreiras mágicas',
'O'),
('Treebeard', 125, 'Ent', 'Fangorn', 'Guardião das árvores', 'M'),
('Elrond', 650, 'Meio-Elfo', 'Valfenda', 'Senhor de Valfenda', 'M'),
('Vesemir', 200, 'Bruxo', 'Kaer Morhen', 'Mestre dos bruxos', 'M'),
('Zoltan', 85, 'Anão', 'Mahakam', 'Mercador de armas', 'O');


-- Missões:
INSERT INTO Missao (xpMissao, dinheiro) VALUES
(100, 200.00),
(200, 500.00),
(150, 300.00),
(150, 300.00),
(300, 800.00),
(250, 600.00),
(180, 400.00),
(120, 250.00),
(80, 150.00),
(100, 200.00),
(80, 150.00),
(90, 100.00),
(70, 80.00);


-- Personagens realizando missões:
INSERT INTO Realiza (idPersonagem, idMissao) VALUES
(1, 3), -- Aragorn completa a Missão 3
```

```sql
(1, 1), -- Aragorn completa a Missão 1
(11, 13),-- Galadriel completa a Missão 13
(2, 11), -- Legolas completa a Missão 11
(3, 12), -- Arya Stark completa a Missão 12
(5, 6), -- Lautrec completa a Missão 6
(5, 1), -- Lautrec completa a Missão 1
(7, 2), -- Ranni completa a Missão 2
(6, 7), -- Radahn completa a Missão 7
(8, 8), -- Blaidd completa a Missão 8
(3, 9), -- Arya Stark completa a Missão 9
(9, 2), -- Boromir completa a Missão 2
(10, 4); -- Faramir completa a Missão 4


-- Itens Únicos (Unico):
INSERT INTO Unico (resistencia, dano, `+dano`, `+resistencia`,
Missao_idMissao) VALUES
(25, 30, 12, 8, 3), -- Relacionado à missão 3
(10, 5, 3, 20, 4), -- Relacionado à missão 4
(15, 10, 5, 15, 2), -- Relacionado à missão 2
(18, 20, 8, 10, 5), -- Relacionado à missão 5
(5, 3, 2, 5, 6), -- Relacionado à missão 6
(20, 25, 10, 5, 3), -- Relacionado à missão 3
(12, 18, 7, 6, 7), -- Relacionado à missão 7
(8, 12, 5, 4, 8); -- Relacionado à missão 8


-- ------------------------------------------------------------
-- Trigger para verificar se o personagem não jogável inserido na tabela
Comum
-- é realmente de um mercador
-- ------------------------------------------------------------
DELIMITER //
CREATE TRIGGER VerificarVendedorMercador
BEFORE INSERT ON Comum
FOR EACH ROW
BEGIN
    DECLARE tipoNPC ENUM('M', 'O');  -- Tipo do personagem não jogável

    -- Obtém o tipo do personagem não jogável
    SELECT tipoNaoJogavel INTO tipoNPC
    FROM projetoISBD.NaoJogavel
    WHERE idPersonagem = NEW.idPersonagem;

    -- Verifica se o personagem não é um mercador ('M')
    IF NOT tipoNPC = 'M' THEN
```

```
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Erro: Apenas mercadores podem vender itens';
    END IF;
END //
DELIMITER ;

-- Itens Comuns (Comum):
INSERT INTO Comum (resistencia, dano, preco, idPersonagem) VALUES
(5, 8, 50.00, 1), -- Relacionado a Corvo de Três Olhos
(8, 12, 80.00, 4), -- Relacionado a Quelana
(6, 4, 40.00, 6), -- Relacionado a Quelaag
(7, 10, 60.00, 7), -- Relacionado a Miriel
(4, 40, 75.00, 9), -- Relacionado a Treebeard
(9, 20, 35.00, 10); -- Relacionado a Elrond


