SAMUEL AUGUSTO SILVA, MATHEUS ANTONIO PEREIRA MENDES

DESENVOLVIMENTO DE UM SISTEMA DE JOGO RPG COM BANCO DE DADOS MYSQL: MODELAGEM E IMPLEMENTAÇÃO DE FUNCIONALIDADES : ANÁLISE E APLICAÇÃO DE STORED PROCEDURES, TRIGGERS E PAPÉIS NO GERENCIAMENTO DE PERSONAGENS E BATALHAS

[UFU Campus Monte Carmelo]

[2025]

SAMUEL AUGUSTO SILVA, MATHEUS ANTONIO PEREIRA MENDES

DESENVOLVIMENTO DE UM SISTEMA DE JOGO RPG COM BANCO DE DADOS MYSQL: MODELAGEM E IMPLEMENTAÇÃO DE FUNCIONALIDADES: ANÁLISE E APLICAÇÃO DE STORED PROCEDURES, TRIGGERS E PAPÉIS NO GERENCIAMENTO DE PERSONAGENS E BATALHAS

[UFU Campus Monte Carmelo]

[2025]

Desenvolvimento de um Sistema de Jogo RPG com Banco de Dados MySQL: Modelagem e Implementação de Funcionalidades

Samuel Augusto Silva, Matheus Antonio Pereira Mendes[1]

[1]Universidade Federal de Uberlândia- (UFU)

samuel.augusto@ufu.br, matheusantoniomendes@ufu.br

Resumo. *Este artigo apresenta o desenvolvimento de um sistema de banco de dados para um RPG medieval utilizando MySQL, com o objetivo de simular as principais mecânicas de um jogo eletrônico diretamente no banco de dados. O projeto foi concebido para gerenciar personagens, habilidades, itens, inventário e batalhas, implementando operações CRUD, stored procedures, triggers, functions e views conforme os requisitos da disciplina. O sistema demonstra como um banco de dados relacional pode ser utilizado para gerenciar aspectos fundamentais de um RPG, servindo como base para expansões futuras com integração a linguagens de programação.*

Abstract. *This article presents the development of a database system for a medieval RPG using MySQL, with the goal of simulating the main mechanics of a video game directly within the database. The project was designed to manage characters, skills, items, inventory, and battles, implementing CRUD operations, stored procedures, triggers, functions, and views in accordance with the course requirements. The system demonstrates how a relational database can be used to manage key aspects of an RPG, serving as a foundation for future expansions with integration into programming languages.*

1. Informações Gerais

Foram criadas oito tabelas principais, incluindo Classe, Personagem, Habilidade, Item, Inventario, Inimigo e Batalha, com relacionamentos bem definidos para garantir a integridade dos dados. Através de stored procedures como SimularBatalha e ComprarItem, automatizamos processos críticos do jogo, enquanto triggers como AtualizarStatusClasse e NomeUnicoPersonagem asseguraram consistência nas operações. Views como RankingPersonagens e InventarioDetalhado facilitaram a consulta de informações relevantes para os ”jogadores”.

As principais dificuldades incluíram a implementação de lógica complexa de jogo em SQL, como o balanceamento de batalhas e a validação de regras de negócio através de triggers. Algumas funcionalidades idealizadas, como sistema de missões e efeitos temporários de itens, não foram implementadas devido à complexidade que exigiriam em um ambiente puramente SQL. Originalmente, o projeto pretendia incluir mais mecânicas de jogo, mas o escopo foi ajustado para focar nas funcionalidades essenciais que poderiam ser realisticamente implementadas dentro das limitações do MySQL.

2. Concepção e Escopo do Projeto

O projeto foi concebido como uma simulação completa de backend para um RPG medieval, onde todas as mecânicas fundamentais do jogo seriam gerenciadas diretamente pelo banco de dados MySQL. A proposta central consistia em desenvolver um sistema que permitisse aos jogadores criar personagens personalizados com classes distintas, cada uma com atributos únicos como força, defesa, vida e mana. Esses personagens poderiam então embarcar em aventuras, coletando diversos tipos de itens - desde armas e armaduras até poções mágicas - e enfrentando inimigos em batalhas estratégicas que considerariam seus atributos e habilidades.

O grande desafio técnico residia em implementar essas mecânicas de jogo utilizando exclusivamente recursos do MySQL, sem recorrer a linguagens de programação externas. Isso nos levou a explorar intensivamente recursos avançados como stored procedures para a lógica do jogo, triggers para validações automáticas e views para apresentação dos dados. O sistema foi projetado para gerenciar desde a progressão dos personagens através de níveis e experiência até a economia básica do jogo, com compra e venda de itens.

Dentre as principais mecânicas implementadas, destacam-se o sistema de criação de personagens com atribuição automática de características conforme a classe escolhida, um sistema de combate que calcula danos e recompensas de forma dinâmica, e um inventário completo que gerencia a posse e uso de itens. A arquitetura do banco foi cuidadosamente planejada para garantir que todas essas funcionalidades interagissem de forma coerente, mantendo a integridade dos dados e a consistência das regras do jogo.

Embora algumas mecânicas mais complexas tenham sido simplificadas devido às limitações do ambiente SQL puro - como o sistema de uso de itens em batalha que foi reduzido a compra e venda básica - o projeto demonstrou com sucesso como os principais elementos de um RPG podem ser modelados e implementados diretamente em um banco de dados relacional. Essa abordagem não apenas validou a viabilidade do conceito, mas também abriu possibilidades para expansões futuras que poderiam integrar o banco a uma aplicação completa.

3. Modelagem do Sistema

A modelagem do banco de dados foi concebida com o objetivo de representar as principais mecânicas de um jogo de RPG, simulando aspectos como criação de personagens, batalhas, uso de habilidades, aquisição de itens e controle de inventário. O modelo foi estruturado utilizando o paradigma relacional, com tabelas interligadas por chaves primárias e estrangeiras, garantindo a integridade referencial e permitindo uma organização lógica e eficiente dos dados.

