Otimizando um marketplace

O marketplace é um e-commerce mediado por uma empresa, em que vários lojistas se inscrevem e vendem seus produtos. Essa loja virtual funciona de forma que o cliente pode acessar um site e comprar itens de diferentes varejistas em uma só plataforma. A plataforma de marketplace XXX está recebendo reclamações sobre o tempo de carregamento nas buscas realizadas no site pelos clientes e lojistas, e, por conta disso, resolveu contratar **VOCÊ**, um especialista em banco de dados, para resolver o problema que já foi diagnosticado pela equipe interna como lentidão nas consultas ao banco de dados.

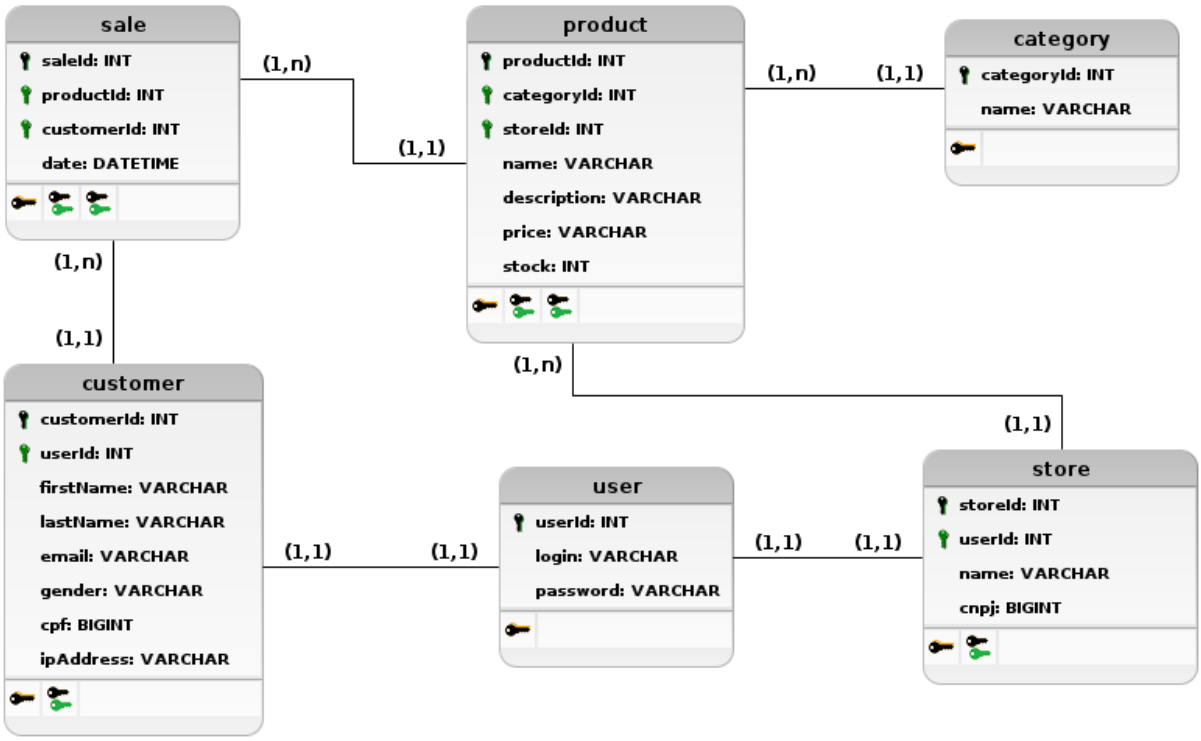
O aprimoramento de desempenho de banco de dados deve ser baseado nos seguintes pilares:

- Otimização de esquema;
- Otimização de consultas;
- Otimização de indexação.

O seu contratante resolveu disponibilizar apenas uma parte de sua base de dados e disponibilizou alguns materiais a serem utilizados como referência durante o processo de otimização.

Modelo Lógico

O modelo a seguir representa a parte da base de dados disponibilizada pelo seu contratante.



Modelo Físico

O modelo a seguir faz o detalhamento das estruturas de dados que foram utilizadas para a construção do banco de dados fisicamente (OBS.: não utilize esses comandos para criar as tabelas na resolução do exercício, e sim o *marketplace.sql* disponibilizado já com os dados).

```
CREATE TABLE category (
  categoryId INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  PRIMARY KEY (categoryId)
);

CREATE TABLE user (
  userId INT NOT NULL AUTO_INCREMENT,
  login VARCHAR(50) NOT NULL,
  password VARCHAR(50) NOT NULL,
  PRIMARY KEY (userId)
);

CREATE TABLE customer (
  customerId INT NOT NULL AUTO_INCREMENT,
  firstName VARCHAR(50) NOT NULL,
  lastName VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
  gender VARCHAR(50) NOT NULL,
  cpf BIGINT NOT NULL,
  ipAddress VARCHAR(20) NOT NULL,
  userId INT NOT NULL,
  PRIMARY KEY (customerId),
  CONSTRAINT customer_user_FK FOREIGN KEY (userId) REFERENCES user (userId)
);
```

```
CREATE TABLE store (
  storeId INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  cnpj BIGINT NOT NULL,
  userId INT NOT NULL,
  PRIMARY KEY (storeId),
    CONSTRAINT store_user_FK FOREIGN KEY (userId) REFERENCES user (userId)
);

CREATE TABLE product (
  productId INT NOT NULL AUTO_INCREMENT,
  categoryId INT NOT NULL,
  storeId INT NOT NULL,
  name VARCHAR(50) NOT NULL,
  description TEXT NOT NULL,
  price VARCHAR(50) NOT NULL,
  stock INT NOT NULL,
  PRIMARY KEY (productId),
  CONSTRAINT product_category_FK FOREIGN KEY (categoryId) REFERENCES category (categoryId),
  CONSTRAINT product_store_FK FOREIGN KEY (storeId) REFERENCES store (storeId)
);

CREATE TABLE sale (
  saleId INT NOT NULL AUTO_INCREMENT,
  productId INT NOT NULL,
  customerId INT NOT NULL,
  date DATETIME NOT NULL,
  PRIMARY KEY (saleId),
  CONSTRAINT sale_product_FK FOREIGN KEY (productId) REFERENCES product (productId),
  CONSTRAINT sale_customer_FK FOREIGN KEY (customerId) REFERENCES customer (customerId)
);
```

