artigo

sábado, 23 de agosto de 2025 10:46

JOIN e seus tipos (gratuito, português, Artigo)
JOIN e seus tipos é um artigo escrito por Danielle Oliveira que explica sobre os tipos de JOINs e como podemos utilizá-los para realizar consultas.

Joins, Views e transações

sábado, 23 de agosto de 2025 14:41

Durante os nossos estudos, vamos conhecer o projeto da empresa Serenatto café e bistrô. Atualmente, a empresa armazena as suas informações em diversas fontes de dados, e nosso trabalho é centralizar todos esses dados em um só lugar e extrair informações essenciais para os gestores da empresa. Então, vamos utilizar o SQLite Online, para criar as nossas tabelas e executar as nossas consultas utilizando a linguagem SQL.

Então, para desenvolver o projeto, vamos iniciar conhecendo as informações necessárias para criar as nossas tabelas:

1. **Produtos**:
   - **ID**: Chave primária.
   - **Nome**: Nome do produto (café, bebidas, pratos, sobremesas, etc.).
   - **Descrição**: Descrição do produto.
   - **Preço**: Preço do produto.
   - **Categoria**: Categoria do produto (café, chá, almoço, jantar, sobremesa, etc.).
2. **Clientes**:
   - **ID**: Chave primária.
   - **Nome**: Nome completo do cliente.
   - **Telefone**: Número de telefone do cliente.
   - **Email**: Endereço de e-mail do cliente.
   - **Endereco**: Endereço do cliente.
3. **Pedidos**:
   - **ID**: Chave primária.
   - **IDCliente**: Chave estrangeira relacionando ao cliente.
   - **DataHoraPedido**: Data e hora do pedido.
   - **Status**: Status do pedido (em andamento, concluído, entregue, etc.).
4. **Itens de Pedido**:
   - **IDPedido**: Chave estrangeira relacionando ao pedido.
   - **IDProduto**: Chave estrangeira relacionando ao produto.
   - **Quantidade**: Quantidade do produto no pedido.
   - **PrecoUnitario**: Preço unitário do produto no momento do pedido.
5. **Colaboradores**:
   - **ID**: Chave primária.
   - **Nome**: Nome completo do colaborador.
   - **Cargo**: Cargo ou posição do colaborador.
   - **DataContratacao**: Data de contratação.
   - **Telefone**: Número de telefone do colaborador.
   - **Email**: Endereço de e-mail do colaborador.
6. **Fornecedores**:
   - **ID**: Chave primária.
   - **Nome**: Nome da empresa do fornecedor.
   - **Contato**: Nome do contato no fornecedor.
   - **Telefone**: Número de telefone do fornecedor.
   - **Email**: Endereço de e-mail do fornecedor.

Create table

sábado, 23 de agosto de 2025 17:05

```
CREATE TABLE colaboradores (
  ID TEXT PRIMARY KEY,
  Nome VARCHAR(255) NOT NULL,
  Cargo VARCHAR(100),
  DataContratacao DATE,
  Telefone VARCHAR(20),
  Email VARCHAR(100),
  Rua VARCHAR(100) NOT NULL,
  Bairro VARCHAR(100) NOT NULL,
  Cidade VARCHAR(100) NOT NULL,
  Estado VARCHAR(2) NOT NULL,
  cep VARCHAR(8) NOT NULL
);
```

INSERT [copulando]

sábado, 23 de agosto de 2025 17:46

```
INSERT INTO colaboradores (ID, Nome, Cargo, DataContratacao, Telefone, Email,
Rua, Bairro, Cidade, Estado, CEP) VALUES
(1, 'Carlos Silva', 'Gerente', '2022-03-15', '115551234', 'carlos.silva@email.com',
'Rua do Comércio - 258', 'Centro', 'São Paulo', 'SP', '01000001');
```

