# Banco de Dados Praceando

## **Estrutura do Banco Normalizado do 2º ano**

O banco de dados do segundo ano é integrado diretamente com o aplicativo Praceando e possui diferentes tabelas que fazem sentido com a lógica do app. As tabelas citadas abaixo foram modeladas para SQL e, portanto, seguem as formas normais.

Vale ressaltar que todas as tabelas adotam abreviações nas colunas para identificar e padronizar de maneira mais eficiente. Também há primary keys e o campo dt\_atualizacao em todas as tabelas e foreing Keys apenas quando é necessário. Abaixo todas as abreviações:

* id para Primary Key

cd para Foreign Key

* ds para descrição
* dt para data
* hr para hora
* is para campo boolean
* qt para quantidade
* nm para nome
* nr para número
* vl para valor
* V para variável
* Ppara parâmetro

### Relacionadas a Usuários

* **Genêro:** tabela dedicada a todo e qualquer gênero que o usuário possa informar dentro do app e possui as colunas id\_genero,ds\_genero e dt\_atualizacao. Cada coluna contém valores indivisíveis, e estão todas relacionadas unicamente a primary key id\_genero.
* **Acesso:** tabela feita para autenticar usuários e identificar seus tipos. Cada coluna contém valores indivisíveis, e estão todas relacionadas unicamente a primary key id\_acesso.
* **Usuário:** tabela dedicada aos dados comuns entre anunciantes e consumidores, possuindo colunas como id\_usuario, cd\_acesso, cd\_inventario\_avatar, cd\_genero, nm\_usuario, ds\_email, ds\_senha, is\_premium, ds\_usuario, dt\_criacao, dt\_desativacao e dt\_atualizacao. Os dados são atômicos e sem repetições, todos valores não-chave dependem da primary key id\_usuario e suas foreing keys estão referenciadas a suas respectivas tabelas.
* **Consumidor:** tabela para dados exclusivos do consumidor, ligada diretamente com usuário. Possui colunas como id\_consumidor, dt\_nascimento, nm\_nickname, nr\_polen e is\_possivel\_anunciar, todas armazenam um valor por registro e todas as colunas estão unicamente relacionadas a id\_consumidor.
* **Anunciante:** tabela para dados exclusivos do anunciante, ligada diretamente com usuário. Possui colunas como id\_anunciante, dt\_nascimento, nm\_empresa, nr\_cnpj, todas armazenam um valor por registro e todas as colunas estão unicamente relacionadas a id\_anunciante.
* Tags do Consumidor: tabela que guarda todos os interesses demonstrados pelos consumidores por meio das tags para recomendações futuras. Possui tabelas como id\_consumidor\_tag, cd\_consumidor e cd\_tag, todas armazenando um valor por registro e relacionadas unicamente a sua primary key.
* Conquistas: A collection registra as conquistas que os usuários podem obter ao realizar atividades ou alcançar metas dentro do aplicativo, com objetivos específicos para consumidores e anunciantes. Possui as colunas **ID\_CONQUISTA, NM\_CONQUISTA, DS\_CONQUISTA, NM\_TIPO, QT\_POLEN, ID\_AVATAR e DT\_ATUALIZACAO.** Cada coluna armazena um valor indivisível, todas estão relacionadas unicamente pela chave primária ID\_CONQUISTAA ideia inicial era que as tabelas usuário, consumidor e anunciante fossem implementadas utilizando herança no SQL. No entanto, devido a erros relacionados a constraints e foreign keys, optou-se por estabelecer ligações diretas entre as tabelas por meio de chaves estrangeiras.
* **Conquista\_usuario**: A collection armazena o histórico de conquistas específicas que cada usuário obteve, permitindo rastrear o progresso e registrar realizações. Possui as colunas **ID\_CONQUISTA\_USER, CD\_USUARIO, CD\_CONQUISTA e DT\_ATUALIZACAO**. Todos os valores são atômicos e relacionados unicamente pela chave primária **ID\_CONQUISTA\_USER.**

