**Lista de Exercícios - DCL e TCL com PostgreSQL**

**Aluno: Douglas Mateus Soares Cândido da Silva**

**Matrícula: 20190020243**

Transações (TCL):

**1.**

**Transação** - é a realização de um conjunto de comandos de uma só vez. Uma transação acontece por completo (todas as operações) ou nada acontece. Também a transação deve garantir um nível de isolamento das demais transações, de maneira que as demais transações somente enxerguem as operações após a transação concluída.

Caso haja um erro qualquer na transação ou falha no sistema o SGBR irá executar um comando ROLLBACK. Transações são uma forma de dar suporte às operações concorrentes, garantindo a segurança e integridade das informações. Garantir que duas solicitações diferentes não efetuarão uma mesma operação ao mesmo tempo.

Ao consultar o banco de dados, uma transação enxerga um snapshot (instantâneo) dos dados, como estes eram no exato momento em que a consulta foi solicitada, desprezando as mudanças ocorridas depois disso.

O PostgreSQL trata a execução de qualquer comando SQL como sendo executado dentro de uma transação.

Na versão 8 apareceram os SAVEPOINTS (pontos de salvamento) , que guardam as informações até eles. Isso salva as operações existentes antes do SAVEPOINT e basta um ROLLBACK TO para continuar com as demais operações.

O PostgreSQL mantém a consistência dos dados utilizando o modelo multiversão MVCC (Multiversion Concurrency Control), que permite que leitura não bloqueie escrita nem escrita bloqueie leitura.

O PostgreSQL também conta com um nível de isolamento chamado serializable (serializável), que é mais rigoroso e emula execução serial das transações.

**BEGIN**;

**UPDATE** contas **SET** saldo = saldo – 100.00 **WHERE** codigo = 5;

SAVEPOINT meu\_ponto\_de\_salvamento;

**UPDATE** contas **SET** saldo = saldo + 100.00 **WHERE** codigo = 5;

*-- ops ... o certo é na conta 6*

**ROLLBACK** **TO** meu\_ponto\_de\_salvamento;

**UPDATE** contas **SET** saldo = saldo + 100.00 **WHERE** conta = 6;

**COMMIT**;

**2.**

**A sigla *ACID*define os quatro pilares necessários para a realização de uma transação em um sistema de banco de dados. São elas:**

* ***Atomicidade;***
* ***Consistência;***
* ***Isolamento;***
* ***Durabilidade.***

No caso, se as quatro propriedades exibidas acima forem seguidas de forma correta, as suas transações estão compatíveis com a estrutura *ACID*. Ou seja, esse é o conceito de ***ACID Compliance,***estar em conformidade com os pilares das transações.

Juntos, os conceitos acima garantem que as transações sejam realizadas de forma confiável.

**Atomicidade**

**Seria a garantia de que a transação será feita totalmente ou não será feita. Nesse caso, a transação não é feita “pela metade”. Se por ventura uma operação da transação falhar, consequentemente, toda a transação falhará.**

Você se lembra das aulas que você teve de Química ou Ciências, na escola? E, caso a resposta seja positiva, você se lembra do conceito de átomos? Em caso negativo, sem problemas!

Um átomo seria uma esfera maciça indivisível, indestrutível e invisível. Ou seja, esse conceito de ser indivisível se encaixa com o de atomicidade, pois, como explicado anteriormente, uma transação não é feita de forma incompleta. Ou ela é feita, ou não é feita.

Um exemplo seria uma pessoa usuária transferir uma quantia para outra pessoa. O dinheiro transferido pode ir totalmente para a conta do destinatário ou não pode. Você não faz um envio de dinheiro pela metade. Isso poderia ocasionar sérios problemas para os envolvidos.

**Consistência**

Seria**a**[**proteção da integridade dos dados**](https://blog.betrybe.com/tecnologia/seguranca-da-informacao/)**. Ou seja, se um banco de dados fizer uma operação que não seja válida, o processo será impedido e retornará para o estado inicial do processo.**

As operações serão legítimas se passarem por todas as verificações do banco de dados, incluindo todas as restrições inseridas nele (gatilhos, funções, procedimentos, entre outros).

Por exemplo, em sua conta, você possui uma quantia de cem reais e deseja realizar um saque de duzentos reais. A operação será impedida, pois, você não possui tal quantia em sua conta bancária para ser sacada.

**Isolamento**

**A capacidade de isolamento seria o fato de uma transação não “atrapalhar” a outra e ocorrer de forma isolada, garantindo que sejam feitas de forma individual.**

Mas, isso quer dizer que não pode acontecer mais de uma transação simultaneamente? Nesse caso, não.

