Transações

Para um SGBD, uma transação e um conjunto de operações que são tratadas como um bloco único e indivisível e também deve prover isolamento entre acessos concorrentes na mesma massa de dados.

Atomicidade – a transação é indivisível, todas suas operações devem ser executadas em caso de sucesso ou nenhuma alteração deve ser refletida na base de dados em caso de falha.

Consistência – a base de dados deve estar consistente antes do início da transação e após o final da transação. Se alguma alteração da transação violar a consistência da base de dados, toda a transação deve ser descartada e a base deve voltar a um estado consistente.

Isolamento – cada transação deve ser executada isolada das demais, uma transação não deve acessar ou alterar dados intermediários de outras transações concorrentes.

Durabilidade – após o final da transação, as alterações realizadas pela transação devem persistir mesmo em caso de falha.

Transações

Uma transação é um conjunto de comandos SQL envolvidos entre os comandos BEGIN e COMMIT ou ROLLBACK.

Exemplo:

```
BEGIN;
DELETE FROM funcionario WHERE secao='MAN';
UPDATE cliente SET nome='Ana' WHERE codigo='02';
COMMIT;
```

Os comandos entre o BEGIN e o COMMIT são chamados de bloco de transação.

O PostgreSQL trata cada comando executado fora de um bloco de transação como sendo executado em uma transação, ou seja, é executado entre um BEGIN e um COMMIT implícito.

É possível controlar os comandos na transação de uma forma mais granular utilizando os pontos de salvamento (savepoints). Os pontos de salvamento permitem descartar partes da transação seletivamente, e efetivar as demais partes.

BEGIN

O comando **BEGIN** inicia um bloco de transação.

```
BEGIN [ WORK | TRANSACTION ] [ transaction_mode [, ...] ]
```

Os parâmetros **WORK** e **TRANSACTION** não tem nenhum efeito.

Onde **transaction_mode** pode ser:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ |
                  READ COMMITTED | READ UNCOMMITTED }
READ WRITE | READ ONLY
```

COMMIT

O comando COMMIT efetiva a transação corrente.

```
COMMIT [ WORK | TRANSACTION ]
```

Os parâmetros WORK e TRANSACTION não tem nenhum efeito.

Exemplo:

```
SELECT * FROM funcionario;
BEGIN WORK;
UPDATE funcionario SET nome='Marcia' WHERE
matricula='45739';
COMMIT WORK;
SELECT * FROM funcionario;
```

ROLLBACK

O comando ROLLBACK aborta a transação corrente.

```
ROLLBACK [ WORK | TRANSACTION ]
```

Os parâmetros WORK e TRANSACTION não tem nenhum efeito.

Exemplo:

```
BEGIN WORK;
DELETE FROM funcionario;
SELECT * FROM funcionario;
ROLLBACK WORK;
SELECT * FROM funcionario;
```

SAVEPOINT

O comando **SAVEPOINT** define um ponto de salvamento na transação corrente.

```
SAVEPOINT savepoint_name
```

Onde **savepoint_name** é o nome do ponto de salvamento.

ROLLBACK TO SAVEPOINT

O comando ROLLBACK TO SAVEPOINT retorna a transação corrente até o ponto de salvamento.

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ]
savepoint_name
```

Os parâmetros WORK e TRANSACTION não tem nenhum efeito.

Exemplo:

```
SELECT * FROM funcionario;
BEGIN WORK;
UPDATE funcionario SET nome='Silvia' WHERE
matricula='15730';
SAVEPOINT teste;
UPDATE funcionario SET nome='Lucio' WHERE
matricula='48026';
SELECT * FROM funcionario;
ROLLBACK TO SAVEPOINT teste;
COMMIT WORK;
SELECT * FROM funcionario;
```

RELEASE SAVEPOINT

O comando **RELEASE SAVEPOINT** elimina um ponto de salvamento sem eliminar os efeitos dos comandos executados depois do estabelecimento do ponto de salvamento.

```
RELEASE [ SAVEPOINT ] savepoint_name
```

