E-fólio A | Folha de resolução para E-fólio

UNIDADE CURRICULAR: **Fundamentos de Bases de Dados**

CÓDIGO: **21053**

DOCENTE: **Paulo Pombinho**

A preencher pelo estudante

NOME: João Paulo Alves Correia Mendes Pires

N.º DE ESTUDANTE: 2203810

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 15 de novembro de 2024

TRABALHO / RESOLUÇÃO:

1) Atomicidade é uma das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) das transações em bases de dados. O princípio de atomicidade estabelece que uma transação deve ser tratada como uma unidade indivisível. Dado este princípio, a transação completa é executada com sucesso e seus efeitos são consolidados (commit), ou nenhuma parte da transação é realizada (rollback).

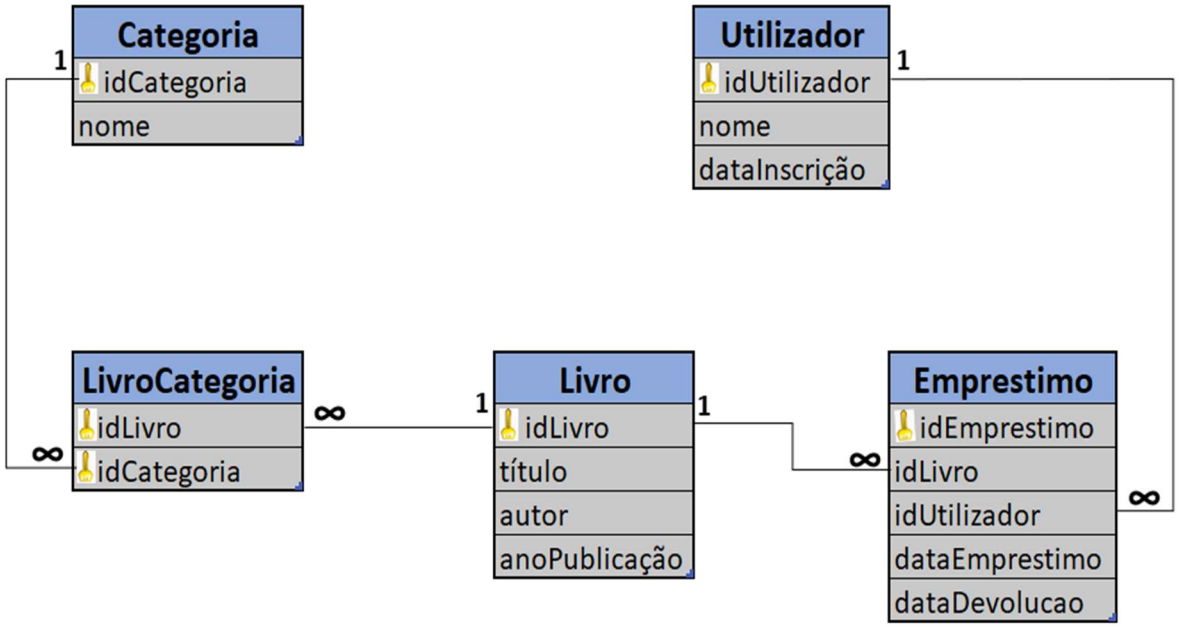
Quando uma transação é iniciada, todas as operações realizadas durante essa transação são mantidas em "buffer" até que a transação seja concluída. Nesse ponto, ocorre o “commit”, onde todas as alterações são efetivamente aplicadas e registradas de forma permanente na base de dados. Por outro lado, se algum problema ocorrer durante a transação, é realizado um “rollback”, que desfaz todas as operações da transação, deixando a base de dados no estado anterior à transação.

A importância da atomicidade está diretamente relacionada à integridade dos dados na base de dados. Sem atomicidade, uma transação incompleta poderia deixar a base de dados em um estado inconsistente, com dados parcialmente atualizados ou corruptos.

Destacam-se dois exemplos:

- Transação de transferência bancária: Numa transação de transferência de dinheiro entre duas contas. Se a transação não fosse atómica, poderia ocorrer uma situação em que o dinheiro é debitado de uma conta, mas não é creditado na outra (devido a uma falha), resultando numa inconsistência e perda de integridade dos dados bancários.
- Atualização de estoque de um produto: Numa transação que envolva atualizar o estoque de um produto na base de dados. Se a transação não fosse atómica, poderia haver uma situação em que o estoque é diminuído, mas o produto não é realmente vendido (por exemplo, devido a uma falha no sistema). Isso levaria a um valor de estoque incorreto, comprometendo a integridade dos dados de inventário.