3.1. Diagrama de Entidade-Relacionamento (DER)

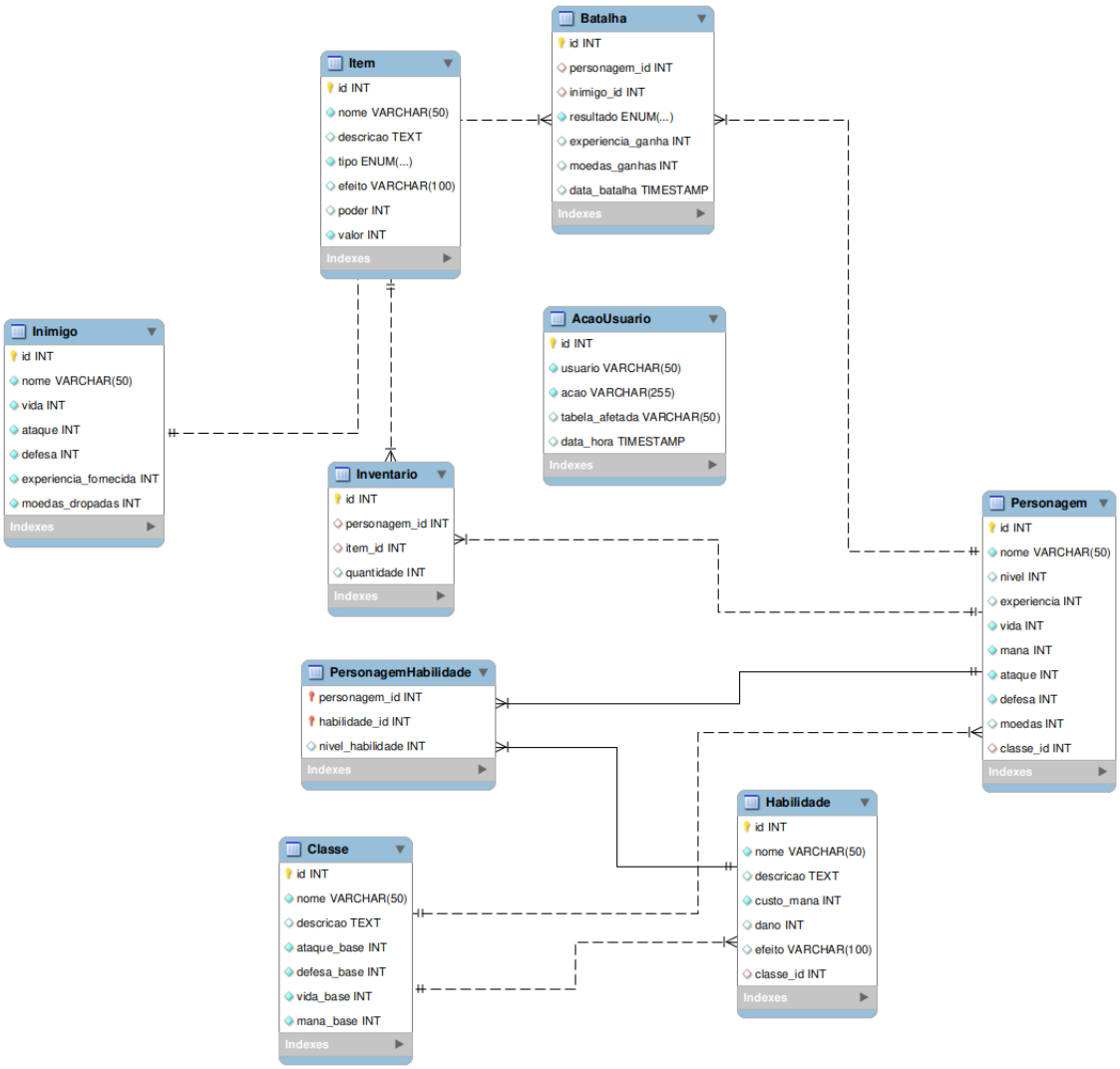

Figura 1: Diagrama de Entidade-Relacionamento

A seguir, apresenta-se uma descrição detalhada das entidades que compõem o modelo:

A modelagem do banco de dados do sistema foi elaborada com base nas principais

necessidades estruturais de um jogo de RPG eletrônico, tendo como objetivo representar de forma relacional as mecânicas fundamentais do gênero. Para isso, foi construído um modelo lógico que abrange as entidades essenciais e seus relacionamentos, permitindo o gerenciamento completo de personagens, classes, habilidades, itens, inimigos, batalhas e ações dos usuários. O diagrama de entidade-relacionamento elaborado orientou toda a estrutura do banco, servindo como referência para a definição das tabelas, chaves primárias, chaves estrangeiras e restrições de integridade.

A entidade Classe representa os arquétipos básicos disponíveis aos jogadores, como Guerreiro ou Mago, e define os atributos iniciais dos personagens, incluindo ataque, defesa, vida e mana base. Essa entidade estabelece uma relação de um para muitos com a entidade Habilidade, que registra técnicas específicas associadas a cada classe. Cada habilidade possui atributos como custo de mana, dano, descrição textual e um efeito adicional, o que possibilita a diferenciação entre as estratégias de combate utilizadas por personagens de classes distintas.

A entidade Personagem constitui o núcleo do sistema, armazenando informações sobre cada jogador, como nome, nível, experiência acumulada, pontos de vida e mana, atributos de combate (ataque e defesa), quantidade de moedas disponíveis e a classe à qual pertence. A ligação entre personagem e classe é de muitos para um, visto que diversos personagens podem compartilhar a mesma classe base. Além disso, os personagens podem adquirir diversas habilidades ao longo da sua evolução, o que motivou a criação da tabela associativa PersonagemHabilidade, responsável por registrar o nível de proficiência de cada personagem em relação às habilidades que aprendeu.

A aquisição e o gerenciamento de itens foram implementados por meio da entidade Item, que define o nome, descrição, tipo (por exemplo, consumível, equipamento, etc.), efeito, poder e valor de cada objeto. Para representar a posse de itens pelos personagens, foi criada a entidade Inventario, que estabelece um relacionamento muitos-para-muitos entre Personagem e Item, armazenando também a quantidade de cada item em posse de um jogador.

Outro aspecto central da modelagem é a entidade Inimigo, que representa os adversários enfrentados pelos personagens em batalhas. Cada inimigo é caracterizado por atributos como vida, ataque, defesa, experiência fornecida e moedas que podem ser obtidas em caso de vitória. A entidade Batalha registra os confrontos ocorridos, vinculando personagens e inimigos, bem como o resultado da luta, a experiência e moedas adquiridas, além da data e hora do evento, permitindo manter um histórico detalhado de todas as interações de combate.

A estrutura foi complementada pela entidade AcaoUsuario, responsável por re-

gistrar operações realizadas no banco de dados. Essa tabela de auditoria armazena informações sobre o usuário que realizou a ação, a tabela afetada, o tipo de modificação e o horário da execução, sendo essencial para fins de segurança e rastreamento.

O modelo relacional foi cuidadosamente projetado para garantir a integridade e a consistência dos dados por meio de diversas restrições. Todas as tabelas apresentam chaves primárias bem definidas e chaves estrangeiras para assegurar os vínculos entre as entidades. Valores padrão foram atribuídos a campos como nível e experiência dos personagens, com o objetivo de padronizar o estado inicial de cada novo registro. Além disso, tipos ENUM foram utilizados em campos com domínios restritos, como o tipo de item e o resultado da batalha, promovendo maior controle sobre os dados inseridos.