### Relacionadas a Eventos

* **Local:**  tabela responsável por possíveis lugares que podem ser utilizados para eventos, sejam parques ou praças. As colunas incluem: id\_local, nm\_local, nr\_lat, nr\_long, hr\_abertura, hr\_fechamento, dt\_desativacao e dt\_atualizacao. Todas as colunas contêm dados indivisíveis e todos os atributos não-chave dependem exclusivamente da chave primária id\_local. Além disso, a Tabela local não possui dependências transitivas entre atributos não-chave.
* **Tag:** meio de categorizar eventos e interesses de usuários, com colunas como id\_tag, nm\_tag, ds\_categoria, dt\_desativacao e dt\_atualizacao. Cada coluna armazena dados indivisíveis e é atendida porque cada coluna não-chave depende exclusivamente da chave primária id\_tag. Também não há dependências transitivas entre colunas não-chave.
* **Evento:** registra dados sobre eventos de anunciantes, incluindo colunas como id\_evento, cd\_local, cd\_anunciante, nm\_evento, ds\_evento, dt\_inicio, hr\_inicio, dt\_fim, hr\_fim, url\_documentacao, dt\_desativacao e dt\_atualizacao. Todas as colunas são atômicas e dependem unicamente da chave primária, também não há dependências transitivas.
* **Tags de Evento:** conecta eventos a tags, facilitando a identificação do propósito de cada evento, com as colunas id\_evento\_tag, cd\_tag, cd\_evento e dt\_atualizacao. Cada coluna é atômica, assim como como cada coluna não-chave depende exclusivamente da chave primária e não possui dependências transitivas.
* **Interesse:** registra interesses dos usuários em eventos específicos, contribuindo para recomendação de outros eventos. Suas colunas são id\_interesse, cd\_consumidor, cd\_evento e dt\_atualizacao. Cada coluna armazena valores indivisíveis e todas as colunas dependem exclusivamente da chave primária. Não há dependências transitivas.
* **Avaliação**: A collection armazena os feedbacks de usuários sobre os eventos, incluindo notas, comentários e reações adicionais, como emojis. Possui as colunas **ID\_AVALIACAO, CD\_EVENTO, CD\_USUARIO, NOTA, COMENTARIO, REACAO e DT\_ATUALIZACAO**. Cada coluna armazena dados indivisíveis e a collection está organizada unicamente pela chave primária **ID\_AVALIACAO**.
* **Recorrência**: A collection define a periodicidade dos eventos que se repetem, como eventos diários, semanais ou mensais, e facilita a exibição automática para os usuários. As colunas incluem **ID\_RECORRENCIA, ID\_EVENTO, NM\_TIPO, DS\_DIAS\_SEMANA, NR\_DIA\_MES e DT\_ATUALIZACAO**. Cada coluna é indivisível, e os dados estão organizados unicamente pela chave primária **ID\_RECORRENCIA**.

### Relacionadas a Produtos

* **Produto:** armazena itens personalizados da empresa, sejam artesanais ou digitais (avatares) tal qualquer usuário pode comprar, com colunas que incluem: id\_produto, qt\_estoque, nm\_produto, ds\_produto, vl\_preco, nm\_categoria, url\_imagem, dt\_desativacao e dt\_atualizacao. Todas as colunas contêm valores atômicos e dependem unicamente da chave primária id\_produto. Além disso, não há dependências transitivas entre as colunas não-chave.
* **Compra:** registra as compras feitas pelos usuários, sejam por produtos ou eventos, com colunas como id\_compra, cd\_usuario, cd\_produto, cd\_evento, dt\_compra, vl\_total, ds\_status e dt\_atualizacao. Todas as colunas contêm valores indivisíveis e cada coluna não-chave depende exclusivamente da chave primária id\_compra. Também há ausência de dependências transitivas entre colunas não-chave.
* **Pagamento:** armazena informações sobre pagamentos realizados, incluindo colunas como id\_pagamento, cd\_compra, dt\_pagamento e dt\_atualizacao. Cada coluna possui dados indivisíveis e todos os atributos não-chave dependem exclusivamente da chave primária id\_pagamento e nãp possuem outras dependências.