Podem ocorrer muitas transações ao mesmo tempo. Porém, elas não podem impactar as outras transações que ocorrem simultaneamente.

Vamos pensar em sites de [e-commerce](https://blog.betrybe.com/tecnologia/amazon/) como exemplo. É um fato afirmar que é um local em que está ocorrendo mais de uma transação simultânea. Você vê um produto que tem somente uma unidade e, outro comprador também comprará ele ao mesmo tempo. E aí, o que pode acontecer, já que só tem apenas um produto?

Nesse caso, o primeiro comprador que finalizar a compra fará que a transação de seu concorrente seja interrompida.

**Durabilidade**

**Seria a preservação dos dados após as operações terem sido realizadas. Ou seja, uma vez que uma transação for efetuada, ela permanecerá dessa forma, mesmo que ocorram problemas graves no sistema, sem precisar de retrabalho.**

Vamos pensar no caso anterior: uma pessoa usuária conseguiu a compra do produto e, suponhamos que devido a uma forte chuva, o servidor foi afetado e ficou um tempo inativo ou o servidor foi reiniciado. Nossa! Os dados da compra serão perdidos e ele terá que fazer de novo? Isso não pode ocorrer em hipótese alguma!

Os dados dessa compra efetuada devem ser exibidos após o servidor voltar de sua inatividade, garantindo a durabilidade do banco de dados.

**3.**

create table if not exists postgres.public.funcionario

(

codigo serial primary key,

nome varchar(100) not null,

idade bigint not null

);

**4.**

begin;

update funcionario set idade = 30 where codigo = 1;

savepoint savepoint\_1;

update funcionario set nome = ‘Douglas’ where codigo = 1;

/\* A operação exige o nome completo. \*/

rollback to savepoint\_1;

update funcionario set nome = ‘Douglas Mateus Soares Cândido da Silva’;

commit;

**O comando COMMIT efetiva a transação corrente. Todas as modificações efetuadas pela transação se tornam visíveis para os outros. A transação será concluída ao executar o comando COMMIT, mas apenas se todas as operações prévias tiverem sido executadas com sucesso, do contrário os dados serão preservados e nenhuma mudança ocorrerá, garantindo o princípio da consistência dos dados.**

**5.**

begin;

update funcionario set idade = 30 where codigo = 1;

savepoint savepoint\_1;

update funcionario set nome = ‘Douglas’ where codigo = 1;

/\* A operação exige o nome completo. \*/

rollback to savepoint\_1;

release savepoint savepoint\_1;

update funcionario set nome = ‘Douglas Mateus Soares Cândido da Silva’;

commit;

**O comando SAVEPOINT cria um snapshot (Uma espécie de foto de como os dados estavam armazenados naquele momento) do estado atual do Banco de Dados e garante que seja possível retornar a esse estado no futuro se o administrador assim desejar. Para retornar a um estado prévio, utiliza-se o comando ROLLBACK TO SAVEPOINT. O comando RELEASE SAVEPOINT destrói um savepoint, assim impossibilitando de executar o comando rollback para tal savepoint que foi destruído.**

**6.**

**Uma tabela bloqueada com o comando LOCK TABLE permanece bloqueada até que você confirme sua transação ou a reverta totalmente ou para um ponto de salvamento antes de você bloquear a tabela; esse comando tem o objetivo de garantir a estabilidade dos dados com a restrição de modificação dos dados da tabela, de forma que os usuários não os alterem no momento que uma transação está sendo processada. Um bloqueio nunca impede que outros usuários consultem a tabela, pois uma consulta nunca coloca um bloqueio em uma tabela. UNLOCK TABLE é o comando utilizado para liberar uma tabela para inserções e modificações.**

Permissões (DCL):

**7.**

create user administrador with password 'adm';

grant all on database postgres to administrador;

SELECT usename AS role\_name,

CASE

WHEN usesuper AND usecreatedb THEN

CAST('superuser, create database' AS pg\_catalog.text)

WHEN usesuper THEN

CAST('superuser' AS pg\_catalog.text)

WHEN usecreatedb THEN

CAST('create database' AS pg\_catalog.text)

ELSE

CAST('' AS pg\_catalog.text)

END role\_attributes

FROM pg\_catalog.pg\_user

ORDER BY role\_name desc;

**8.**

create user usu;

grant all on database postgres to usu;

alter user usu with password 'usu';

alter user usu rename to administrador2;

revoke all on database postgres from administrador2;

drop user administrador2;

**9.**

create group adms with user administrador;

**10.**

create user douglas with password 'douglas';

grant select, insert, update, delete on all tables in schema public to douglas;

**11.**

revoke delete on all tables in schema public from douglas;