Para lidar com regras de negócio mais complexas, o projeto incorporou diversas triggers, como a trigger AtualizarStatusClasse, que inicializa os atributos do personagem de acordo com a classe escolhida; NomeUnicoPersonagem, que impede a criação de personagens com nomes duplicados; e VerificarNivelMaximo, que restringe a evolução dos personagens além de um limite predefinido. Esses mecanismos de automação interna foram fundamentais para garantir a validade lógica das operações e a aderência às mecânicas previstas para o jogo.

Por fim, a modelagem foi orientada por princípios de normalização, assegurando a eliminação de redundâncias e a clara separação entre responsabilidades das entidades. Medidas de desempenho, como a criação de índices nas chaves primárias e estrangeiras, foram adotadas para otimizar o tempo de resposta das consultas. A estrutura geral do banco foi concebida com extensibilidade em mente, permitindo futuras expansões como sistemas de missões, lojas, eventos e funcionalidades multijogador. Dessa forma, o modelo desenvolvido oferece uma base sólida e funcional para o gerenciamento de um RPG eletrônico diretamente no MySQL, promovendo uma experiência rica e personalizável a partir da camada de banco de dados.

4. Desenvolvimento Detalhado do Sistema RPG

Nesta seção, detalharemos a implementação técnica do banco de dados do RPG

4.1. Operações CRUD: Tabela Classe

A tabela Classe representa os atributos fundamentais que moldam os personagens do jogo. Cada classe define as características base como ataque, defesa, vida e mana, que são essenciais para o equilíbrio e a identidade dos personagens no universo do RPG. As operações CRUD nessa tabela permitem adicionar novas classes ao sistema, consultar os dados de classes existentes, alterar seus atributos conforme ajustes de balanceamento, e eventualmente remover classes obsoletas ou descontinuadas. Essas ações são comuns em

jogos que evoluem com o tempo, com novas classes sendo introduzidas ou reequilibradas conforme a experiência dos jogadores.

```
0
1  INSERT INTO Item (nome, descricao, tipo, efeito, poder, valor)
2  VALUES ('Poção de Força', 'Aumenta a defesa em 100 pontos', 'consumivel', NULL,
       ↪ 100, 800);
3
4  SELECT * FROM Item;
5
6  SELECT * FROM Item WHERE id = 1;
7
8  UPDATE Item SET valor = 60 WHERE id = 1;
9
10 DELETE FROM Item WHERE id = 21;
```

4.2. Operações CRUD: Tabela Personagem

A tabela Personagem armazena os heróis do jogador, com atributos que evoluem conforme ganham experiência em batalhas e missões. Esse é o núcleo do sistema RPG, pois representa o protagonista da jornada. As operações CRUD nesta tabela são fundamentais para registrar novos personagens, consultar seus atributos e progresso, ajustar atributos manualmente em situações excepcionais (como testes ou correções), ou excluir personagens de contas inativas ou para reinicializações. O vínculo com a tabela Classe garante que cada personagem herde uma base coerente com sua classe.

```
0
1  INSERT INTO Personagem (nome, nivel, experiencia, vida, mana, ataque, defesa,
       ↪ moedas, classe_id)
2  VALUES ('Jack', 0, 1, 100, 50, 10, 10, 100, 1);
3
4  SELECT * FROM Personagem;
5
6  SELECT * FROM Personagem WHERE id = 1;
7
8  -- Atualização de nível (ex: após batalha)
9  UPDATE Personagem SET nivel = 2 WHERE id = 1;
10
11 DELETE FROM Personagem WHERE id = 21;
```

4.3. Operações CRUD: Tabela Item

A tabela Item representa os recursos utilizáveis pelos personagens, incluindo poções, armas, armaduras e itens especiais. Cada item possui atributos como tipo, valor e efeitos,

que impactam diretamente a performance do personagem em jogo. O CRUD dessa tabela permite adicionar novos itens ao universo do RPG (por exemplo, como recompensas ou atualizações), visualizar o catálogo de itens, balancear seus efeitos ou valores, e excluir aqueles que foram substituídos ou considerados desequilibrados.

```
0
1  -- Inserção de item
2  INSERT INTO Item (nome, descricao, tipo, efeito, poder, valor)
3  VALUES ('Poção de Força', 'Aumenta a defesa em 100 pontos', 'consumivel', NULL,
       ↪ 100, 800);
4
5  SELECT * FROM Item;
6
7  SELECT * FROM Item WHERE id = 1;
8
9  UPDATE Item SET valor = 60 WHERE id = 1;
10
11 DELETE FROM Item WHERE id = 21;
```

4.4. Operações CRUD: Tabela Inimigo

A tabela Inimigo representa os adversários encontrados pelos personagens durante o jogo. Cada inimigo possui atributos como vida, ataque, defesa e recompensas em experiência e moedas ao ser derrotado. Esses dados são essenciais para simular os desafios enfrentados pelos jogadores e para manter o equilíbrio entre risco e recompensa. As operações CRUD permitem cadastrar novos inimigos (por exemplo, para missões ou chefes de fase), consultar todos os inimigos existentes, ajustar seus atributos com base em testes de balanceamento ou feedback dos jogadores, e remover inimigos que não são mais utilizados ou foram substituídos.

```
0
1  INSERT INTO Inimigo (nome, vida, ataque, defesa, experiencia_fornecida,
       ↪ moedas_dropadas)
2  VALUES ('Malbog', 250, 50, 50, 350, 380);
3
4  SELECT * FROM Inimigo;
5
6  SELECT * FROM Inimigo WHERE id = 1;
7
8  UPDATE Inimigo SET vida = 110 WHERE id = 1;
9
10 DELETE FROM Inimigo WHERE id = 21;
```

4.5. Operações CRUD: Tabela Habilidade

A tabela Habilidade armazena os poderes especiais que os personagens podem aprender e usar. Cada habilidade possui um custo de mana, um efeito específico e pode estar vinculada a uma classe, reforçando a identidade estratégica de cada arquétipo. O CRUD nesta tabela permite cadastrar novas habilidades à medida que o jogo evolui, consultar habilidades já criadas (inclusive associadas às classes), ajustar efeitos ou custos de mana conforme necessário, e excluir habilidades que não são mais utilizadas ou foram substituídas.

```
0
1  INSERT INTO Habilidade (nome, descricao, custo_mana, dano, efeito, classe_id)
2  VALUES ('O chamado do vazio', 'Conjura um guerreiro esqueleto', 20, 35, 'Dano
       ↪ extra em arqueiros', 22);
3
4  SELECT * FROM Habilidade;
5
6  -- Consulta específica com nome da classe
7  SELECT h.*, c.nome AS classe_nome
8  FROM Habilidade h
9  JOIN Classe c ON h.classe_id = c.id
10 WHERE h.id = 21;
11
12 -- Atualização de dano e custo de mana
13 UPDATE Habilidade
14 SET custo_mana = 30, dano = 40
15 WHERE id = 21;
16
17 -- Exclusão com verificação de dependências
18 DELETE FROM Habilidade
19 WHERE id = 21
20 AND NOT EXISTS (SELECT 1 FROM PersonagemHabilidade WHERE habilidade_id = 21);
```