Consultas

As consultas exibidas a seguir foram disponibilizadas para você, especialista em banco de dados, pela plataforma de marketplace.

```
-- Consulta #1
-- Utilizada para verificar número de compras em 2015.
SELECT SQL_NO_CACHE COUNT(*) AS `Número de compras em 2015`
FROM sale
WHERE YEAR(date) = 2015
GROUP BY YEAR(date);

-- Consulta #2
-- Utilizada para verificar o número de produtos com valor menor
-- que R$ 1.000,00 e com estoque baixo (menor que 100).
-- Dentro da aplicação, o usuário pode escolher diferentes faixas
-- de preço e diferentes quantidades em estoque.
SELECT SQL_NO_CACHE COUNT(*) AS `Quantidade`
FROM product
WHERE price BETWEEN 0 AND 1000
    AND stock <= 100;

-- Consulta #3
-- Utilizada para buscar os nomes das 10 primeiras categorias cadastradas.
SELECT SQL_NO_CACHE name AS `Nome da categoria`
FROM category
LIMIT 10;
```

```
-- Consulta #4
-- Utilizada para buscar os e-mails dos usuários que fizeram compras
-- no intervalo de 1 ano até a data atual e exibir em um relatório paginado
-- com 50 itens por página.
SELECT SQL_NO_CACHE *
FROM sale s
INNER JOIN customer c
ON s.customerId = c.customerId
WHERE date > CURDATE() - INTERVAL 1 YEAR;

-- Consulta #5
-- Utilizada para buscar login e password de um usuário específico
SELECT SQL_NO_CACHE login, password
FROM user
WHERE userId = 11100;

-- Consulta #6
-- Utilizada para exibir um relatório não paginado com o nome
-- de todas as lojas cadastradas.
SELECT SQL_NO_CACHE *
FROM store;
```

Base de dados

A parte da base de dados disponibilizada é composta por 6 tabelas, sendo elas:

- **category**→Responsável por armazenar as categorias dos produtos. Possui 1.000 registros.
- **customer**→Reponsável por armazenar os dados dos clientes. Possui 15.000 registros.
- **product**→Responsável por armazar os produtos cadastrados pelos lojistas. Possui 100.000 registros.
- **sale**→Responsável por armazenar as vendas realizadas (dados de 2011 até 2019). Possui 2.000.000 registros.
- **store**→Responsável por armazenar os dados dos lojistas. Possui 5.000 registros.
- **user**→Responsável por armazenar os dados de login dos usuários (clientes e lojistas). Possui 20.000 registros.

Exercício

Os seguintes problemas devem ser resolvidos:

- Pelo menos uma coluna envolvida nas consultas possui tipagem inadequada que se propagou desde o modelo lógico.
- Pelo menos duas das consultas disponibilizadas possuem problemas no modo como foram construídas para atender à demanda que necessitavam.
- Pelo menos dois índices importantes não existe.

Deve-se provar quantitativamente que houve aumento no desempenho das consultas. Para fazer isso, realize *benchmarks* com cada uma das consultas e observe a taxa de transferência média e a latência média de cada uma delas antes de fazer modificações no esquema, nas consultas e/ou em índices. Após as modificações, realize novos *benchmarks* para verificar a melhoria obtida.

Sendo assim, para cada uma das consultas, deve-se responder se foi necessário ou não aplicar alguma melhoria. Caso tenha sido necessário, explicar o porquê, listar comandos necessários para executar dentro do SGBD MySQL e colocar, em números, a porcentagem de aumento da taxa de transferência média e a porcentagem de redução da latência média.

Exemplo de resposta:

Foi possível aumentar o desempenho da consulta X alterando o tipo da coluna COLUNA1 da tabela TABELA1 de TIPO1 para TIPO2, que é um tipo mais adequado para os dados que a coluna armazena.

O desempenho também foi melhorado criando um índice para a coluna COLUNA2, que é necessário tendo em vista o filtro que sempre é feito nessa com base nessa coluna.

```
ALTER TABLE tabela1 MODIFY coluna1 TIPO2 NOT NULL;
CREATE INDEX coluna2_idx ON tabela1(`coluna2`);
```

Testes realizados

Antes	**Depois**
Taxa de transferência média: 1.997,20 t/s	Taxa de transferência média: 9.596,96 t/s
Latência média: 8,00 ms	Latência média: 1,67 ms

Aumento da taxa de transferência média: 380,52% (ou seja, 1.997,20 + 380,52% = 9.596,96)

Diminuição da latência média: 79,12% (ou seja, 8,00 - 79,12% = 1,67)

Dicas para resolução

- Manter todas as consultas no único arquivo *consultas.lua* disponibilizado, e, a cada execução do S*ysbench* deixar as consultas que não devem ser testadas "comentadas" e a consulta que deve ser testada "descomentada".

- Para calcular a porcentagem de aumento/diminuição, pode-se utilizar (onde *vn* corresponde ao valor novo e *va* corresponde ao valor antigo):
  - ▼ $porcentagem = ((vn/va) - 1) * 100$