OPERADORES

sábado, 23 de agosto de 2025 17:18

UNION
UNION ALL
EXCEPT
INTERSECT

Além do UNION e UNION ALL, em alguns SGBDs podemos ter também os operadores EXCEPT e INTERSECT, que são dois operadores relacionais que são usados em consultas SQL para realizar operações de conjunto em tabelas ou resultados de consultas. Eles permitem que você compare conjuntos de dados entre duas consultas e extraia informações específicas com base nas diferenças ou semelhanças entre esses conjuntos.
1 - EXCEPT:
O operador EXCEPT é usado para retornar todas as linhas que estão presentes na primeira consulta (conjunto A) e não estão presentes na segunda consulta (conjunto B). Em outras palavras, ele subtrai o conjunto B do conjunto A.
O operador INTERSECT é usado para retornar todas as linhas que estão presentes tanto na primeira consulta (conjunto A) quanto na segunda consulta (conjunto B). Em outras palavras, ele retorna a interseção dos dois conjuntos.

Observações importantes:
As duas consultas dentro de EXCEPT ou INTERSECT devem retornar o mesmo número de colunas e essas colunas devem ter tipos de dados compatíveis.
EXCEPT e INTERSECT geralmente são usados em consultas SELECT, mas também podem ser usados em subconsultas ou em combinação com outros operadores SQL, como UNION.
Esses operadores são úteis para encontrar diferenças ou semelhanças entre conjuntos de dados, o que pode ser valioso em situações como validação de dados, comparação de listas e muito mais. Eles são especialmente úteis quando você precisa realizar operações complexas de conjunto em dados de tabelas relacionadas.

Desafio 1

sábado, 23 de agosto de 2025 17:47

Você é um professor e deseja identificar o aluno que obteve a maior nota em sua disciplina. Você tem duas tabelas em seu banco de dados: "Alunos" e "Notas". A tabela "Notas" registra as notas dos alunos em sua disciplina. Seu desafio é encontrar o aluno com a maior nota.
Tabela "Alunos":

- ID_aluno (chave primária)
- Nome
- Curso

Tabela "Notas":

- ID_nota (chave primária)
- ID_aluno (chave estrangeira)
- Nota

Seu desafio é criar uma consulta SQL que retorne o nome do aluno que obteve a maior nota em sua disciplina.

O primeiro passo foi encontrar a maior nota registrada.

```
SELECT MAX(Nota)
    FROM Notas
```

Em seguida, encontrar o ID do aluno que obteve essa nota máxima.

```
SELECT ID_aluno
 FROM Notas
 WHERE Nota = (
    SELECT MAX(Nota)
    FROM Notas
 )
```

Por fim, retorne o nome do aluno que obteve a maior nota.

```
SELECT Nome
FROM Alunos
WHERE ID_aluno = (
  SELECT ID_aluno
  FROM Notas
  WHERE Nota = (
    SELECT MAX(Nota)
    FROM Notas
  )
);
```

JOIN

sábado, 23 de agosto de 2025 11:10

Imagine que você tem um quebra-cabeça: cada peça representa uma informação que está guardada em uma tabela diferente no seu banco de dados. Para ver a imagem completa, você precisa encaixar essas peças. No mundo dos bancos de dados, esse "encaixe" é feito através de uma operação chamada JOIN.

O que é JOIN?
JOIN é um comando usado em SQL (a linguagem de consulta de bancos de dados) para combinar linhas de duas ou mais tabelas, baseado em uma coluna relacionada entre elas. Isso é muito útil quando você quer ver informações que estão distribuídas em diferentes tabelas.

Tipos de JOIN
Há vários tipos de JOIN, mas vamos focar nos mais comuns:
INNER JOIN: Combina linhas de duas tabelas quando há uma correspondência entre as colunas especificadas. Se não houver correspondência, a linha não aparece no resultado.
LEFT JOIN (ou LEFT OUTER JOIN): Retorna todas as linhas da tabela da esquerda (a primeira mencionada) e as linhas correspondentes da tabela da direita. Se não houver correspondência, os resultados da tabela da direita terão valores NULL.
RIGHT JOIN (ou RIGHT OUTER JOIN): É o oposto do LEFT JOIN. Retorna todas as linhas da tabela da direita e as correspondentes da esquerda. Novamente, se não houver correspondência, os resultados da tabela da esquerda terão valores NULL.
FULL JOIN (ou FULL OUTER JOIN): Combina as linhas de ambas as tabelas. Se não houver correspondência, ainda assim as linhas serão mostradas, com NULL no lado sem correspondência.