4.6. Operações CRUD: Tabela PersonagemHabilidade

A tabela PersonagemHabilidade representa a relação entre personagens e suas habilidades, incluindo o nível de domínio de cada habilidade. Esta associação permite uma personalização profunda dos personagens, pois cada um pode evoluir suas habilidades de forma única. O CRUD dessa tabela é utilizado para associar habilidades aos personagens (aprendizado), consultar quais habilidades um personagem possui, melhorar o nível de uma habilidade (treinamento) e remover habilidades em caso de reconfiguração ou troca de estratégia.

```
0
1  -- Associação de habilidade ao personagem
```

```sql
INSERT INTO PersonagemHabilidade (personagem_id, habilidade_id,
    ↪ nivel_habilidade)
VALUES (22, 5, 3);

-- Consulta das habilidades de um personagem
SELECT h.nome, h.descricao, ph.nivel_habilidade
FROM PersonagemHabilidade ph
JOIN Habilidade h ON ph.habilidade_id = h.id
WHERE ph.personagem_id = 22;

-- Consulta dos personagens que possuem uma habilidade específica
SELECT p.nome, p.nivel, ph.nivel_habilidade
FROM PersonagemHabilidade ph
JOIN Personagem p ON ph.personagem_id = p.id
WHERE ph.habilidade_id = 5;

-- Atualização de nível da habilidade
UPDATE PersonagemHabilidade
SET nivel_habilidade = nivel_habilidade + 1
WHERE personagem_id = 22 AND habilidade_id = 3;

-- Exclusão da associação
DELETE FROM PersonagemHabilidade
WHERE personagem_id = 22 AND habilidade_id = 5;
```

4.7. Operações CRUD: Tabela Inventario

A tabela Inventario controla os itens que cada personagem possui, registrando a quantidade de cada item em posse. Esta estrutura é essencial para jogos de RPG que envolvem gestão de recursos, pois permite o uso estratégico de consumíveis, armas e armaduras. As operações CRUD permitem adicionar itens ao inventário (ganhos, compras), consultar os itens de um personagem, atualizar a quantidade (uso ou venda), e remover itens do inventário quando esgotados ou descartados.

```sql


INSERT INTO Inventario (personagem_id, item_id, quantidade)
VALUES (2, 5, 3);

-- Consulta de itens com detalhes
SELECT i.nome, i.tipo, inv.quantidade, i.efeito
FROM Inventario inv
JOIN Item i ON inv.item_id = i.id
WHERE inv.personagem_id = 1;

-- Verificação de quantidade de item específico
```

```sql
SELECT quantidade
FROM Inventario
WHERE personagem_id = 1 AND item_id = 1;

-- Atualização de quantidade
UPDATE Inventario
SET quantidade = 50
WHERE personagem_id = 1 AND item_id = 1;

-- Remoção de item específico
DELETE FROM Inventario
WHERE personagem_id = 2 AND item_id = 5;

-- Remoção em massa de consumíveis
DELETE FROM Inventario
WHERE personagem_id = 1
AND item_id IN (SELECT id FROM Item WHERE tipo = 'consumivel');
```

4.8. Operações CRUD: Tabela Batalha

A tabela Batalha registra os confrontos entre personagens e inimigos. Cada entrada representa uma tentativa (vitória ou derrota), armazenando dados importantes como o inimigo enfrentado, a experiência e moedas obtidas. Esse histórico é crucial para estatísticas, progresso do jogador e mecânicas de evolução. As operações CRUD permitem registrar novas batalhas, consultar o histórico de um personagem, corrigir registros incorretos e excluir dados antigos ou específicos (como todas as derrotas, para recomeçar uma saga).

```sql

-- Registro de nova batalha
INSERT INTO Batalha (personagem_id, inimigo_id, resultado, experiencia_ganha,
    ↪ moedas_ganhas)
VALUES (1, 5, 'vitória', 150, 30);

-- Consulta de batalhas de um personagem
SELECT b.*, i.nome AS inimigo_nome
FROM Batalha b
JOIN Inimigo i ON b.inimigo_id = i.id
WHERE b.personagem_id = 1
ORDER BY b.data_batalha DESC;

-- Estatísticas agregadas de batalha
SELECT
    COUNT(*) AS total_batalhas,
    SUM(CASE WHEN resultado = 'vitória' THEN 1 ELSE 0 END) AS vitorias,
    SUM(CASE WHEN resultado = 'derrota' THEN 1 ELSE 0 END) AS derrotas,
    SUM(experiencia_ganha) AS exp_total,
```

```
    SUM(moedas_ganhas) AS moedas_total
FROM Batalha
WHERE personagem_id = 1;

-- Correção de experiência ganha
UPDATE Batalha
SET experiencia_ganha = 200
WHERE id = 1;

-- Exclusão de batalhas antigas
DELETE FROM Batalha
WHERE data_batalha < DATE_SUB(NOW(), INTERVAL 30 DAY);

-- Exclusão de derrotas específicas
DELETE FROM Batalha
WHERE personagem_id = 1 AND resultado = 'derrota';
```

5. Procedimentos Armazenados (Procedures)

5.1. SimularBatalha

A procedure SimularBatalha simula o confronto entre um personagem e um inimigo, reproduzindo uma lógica típica de batalhas em jogos de RPG. Nessa lógica, o dano causado por cada parte é calculado com base na diferença entre o ataque de um e a defesa do outro, garantindo que o dano mínimo nunca seja negativo. A procedure avalia turno a turno até que um dos lados seja derrotado. Ao final, atualiza o banco com os resultados da batalha, como ganho de experiência e moedas em caso de vitória. A seguir, observamos o trecho que representa a execução da lógica principal do combate, incluindo o cálculo de dano, o laço de ataque e a verificação de resultado:

```

SET dano_personagem := GREATEST(p_ataque - i_defesa, 0);
SET dano_inimigo := GREATEST(i_ataque - p_defesa, 0);

WHILE p_vida > 0 AND i_vida > 0 DO
    SET i_vida := i_vida - dano_personagem;
    IF i_vida > 0 THEN
        SET p_vida := p_vida - dano_inimigo;
    END IF;
END WHILE;

IF p_vida > 0 THEN
    SET resultado := 'vitória';
    SET experiencia_ganha := i_experiencia;
    SET moedas_ganhas := i_moedas;
```

```
ELSE
    SET resultado := 'derrota';
    SET experiencia_ganha := 0;
    SET moedas_ganhas := 0;
END IF;
```