Exemplo:

```
SELECT * FROM funcionario;
BEGIN WORK;
SAVEPOINT teste;
UPDATE funcionario SET nome='Lucio' WHERE
matricula='48026';
SELECT * FROM funcionario;
RELEASE SAVEPOINT teste;
ROLLBACK TO SAVEPOINT teste;
COMMIT WORK;
SELECT * FROM funcionario;
```

SELECT FOR UPDATE

A cláusula FOR UPDATE do comando SELECT permite bloquear para alteração, até o fim da transação, os registros selecionados das tabelas indicadas, impedindo que outras transações executem um UPDATE, DELETE, SELECT FOR UPDATE ou SELECT FOR SHARE com esses registros. Se não for especificada nenhuma tabela na cláusula FOR UPDATE, os registros de todas as tabelas envolvidas serão bloqueados.

Exemplo:

```
SELECT * FROM funcionario;
BEGIN WORK;
SELECT * FROM funcionario WHERE matricula='48026' FOR
UPDATE;
UPDATE funcionario SET nome='Luiz Carlos' WHERE
matricula='48026';
SELECT * FROM funcionario;
COMMIT WORK;
SELECT * FROM funcionario;
```

SELECT FOR SHARE

A cláusula FOR SHARE do comando SELECT permite bloquear para consulta, até o fim da transação, os registros selecionados das tabelas indicadas, impedindo que outras transações executem um UPDATE, DELETE ou SELECT FOR UPDATE com esses registros. Se não for especificada nenhuma tabela na cláusula FOR SHARE, os registros de todas as tabelas envolvidas serão bloqueados.

Exemplo:

```
SELECT * FROM funcionario;
BEGIN WORK;
SELECT * FROM funcionario WHERE matricula='48026' FOR
SHARE;
COMMIT WORK;
```

NOWAIT

Quanto utilizada a cláusula NOWAIT com SELECT FOR UPDATE ou SELECT FOR SHARE, se não for possível obter o bloqueio imediatamente, o SGBD gera um erro na execução do comando. Sem a cláusula NOWAIT, se não for possível obter o bloqueio, o SGBD irá aguardar o sucesso do bloqueio pelo tempo definido na configuração antes de gerar o erro, ou mesmo aguardar indefinidamente se a configuração não determinar um tempo limite para o sucesso do bloqueio.

LOCK

O comando **LOCK** permite bloquear uma tabela até o fim da transação.

```
LOCK [ TABLE ] name [, ...] [ IN lockmode MODE ] [ NOWAIT ]
```

Onde **lockmode** pode ser:

```
    ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE
  | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

O comando **LOCK** trata de bloqueios de tabela, portanto, o uso de modos de bloqueio de linhas (contendo **ROW**) não fazem o bloqueio por linhas.

O comando **LOCK** não existe no padrão SQL.

Exemplo:

