

# Banco de Dados 2

04 – Transações, WAL e MVCC.

# Mecanismos de Recuperação de Paradas e Falhas

- Um **sistema de computação**, como qualquer outro dispositivo elétrico ou mecânico, está **sujeito a falhas**.
- Possíveis causas de falhas:
  - Quebra do disco,
  - Queda de força,
  - Erros de Software,
  - Incêndio,
  - Sabotagem etc.

# Mecanismos de Recuperação de Paradas e Falhas

- Falhas levam o banco de dados a perder informações.
- Uma parte essencial do sistema de banco de dados é um **esquema de recuperação** responsável pela **detecção de falhas** e pela **restauração do banco de dados** para um **estado consistente** que existia antes da ocorrência da falha.

# Mecanismos de Recuperação de Paradas e Falhas

- Frequentemente, diversas operações do banco de dados (**BD**) formam uma **única unidade lógica de trabalho**.
- Exemplo: a transferência de fundos, na qual uma conta é debitada e a outra é creditada.
- É essencial à **consistência** do **BD** que o crédito e o débito ocorram, ou que nenhum dos dois ocorra.

# Atomicidade

- A transferência de fundos deve ocorrer por inteiro ou não deve ocorrer.
- Este requerimento “**tudo ou nada**” é chamado **atomicidade**.
- Uma **transação** é uma **coleção de operações** que executa **uma única função lógica** numa aplicação de **BD**.

# Transação

- Cada **transação** é uma **unidade de atomicidade**.
- Independentemente da **possibilidade de falhas** **uma transação preserva a atomicidade**.

# Tipos de Falhas

- [1] – Erros Lógicos.
  - A **transação** não pode mais continuar com sua execução normal devido a alguma **condição interna**, como uma entrada com erro, dado não encontrado, overflow ou limite de reservas excedido.
- [2] – Erros do Sistema.
  - O sistema entrou num estado indesejável (por exemplo, travamento mútuo [*“dead lock”*]). A transação deve ser interrompida, mas pode ser reexecutada mais tarde.

# Tipos de Falhas

- **[3] – Queda do Sistema.**
  - Disfunções do *hardware* causam a perda de conteúdo do armazenamento volátil.
- **[4] – Falha no Disco.**
  - Um bloco do disco perde seu conteúdo como resultado de uma falha durante a transferência de dados.



# Modelo de Transação

- Uma **transação** é uma **unidade de programa** que faz o **acesso** e **possivelmente atualiza** vários itens de dados.
- As **transações não** devem violar qualquer restrição de consistência do BD.
- Se era **consistente** quando uma **transação** iniciou, o **BD** precisa ser **consistente** quando a **transação** **terminar com sucesso**.

# Modelo de Transação

- Entretanto, durante a execução de uma **transação**, pode ser necessário temporariamente permitir a inconsistência.
- Essa **inconsistência temporária**, ainda que necessária, pode levar a dificuldades se ocorrer uma **falha**.

# Exemplo do Sistema Bancário Simplificado

- Considere um **sistema bancário** um pouco **simplificado** com diversas **contas** e um **conjunto de transações** que faça o **acesso** àquelas **contas** e as **atualize**.
- Seja **T** uma **transação** que transfira **\$50** da **conta A** para a **conta B**.

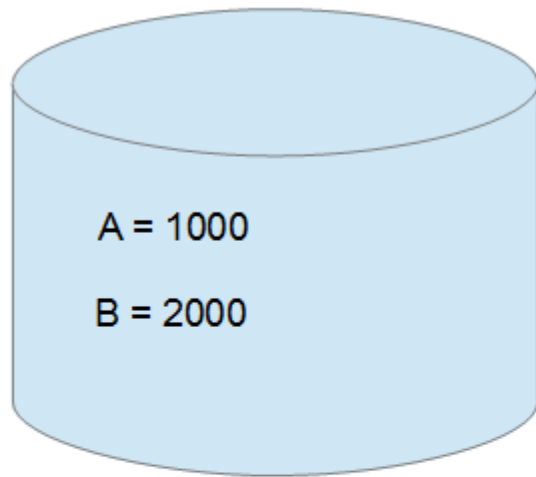
# Exemplo do Sistema Bancário Simplificado

- Esta transação pode ser definida como:
  - T: **read**(A, saldoA)
  - $\text{saldoA} = \text{saldoA} - 50$
  - **write**(A, saldoA)
  - **read**(B, saldoB)
  - $\text{saldoB} = \text{saldoB} + 50$
  - **write**(B, saldoB)

# Exemplo do Sistema Bancário Simplificado

- A restrição de consistência é que **a soma de A e B não deve mudar com a execução de uma transação.**
- Suponha que logo **antes da transação T**, os valores das **contas A e B** sejam **\$ 1.000** e **\$ 2.000**, respectivamente.