### Relacionadas a Sustentabilidade

* **Frases Sustentáveis:** possui uma frase relacionada a sustentabilidade para cada dia do ano, com colunas que incluem: id\_frase\_sustentavel, ds\_frase\_sustentavel, dt\_frase\_sustentavel e dt\_atualizacao. Todas as colunas contêm valores atômicos e dependem unicamente da chave primária id\_frase\_sustentavel. Além disso, não há dependências transitivas entre as colunas não-chave.

## **Observações sobre o Banco do 1º Ano**

Inicialmente, o banco foi estruturado junto com o 1º Ano, resultando na criação do Script 1. Posteriormente, foi necessário remodelá-lo para atender às demandas do aplicativo, o que levou ao desenvolvimento do Script 2.

### Primeira Forma Normal (1NF) - Eliminação de Grupos Repetidos

A 1NF exige que todas as colunas de uma tabela armazenem valores atômicos, ou seja, sem listas ou conjuntos de valores em uma única coluna.

* **Script 1 e Script 2**: Ambos os scripts seguem a 1NF, pois as colunas de todas as tabelas armazenam valores únicos e atômicos. Em relação às colunas de data e quantidade, não houve alteração estrutural que afetasse a 1NF.

### Segunda Forma Normal (2NF) - Eliminação de Dependências Parciais

A 2NF requer que cada tabela esteja na 1NF e que cada coluna dependa inteiramente da chave primária, sem dependências parciais (ou seja, dependências de parte de uma chave composta).

* **Identificação de FK e Modificação de Nomes**: O script 2 altera o nome de várias colunas para seguir uma convenção de nomenclatura (cd para chaves estrangeiras, nm para nomes, ds para descrições, etc.), o que facilita a identificação de chaves e dependências.
* **Separação de Responsabilidades**: No Script 1, as tabelas **dados\_consumidor** e **dados\_anunciante** incluem atributos que podem se sobrepor aos usuários em geral, como id\_usuario. No Script 2, eles são normalizados em consumidor e anunciante que fazem referência direta a **usuario**.

### Terceira Forma Normal (3NF) - Eliminação de Dependências Transitivas

A 3NF exige que cada tabela esteja na 2NF e que nenhuma coluna não-chave dependa de outra coluna não-chave.

* **Separação das Tabelas acesso e usuario\_tag**: No Script 1, usuario armazenava algumas informações que poderiam ser categorizadas separadamente. No Script 2, as tabelas acesso e usuario\_tag foram criadas para que:
  + acesso gere dados relacionados ao acesso do usuário e evita dependências transitivas no usuario.
  + usuario\_tag cria uma associação entre usuario e tag, permitindo que usuario dependa apenas de colunas próprias sem dependências externas transitivas.

### Considerações Finais e Otimização

Além da normalização, o script 2 se destaca em:

* **Padronização de Nomeclatura**: Nomes de colunas seguem um padrão (ex.: id para PK, cd para FK), facilitando a leitura e manutenção.
* **Tabelas de Log**: No Script 1, havia mais uso de logs e triggers, que foram simplificados no Script 2 para reduzir a complexidade e focar na normalização do banco.
* **Remoção de Views Desnecessárias**: As **views** consumidor e anunciante foram removidas para evitar complexidade desnecessária e centralizar os dados nas tabelas consumidor e anunciante.

## **Functions**

### FNC\_CALCULAR\_IDADE

**Descrição:** Esta função calcula a idade de um usuário com base na data de nascimento fornecida. Ela subtrai o ano de nascimento do usuário do ano atual para determinar a idade em anos.

**Parâmetros:**

* p\_dt\_nascimento (DATE): Data de nascimento do usuário no formato DATE.