Motivação para Usar JOIN
A principal razão para usar JOIN é que, em muitos casos, os dados que precisamos estão espalhados em várias tabelas. JOINs nos permitem combinar esses dados de maneira significativa para relatórios, análises e tomada de decisão.

Conclusão
JOIN é uma ferramenta poderosa em SQL que nos ajuda a conectar as peças do quebra-cabeça dos nossos dados. Se você está aprendendo sobre bancos de dados, entender como os JOINs funcionam é essencial para manipular e compreender as informações de forma eficaz. Experimente os diferentes tipos de JOIN e veja como eles podem resolver problemas complexos de junção de dados!

Desafio 2

sábado, 23 de agosto de 2025 13:58

LEFT JOIN
LEFT JOIN
FULL JOIN

Você é um(a) gerente de uma loja online e deseja identificar os clientes que fizeram pedidos e aqueles que ainda não fizeram nenhum pedido em sua loja. Você tem duas tabelas em seu banco de dados: "Clientes" e "Pedidos". A tabela "Clientes" contém informações sobre os clientes, enquanto a tabela "Pedidos" registra informações sobre os pedidos feitos por esses clientes. Seu desafio é encontrar os clientes que não fizeram pedidos em sua loja.
Tabela "Clientes":
ID (chave primária)
Nome
Email
Tabela "Pedidos":
ID (chave primária)
IDcliente (chave estrangeira)
DataPedido
Seu desafio é criar uma consulta SQL que retorne o nome dos clientes que ainda não fizeram pedidos em sua loja.

Aqui temos um exemplo de como podemos resolver este desafio:
Realize um RIGHT JOIN entre as tabelas "Clientes" e "Pedidos" para encontrar todos os clientes, incluindo aqueles que não fizeram pedidos.

```
SELECT *
FROM Pedidos p
RIGHT JOIN clientes c
ON c.ID = p.IDcliente
```

Filtre os resultados para encontrar os clientes que não têm pedidos (ou seja, onde o campo ID_pedido é nulo).

```
SELECT *
FROM Pedidos p
RIGHT JOIN clientes c
ON c.ID = p.IDcliente
WHERE p.IDcliente IS NULL;
```

Retorne o nome dos clientes que não fizeram pedidos.

```
SELECT c.nome
FROM Pedidos p
RIGHT JOIN clientes c
ON c.ID = p.IDcliente
WHERE p.IDcliente IS NULL;
```

A tabela Pedidos contém informações sobre os pedidos feitos pelos clientes,

enquanto a tabela ItensMenu contém detalhes dos itens disponíveis no menu. Para compreender melhor quais itens do menu não estão vendendo bem, seu gerente pede que você crie uma consulta SQL que liste todos os itens do menu que não tiveram nenhum pedido nas últimas semanas

```
SELECT *
FROM ItensMenu
LEFT JOIN Pedidos
ON ItensMenu.id = Pedidos.item_id
WHERE Pedidos.id IS NULL;
```

Esta consulta é apropriada porque o LEFT JOIN inclui todos os registros da tabela ItensMenu e os registros correspondentes da tabela Pedidos, onde existem. A cláusula WHERE Pedidos.id IS NULL garante que serão listados apenas os itens que não têm pedidos associados.

CROSS JOIN

quinta-feira, 21 de agosto de 2025 17:07

O CROSS JOIN no SQL é um tipo de junção que combina todas as linhas de uma tabela com todas as linhas de outra tabela, criando um produto cartesiano. Em outras palavras, ele forma todas as combinações possíveis de registros entre as duas tabelas envolvidas na junção. O resultado é uma tabela resultante que contém o número de linhas igual ao produto do número de linhas das duas tabelas originais.