```
BEGIN WORK;
LOCK funcionario IN ACCESS EXCLUSIVE MODE;
COMMIT WORK;
```

LOCK

ACCESS SHARE

Conflita apenas com o modo de bloqueio ACCESS EXCLUSIVE.

O comando SELECT e o comando ANALYZE obtêm um bloqueio neste modo nas tabelas referenciadas. Em geral, qualquer comando que apenas lê a tabela sem modificá-la obtém este modo de bloqueio.

ROW SHARE

Conflita com os modos de bloqueio EXCLUSIVE e ACCESS EXCLUSIVE.

O comando SELECT FOR UPDATE obtém o bloqueio neste modo na(s) tabela(s) de destino (além do bloqueio no modo ACCESS SHARE para as demais tabelas referenciadas mas não selecionadas FOR UPDATE).

LOCK

ROW EXCLUSIVE

Conflita com os modos de bloqueio SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE e ACCESS EXCLUSIVE.

Os comandos UPDATE, DELETE e INSERT obtêm este modo de bloqueio na tabela de destino (além do modo de bloqueio ACCESS SHARE nas outras tabelas referenciadas). Em geral, este modo de bloqueio é obtido por todos os comandos que alteram os dados da tabela.

SHARE UPDATE EXCLUSIVE

Conflita com os modos de bloqueio SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE e ACCESS EXCLUSIVE. Este modo protege a tabela contra alterações simultâneas no esquema e a execução do comando VACUUM.

Obtida pelo comando VACUUM (sem a opção FULL).

LOCK

SHARE

Conflita com os modos de bloqueio ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE e ACCESS EXCLUSIVE. Este modo protege a tabela contra alterações simultâneas nos dados.

Obtido pelo comando CREATE INDEX.

SHARE ROW EXCLUSIVE

Conflita com os modos de bloqueio ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE e ACCESS EXCLUSIVE.

Este modo de bloqueio não é obtido automaticamente por nenhum comando do PostgreSQL.

LOCK

EXCLUSIVE

Conflita com os modos de bloqueio ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE e ACCESS EXCLUSIVE. Este modo permite apenas bloqueios ACCESS SHARE simultâneos, ou seja, somente leituras da tabela podem prosseguir em paralelo com uma transação que obteve este modo de bloqueio.

Este modo de bloqueio não é obtido automaticamente por nenhum comando do PostgreSQL. Entretanto, é obtido em alguns catálogos do sistema em algumas operações.

LOCK

ACCESS EXCLUSIVE

Conflita com todos os modos de bloqueio (ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE e ACCESS EXCLUSIVE). Este modo garante que a transação que o obteve é a única acessando a tabela de qualquer forma.

Obtido pelos comandos ALTER TABLE, DROP TABLE, REINDEX, CLUSTER e VACUUM FULL. Este é, também, o modo de bloqueio padrão para o comando LOCK TABLE sem a especificação explícita do modo.

Somente o bloqueio ACCESS EXCLUSIVE bloqueia o comando SELECT (sem a cláusula FOR UPDATE).

Commit em Duas Fases

O protocolo de Commit em duas fases (2PC) é um protocolo que permite uma transação distribuída entre dois ou mais bancos de dados.

Esse protocolo necessita de um gerenciador de transações, chamado de coordenador e os demais bancos de dados são chamados subordinados.

Fase I – o coordenador envia para os subordinados uma mensagem para preparar a efetivação da transação, cada subordinado responde ao coordenador com uma mensagem indicando se o COMMIT foi preparado com sucesso ou se a transação deve ser abortada.

Fase II – se todos os subordinados responderam que a transação foi preparada, então o coordenador envia mensagem para todos subordinados para efetivar a transação, se algum subordinado não respondeu ou enviou mensagem para abortar a transação, o coordenador envia para os subordinados uma mensagem para abortar a transação.

PREPARE TRANSACTION

O comando PREPARE TRANSACTION prepara a transação corrente para efetivação em duas fases.

```
PREPARE TRANSACTION transaction_id
```

Este comando armazena o estado da transação em disco e a transação deixa de estar associada a sessão corrente. Os efeitos da transação preparada não poderão ser vistos por nenhuma outra transação até que a transação seja efetivada.

COMMIT PREPARED

O comando COMMIT PREPARED efetiva uma transação preparada para efetivação em duas fases.

```
COMMIT PREPARED transaction_id
```

Exemplo:

```
SELECT * FROM funcionario;
BEGIN TRANSACTION;
UPDATE funcionario SET nome='Lucas' WHERE
matricula='48026';
PREPARE TRANSACTION 'teste';
\q
SELECT * FROM funcionario;
COMMIT PREPARED 'teste';
SELECT * FROM funcionario;
```

ROLLBACK PREPARED

O comando ROLLBACK PREPARED aborta uma transação preparada para efetivação em duas fases.