**Retorno:** A função retorna um valor inteiro (INTEGER) que representa a idade do usuário em anos completos.

**Funcionamento:**

1. Calcula a diferença em anos entre a data atual (CURRENT\_DATE) e a data de nascimento fornecida (dt\_nascimento).
2. Usa a função extract(year from age(...)) para extrair a diferença em anos completos.
3. Retorna o valor calculado, que representa a idade do usuário.

* **Utilização:** Ideal para obter a idade de usuários, especialmente em contextos de relatórios, validação de idade ou cálculos de faixa etária.

### FNC\_INSERIR\_ID\_AVATAR

**Descrição:** Esta função atualiza o avatar associado a um usuário no banco de dados, usando os identificadores de usuário e avatar fornecidos. Ao atualizar o avatar, a função também registra a data da última atualização.

**Parâmetros:**

* p\_id\_usuario (INTEGER): Identificador único do usuário cuja informação será atualizada.
* p\_id\_avatar (INTEGER): Identificador único do avatar a ser atribuído ao usuário.

**Retorno:**

* A função não retorna valor (VOID).

**Funcionamento:**

1. Atualiza a tabela **usuario** para o usuário correspondente a p\_id\_usuario.
2. Atribui ao campo cd\_inventario\_avatar o valor de p\_id\_avatar, especificando qual avatar o usuário escolheu.
3. Define o campo dt\_atualizacao como a data atual (CURRENT\_DATE) para registrar o momento da última alteração de avatar.
4. A alteração afeta apenas o registro do usuário indicado pelo p\_id\_usuario.

**Utilização:** Essa função é especialmente útil em sistemas onde usuários podem escolher ou trocar seus avatares, facilitando o rastreamento e o gerenciamento de mudanças.

## **Triggers**

### TRG\_DESATIVAR\_USUARIO

**Descrição:**  
Essas implementações desativam o usuário e, caso o usuário seja um anunciante, desativam também os eventos associados a ele.

* **Trigger** trg\_desativar\_usuario: ativada antes de qualquer atualização na coluna dt\_desativacao da tabela usuario.
* **Função** fnc\_desativar\_usuario: realiza as atualizações de dt\_desativacao em usuario e evento.

**Funcionamento:**

1. Atualiza o campo dt\_desativacao do usuário para a data e hora atuais (NOW()).
2. Se o campo nr\_cnpj do usuário não for nulo, atualiza também dt\_desativacao dos eventos onde cd\_anunciante coincide com o id\_usuario desativado.

### TRG\_ATUALIZA\_QT\_INTERESSE

**Descrição:**  
Atualizam o contador de interesse (qt\_interesse) em um evento sempre que uma nova entrada é inserida na tabela interesse.

* **Trigger** trg\_atualiza\_qt\_interesse: ativada após cada INSERT na tabela interesse.
* **Função** fnc\_atualiza\_qt\_interesse: aumenta qt\_interesse em 1 para o evento correspondente.

**Funcionamento:**

1. Incrementa o campo qt\_interesse da tabela evento para o evento associado (cd\_evento) da nova entrada em interesse.

### FNC\_LOG\_GERAL

**Descrição:**  
Esta função cria logs detalhados de todas as operações (INSERT, UPDATE, DELETE) nas tabelas alvo. A função utiliza a extensão hstore para registrar as alterações feitas em cada campo, com o valor antigo (vl\_campo\_old) e o valor novo (vl\_campo\_new).

**Parâmetros de Contexto:**

* v\_nm\_tabela\_log: tabela de log específica onde a entrada será inserida, gerada dinamicamente com o prefixo log\_ e o nome da tabela.
* v\_nm\_pk e v\_nm\_fk: identificadores dinâmicos da chave primária e da chave estrangeira, respectivamente, baseados na tabela de origem.

**Funcionamento:**

1. **INSERT**: insere uma entrada de log com o ID do registro e o nome do usuário para cada nova linha na tabela de origem.
2. **UPDATE**: compara valores new e old em busca de mudanças usando hstore.
   * Para cada campo alterado, registra o nome do campo, valores antigos e novos, usuário e operação.
3. **DELETE**: insere uma entrada de log para registrar a exclusão do registro.