SINTAXE

```
SELECT colunas
FROM tabela1
CROSS JOIN tabela2;
```

Pontos importantes sobre o CROSS JOIN:

1. **Combinação Completa:** O CROSS JOIN não utiliza uma condição de correspondência específica, ele simplesmente combina todas as linhas da primeira tabela com todas as linhas da segunda tabela. Isso significa que ele gera todas as combinações possíveis, mesmo que não haja relação lógica entre os dados.
2. **Grande Número de Linhas:** O uso imprudente do CROSS JOIN pode resultar em um grande número de linhas na tabela resultante, o que pode afetar o desempenho e a eficiência das consultas.
3. **Uso Limitado:** O CROSS JOIN é usado com menos frequência do que outros tipos de junções, como INNER JOIN, LEFT JOIN e RIGHT JOIN, porque geralmente não é necessário combinar todas as linhas de duas tabelas. No entanto, pode ser útil em casos específicos, como ao gerar combinações de valores para análises estatísticas.

View

sábado, 23 de agosto de 2025 12:01

Uma VIEW em SQL é uma consulta armazenada que cria uma representação virtual de uma tabela a partir dos resultados de uma consulta SQL.

```
CREATE VIEW ViewTotalPorPedido AS
SELECT
   P.ID AS ID_Pedido,
   P.DataHoraPedido,
   C.Nome AS NomeCliente,
   SUM(IP.Quantidade * IP.PrecoUnitario) AS TotalPorPedido
FROM Pedidos AS P
JOIN Clientes AS C ON P.ID_Cliente = C.ID
JOIN ItensPedido AS IP ON P.ID = IP.ID_Pedido
GROUP BY P.ID, P.DataHoraPedido, C.Nome;
```

Na sequência, passamos SELECT nome, endereco FROM clientes. Dessa forma, estamos informando que queremos exatamente essas informações, nome e endereço, da tabela de clientes.

```
CREATE VIEW ViewCliente AS
SELECT nome, endereco FROM clientes;
```

Ao executar, obtemos uma VIEW que tem armazenado virtualmente essas informações dos nossos clientes.
Visualização dos 5 primeiros registros. Para conferir o retorno na íntegra, execute-o na sua máquina.

nome	endereco
Maria Silva	Rua das Flores, 123, Cidade A
João Pereira	Av. Principal, 456, Cidade B
Ana Rodrigues	Rua Central, 789, Cidade C
Pedro Alves	Travessa dos Sonhos, 58, Cidade D
Sofia Santos	Alamedas das Artes, 321, Cidade E

Se visualizarmos na lateral esquerda, abaixo das nossas tabelas, temos agora uma área de View, onde consta ViewCliente.
Podemos chamar a nossa view normalmente como uma tabela:

```
SELECT * FROM ViewCliente;
```

Agora, vamos criar uma *view* com a consulta que calcula o valor total dos nossos pedidos. A chamaremos de `ViewValorTotalPedido`.

```
CREATE VIEW ViewValorTotalPedido AS
SELECT p.id, c.nome, p.datahorapedido, SUM(ip.precounitario) AS ValorTotalPedido
FROM clientes c
JOIN pedidos p ON c.id = p.idcliente
JOIN itenspedidos ip ON p.id = ip.idpedido
GROUP BY p.id, c.nome;
```

Feito isso, se olharmos a lateral esquerda novamente, veremos ViewValorTotalPedido dentro de View.
Se executarmos a consulta passando ViewValorTotalPedido, obteremos exatamente o resultado da nossa consulta.

```
SELECT * FROM ViewValorTotalPedido;
```

Visualização dos 5 primeiros registros. Para conferir o retorno na íntegra, execute-o na sua máquina.