-- ------------------------------------------------------------
-- Trigger para verificar se o personagem não jogável inserido na tabela Compra
-- é realmente de um mercador ou se ele está relacionado ao item
-- ------------------------------------------------------------
DELIMITER //
CREATE TRIGGER VerificarMercadorEEstoque
BEFORE INSERT ON Compra
FOR EACH ROW
BEGIN
    DECLARE tipoNPC ENUM('M', 'O');  -- Tipo do personagem não jogável (mercador ou outro)
    DECLARE itemExiste INT;  -- Variável para verificar se o item está no estoque

    -- Verifica se o personagem não jogável é um mercador
    SELECT tipoNaoJogavel INTO tipoNPC
    FROM projetoISBD.NaoJogavel
    WHERE idPersonagem = NEW.idPersonagemNaoJogavel;

    -- Se o personagem não for um mercador, bloqueia a compra
    IF NOT tipoNPC = 'M' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Erro: Apenas mercadores podem vender itens';
    END IF;

    -- Verifica se o mercador tem o item em estoque
    SELECT COUNT(*) INTO itemExiste
    FROM projetoISBD.Comum
    WHERE idItem = NEW.idItem AND idPersonagem =
```

```
NEW.idPersonagemNaoJogavel;

    -- Se o item não estiver no estoque do mercador, bloqueia a compra
    IF itemExiste = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Erro: O mercador não possui esse item em
estoque';
    END IF;
END //
DELIMITER ;

-- Transações (Compra):
INSERT INTO Compra (idPersonagemJogavel, idItem, idPersonagemNaoJogavel)
VALUES
(1, 1, 1),   -- Jogavel 1 (Aragorn) compra Comum 1 de NPC 1 (Corvo de
Três Olhos)
(9, 3, 6),   -- Jogavel 9 (Boromir) compra Comum 3 de NPC 6 (Quelaag)
(4, 2, 4),   -- Jogavel 4 (Artorias) compra Comum 2 de NPC 4 (Quelana)
(6, 4, 7),   -- Jogavel 6 (Radahn) compra Comum 4 de NPC 7 (Miriel)
(2, 5, 9),   -- Jogavel 2 (Legolas) compra Comum 5 de NPC 9 (Treebeard)
(10, 4, 7);  -- Jogavel 10 (Faramir) compra Comum 7 de NPC 7 (Miriel)


-- EXEMPLOS DE RESTRIÇÃO
-- Exemplo (Trigger): Relacionamento com personagens não jogáveis do tipo
Outro ('O')
INSERT INTO Comum (resistencia, dano, preco, idPersonagem) VALUES
(4, 3, 20.00, 8), -- Relacionado a Thops
(3, 5, 30.00, 2), -- Relacionado a Ferreiro Tobho
(10, 15, 100.00, 3), -- Relacionado à Andre of Astora
(5, 8, 500.00, 2), -- Relacionado a Ferreiro Tobho
(2, 1, 10.00, 5); -- Relacionado a Siegmeyer


-- Exemplos (Trigger): Relacionamento com personagens não jogáveis do
tipo Outro ('O')
-- ou que não tenha o item especificado
-- 1 -> Tentativa de inserir um personagem não jogável que seja Outro
('O')
INSERT INTO Compra (idPersonagemJogavel, idItem, idPersonagemNaoJogavel)
VALUES
(8, 8, 8),   -- Jogavel 8 (Blaidd) compra Comum 8 de NPC 8 (Thops) -
Outro
(1, 3, 3),   -- Jogavel 1 (Aragorn) compra Comum 3 de NPC 3 (Andre of
Astora) - Outro
(7, 5, 5),   -- Jogavel 7 (Ranni) compra Comum 5 de NPC 5 (Siegmeyer) -
```

```sql
Outro
(5, 3, 3);  -- Jogavel 5 (Lautrec) compra Comum 3 de NPC 3 (Andre of
Astora) - Outro

-- 2 -> Tentativa de inserir um personagem não jogável que não tenha o
item
INSERT INTO Compra (idPersonagemJogavel, idItem, idPersonagemNaoJogavel)
VALUES
(5, 3, 1); -- Jogavel 5 (Lautrec) compra Comum 3 de NPC 1 (Corvo de Três
Olhos) - Não possui item