**Observação:** É necessária a extensão hstore para armazenar e processar alterações entre new e old. Caso não exista, basta dar o comando create extension if not exists hstore;

Essas tabelas de log (log\_usuario, log\_local, log\_tag, log\_evento, log\_frase\_sustentavel, log\_produto, log\_compra, log\_pagamento) armazenam os registros gerados por fnc\_log\_geral. Cada tabela de log inclui as colunas:

* id\_log: chave primária do log.
* cd\_<tabela>: chave estrangeira para a tabela de origem.
* nm\_campo: campo alterado na operação.
* vl\_campo\_old e vl\_campo\_new: valores antigos e novos para o campo alterado.
* nm\_usuario: nome do usuário que realizou a ação.
* nm\_operacao: tipo da operação realizada (INSERT, UPDATE, DELETE).
* dt\_operacao: data e hora da operação.

Cada tabela de log tem uma trigger associada (como trg\_log\_usuario, trg\_log\_local, etc.) que chama a fnc\_log\_geral após uma operação de INSERT, UPDATE ou DELETE na respectiva tabela de origem.

### **TRG\_DESATIVAR\_USUARIO**

Descrição: A função FNC\_DESATIVAR\_USUARIO define uma lógica de atualização automática da tabela evento quando o campo **dt\_desativacao** de um usuário na tabela **usuario** é atualizado pela primeira vez. Seu objetivo é desativar os eventos associados ao usuário desativado, registrando a data e hora da desativação.

Parâmetros de Contexto: Não há parâmetros externos, pois a função é acionada pela trigger **TRG\_DESATIVAR\_USUARIO** com os dados da linha de **usuario.**

Funcionamento:

A função verifica se o campo **dt\_desativacao** do registro antigo (OLD) é NULL, ou seja, se a desativação do usuário está ocorrendo pela primeira vez.

Caso **dt\_desativacao** seja NULL, a função:

Atualiza o campo **dt\_desativacao** na tabela evento, definindo a data e hora atuais (NOW()) para todos os eventos associados ao usuário, identificado pela correspondência entre **cd\_anunciante** (na tabela evento) e **id\_usuario** (na tabela **usuario**).

Em seguida, a função retorna o novo estado do registro (NEW), permitindo a continuidade do processo de atualização do campo **dt\_desativacao** na tabela **usuario.**

Trigger Associada: A trigger **TRG\_DESATIVAR\_USUARIO** é configurada para ativar a função **FNC\_DESATIVAR\_USUARIO** antes de qualquer atualização do campo **dt\_desativacao** na tabela **usuario**. Ela verifica se o campo está sendo modificado e aciona a função para desativar os eventos vinculados ao usuário.

Estrutura:

Trigger: **TRG\_DESATIVAR\_USUARIO**

Tabela: **usuario**

Coluna de Monitoramento: **dt\_desativacao**

Operação: **BEFORE UPDATE**

### **TRG\_DESATIVAR\_LOCAL**

## **Procedures**

### PRC\_INSERIR\_EVENTO\_TAGS

**Descrição:** Esta procedure insere um novo evento na tabela evento, com seus detalhes e associando-o a tags específicas. Ela valida se cada tag fornecida existe na tabela tag, lançando um erro se uma tag não for encontrada.

**Parâmetros:**

* p\_nm\_evento (VARCHAR): Nome do evento.
* p\_ds\_evento (TEXT): Descrição do evento.
* p\_dt\_inicio (DATE): Data de início do evento.
* p\_hr\_inicio (TIME): Hora de início do evento.
* p\_dt\_fim (DATE): Data de término do evento.
* p\_hr\_fim (TIME): Hora de término do evento.
* p\_url\_documentacao (TEXT): URL de documentação do evento.
* p\_cd\_local (INTEGER): Código do local onde o evento ocorrerá.
* p\_cd\_anunciante (INTEGER): Código do anunciante do evento.
* p\_tags (VARCHAR[]): Lista de tags associadas ao evento.