id	nome	datahorapedido	ValorTotalPedido
1	Ana Maria Silva	2023-01-02 08:15:00	9
10	Helena Lima	2023-01-06 10:15:00	18
101	Isabel Gonçalves	2023-04-09 12:45:00	29.5
102	Diogo Fernandes	2023-04-09 13:30:00	25

103	Carlos Gomes	2023-05-01 08:15:00	19.5

Criamos a nossa view com um comando que será executado frequentemente. Com essa tabela virtual, conseguiremos trabalhar normalmente como em outras tabelas.
Na sequência, vamos explorar essa view e realizar diversas consultas e filtros que vão trazer resultados importantes sobre o faturamento do Serenatto Café & Bistrô!

trigger

sábado, 23 de agosto de 2025 18:30

A trigger automatiza o cálculo do faturamento em tempo real, garantindo a atualização diária dos dados. Isso otimiza as operações, reduz erros humanos e fornece informações precisas para decisões estratégicas. Além disso, a segurança e rastreabilidade dos dados são aprimoradas, facilitando auditorias e permitindo a reutilização da trigger em outras unidades.

TRIGGER
Uma Trigger é um procedimento armazenado que é executado automaticamente em resposta a um evento específico em uma tabela.

```
CREATE TRIGGER CalculoFaturamentoDiario
AFTER INSERT ON ItensPedido
FOR EACH ROW
BEGIN
   DELETE FROM FaturamentoDiario ;
   INSERT INTO FaturamentoDiario (Dia, FaturamentoTotal)
   SELECT
    DATE(P.DataHoraPedido) AS Dia,
    SUM(IP.Quantidade * IP.PrecoUnitario) AS Faturamento
  FROM Pedidos AS P
  JOIN ItensPedido AS IP ON P.ID = IP.ID_Pedido
  GROUP BY Dia
  ORDER BY Dia;
END;
```

CASCADE vs. RESTRICT em bancos de dados relacionais

sábado, 23 de agosto de 2025 09:49

Quando você está aprendendo sobre bancos de dados, especialmente os do tipo relacional, você descobre que eles são como um grande quebra-cabeça onde cada peça é uma informação que precisa se encaixar perfeitamente com as outras. Para manter esse quebra-cabeça organizado, existem regras que nos ajudam a entender o que acontece quando queremos alterar ou remover uma dessas peças. Duas dessas regras são chamadas de CASCADE e RESTRICT, e elas são aplicadas quando temos operações que afetam várias partes do quebra-cabeça ao mesmo tempo, ou seja, quando queremos deletar ou atualizar dados que estão ligados por uma relação chamada de chave estrangeira. Vamos entender um pouco sobre cada uma delas.

CASCADE

Imagine que temos um banco de dados de uma escola e uma tabela chamada "Turmas" que está relacionada com outra tabela chamada "Alunos". Se a turma A for deletada, o que acontece com os alunos que estão nessa turma? CASCADE é como uma onda que leva tudo que está conectado com ela. Se deletarmos a turma A e tivermos configurado CASCADE para a relação entre as tabelas, todos os alunos relacionados à turma A também serão deletados automaticamente.

Exemplo de código com CASCADE:

```
ALTER TABLE Alunos
ADD CONSTRAINT fk_turma
FOREIGN KEY (turma_id) REFERENCES Turmas(turma_id)
ON DELETE CASCADE;
```

RESTRICT

Por outro lado, RESTRICT age como um guarda de trânsito que diz "Você não pode passar!". Usando o mesmo exemplo da escola, se tentarmos deletar a turma A e houver alunos relacionados a ela com a regra RESTRICT aplicada, o banco de dados vai impedir essa ação e mostrará um erro. Isso garante que não vamos perder informações importantes acidentalmente.