-- Exemplo: valor DEFAULT de "saldo"
INSERT INTO Jogavel (nome, idade, raca, localOrigem, xpJogador,
resistencia, furtividade, precisao, magia, dano, idClasse) VALUES
('Solaire', 45, 'Humano', 'Astora', 300, 8, 3, 7, 4, 6, 1);


-- Exemplo: nome não pode ser repetido (UNIQUE)
-- Error Code: 1062. Duplicate entry 'Solaire' for key 'Jogavel.nome'
INSERT INTO Jogavel (nome, idade, raca, localOrigem, xpJogador,
resistencia, furtividade, precisao, magia, dano, idClasse) VALUES
('Solaire', 70, 'Druida', 'Astalavista', 320, 8, 3, 7, 4, 6, 1);
```

d) Exemplos de modificação de dados em 5 tabelas. Mostre pelo menos um exemplo com UPDATE aninhado, envolvendo mais de uma tabela. Não esqueça de descrever o que cada comando faz.

```sql
-- Auxiliares (ver modificações)
SELECT * FROM Jogavel;                                  -- Aux do 1º update
SELECT * FROM Missao;                                   -- Aux do 2º update
SELECT * FROM NaoJogavel;                               -- Aux do 3º update
SELECT * FROM Guerreiro;                                -- Aux do 4º update
SELECT C.idPersonagemJogavel, C.idItem, CO.dano     -- Aux do 5º update
FROM Compra C
INNER JOIN Comum CO ON C.idItem = CO.idItem
WHERE C.idPersonagemJogavel = 1;


-- Atualizar saldo de um jogador
UPDATE Jogavel SET saldo = saldo + 55.55 WHERE IdPersonagem = 1;


-- Atualizar experiência de missão
UPDATE Missao SET xpMissao = xpMissao + 33 WHERE IdMissao = 1;


-- Atualizar tipo de não jogável
UPDATE NaoJogavel SET tipoNaoJogavel = 'O' WHERE IdPersonagem = 1;


-- Atualizar resistência bônus do Guerreiro 1
UPDATE Guerreiro SET `+resistencia` = 10 WHERE IdClasse = 1;


-- Atualização aninhada: Aumentar o dano de todos os itens comprados por
um jogador (desativando o safe updates)
SET SQL_SAFE_UPDATES = 0;
UPDATE Comum SET dano = dano + 1 WHERE idItem IN (SELECT idItem FROM
Compra WHERE IdPersonagemJogavel = 1);
SET SQL_SAFE_UPDATES = 1;

-- Atualização aninhada: Aumentar o dano de todos os itens comprados por
um jogador
UPDATE Comum c
JOIN Compra cp ON c.idItem = cp.idItem
SET c.dano = c.dano + 1
WHERE cp.IdPersonagemJogavel = 1;
```

e) Exemplos de exclusão de dados em 5 tabelas. Mostre pelo menos um exemplo com DELETE aninhado, envolvendo mais de uma tabela. Não esqueça de descrever o que cada comando faz

```
-- Auxiliares (ver modificações)
SELECT C.idPersonagemJogavel, C.idItem, CO.dano    -- Aux do 1° delete
FROM Compra C
INNER JOIN Comum CO ON C.idItem = CO.idItem
WHERE C.idPersonagemJogavel = 1;
SELECT * FROM Realiza;                             -- Aux do 2° delete
SELECT * FROM Jogavel;                             -- Aux do 3° delete
SELECT * FROM Missao;                              -- Aux do 4° delete
SELECT * FROM NaoJogavel;                          -- Aux do 5° e 6°
delete

-- Exclusão aninhada: Excluir todos os itens comprados por um jogador
(desativando o safe updates)
SET SQL_SAFE_UPDATES = 0;
DELETE FROM Comum WHERE idItem IN (SELECT idItem FROM Compra WHERE
IdPersonagemJogavel = 1);
SET SQL_SAFE_UPDATES = 1;