**Funcionamento:**

1. Insere um novo evento na tabela evento e armazena o ID do evento em v\_id\_evento.
2. Para cada tag em p\_tags, verifica se a tag existe na tabela tag.
   * Se a tag não existir, lança um erro com a mensagem "Tag não encontrada".
3. Insere uma entrada na tabela evento\_tag para associar cada tag ao evento.

**Utilização:** Essa procedure é útil para registrar eventos com múltiplas tags, associando-os a informações contextuais e locais relevantes.

### PRC\_INSERIR\_INTERESSE\_TAGS

**Descrição:** Esta procedure registra o interesse de um usuário em um evento, associando-o a tags específicas de interesse. Valida se cada tag existe na tabela tag, lançando um erro caso contrário.

**Parâmetros:**

* p\_cd\_consumidor (INTEGER): Código do consumidor interessado no evento.
* p\_cd\_evento (INTEGER): Código do evento no qual o usuário está interessado.
* p\_tags (VARCHAR[]): Lista de tags que representam os interesses do usuário.

**Funcionamento:**

1. Insere uma entrada na tabela interesse para registrar o interesse do consumidor no evento.
2. Para cada tag em p\_tags, verifica se a tag existe na tabela tag.
   * Se a tag não existir, lança um erro com a mensagem "Tag não encontrada".
3. Insere uma entrada na tabela usuario\_tag para associar cada tag ao consumidor.

**Utilização:** Essa procedure é ideal para sistemas onde usuários podem demonstrar interesse em eventos específicos e associar esse interesse a tags de preferência.

### PRC\_ATUALIZAR\_STATUS\_COMPRA (para Compras)

**Descrição:** Esta procedure insere um registro de compra na tabela compra e atualiza o estoque do produto adquirido.

**Parâmetros:**

* p\_cd\_usuario (INTEGER): Código do usuário que realizou a compra.
* p\_cd\_produto (INTEGER): Código do produto adquirido.
* p\_cd\_evento (INTEGER): Código do evento associado à compra, se aplicável.
* p\_vl\_total (INTEGER): Valor total da compra.

**Funcionamento:**

1. Armazena a data e hora atual em v\_dt\_compra.
2. Insere um novo registro de compra na tabela compra com os dados do usuário, produto, evento, data e valor total.
3. Atualiza o estoque do produto (qt\_estoque) na tabela produto, subtraindo 1 unidade.
4. Executa COMMIT para confirmar a transação.

**Utilização:** Essencial para registrar compras de produtos com controle de estoque, permitindo registrar informações de venda e manter a contagem de estoque atualizada.

### PRC\_ATUALIZAR\_STATUS\_COMPRA

**Descrição:** Esta procedure registra o pagamento de uma compra e atualiza o status da compra para **Concluída**.

**Parâmetros:**

* p\_cd\_compra (INTEGER): Código da compra para a qual o pagamento está sendo registrado.

**Funcionamento:**

1. Obtém a data e hora atual em v\_dt\_pagamento.
2. Insere um registro de pagamento na tabela pagamento associado ao código da compra.
3. Atualiza o status da compra na tabela compra para "Concluída".
4. Executa COMMIT para confirmar a transação.

**Utilização:** Esta procedure é útil para sistemas de compras, permitindo registrar pagamentos e atualizar o status da compra como completa.