Exemplo de código com RESTRICT:

```
ALTER TABLE Alunos
ADD CONSTRAINT fk_turma
FOREIGN KEY (turma_id) REFERENCES Turmas(turma_id)
ON DELETE RESTRICT;
```

Operações de Grande Escala

Quando trabalhamos com muitos dados, como em bancos de dados de grandes empresas, é crucial escolher entre CASCADE e RESTRICT com cuidado. CASCADE pode ser útil, mas também perigoso se deletarmos dados que não deveríamos. RESTRICT é mais seguro, mas pode dificultar a atualização e a exclusão de dados em massa.

TRANSACTION

sábado, 23 de agosto de 2025 10:08

Geralmente, a cláusula WHERE é empregada para aplicar filtros durante deleções ou atualizações, manipulando apenas o dado específico desejado. No entanto, e se atualizarmos sem usar a cláusula WHERE, ou se o computador travar no meio do processo, causando uma falha na execução? Mesmo ao usar o filtro, existe a possibilidade de execução inadequada.

As transações são como abrir um bloco no SQL, permitindo a execução de vários comandos, como DELETE e UPDATE. Se um desses comandos falhar, todos os outros também falharão, consolidando vários comandos em um único bloco. Suponhamos que realizamos um UPDATE:

```
UPDATE Pedidos SET Status = 'Concluído'
```

Iniciamos a transação com o BEGIN, que, traduzindo para o português, significa "início". Ao executar este comando, iniciamos um bloco, criando, assim, uma aba virtual.

```
BEGIN TRANSACTION;
```

Tomemos, por exemplo, um UPDATE, um DELETE e um SELECT.

```
BEGIN TRANSACTION;
SELECT * FROM clientes
UPDATE Pedidos SET Status = 'Concluído'
DELETE FROM clientes
```

Podemos utilizar o ROLLBACK. Com o ROLLBACK, todos esses comandos que executamos, serão revertidos.

```
ROLLBACK;
```

Por fim, utilizamos o COMMIT; para confirmar que desejamos que essas ações, neste caso, limitadas ao UPDATE, sejam confirmadas fisicamente no nosso banco de dados.

```
COMMIT;
```

As **transações(TRANSACTIONS)** são um outro recurso, uma camada a mais de segurança quando formos atualizar os nossos dados, manipular como atualizar ou até deletar.

COMMIT e ROLLBACK

sábado, 23 de agosto de 2025 10:18

As transações são uma parte fundamental da gestão de bancos de dados, permitindo que um conjunto de operações seja tratado como uma única unidade lógica e atômica. Isso significa que ou todas as operações de uma transação são concluídas com sucesso e confirmadas, ou nenhuma delas é realizada (rollback), para manter a consistência dos dados. Aqui estão os conceitos essenciais sobre transações:

Iniciando uma Transação:
Em SQL, você pode iniciar uma transação usando a instrução BEGIN TRANSACTION, BEGIN, ou simplesmente START TRANSACTION, dependendo do sistema de gerenciamento de banco de dados (DBMS) que você está usando. Por exemplo:

```
BEGIN TRANSACTION;
-- ou
BEGIN;
-- ou
START TRANSACTION;
```

Funcionamento do COMMIT:
O comando COMMIT é usado para confirmar todas as operações realizadas dentro de uma transação. Quando você emite o comando COMMIT, as alterações feitas nas tabelas durante a transação se tornam permanentes no banco de dados.

```
COMMIT;
```

Funcionamento do ROLLBACK:
O comando ROLLBACK é usado para desfazer todas as operações realizadas dentro de uma transação. Isso significa que todas as alterações feitas durante a transação são revertidas e o estado do banco de dados volta ao que estava antes do início da transação.

```
ROLLBACK;
```