-- Exclusão aninhada: Excluir todos os itens comprados por um jogador
DELETE c
FROM Comum c
JOIN Compra cp ON c.idItem = cp.idItem
WHERE cp.IdPersonagemJogavel = 1;


-- Excluir realizações de missões
DELETE FROM Realiza WHERE IdPersonagem = 1;


-- Excluir um jogador
DELETE FROM Jogavel WHERE IdPersonagem = 1;


-- Excluir uma missão
DELETE FROM Missao WHERE idMissao = 1;


-- Excluir um não jogável com item Comum (ERRO - Restrict)
DELETE FROM NaoJogavel WHERE IdPersonagem = 1;
```

```sql
-- Excluir um não jogável que não possui item Comum
DELETE FROM NaoJogavel WHERE IdPersonagem = 9;
```

f) Exemplos de, pelo menos, 12 consultas. Inclua consultas simples e complexas,envolvendo todas as cláusulas do comando de consulta (`SELECT`, `FROM`, `WHERE`, `ORDER BY`, `GROUP BY`, `HAVING`), os operadores (`JOIN`, `OUTER JOIN`, `UNION`, `AND`, `OR`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `IS NULL`, `ANY`/`SOME`, `ALL`, `EXISTS`), além de funções agregadas e consultas aninhadas (subconsultas). Não faça aninhamentos "forçados", somente os use em situações onde é difícil escrever uma consulta sem aninhamento. Será avaliado o nível de complexidade das consultas apresentadas. Não se esqueça de descrever em detalhes o que cada consulta recupera (ex: recupera o nome e o endereço dos gerentes dos departamentos que controlam os projetos localizados em Lavras).

1. **F1**: consulta com `INNER JOIN`
2. **F2**: consulta com `OUTER JOIN`
3. **F3**: consulta com `ORDER BY`
4. **F4**: consulta com `GROUP BY`
5. **F5**: consulta com `HAVING`
6. **F6**: consulta com `UNION`
7. **F7**: consulta com `IN`
8. **F8**: consulta com `LIKE`
9. **F9**: consulta com `IS NULL`
10. **F10**: consulta com `ANY` ou `SOME`
11. **F11**: consulta com `ALL`
12. **F12**: consulta com `EXISTS`

OBS: Outras consultas a seu critério. Inclua consultas com os operadores `AND`, `OR`, `NOT`, `BETWEEN`, se ainda não apareceram nas consultas de **F1** a **F12**.

```sql
-- F1 - Recupera o nome dos jogadores e o tipo de classe que eles pertencem
SELECT J.nome, C.tipoClasse
FROM Jogavel J
INNER JOIN Classe C ON J.idClasse = C.idClasse;


-- F2 - Recupera todos os jogadores e suas missões, incluindo jogadores sem missões
SELECT J.nome, R.idMissao
FROM Jogavel J
LEFT JOIN Realiza R ON J.idPersonagem = R.idPersonagem;


-- F3 - Recupera os jogadores ordenados por saldo decrescente
SELECT idPersonagem, nome, saldo
FROM Jogavel
```

```sql
ORDER BY saldo DESC;


-- F4 - Recupera a quantidade de jogadores por classe
SELECT idClasse, COUNT(*) AS quantidade
FROM Jogavel
GROUP BY idClasse;

-- F4 (MELHORADA) - Recupera a quantidade de jogadores por classe,
mostrando o tipo de classe (GUE, ARQ, etc.)
SELECT C.tipoClasse, COUNT(*) AS quantidade
FROM Jogavel J
INNER JOIN Classe C ON J.idClasse = C.idClasse
GROUP BY C.tipoClasse;


-- F5 - Recupera as classes com mais de 5 jogadores
SELECT idClasse, COUNT(*) AS quantidade
FROM Jogavel
GROUP BY idClasse
HAVING COUNT(*) > 3;