2)



Justificação: Para estabelecer ligações com chaves estrangeiras considerei atributos com o mesmo nome nas tabelas, dado que se infere que os tipos de dados devem ser idênticos, o tamanho dos campos deve ser igual e o campo referenciado é chave primária ou tem um índice único na tabela de origem.
Foi utlizada a seguinte lógica para estabelecer as ligações:

Tabela "Livro": Chave Primária: idLivro, identificador dado no enunciado.
Esta tabela armazena as informações básicas de cada livro da biblioteca.
Relacionamento através de chaves estrangeiras: Um livro pode pertencer a várias categorias (1:N com LivroCategoria) e pode ter vários empréstimos (1:N com Emprestimo).

Tabela "Categoria": Chave Primária: idCategoria
Esta tabela armazena as diferentes categorias de livros.
Relacionamento através de chaves estrangeiras: Uma categoria pode estar associada a vários livros (1:N com LivroCategoria).

Tabela "LivroCategoria": Chaves Primárias Compostas: idLivro, idCategoria
Esta tabela associa categorias e livros, apesar de um livro poder pertencer a várias categorias e cada categoria conter vários livros é possível formar a chave primária composta através da combinação dos dois atributos.
Relacionamento através de chaves estrangeiras: idLivro referencia Livro(idLivro) e idCategoria referencia Categoria(idCategoria).
Esta é uma tabela de junção que representa a relação “muitos para muitos” entre livros e categorias.

Tabela "Utilizador": Chave Primária: idUtilizador

Esta tabela armazena informações sobre os utilizadores da biblioteca.
Relacionamento através de chaves estrangeiras: Um utilizador pode ter vários empréstimos (1:N com Emprestimo).

Tabela "Emprestimo": Chave Primária: idEmprestimo
Relacionamento através de chaves estrangeiras: idLivro referencia Livro(idLivro) e idUtilizador referencia Utilizador(idUtilizador).
Esta tabela regista todos os empréstimos de livros, ligando cada utilizador a cada livro.

Lógica relacional:
Um livro pode pertencer a várias categorias.
Cada entrada em LivroCategoria deve corresponder a um livro existente.
Uma categoria pode estar associada a vários livros.
Cada entrada em LivroCategoria deve corresponder a uma categoria existente.
Um livro pode ter vários empréstimos (em momentos diferentes).
Cada empréstimo está associado a exatamente um livro.
Um utilizador pode ter vários empréstimos.
Cada empréstimo está associado a exatamente um utilizador.

3) a)

```
SELECT Livro.título, Utilizador.nome
FROM Emprestimo
JOIN Livro ON Emprestimo.idLivro = Livro.idLivro
JOIN Utilizador ON Emprestimo.idUtilizador = Utilizador.idUtilizador
WHERE Emprestimo.dataDevolucao IS NULL;
```

Comecei a seleção dos campos pedidos no enunciado pela tabela Emprestimo para verificar quais empréstimos ainda não têm uma data de devolução.

```
"SELECT Livro.título, Utilizador.nome
FROM Emprestimo"
```

Depois uni a tabela Emprestimo com a tabela Livro, vinculando cada empréstimo ao livro correspondente através do campo idLivro.

"JOIN Livro ON Emprestimo.idLivro = Livro.idLivro"

Juntei a tabela Emprestimo com a tabela Utilizador, vinculando cada empréstimo ao utilizador que o realizou através do campo idUtilizador.

"JOIN Utilizador ON Emprestimo.idUtilizador = Utilizador.idUtilizador"

No fim filtrei os registros para retornar apenas os empréstimos onde dataDevolucao é NULL, indicando que o livro ainda não foi devolvido.

"WHERE Emprestimo.dataDevolucao IS NULL"

b)

```
SELECT Utilizador.nome, COUNT(Emprestimo.idLivro) AS livrosEmprestados
FROM Emprestimo
JOIN Utilizador ON Emprestimo.idUtilizador = Utilizador.idUtilizador
GROUP BY Utilizador.nome
HAVING COUNT(Emprestimo.idLivro) > 3
ORDER BY livrosEmprestados DESC;
```

Comecei com a tabela Emprestimo, onde estão registradas todas as informações dos empréstimos realizados incluindo a contabilização de empréstimos. (FROM Emprestimo) e utilizei (COUNT(Emprestimo.idLivro) AS livrosEmprestados) para calcular o número total de livros emprestados por cada utilizador e dar o nome à coluna a apresentar.