Vantagens de Usar Transações:
Atomicidade: As transações garantem que todas as operações sejam concluídas com sucesso ou que nenhuma delas seja executada. Isso ajuda a manter a integridade dos dados, evitando estados inconsistentes no banco de dados.
Consistência: As transações garantem que o banco de dados permaneça em um estado consistente antes e depois da execução de uma sequência de operações. Se algo der errado, a transação pode ser revertida (rollback) para evitar inconsistências.
Isolamento: O isolamento permite que várias transações ocorram simultaneamente sem interferir umas nas outras. Os DBMSs geralmente oferecem níveis de isolamento para controlar o grau de concorrência e proteger a consistência dos dados.
Durabilidade: O comando COMMIT garante que as alterações em uma transação sejam permanentes, mesmo em caso de falha do sistema ou reinicialização do banco de dados.
Gerenciamento de Erros: As transações permitem que os erros sejam gerenciados de forma controlada. Se algo der errado, você pode optar por reverter a transação e manter o banco de dados em um estado consistente.
Trabalho em Equipe: Em ambientes multiusuário, as transações ajudam a coordenar o trabalho de várias sessões de usuários, garantindo que as operações sejam executadas de maneira ordenada e consistente.

Desvantagens de Usar Transações:
Bloqueio: Em sistemas com alto volume de transações concorrentes, o uso indevido de transações pode levar a bloqueios, onde uma transação aguarda a conclusão de outra, afetando o desempenho.
Complexidade: O uso de transações pode aumentar a complexidade do código e do design do banco de dados.
Uso de Recursos: Transações podem consumir recursos significativos do sistema, especialmente se não forem gerenciadas adequadamente.

Em resumo, as transações desempenham um papel fundamental na manutenção da consistência e da integridade dos dados em um banco de dados. Elas fornecem controle e segurança ao gerenciar operações complexas, evitando estados inconsistentes e permitindo o gerenciamento adequado de erros e conflitos em ambientes de banco de dados. No entanto, é importante usá-las com sabedoria e considerar as implicações de desempenho ao projetar sistemas que dependem fortemente delas.

Desafio 3

sábado, 23 de agosto de 2025 10:34

1- Traga todos os dados da cliente Maria Silva.
2- Busque o ID do pedidov e o ID do cliente dos pedidos onde o status esteja como entregue.
3- Retorne todos os produtos onde o preço seja maior que 10 e menor que 15.
4- Busque o nome e cargo dos colaboradores que foram contratados entre 2022-01-01 e 2022-06-31.
5- Recupere o nome do cliente que fez o primeiro pedido.
6- Liste os produtos que nunca foram pedidos.
7- Liste os nomes dos clientes que fizeram pedidos entre 2023-01-01 e 2023-12-31.
8- Recupere os nomes dos produtos que estão em menos de 15 pedidos.
9- Liste os produtos e o ID do pedido que foram realizados pelo cliente "Pedro Alves" ou pela cliente "Ana Rodrigues".
10- Recupere o nome e o ID do cliente que mais comprou(valor) no Serenatto.

1 -
```sql
SELECT *
FROM clientes
WHERE nome = 'Maria Silva'
```
2 -
```sql
SELECT id, idcliente
FROM pedidos
WHERE status = 'Entregue'
```
3 -
```sql
SELECT *
FROM produtos
WHERE preco > 10 and preco < 15
```
4 -
```sql
SELECT nome, cargo
FROM colaboradores
WHERE datacontratacao BETWEEN '2022-01-01' AND '2022-06-31';
```
5 -
```sql
SELECT nome
FROM clientes
WHERE id = (
SELECT idcliente
        FROM pedidos
    ORDER BY datahorapedido asc
       LIMIT 1);
```
6 -
```sql
SELECT nome
FROM produtos
WHERE id NOT IN (
SELECT idProduto
    FROM ItensPedido);
```
7 -
```sql
SELECT c.nome
FROM clientes AS c
INNER JOIN pedidos AS p ON c.id = p.idCliente
WHERE p.datahorapedido
BETWEEN '2023-01-01' AND '2023-01-31';
```
8 -

```sql
SELECT p.nome
FROM produtos AS p
INNER JOIN itensPedido AS ip
ON p.id = IP.idProduto
GROUP BY p.nome
HAVING COUNT(ip.idPedido) < 15;
```