5.2. VerificarSubidaNivel

O procedimento VerificarSubidaNivel é fundamental para implementar a evolução dos personagens ao longo do jogo. Ele verifica se o personagem possui pontos de experiência suficientes para subir de nível, e ao atingir o requisito, realiza a atualização do nível e incrementa seus atributos com base na classe à qual pertence. Isso reflete diretamente a mecânica tradicional dos RPGs, onde o crescimento do personagem é condicionado ao acúmulo de experiência por meio de batalhas. O trecho abaixo demonstra a verificação do nível atual em relação à experiência acumulada e a subsequente evolução do personagem:

```

SELECT experiencia, nivel INTO exp_atual, nivel_atual
FROM Personagem WHERE id = personagem_id;

SET exp_necessaria := nivel_atual * 100;

WHILE exp_atual >= exp_necessaria DO
    SET nivel_atual := nivel_atual + 1;
    SET exp_necessaria := nivel_atual * 100;

    SELECT c.ataque_base, c.vida_base, c.mana_base
    INTO ataque_up, vida_up, mana_up
    FROM Classe c
    JOIN Personagem p ON p.classe_id = c.id
    WHERE p.id = personagem_id;

    UPDATE Personagem
    SET nivel = nivel_atual,
        ataque = ataque + ataque_up,
        vida = vida + vida_up,
        mana = mana + mana_up
    WHERE id = personagem_id;
END WHILE;
```

5.3. ComprarItem

A procedure ComprarItem representa o sistema de aquisição de itens no RPG. Essa funcionalidade garante que o personagem tenha moedas suficientes, debita o valor da compra

e atualiza o inventário. Caso o item já esteja presente no inventário, apenas incrementa a quantidade; caso contrário, insere um novo registro. Isso garante uma experiência de compra próxima à realidade dos jogos, onde a economia e a gestão de inventário são aspectos fundamentais. No trecho a seguir, observa-se a lógica de verificação do saldo, o desconto das moedas e a decisão entre atualizar ou inserir no inventário:

```
0
1  SELECT moedas INTO moedas_personagem
2  FROM Personagem WHERE id = personagem_id;
3
4  SELECT valor INTO valor_item FROM Item WHERE id = item_id;
5
6  IF moedas_personagem < (valor_item * quantidade_desejada) THEN
7      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Moedas insuficientes';
8  END IF;
9
10 UPDATE Personagem
11 SET moedas = moedas - (valor_item * quantidade_desejada)
12 WHERE id = personagem_id;
13
14 SELECT COUNT(*) INTO existe
15 FROM Inventario
16 WHERE personagem_id = personagem_id AND item_id = item_id;
17
18 IF existe > 0 THEN
19     UPDATE Inventario
20     SET quantidade = quantidade + quantidade_desejada
21     WHERE personagem_id = personagem_id AND item_id = item_id;
22 ELSE
23     INSERT INTO Inventario (personagem_id, item_id, quantidade)
24     VALUES (personagem_id, item_id, quantidade_desejada);
25 END IF;
```

5.4. ResetarPersonagem

A procedure ResetarPersonagem foi desenvolvida para simular um recurso comum em RPGs: o reset de status do personagem. Essa funcionalidade é útil quando o jogador deseja recomeçar sua progressão, seja por estratégia ou por falha anterior. A procedure redefine os principais atributos do personagem (nível, experiência, vida e mana) para seus valores base. Essa ação pode ser vista, por exemplo, após derrotas consecutivas ou como penalidade por falhar em uma missão importante. A seguir, observamos o trecho que realiza essa lógica de redefinição dos atributos, após verificar se o personagem existe:

```
0
1  UPDATE Personagem
```

```
SET nivel = 1,
    experiencia = 0,
    vida = 100,
    mana = 100
WHERE id = p_id;
```

5.5. AdicionarExperiencia

A procedure AdicionarExperiencia representa a mecânica fundamental de evolução nos jogos de RPG: a acumulação de pontos de experiência (XP). Ao vencer batalhas, cumprir tarefas ou realizar ações significativas, o personagem ganha experiência que, posteriormente, contribui para sua evolução de nível. Esta procedure verifica se o personagem existe e, em seguida, soma a quantidade de experiência recebida ao seu total atual. O trecho abaixo mostra exatamente essa lógica central de atualização do XP:

```

UPDATE Personagem
SET experiencia = experiencia + p_experiencia
WHERE id = p_personagem_id;
```

6. Funções (Functions)

6.1. CalcularQuantidadeItem

A função Calcular Quantidade Item simula a contagem total de um determinado item no inventário de um personagem. Nos RPGs, é comum que um jogador possua várias unidades de um mesmo item, como poções ou equipamentos duplicados. Essa função retorna a soma de todas essas unidades com base no ID do personagem e do item, utilizando COALESCE para garantir que o valor retornado seja zero caso não exista nenhum item registrado. A seguir, observamos o trecho que exemplifica essa lógica de forma direta:

```

RETURN (SELECT COALESCE(SUM(quantidade), 0)
        FROM Inventario
        WHERE personagem_id = p_personagem_id
          AND item_id = p_item_id);
```

6.2. ExperienciaProximoNivel

Essa função representa uma das lógicas centrais da progressão em RPGs: calcular a quantidade de experiência necessária para alcançar o próximo nível. A mecânica considera o nível atual do personagem e multiplica esse valor por um fator fixo (neste caso, 1000) para determinar o novo limite de XP. Essa abordagem foi escolhida por ser simples e permitir uma curva de progressão previsível. O trecho a seguir ilustra o cálculo da experiência exigida:

```
0
1 SET exp_necessaria = (nivel_atual * 1000);
```

6.3. CalcularNivelEX

Complementando a função anterior, CalcularNivelEX realiza o processo inverso: determina qual o nível correspondente à quantidade total de experiência acumulada. Essa função é útil quando se deseja recalcular o nível de um personagem com base em sua experiência total — por exemplo, em situações de importação de dados ou correção de estado. A lógica se baseia em uma divisão simples com arredondamento para baixo e incremento de um nível inicial:

```
0
1 RETURN FLOOR(experiencia / 1000) + 1;
```