Juntei a tabela Emprestimo com Utilizador para obter o nome dos utilizadores que realizaram os empréstimos. (JOIN Utilizador ON Emprestimo.idUtilizador = Utilizador.idUtilizador)

Agrupei os resultados pelo nome do utilizador, para que cada linha do resultado final represente um utilizador. (GROUP BY Utilizador.nome)

No fim filtrei os utilizadores que requisitaram mais de 3 livros. (HAVING COUNT(Emprestimo.idLivro) > 3) e ordenei de forma decrescente pelo número de livros emprestados. (ORDER BY livrosEmprestados DESC)

c)

```
SELECT Livro.título, Categoria.nome, Livro.anoPublicacao,
COUNT(Emprestimo.idEmprestimo) AS total_emprestimos
FROM Livro
JOIN LivroCategoria ON Livro.idLivro = LivroCategoria.idLivro
JOIN Categoria ON LivroCategoria.idCategoria = Categoria.idCategoria
JOIN Emprestimo ON Livro.idLivro = Emprestimo.idLivro
WHERE Livro.anoPublicacao < 2000
   AND Categoria.nome = 'História'
   AND NOT EXISTS (SELECT 1
      FROM Emprestimo
      WHERE Emprestimo.idLivro = Livro.idLivro
        AND Emprestimo.dataDevolucao IS NOT NULL
        AND (Emprestimo.dataDevolucao - Emprestimo.dataEmprestimo) > 30
   )
GROUP BY Livro.título, Categoria.nome, Livro.anoPublicacao
HAVING COUNT(Emprestimo.idEmprestimo) >= 1
```

Selecionei as colunas referentes ao título do livro, nome da categoria e ano de publicação do livro. “SELECT Livro.título, Categoria.nome, Livro.anoPublicacao,”

Contei o número de empréstimos para cada livro como “total_emprestimos”.
“COUNT(Emprestimo.idEmprestimo) AS total_emprestimos”

Ambas as ações selecionaram dados da tabela Livro. “FROM Livro”

Em seguida, uni a tabela Livro com as seguintes tabelas:

- LivroCategoria: Para associar os livros com suas categorias respetivas
- Categoria: Para recuperar os nomes das categorias
- Emprestimo: Para contar o número de empréstimos para cada livro

“JOIN LivroCategoria ON Livro.idLivro = LivroCategoria.idLivro
JOIN Categoria ON LivroCategoria.idCategoria = Categoria.idCategoria
JOIN Emprestimo ON Livro.idLivro = Emprestimo.idLivro”

Na consulta filtrei por livros publicados antes do ano 2000, na categoria "História" e excluí livros que foram devolvidos com um atraso superior a 30 dias em qualquer um dos seus empréstimos.

“WHERE Livro.anoPublicacao < 2000
AND Categoria.nome = 'História'
AND NOT EXISTS (SELECT 1
FROM Emprestimo
WHERE Emprestimo.idLivro = Livro.idLivro
AND Emprestimo.dataDevolucao IS NOT NULL
AND (Emprestimo.dataDevolucao - Emprestimo.dataEmprestimo) > 30
)”

Agrupei os resultados são pelo título do livro, nome da categoria e ano de publicação. “GROUP BY Livro.título, Categoria.nome, Livro.anoPublicacao”

Incluí na consulta somente livros tenham sido requisitados pelo menos uma vez. “HAVING COUNT(Emprestimo.idEmprestimo) >= 1”

d)

```
UPDATE Utilizador
SET dataInscrição = '2024-01-01'
WHERE idUtilizador IN (
   SELECT idUtilizador
   FROM Emprestimo
   GROUP BY idUtilizador
   HAVING COUNT(idEmprestimo) > 5
);
```

Primeiro especifiquei que se atualizar a tabela Utilizador com o valor na dataInscrição de '2024-01-01'.

“UPDATE Utilizador
SET dataInscrição = '2024-01-01'”

Seguidamente filtrei para atualizar apenas os registros dos utilizadores cujo idUtilizador está na consulta. “WHERE idUtilizador IN (”

Na consulta selecionei os IDs dos utilizadores da tabela Emprestimo.

“SELECT idUtilizador
FROM Emprestimo”

Agrupei os registos por idUtilizador para calcular o total de empréstimos por utilizador. “ GROUP BY idUtilizador”

Filtrei os utilizadores com mais de 5 empréstimos.
“HAVING COUNT(idEmprestimo) > 5
);”