9 -

```sql
SELECT p.nome, ip.idpedido
FROM produtos AS p
INNER JOIN itensPedido AS ip ON P.id = IP.idProduto
INNER JOIN pedidos AS pd ON ip.idPedido = pd.id
INNER JOIN clientes AS c ON pd.idCliente = C.id
WHERE c.nome IN ('Pedro Alves', 'Ana Rodrigues');
```

10 -

```sql
SELECT p.idCliente, c.nome, SUM(ip.quantidade * ip.precounitario) as Valortotal
FROM clientes c
INNER JOIN pedidos p ON c.id = p.idcliente
INNER JOIN itensPedido ip ON p.id = ip.idpedido
GROUP BY p.idCliente
ORDER BY Valortotal DESC
LIMIT 1;
```

De <https://cursos.alura.com.br/course/realizando-consultas-sql-joins-views-transacoes/task/149273>

update, delete e transações

sábado, 23 de agosto de 2025 10:41

Como atualizar os nossos dados utilizando o UPDATE, como excluir os dados com o DELETE, e como garantir uma camada a mais de segurança para os nossos dados.

Exemplos:

UPDATE
Com o UPDATE realizamos a atualização dos nossos dados.

```
UPDATE PRODUTOS SET Descricao = 'Croissant recheado com amêndoas.'
WHERE ID = 28;
```

Com este comando, estamos atualizando a descrição do produto que possui o ID 28.

DELETE
Com o DELETE realizamos a exclusão dos nossos dados das nossas tabelas.

```
DELETE FROM Produtos WHERE ID = 31;
```

Com este comando, estamos excluindo o produto, que possui o ID 31.

TRANSAÇÕES
Transações no SQL são usadas para garantir a consistência e integridade dos dados, permitindo que um conjunto de operações SQL seja tratado como uma única unidade atômica. Existem três principais comandos relacionados a transações no SQL: BEGIN TRANSACTION, COMMIT e ROLLBACK.

```
BEGIN TRANSACTION ;
```

Para encerrar uma transação, podemos utilizar o COMMIT ou ROLLBACK.

```
COMMIT ;
```

Para confirmar as alterações.

```
ROLLBACK;
```

Para reverter as alterações.

Desafio 4

sábado, 23 de agosto de 2025 10:48

Agora que você já concluiu os seus estudos no curso **Realizando consultas com SQL: Joins, Views e transações**, chegou o momento de continuar a desenvolver o desafio, onde você pode utilizar os conhecimentos adquiridos para montar novas consultas.
Para desenvolver esse desafio, você irá utilizar o banco de dados criado no desafio 1, mas não se preocupe, se você não construiu ou não tem mais o banco de dados disponível, baixe o banco já estruturado aqui e importe no SQLite online!

Contexto
Agora, vamos avançar na exploração do nosso banco de dados, abordando consultas mais complexas e avançadas que envolvem subconsultas, **JOIN**, views. Essas técnicas são poderosas e podem ser úteis para lidar com requisitos complexos em sistemas de gerenciamento escolar.

Desafio
Consulta 1: Buscar o nome do professor e a turma que ele é orientador
Consulta 2: Retornar o nome e a nota do aluno que possui a melhor nota na disciplina de Matemática
Consulta 3: Identificar o total de alunos por turma
Consulta 4: Listar os Alunos e as disciplinas em que estão matriculados
Consulta 5: Criar uma view que apresenta o nome, a disciplina e a nota dos aluno

Glossário SQL

sábado, 23 de agosto de 2025 10:51

GLOSSÁRI
O+SQL+-+...

links

sábado, 23 de agosto de 2025 16:32

https://awari.com.br/como-criar-uma-view-sql-passo-a-passo-guia-completo-para-iniciantes/

https://www.sqltutorial.org/sql-views/

https://www.alura.com.br/artigos/join-em-sql

https://www.youtube.com/watch?v=ANkrrduTtu0&t=55s

https://www.youtube.com/watch?v=tyJ476aNCYU&list=PLZumiKXZED9X9m5IJoT5N7k2eJAJzEbVq

https://sqliteonline.com/