-- F5 (MELHORADA) - Recupera as classes com mais de 5 jogadores,
mostrando o tipo de classe (GUE, ARQ, etc.)
SELECT C.tipoClasse, COUNT(*) AS quantidade
FROM Jogavel J
INNER JOIN Classe C ON J.idClasse = C.idClasse
GROUP BY C.tipoClasse
HAVING COUNT(*) > 3;


-- F6 - Recupera os nomes de todos os personagens, sejam jogáveis ou não
SELECT nome FROM Jogavel
UNION
SELECT nome FROM NaoJogavel;


-- F7 - Recupera os jogadores que pertencem às classes 1 ou 2
SELECT nome, idClasse
FROM Jogavel
WHERE idClasse IN (1, 2);


-- F8 - Recupera os jogadores cujos nomes começam com 'Ra'
SELECT *
FROM Jogavel
```

```sql
WHERE nome LIKE 'Ra%';


-- F9 - Recupera as missões que ainda não foram realizadas por nenhum
jogador
SELECT M.idMissao, M.xpMissao, M.dinheiro
FROM Missao M
LEFT JOIN Realiza R ON M.idMissao = R.idMissao
WHERE R.idMissao IS NULL;


-- F10 - Recupera os jogadores que têm saldo maior que qualquer saldo dos
jogadores da classe 1
SELECT *
FROM Jogavel
WHERE saldo > ANY (SELECT saldo FROM Jogavel WHERE idClasse = 1);


-- F11 - Recupera os jogadores que têm saldo maior que todos os saldos
dos jogadores da classe 1
SELECT nome, saldo
FROM Jogavel
WHERE saldo >= ALL (SELECT saldo FROM Jogavel WHERE idClasse = 1);


-- F12 - Recupera os jogadores que realizaram pelo menos uma missão
SELECT idPersonagem, nome
FROM Jogavel J
WHERE EXISTS (SELECT * FROM Realiza R WHERE R.idPersonagem =
J.idPersonagem);


-- AND - Recupera os jogadores da classe 1 com saldo maior que 50
SELECT idPersonagem, nome, idClasse, saldo
FROM Jogavel
WHERE idClasse = 1 AND saldo > 50;

-- OR - Recupera os jogadores da classe 1 ou da classe 2
SELECT idPersonagem, nome, idClasse
FROM Jogavel
WHERE idClasse = 1 OR idClasse = 2;

-- NOT - Recupera os jogadores que não pertencem à classe 1
SELECT idPersonagem, nome, idClasse
FROM Jogavel
WHERE NOT idClasse = 1;
```

```sql
-- BETWEEN - Recupera os jogadores com saldo entre 50 e 100
SELECT idPersonagem, nome, saldo
FROM Jogavel
WHERE saldo BETWEEN 0 AND 100;
```

g) Exemplos de criação de 3 visões (*Views*). Inclua também exemplos de como usar cada uma das visões. Não esqueça de descrever o que cada comando faz

```sql
-- Cria uma view que retorna o nome dos jogadores e o tipo de classe que eles pertencem
CREATE VIEW View_Jogadores_Classes AS
SELECT J.nome, C.tipoClasse
FROM Jogavel J
INNER JOIN Classe C ON J.idClasse = C.idClasse;

-- UTILIZANDO A VIEW
-- Seleciona todos os jogadores
SELECT * FROM View_Jogadores_Classes;

-- Seleciona todos os jogadores da classe Ladrão (LAD)
SELECT * FROM View_Jogadores_Classes WHERE tipoClasse = 'LAD';


-- Cria uma view que retorna as missões que ainda não foram realizadas por nenhum jogador
CREATE VIEW View_Missoes_Nao_Realizadas AS
SELECT M.idMissao, M.xpMissao, M.dinheiro
FROM Missao M
LEFT JOIN Realiza R ON M.idMissao = R.idMissao
WHERE R.idMissao IS NULL;

-- UTILIZANDO A VIEW
-- Seleciona todas as missões não realizadas
SELECT * FROM View_Missoes_Nao_Realizadas;