7. Triggers de Integridade

7.1. Atualização automática de atributos com base na classe

A trigger AtualizarStatusClasse é executada automaticamente antes da inserção de um novo personagem e tem como objetivo simular a lógica clássica de criação de personagens em jogos de RPG: ao escolher uma classe (como guerreiro, mago ou arqueiro), os atributos base do personagem são definidos conforme os valores pré-configurados para essa classe. Além disso, ela define o nível e a experiência iniciais caso não sejam informados. A seguir, vemos o trecho responsável por aplicar esses atributos ao novo personagem:

```
0
1 SELECT ataque_base, defesa_base, vida_base, mana_base
2 INTO atk, def, hp, mp
3 FROM Classe
4 WHERE id = NEW.classe_id;
```

7.2. Prevenção de nível negativo

Para garantir a integridade dos dados, especialmente em atualizações manuais ou por bugs em procedures, a trigger verificar nivel personagem impede que o nível de um personagem seja alterado para um valor inferior a 1. Caso essa condição ocorra, a trigger cancela a operação e retorna uma mensagem de erro personalizada. A lógica de verificação é simples e eficaz, como demonstrado abaixo:

```
0
1 IF NEW.nivel < 1 THEN
2     SIGNAL SQLSTATE '45000'
3     SET MESSAGE_TEXT = 'Erro: O nível do personagem não pode ser menor que 1.';
4 END IF;
```

7.3. Restrições na inserção de níveis e experiência

Além da proteção contra atualizações incorretas, triggers adicionais também atuam antes da inserção de novos personagens para evitar valores inválidos. Triggers como verificar nivel insert e verificar experiencia insert impedem respectivamente níveis abaixo de 1 e experiências negativas no momento da criação do personagem. Isso assegura que o banco de dados mantenha integridade desde o início. A verificação abaixo mostra como isso é feito para o campo de experiência:

```
0
1 IF NEW.experiencia < 0 THEN
2     SIGNAL SQLSTATE '45000'
3     SET MESSAGE_TEXT = 'Experiência não pode ser negativa';
4 END IF;
```

7.4. Garantia de unicidade no nome do personagem

Em jogos, o nome do personagem é frequentemente utilizado como identidade única dentro do universo do jogo. A trigger NomeUnicoPersonagem assegura que não haja dois personagens com o mesmo nome, rejeitando automaticamente inserções duplicadas. Isso evita confusões e garante uma experiência de jogo mais autêntica. O trecho abaixo mostra a verificação feita:

```
0
1 IF EXISTS (SELECT 1 FROM Personagem WHERE nome = NEW.nome) THEN
2     SIGNAL SQLSTATE '45000'
3     SET MESSAGE_TEXT = 'Nome de personagem já existe!';
4 END IF;
```

7.5. Validação de referências em batalhas

Para manter consistência nas relações entre personagens, inimigos e batalhas, a trigger validar batalha verifica se tanto o personagem quanto o inimigo envolvidos em uma batalha realmente existem nas tabelas correspondentes antes de permitir o registro. Essa verificação impede inserções inválidas ou com IDs corrompidos. O trecho a seguir exemplifica a lógica para o personagem:

```
0
1 IF NEW.personagem_id IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Personagem WHERE
      ↪ id = NEW.personagem_id) THEN
2     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Personagem não existe';
3 END IF;
```

7.6. Limite máximo de nível

Por fim, a trigger VerificarNivelMaximo implementa uma restrição clássica de RPG: impedir que personagens ultrapassem o nível máximo permitido — neste caso, 100. Se uma tentativa de atualização tentar exceder esse valor, o nível é automaticamente ajustado para 100 e uma exceção é lançada. Isso evita trapaças e mantém a progressão balanceada.

```
0
1 IF NEW.nivel > 100 THEN
2     SET NEW.nivel = 100;
3     SIGNAL SQLSTATE '45000'
4     SET MESSAGE_TEXT = 'Nível máximo atingido: 100';
5 END IF;
```

8. Views Úteis

8.1. Ranking de Personagens

A view RankingPersonagens oferece uma visão panorâmica e hierárquica dos personagens com base em seus atributos e desempenho. Essa view é essencial para montar um ranking dos jogadores mais poderosos, considerando o nível, a experiência, atributos como ataque e defesa, recursos como moedas, e também seu histórico de vitórias e derrotas em batalhas. A apresentação dos pontos de vida e mana é feita de forma amigável, com um formato do tipo ”atual/base”, garantindo clareza na exibição.

```
0
1
2 CREATE VIEW RankingPersonagens AS
```

```sql
SELECT
    p.id,
    p.nome,
    c.nome AS classe,
    p.nivel,
    p.experiencia,
    CONCAT(p.vida, '/', (SELECT vida_base FROM Classe WHERE id = p.classe_id))
    ↪ AS vida,
    CONCAT(p.mana, '/', (SELECT mana_base FROM Classe WHERE id = p.classe_id))
    ↪ AS mana,
    p.ataque,
    p.defesa,
    p.moedas,
    (SELECT COUNT(*) FROM Batalha WHERE personagem_id = p.id AND resultado = '
    ↪ vitória') AS vitorias,
    (SELECT COUNT(*) FROM Batalha WHERE personagem_id = p.id AND resultado = '
    ↪ derrota') AS derrotas
FROM Personagem p
JOIN Classe c ON p.classe_id = c.id
ORDER BY p.nivel DESC, p.experiencia DESC;
```

8.2. Inventário Detalhado

A view InventarioDetalhado fornece uma análise detalhada dos itens em posse de cada personagem. Ao cruzar as tabelas Inventario, Item e Personagem, a view mostra quais itens cada jogador possui, sua quantidade, o tipo e efeito do item, bem como o valor individual e o valor total acumulado com base nas quantidades. Essa estrutura facilita análises financeiras, logísticas ou até mecânicas de jogo relacionadas ao uso ou venda de itens.

```sql

CREATE VIEW InventarioDetalhado AS
SELECT
    p.nome AS personagem,
    i.nome AS item,
    inv.quantidade,
    i.tipo,
    i.poder,
    i.efeito,
    i.valor,
    (inv.quantidade * i.valor) AS valor_total
FROM Inventario inv
JOIN Personagem p ON inv.personagem_id = p.id
JOIN Item i ON inv.item_id = i.id
ORDER BY p.nome, i.tipo, i.nome;
```