## Principais Funções do RPA

* **Funções conexao\_1ano() e conexao\_2ano()**: Ambas são funções simples que criam e retornam uma conexão com os respectivos bancos de dados usando as URLs de conexão armazenadas nas variáveis de ambiente. Em caso de falha, capturam e exibem o erro sem interromper completamente a execução.
* **Função insert()**: Recebe uma tabela e valores para inserir no banco de dados. Constrói um comando SQL com placeholders para evitar injeções SQL e insere os valores na tabela do 2º ano. Em caso de erro, executa um rollback para desfazer qualquer alteração parcial.
* **Função update()**: Recebe uma tabela, campo e valor a serem atualizados, e um identificador para localizar o registro correto. Realiza um comando SQL UPDATE no banco de dados do 2º ano. Se algum problema ocorrer, realiza um rollback para desfazer a atualização e imprime o erro.
* **Funções soft\_delete() e hard\_delete()**: São funções que excluem dados de acordo com dois métodos. O soft\_delete remove o registro de forma lógica (marcando-o como excluído), enquanto hard\_delete define uma data de desativação, sem excluir fisicamente o dado, para que possa ser reativado se necessário.
* **Função count()**: Compara a quantidade de registros entre as tabelas do 1º e 2º ano. Retorna as contagens de cada banco, para que a função buscar() possa verificar e sincronizar as diferenças de registros entre os bancos de dados.
* **Função buscar()**: É a função mais completa, projetada para garantir que as tabelas entre os bancos de dados estejam sincronizadas. Verifica se há registros faltantes ou diferentes entre as tabelas e realiza as operações necessárias:
  + **Inserção/Atualização**: Se houver registros no banco do 1º ano que não estão no banco do 2º ano, realiza inserções ou atualizações. Compara também valores de cada coluna, garantindo que ambos os bancos estejam em sincronia.
  + **Exclusão**: Caso existam registros no banco do 2º ano que não estão no do 1º, realiza uma exclusão lógica (soft delete) ou física (hard delete), dependendo da tabela.

O RPA foi projetado para lidar com exceções de forma robusta, capturando erros de conexão, erros durante inserções ou atualizações e falhas de configuração das variáveis de ambiente. Sempre que ocorre um erro, ele é exibido para o usuário, e a função que o gerou executa um rollback antes de fechar a conexão, garantindo que o banco de dados não fique com dados incompletos ou inconsistentes. Além disso, o RPA inclui queries SQL para consultas padronizadas, como query\_produto e query\_avatar, usadas para recuperar registros de produtos e avatares de maneira específica. Elas verificam se há atualizações em um período recente (últimos 30 minutos), mantendo o banco de dados sempre atualizado com o mínimo de impacto nos dados.

A manutenção do RPA envolve principalmente a atualização do arquivo .**env** com novas credenciais, quando necessário, e a adição de campos ou tabelas à medida que o banco de dados evolui. Além disso, o RPA foi projetado para ser escalável, de modo que novas funcionalidades possam ser adicionadas sem a necessidade de modificar muito o código existente.

## **Indíces**

No intuito de otimizar a performance das consultas em nosso banco de dados, foram criados índices nas colunas que representam as chaves primárias das tabelas usuario, evento e compra. Esses índices são fundamentais para melhorar a eficiência das operações de busca, especialmente nas tabelas mais utilizadas do sistema.

### Detalhes dos Índices Criados:

1. **Índice na Tabela usuario**:
   * **Nome do Índice**: **idx\_usuario**
   * **Coluna**: **id\_usuario**
   * **Descrição**: Este índice permite um acesso mais rápido às informações dos usuários, facilitando as consultas que filtram ou buscam por usuários específicos. Com um grande volume de dados na tabela usuario, esse índice se torna essencial para garantir uma resposta ágil em operações frequentes.
2. **Índice na Tabela evento**:
   * **Nome do Índice**: **idx\_evento**
   * **Coluna**: **id\_evento**
   * **Descrição**: A tabela evento armazena informações sobre diversos eventos, e este índice otimiza as buscas relacionadas a eventos específicos. Ao criar um índice sobre a coluna **id\_evento**, conseguimos acelerar significativamente as consultas que dependem desta informação.
3. **Índice na Tabela compra**:
   * **Nome do Índice**: **idx\_compra**
   * **Coluna**: **id\_compra**
   * **Descrição**: A tabela compra contém registros de transações e, com o índice na coluna **id\_compra**, garantimos um acesso mais eficiente a essas informações. Isso é especialmente importante em operações que envolvem análises de compra, permitindo que o sistema responda rapidamente às consultas.