-- Seleciona as missões não realizadas que oferecem mais de 100 pontos de experiência
SELECT * FROM View_Missoes_Nao_Realizadas WHERE xpMissao > 100;


-- Cria uma view que retorna os itens comuns e seus preços, juntamente com o nome do vendedor (personagem não jogável)
CREATE VIEW View_Itens_Comuns_Precos AS
SELECT C.idItem, C.preco, NJ.nome AS vendedor
FROM Comum C
INNER JOIN NaoJogavel NJ ON C.idPersonagem = NJ.idPersonagem;

-- UTILIZANDO A VIEW
-- Seleciona todos os itens comuns
SELECT * FROM View_Itens_Comuns_Precos;
```

```sql
-- Seleciona os itens comuns com preço superior a 75.00
SELECT * FROM View_Itens_Comuns_Precos WHERE preco > 75.00;
```

h) Exemplos de criação de usuários (pelo menos 2), concessão (GRANT) e revocação (REVOKE) de permissão de acesso

```sql
-- Cria um usuário chamado 'admin' com a senha 'admin123'
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin123';

-- Cria um usuário chamado 'leitor' com a senha 'leitor123'
CREATE USER 'leitor'@'localhost' IDENTIFIED BY 'leitor123';

-- Concede todas as permissões (ALL PRIVILEGES) no banco de dados 'projetoISBD' ao usuário 'admin'
GRANT ALL PRIVILEGES ON projetoISBD.* TO 'admin'@'localhost';

-- Concede permissão de leitura (SELECT) no banco de dados 'projetoISBD' ao usuário 'leitor'
GRANT SELECT ON projetoISBD.* TO 'leitor'@'localhost';

-- O usuário 'admin' pode inserir um novo jogador na tabela 'Jogavel'
INSERT INTO Jogavel (nome, idade, raca, localOrigem, saldo, xpJogador, resistencia, furtividade, precisao, magia, dano, idClasse)
VALUES ('NovoJogador', 25, 'Humano', 'SP', 100.00, 0, 10, 5, 7, 6, 8, 1);

-- O usuário 'leitor' pode consultar os jogadores na tabela 'Jogavel'
SELECT * FROM Jogavel;

-- Mostra as permissões concedidas ao usuário 'admin'
SHOW GRANTS FOR 'admin'@'localhost';

-- Mostra as permissões concedidas ao usuário 'leitor'
SHOW GRANTS FOR 'leitor'@'localhost';

-- Revoga a permissão de leitura (SELECT) no banco de dados 'projetoISBD' do usuário 'leitor'
REVOKE SELECT ON projetoISBD.* FROM 'leitor'@'localhost';

-- Revoga todas as permissões (ALL PRIVILEGES) no banco de dados 'projetoISBD' do usuário 'admin'
REVOKE ALL PRIVILEGES ON projetoISBD.* FROM 'admin'@'localhost';
```

i) Exemplos de 3 procedimentos/funções, com e sem parâmetros, de entrada e de saída, contendo alguns comandos tais como `IF`, `CASE WHEN`, `WHILE`, declaração de variáveis e funções prontas. Os procedimentos devem ter aplicação no seu projeto. Apresente exemplos de testes dos procedimentos/funções, mostrando como eles são executados.

```
--
-------------------------------------------------------------------------
-- Classifica uma missão como 'Fácil', 'Média' ou 'Difícil' com base na
experiência (XP) que concede.
--
-------------------------------------------------------------------------

DELIMITER //
CREATE PROCEDURE ClassificarDificuldadeMissao(
    IN aux_idMissao INT,
    OUT dificuldade VARCHAR(20)
)
BEGIN
    DECLARE aux_xpMissao DECIMAL(3);

    -- Obtém o XP da missão
    SELECT xpMissao INTO aux_xpMissao
    FROM Missao
    WHERE idMissao = aux_idMissao;