8.3. Histórico de Batalhas

A view HistoricoBatalhas permite o rastreamento completo das batalhas realizadas, reunindo informações cruciais como o nome do personagem envolvido, o inimigo enfrentado, o resultado da luta, experiência e moedas obtidas, além da data da batalha. Um diferencial dessa view é a atribuição de uma cor simbólica à coluna cor resultado, indicando vitória com verde e derrota com vermelho, o que pode ser aproveitado visualmente por interfaces web ou painéis de dados:

```
0
1  CREATE VIEW HistoricoBatalhas AS
2  SELECT
3      b.id,
4      p.nome AS personagem,
5      i.nome AS inimigo,
6      b.resultado,
7      b.experiencia_ganha,
8      b.moedas_ganhas,
9      b.data_batalha,
10     CASE
11         WHEN b.resultado = 'vitória' THEN 'green'
12         ELSE 'red'
13     END AS cor_resultado
14 FROM Batalha b
15 LEFT JOIN Personagem p ON b.personagem_id = p.id
16 LEFT JOIN Inimigo i ON b.inimigo_id = i.id
17 ORDER BY b.data_batalha DESC;
```

8.4. Personagens por Classe

A view PersonagensPorClasse organiza todos os personagens cadastrados no sistema de acordo com a sua respectiva classe. Essa segmentação é ideal para exibições que filtram os personagens por tipo (como magos, guerreiros, arqueiros, etc.), e também permite análises rápidas sobre níveis médios por classe, distribuição de jogadores, entre outros.

```
0
1  CREATE VIEW PersonagensPorClasse AS
2  SELECT
3      p.id AS personagem_id,
4      p.nome AS personagem_nome,
5      c.nome AS classe_nome,
6      p.nivel,
7      p.experiencia,
8      p.vida,
9      p.mana,
10     p.ataque,
```

```
    p.defesa,
    p.moedas
FROM Personagem p
JOIN Classe c ON p.classe_id = c.id
ORDER BY c.nome, p.nivel DESC;
```

9. Criação dos Papéis (Roles)

No contexto de um sistema de RPG com banco de dados, os roles (ou papéis) representam os diferentes níveis de permissão que usuários podem ter de acordo com sua função dentro do mundo do jogo. Isso reflete o que vemos em muitos MMOs e RPGs multiplayer, onde há desde desenvolvedores e administradores até jogadores experientes ou líderes de clãs.

9.1. Administrador do Servidor (rpg root)

O papel de Administrador do Servidor é equivalente a um desenvolvedor principal ou dono do jogo. Ele possui acesso completo a todos os dados e funcionalidades do sistema: pode criar, editar e apagar qualquer registro — desde personagens até configurações de regras, itens e batalhas.

Exemplo no jogo real: Imagine um desenvolvedor como o ”Samuel”, que está testando a economia do jogo. Ele pode modificar diretamente os valores dos itens, corrigir bugs no banco de dados e até resetar personagens manualmente se necessário.

Permissões atribuídas: ALL PRIVILEGES em todas as tabelas e procedures do banco RPG.

9.2. Mestre da Guilda (rpg guild master)

O Mestre da Guilda tem um papel de liderança entre os jogadores. No sistema, isso é traduzido em permissões intermediárias: ele pode criar ou editar personagens (como um mestre que ajuda seus membros), administrar o inventário da guilda, e gerenciar as habilidades dos seus aliados. Também pode executar algumas procedures estratégicas, como ComprarItem ou VerificarSubidaNivel, mas não tem acesso total ao sistema.

Exemplo no jogo real: Leonardo é o mestre da guilda “Dragões de Prata”. Ele pode adicionar equipamentos aos membros da guilda, treinar seus personagens com novas habilidades e garantir que estejam prontos para eventos importantes no servidor, como guerras de guilda.

Permissões: leitura e escrita nas tabelas de itens, inventário, personagens, habilidades; execução de procedures específicas.

9.3. Campeão do Servidor (rpg champion)

O Campeão do Servidor representa jogadores muito experientes e reconhecidos por suas conquistas. Apesar de não poder alterar os dados diretamente, ele tem acesso privilegiado de leitura para consultar informações complexas do jogo, como o ranking, os detalhes das batalhas e estatísticas. Ele também pode executar funções e procedures analíticas, como SimularBatalha — útil para estudar estratégias.

Exemplo no jogo real: Vitor é o campeão do último torneio PvP. Ele não precisa alterar nada, mas tem acesso a dados privilegiados que outros jogadores comuns não veem, como a fórmula de dano das habilidades e o histórico completo das batalhas. Ele usa isso para melhorar seu desempenho e compartilhar dicas com outros jogadores.

Permissões: leitura em todas as tabelas e execução de algumas funções estratégicas.

9.4. Pro Player (rpg pro player)

O Pro Player é um jogador dedicado, mas com acesso mais limitado. Seu papel é puramente consultivo: pode ver informações sobre personagens, classes, inimigos e itens, mas não tem permissão para alterar dados ou executar funções administrativas.

Exemplo no jogo real: Matheus é um jogador assíduo que consulta dados para planejar o crescimento do seu personagem. Ele pode acessar as fichas dos inimigos antes de uma batalha ou estudar os efeitos dos itens que pretende comprar. No entanto, não pode interferir nos dados do jogo nem fazer simulações avançadas.

Permissões: apenas leitura básica em tabelas de jogo e execução da função de experiência necessária para subir de nível.

```
0
1  Criação dos Usuários
2
3
4  -- Usuário 1: Administrador do Servidor (Root/Dev)
5
6  CREATE USER 'samuel'@'localhost' IDENTIFIED BY '123';
7  GRANT 'rpg_root' TO 'samuel'@'localhost';
8  SET DEFAULT ROLE 'rpg_root' TO 'samuel'@'localhost';
9
10 -- Usuário 2: Mestre da Guilda
11
12 CREATE USER 'leonardo'@'localhost' IDENTIFIED BY '456';
13 GRANT 'rpg_guild_master' TO 'leonardo'@'localhost';
14 SET DEFAULT ROLE 'rpg_guild_master' TO 'leonardo'@'localhost';
15
16 -- Usuário 3: Campeão do Servidor
```

```
17
18 CREATE USER 'vitor'@'localhost' IDENTIFIED BY '789';
19 GRANT 'rpg_champion' TO 'vitor'@'localhost';
20 SET DEFAULT ROLE 'rpg_champion' TO 'vitor'@'localhost';
21
22 -- Usuário 4: Pro Player
23
24
25 CREATE USER 'matheus'@'localhost' IDENTIFIED BY '111';
26 GRANT 'rpg_pro_player' TO 'matheus'@'localhost';
27 SET DEFAULT ROLE 'rpg_pro_player' TO 'matheus'@'localhost';
28
29 FLUSH PRIVILEGES;
```

10. Monitoramento de Ações dos Usuários: Log Automatizado

10.1. Função Extra

Embora o projeto inicial do banco de dados do sistema de RPG já contemplasse um conjunto completo de funcionalidades essenciais para o gerenciamento de personagens, batalhas, inventário e progressão, surgiu a necessidade — a posteriori — de incluir uma camada de rastreabilidade para ações sensíveis realizadas por usuários no banco. Essa demanda refletiu uma preocupação comum em sistemas multiusuários: a transparência nas alterações e a capacidade de identificar quem realizou determinada modificação.