```
ROLLBACK PREPARED transaction_id
```

Exemplo:

```
SELECT * FROM funcionario;
BEGIN TRANSACTION;
UPDATE funcionario SET nome='Luiz Mauricio' WHERE
matricula='26360';
PREPARE TRANSACTION 'teste';
\q
SELECT * FROM funcionario;
ROLLBACK PREPARED 'teste';
SELECT * FROM funcionario;
```

Verificação das Restrições (CONTRAINTS)

No início de uma transação, o banco de dados deve estar em um estado consistente e ao final da transação, banco também deve estar em um estado consistente. Porém, é possível adiar a verificação das restrições dentro de uma transação, permitindo que durante a transação, as alterações levem os dados a um estado inconsistente, ferindo restrições que devem ser novamente satisfeitas antes do final da transação.

Para que uma restrição possa ter sua verificação adiada, deve ser usada a cláusula DEFERRABLE na sua criação. Por padrão, as restrições são NOT DEFERRABLE, não podem ter sua verificação adiada.

Restrições que podem ser adiadas podem ser criadas com INITIALLY IMMEDIATE, sendo, a princípio, verificadas a cada instrução executada, esse é o padrão na criação de restrições, ou INITIALLY DEFERRED, sendo verificada no final da transação.

O PostgreSQL permite apenas o adiamento da verificação das restrições UNIQUE, PRIMARY KEY, EXCLUDE e REFERENCES. Todas as outras restrições são verificadas após a execução de cada instrução.

Verificação das Restrições

Exemplo:

```
ALTER TABLE entrada DROP CONSTRAINT entrada_produto_fkey;
ALTER TABLE entrada ADD CONSTRAINT entrada_produto_fkey
FOREIGN KEY (produto) REFERENCES produto DEFERRABLE
INITIALLY DEFERRED;
BEGIN;
INSERT INTO entrada (data,produto,quantidade) VALUES
('2010-03-03','07',10);
INSERT INTO produto (codigo,descricao) VALUES
('07','Fichario');
COMMIT;
```

Verificação das Restrições

Se a verificação de alguma restrição e adiada e o banco de dados passa a um estado inconsistente durante a transação, não será possível efetivar a transação antes de resolver as inconsistências.

Exemplo:

```
BEGIN;
INSERT INTO entrada (data,produto,quantidade) VALUES
('2010-03-05','08',4);
COMMIT;
```

SET CONSTRAINTS

O comando **SET CONSTRAINTS** permite determinar o momento da execução das restrições que podem ser adiadas.

```
SET CONSTRAINTS {ALL|name [,...]} {DEFERRED|IMMEDIATE}
```

Com a opção **DEFERRED**, o comando determina que as restrições especificadas serão verificadas apenas no final da transação. A opção **IMMEDIATE** determina que as restrições serão verificadas a cada instrução executada.

Apenas restrições **DEFERRABLE** podem ser afetadas por esse comando. Restrições **NOT DEFERRABLE** sempre serão verificadas a cada instrução executada.

Esse comando deve ser usado dentro de uma transação e o seu efeito é válido apenas dentro dessa transação.

SET CONSTRAINTS

Exemplo:

```
ALTER TABLE saida DROP CONSTRAINT saida_produto_fkey;
ALTER TABLE saida ADD CONSTRAINT saida_produto_fkey
FOREIGN KEY (produto) REFERENCES produto DEFERRABLE;
BEGIN;
SET CONSTRAINTS saida_produto_fkey DEFERRED;
INSERT INTO saida (data,produto,quantidade) VALUES
('2010-03-01','08',6);
INSERT INTO produto (codigo,descricao) VALUES
('08','Esquadro');
COMMIT;
```

SET CONSTRAINTS

O comando SET CONSTRAINTS com a opção IMMEDIATE faz a verificação imediata das restrições especificadas.

Exemplo:

```
BEGIN;
SET CONSTRAINTS saida_produto_fkey DEFERRED;
INSERT INTO saida (data,produto,quantidade) VALUES
('2010-03-12','09',4);
SET CONSTRAINTS saida_produto_fkey IMMEDIATE;
ROLLBACK;
```