**Importância dos Índices**

Os índices desempenham um papel crucial na otimização do banco de dados, pois reduzem o tempo necessário para localizar e acessar registros específicos. Em um ambiente onde as tabelas contêm um volume significativo de dados, como é o caso das tabelas **usuario**, evento e compra, a implementação de índices adequados é vital para manter a performance e garantir que as operações do sistema sejam realizadas de maneira eficiente e eficaz.

## **Querys complexas do MongoDB**

### **Media\_notas\_evento**

**Descrição**: Calcula a média das notas dadas a cada evento, ordenando os eventos pela média de notas.

**Funcionamento**:

1. Agrupa os documentos na collection **avaliacao** por **cd\_evento**.
2. Calcula a média das notas para cada evento com $avg.
3. Ordena os eventos de acordo com a média de notas em ordem decrescente.

### **Eventos\_populares**

**Descrição**: Esta query agrupa as avaliações por evento, mostrando o número total de avaliações, o nome e a média de nota de cada evento, e exibe os cinco eventos mais avaliados. Essa consulta permite identificar os eventos que tiveram maior engajamento em termos de avaliações e a qualidade média percebida pelos participantes.

**Funcionamento**:

1. **Lookup**: Realiza uma junção com a collection evento para adicionar detalhes do evento (como nome) aos documentos da collection **avaliacao**, com base no campo **cd\_evento.**
2. **Unwind**: Expande o campo evento de um **array** para um único objeto, permitindo acessar diretamente os detalhes do evento.
3. **Match**: Filtra os documentos para incluir apenas eventos do anunciante específico com **cd\_anunciante** igual a 15.
4. **Group**: Agrupa os documentos por cd\_evento, calculando o número total de avaliações ($sum), o nome do evento **($first)**, e a média das notas **($avg).**
5. **Sort**: Ordena os eventos em ordem decrescente pelo total de avaliações.
6. **Limit**: Limita o resultado aos cinco eventos mais avaliados.

### **Média\_nota\_total**

**Descrição**: Calcula a média geral das notas para todos os eventos do anunciante, permitindo avaliar a satisfação geral dos usuários em relação aos eventos oferecidos pelo anunciante específico.

**Funcionamento**:

1. **Lookup**: Realiza uma junção com a collection evento para obter detalhes de cada evento associado à avaliação.
2. **Unwind**: Expande o campo evento para acessar diretamente as informações relacionadas ao evento.
3. **Match**: Filtra os documentos para considerar apenas os eventos do anunciante com **cd\_anunciante** igual a 15.
4. **Group**: Agrupa todos os documentos (sem um campo específico), calculando a média geral das notas **($avg)** em todos os eventos do anunciante.
5. **Resultado**: Retorna o valor da média geral de nota para o anunciante.

### **Média\_notas\_evento**

**Descrição**: Calcula a média das notas para cada evento do anunciante, permitindo identificar quais eventos foram mais bem avaliados pelos usuários, ordenados pela média de nota.

**Funcionamento**:

1. **Lookup**: Realiza uma junção com a collection evento para incluir detalhes do evento.
2. **Unwind**: Expande o campo evento para acessar diretamente os detalhes do evento.
3. **Match**: Filtra os documentos para incluir apenas eventos do anunciante com **cd\_anunciante** igual a 15.
4. **Group**: Agrupa os documentos por **cd\_evento**, calculando a média das notas para cada evento com **$avg.**
5. **Sort**: Ordena os eventos em ordem decrescente pela média das notas, permitindo que os eventos mais bem avaliados apareçam primeiro.

Essas explicações fornecem uma visão clara dos objetivos e processos de cada query complexa utilizada no **MongoDB**, ajudando a contextualizar a funcionalidade de cada uma na análise de dados para o aplicativo.