    -- Classifica a dificuldade usando CASE WHEN
    SET dificuldade = CASE
        WHEN aux_xpMissao <= 50 THEN 'Fácil' -- Classifica a missão como
'Fácil' caso o XP for menor ou igual a 50
        WHEN aux_xpMissao BETWEEN 51 AND 150 THEN 'Média' -- Classifica a
missão como 'Média' caso o XP estiver entre 51 e 150
        ELSE 'Difícil' -- Classifica como 'Difícil' caso nenhum dos casos
anteriores for verdadeiro
    END;
END //
DELIMITER ;

DROP PROCEDURE ClassificarDificuldadeMissao; -- Exclui o procedimento

-- TESTE
SELECT * FROM Missao;

-- Executar o procedimento
CALL ClassificarDificuldadeMissao(3, @dificuldade);
SELECT @dificuldade AS dificuldade; -- Deve retornar "Média"
```

```
CALL ClassificarDificuldadeMissao(2, @dificuldade);
SELECT @dificuldade AS dificuldade; -- Deve retornar "Difícil"



--
-------------------------------------------------------------------------
------
-- Consulta a experiência (xpJogador) do jogador e retorna o nível
calculado.
--
-------------------------------------------------------------------------
------

DELIMITER //
CREATE FUNCTION CalcularNivelJogador(
    aux_idPersonagem INT
)
RETURNS INT
BEGIN
    DECLARE aux_xpJogador DECIMAL(5);  -- Experiência do jogador
    DECLARE nivel INT DEFAULT 1;   -- Nível inicial
    DECLARE xpNecessario INT DEFAULT 100;  -- XP necessário para o
próximo nível

    -- 1. Obter a experiência do jogador
    SELECT xpJogador INTO aux_xpJogador
    FROM Jogavel
    WHERE idPersonagem = aux_idPersonagem;

    -- 2. Calcular o nível usando WHILE
    WHILE aux_xpJogador >= xpNecessario DO
        SET nivel = nivel + 1;  -- Incrementar o nível
        SET aux_xpJogador = aux_xpJogador - xpNecessario;  -- Subtrair XP
gasto
        SET xpNecessario = xpNecessario + 50;  -- Aumentar XP necessário
para o próximo nível
    END WHILE;

    -- 3. Retornar o nível calculado
    RETURN nivel;
END //
DELIMITER ;

DROP FUNCTION CalcularNivelJogador; -- Exclui a função

-- TESTE
```

```sql
SELECT * FROM Jogavel;

-- Calcular o nível do jogador com idPersonagem = 5
SELECT nome, CalcularNivelJogador(5) AS nivel
FROM Jogavel
WHERE idPersonagem = 5; -- Deve retornar nivel = 2

-- Teste chamando só a função
Select CalcularNivelJogador(5);


--
-------------------------------------------------------------------------
------
-- Retorna uma classificação para cada jogador (Jogável) de acordo com a
quantidade
-- da sua experiência (XP)
--
-------------------------------------------------------------------------
------

DELIMITER //
CREATE PROCEDURE ClassificarJogadores()
BEGIN
    SELECT
        idPersonagem,
        nome,
        xpJogador,
        CASE
            WHEN xpJogador >= 350 THEN 'Lendário' -- Classificação para
xp acima ou igual a 350
            WHEN xpJogador >= 200 THEN 'Veterano' -- Classificação para
xp entre 200 e 349
            WHEN xpJogador >= 50 THEN 'Intermediário' -- Classificação
para xp entre 50 e 199
            ELSE 'Iniciante' -- Classificação para xp abaixo de 50
        END AS classificação
    FROM projetoISBD.Jogavel
    ORDER BY xpJogador DESC;
END //
DELIMITER ;

DROP PROCEDURE ClassificarJogadores; -- Exclui o procedimento

-- TESTE
-- Deve retornar uma tabela com todos os personagens jogáveis
```

```
classificados de acordo com sua experiência (XP)
-- Os dados devem estar ordenados do maior para o menor (decrescente), ou
seja, na ordem: 'Lendário', 'Veterano', 'Intermediário' e 'Iniciante'
CALL ClassificarJogadores();