# Exemplo do Sistema Bancário Simplificado



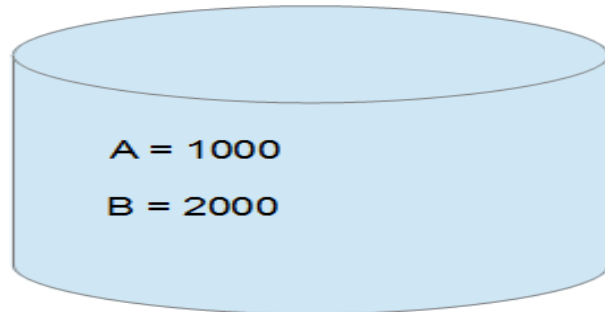
Disco



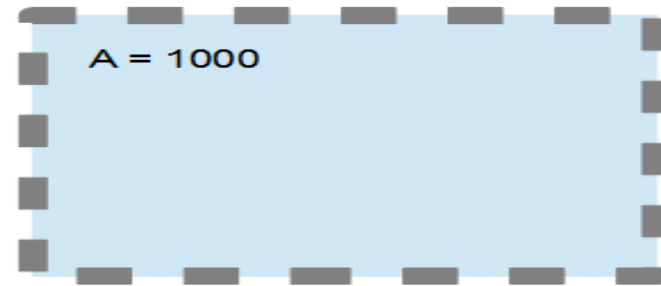
Memória Principal

**Situação antes da execução da Transação T.**

# Exemplo do Sistema Bancário Simplificado



Disco



Memória Principal

**T: read**(A, saldoA) ←

saldoA = saldoA - 50

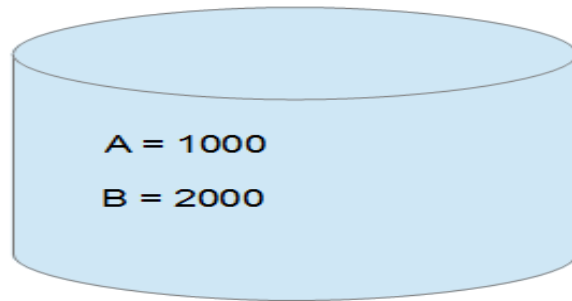
**write**(A, saldoA)

**read**(B, saldoB)

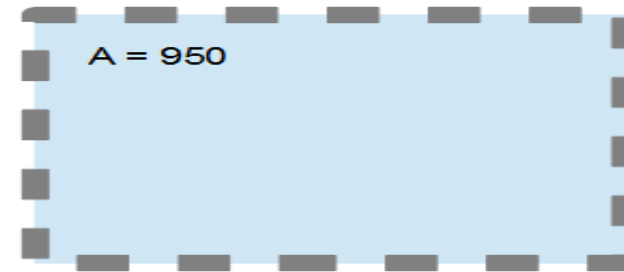
saldoB = saldoB + 50

**write**(B, saldoB)

# Exemplo do Sistema Bancário Simplificado

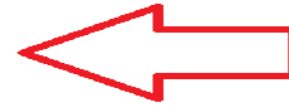


Disco



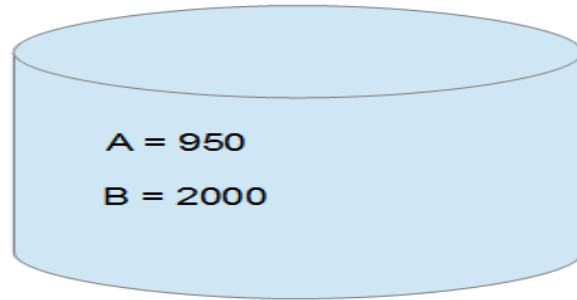
Memória Principal

**T:** **read**(A, saldoA)  
saldoA = saldoA - 50  
**write**(A, saldoA)  
**read**(B, saldoB)  
saldoB = saldoB + 50  
**write**(B, saldoB)