Para isso, foi idealizada e implementada a tabela AcaoUsuario, responsável por registrar logs automáticos de ações realizadas por usuários, tais como inserções, edições ou exclusões em tabelas críticas do sistema. Este controle se torna especialmente útil em um cenário de múltiplos papéis com diferentes permissões, como visto anteriormente.

```
0
1 CREATE TABLE AcaoUsuario (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     usuario VARCHAR(50) NOT NULL,
4     acao VARCHAR(255) NOT NULL,
5     tabela_afetada VARCHAR(50),
6     data_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP
7 );
```

Junto à tabela, foi criada uma trigger (gatilho) específica para registrar automaticamente a criação de novos itens no banco de dados, ação que pode ser executada por usuários com papel de rpg root ou rpg guild master:

```
0
1 DELIMITER //
2 CREATE TRIGGER log_acoes_item
3 AFTER INSERT ON Item
4 FOR EACH ROW
5 BEGIN
6     INSERT INTO AcaoUsuario (usuario, acao, tabela_afetada)
7     VALUES (CURRENT_USER(), CONCAT('Criou novo item: ', NEW.nome), 'Item');
8 END //
9 DELIMITER ;
```

Essa solução oferece uma camada importante de accountability, permitindo identificar autores de modificações e reforçando a segurança do sistema. Além disso, abre espaço para futuras expansões, como o log de atualizações ou exclusões em outras tabelas-chave.

11. Conclusão

11.1. Dificuldades Encontradas

Durante o desenvolvimento do sistema de RPG utilizando apenas o MySQL, diversas dificuldades técnicas surgiram. A complexidade das Stored Procedures, em especial a SimularBatalha, exigiu inúmeras iterações e testes até que se alcançasse um balanceamento adequado do sistema de combate, totalizando cerca de 15 revisões sucessivas.

Outro desafio relevante foi o gerenciamento de transações, como no caso da procedure ComprarItem, que demandou o uso cuidadoso de COMMIT e ROLLBACK para garantir a integridade dos dados durante acessos simultâneos. Em um ambiente onde múltiplos jogadores podem tentar comprar itens ao mesmo tempo, essas operações tornaram-se críticas.

Também foi identificado que o SQL puro apresenta limitações estruturais ao tentar simular mecânicas de jogos complexos. Por exemplo, funcionalidades como “efeitos temporários de itens” ou ações combinadas entre triggers e procedures provaram-se extremamente difíceis de implementar diretamente no banco, levando à decisão de simplificar ou remover algumas ideias originais. Isso evidenciou a importância de alinhar as expectativas de design com as capacidades reais da tecnologia utilizada.

11.2. Limitações do Sistema

Sistema de missões Falta de suporte nativo para eventos temporizados no MySQL Combate em grupo (party) Dificuldade em modelar relacionamentos muitos-para-muitos com

lógica transacional Efeitos cumulativos de itens Ausência de estruturas dinâmicas para armazenar estados temporário

Essas limitações refletem os limites naturais de se desenvolver um jogo inteiramente dentro de um sistema gerenciador de banco de dados relacional.

11.3. Avaliação Individual

Samuel:

”O maior aprendizado foi descobrir como modelar lógica de jogo em SQL. A procedure de batalha foi especialmente desafiadora, mas ao final, ver o cálculo completo de dano funcionando apenas com MySQL foi recompensador. Percebi que triggers são poderosas, mas exigem planejamento. Também percebi que deveríamos ter pensado melhor na ideia de criar um exemplo de jogo direto no banco, pois algumas funcionalidades que imaginamos ficaram complexas demais e tivemos que adaptar. Isso gerou certa inconsistência com a ideia original, mas no geral, considero que foi um bom trabalho.”

Matheus:

”Fiquei surpreso com a flexibilidade do MySQL para criar mecânicas de RPG. A view RankingPersonagens mostrou como SQL pode gerar relatórios complexos com poucas linhas. Por outro lado, a falta de arrays nativos dificultou implementar inventários mais elaborados — mas faz sentido, dado que estamos tentando criar um RPG em banco de dados puro.”

11.4. Conquistas e Trabalhos Futuros

Sistema de combate funcional com cálculo automático de dano;

Progressão de níveis integrada à experiência de batalha;

Inventário dinâmico com tipos variados de itens e efeitos;

Controle de acesso com diferentes papéis de usuários (administrador, mestre de guilda, etc.);

Views analíticas para ranking, histórico de batalha e status dos personagens;

Trigger de log automatizado para rastrear alterações no sistema.

11.5. Como melhorias futuras, foram propostas

Adição de sistema de guildas com as tabelas Guilda e MembroGuilda;

Implementação de missões diárias, via eventos agendados;

Inclusão de uma loja de itens com sistema de preços dinâmicos;

Criação de uma funcionalidade de localização dos personagens (ex.: cidade natal);

Sistema de recompensas por nível (ex.: 200 moedas ao atingir o nível 10).

11.6. Considerações Finais

Este projeto demonstrou que é possível prototipar um sistema de RPG inteiramente no banco de dados, utilizando recursos como procedures, triggers, views e funções. Embora algumas funcionalidades tenham exigido simplificações ou adaptações, o trabalho serviu como uma excelente experiência de design de sistemas, evidenciando os dilemas entre complexidade, desempenho e viabilidade técnica.

A conclusão geral é que bancos de dados relacionais podem ir além do armazenamento e oferecer capacidade lógica significativa, desde que suas limitações sejam compreendidas e bem gerenciadas. Para prototipagem, controle de regras e análise de dados, o SQL se mostrou surpreendentemente poderoso.

12. Referências

ELMASRI, Ramez; NAVATHE, Shamkant B. *Sistemas de banco de dados*. 6. ed. Tradução de Marília Guimarães Pinheiro, Cláudio César Canhette, Glenda Cristina Valim Melo, Claudia Vicei Amadeu e Rinaldo Macedo de Morais. Revisão técnica de Luis Ricardo de Figueiredo. São Paulo: Pearson Education do Brasil, 2011.

MATERIAL de apoio apresentado pelo professor em sala de aula, incluindo slides, anotações e exemplos de código SQL.

Segue o link do repositório no GitHub com os códigos do projeto RPG: o arquivo RPG.sql contém as tabelas base e as inserções iniciais, enquanto o arquivo RPG Operacoes.sql reúne toda a lógica das operações no banco de dados.

⟨https://github.com/SamuelAugusto633/Projeto-Final---Banco-de-Dados-2-RPG-Medieval-.git⟩