--
------------------------------------------------------------------------
------
-- Retorna uma relação de itens e seus vendedores (estoque)
--
------------------------------------------------------------------------
------

DELIMITER //
CREATE PROCEDURE ListarItensComunsEVendedores()
BEGIN
    SELECT C.idItem, C.resistencia, C.dano, C.preco, N.nome AS
NomeVendedor
    FROM projetoISBD.Comum C
    JOIN projetoISBD.NaoJogavel N ON C.idPersonagem = N.idPersonagem
    ORDER BY N.nome;
END //
DELIMITER ;

-- TESTE
CALL ListarItensComunsEVendedores();
```

j) Exemplos de 3 *triggers*, um para cada evento (inserção, alteração e exclusão). Os *triggers* devem ter aplicação no seu projeto. Apresente exemplos de testes dos *triggers*, mostrando casos em que eles são disparados e casos em que não são disparados.

```
-- ------------------------------------------------
-- Trigger que impede que seja inserido um saldo negativo para o
personagem Jogavel
-- ------------------------------------------------

DELIMITER //
CREATE TRIGGER before_insert_jogavel
BEFORE INSERT ON Jogavel
FOR EACH ROW
BEGIN
    IF NEW.saldo < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'O saldo do personagem não pode ser
negativo.';
    END IF;
END //
DELIMITER ;

-- Teste de Trigger
INSERT INTO Jogavel (nome, idade, raca, localOrigem, saldo, xpJogador,
resistencia, furtividade, precisao, magia, dano, idClasse)
VALUES ('HeróiSombrio', 30, 'Elfo', 'Floresta Encantada', -50.00, 10.00,
5, 4, 6, 3, 7, 1);


-- ------------------------------------------------
-- Trigger que impede redução da experiência (xp) do personagem Jogavel
-- ------------------------------------------------

DELIMITER //
CREATE TRIGGER before_update_xpJogador
BEFORE UPDATE ON Jogavel
FOR EACH ROW
BEGIN
    IF NEW.xpJogador < OLD.xpJogador THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Não é permitido reduzir a experiência do
jogador.';
    END IF;
END //
DELIMITER ;
```

```sql
-- Teste de Trigger
INSERT INTO Jogavel (nome, idade, raca, localOrigem, saldo, xpJogador,
resistencia, furtividade, precisao, magia, dano, idClasse)
VALUES ('GuerreiroLendario', 35, 'Orc', 'Montanhas Sombrias', 200.00,
50.00, 7, 5, 8, 2, 9, 2);

UPDATE Jogavel
SET xpJogador = 30.00
WHERE nome = 'GuerreiroLendario';


-- ------------------------------------------------
-- Trigger que atualiza automaticamente o saldo de um personagem Jogavel
após uma compra
-- ------------------------------------------------

DELIMITER //
CREATE TRIGGER AtualizarSaldoAposCompra
AFTER INSERT ON Compra
FOR EACH ROW
BEGIN
    DECLARE preco_item DECIMAL(6,2);

    -- Obtém o preço do item comprado
    SELECT preco INTO preco_item
    FROM Comum
    WHERE idItem = NEW.idItem;

    -- Atualiza o saldo do jogador que fez a compra
    UPDATE Jogavel
    SET saldo = saldo - preco_item
    WHERE idPersonagem = NEW.idPersonagemJogavel;
END //
DELIMITER ;

-- TESTE
-- Verificando saldo do personagem 1
SELECT idPersonagem, nome, saldo
FROM Jogavel
WHERE idPersonagem = 1;

-- Verificando preço do item Comum 3
SELECT idItem, preco
FROM Comum
WHERE idItem = 3;
```

```sql
-- Realizando Compra com o personagem Não Jogavel 4
INSERT INTO Compra (idPersonagemJogavel, idItem, idPersonagemNaoJogavel)
VALUES (1, 3, 4);
```