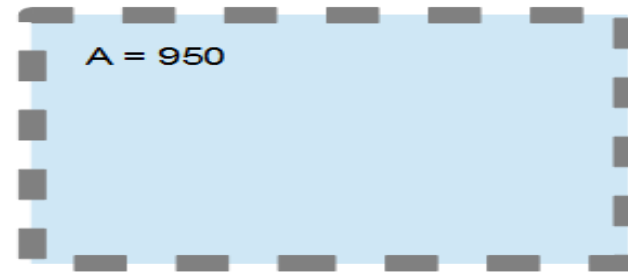




# Exemplo do Sistema Bancário Simplificado

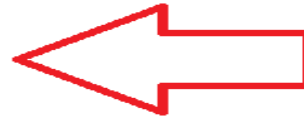


Disco



Memória Principal

T: **read**(A, saldoA)  
saldoA = saldoA - 50  
**write**(A, saldoA)  
**read**(B, saldoB)  
saldoB = saldoB + 50  
**write**(B, saldoB)



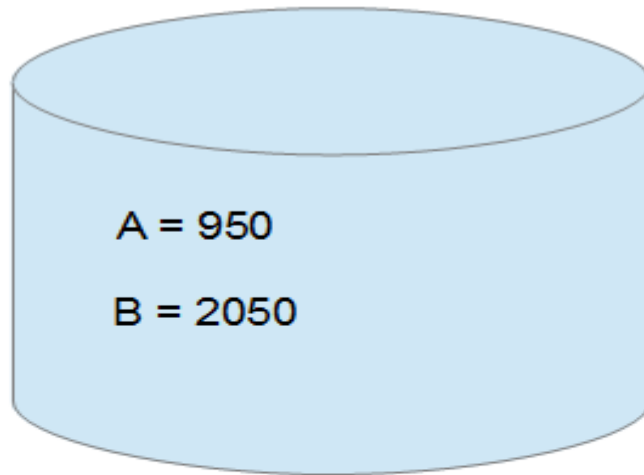
# Exemplo do Sistema Bancário Simplificado

- Nesse ponto uma falha resultaria em um **estado de inconsistência** para nosso banco de dados (BD).
- A soma dos saldos das contas A e B antes da Transação T era **\$ 3.000** (\$ 1.000 + \$ 2.000), mas após T passa a ser **\$ 2.950** (\$ 950 + \$ 2.000).

# Exemplo do Sistema Bancário Simplificado

- O estado de inconsistência decorre da violação da atomicidade de nossa transação: **ocorreu um débito na conta A, mas não houve o crédito na conta B.**
- Em uma **transação** deve ocorrer tudo (débito e crédito no caso de nosso exemplo) e não apenas uma parte (somente o débito).

# Exemplo do Sistema Bancário Simplificado



Disco



Memória Principal

# Exemplo do Sistema Bancário Simplificado

- **Caso nenhuma falha ocorresse** durante a execução da transação T, o slide anterior seria o resultado final de T.
- **Um estado consistente do BD:**
  - **Antes de T:**  $\text{saldoA} + \text{saldoB} = 1.000 + 2.000 = 3.000.$
  - **Depois de T:**  $\text{saldoA} + \text{saldoB} = 950 + 2.050 = 3.000.$

# Exemplo do Sistema Bancário Simplificado

- Dois importantes requerimentos que usamos em **transações**:
  - “**Corretude**” (“**Correctness**”): Cada transação precisa ser um programa que preserva a consistência do banco de dados.
  - “**Atomicidade**”: Todas as operações associadas a uma transação precisam ser executadas até o final, ou nenhuma deve ser executada.

# Exemplo do Sistema Bancário Simplificado

- Assegurar a “**corretude**” é **responsabilidade** do **programador** que codifica uma transação.
- Assegurar a **atomicidade** é **responsabilidade** do próprio **sistema de banco de dados**.

# Exemplo do Sistema Bancário Simplificado

- Na **ausência de falhas**, todas as transações são completadas com sucesso e a atomicidade é respeitada facilmente.
- Nem sempre uma transação pode completar sua execução com sucesso. Tal **transação** é chamada **abortada**.
- Uma **transação abortada** não deve ter efeito sobre o **estado do banco de dados (BD)**.



# Exemplo do Sistema Bancário Simplificado

- Em caso de **transação abortada**, o **BD** precisa ser **restaurado ao estado que se encontrava logo antes da transação** em questão ter se iniciado.
- Dizemos que tal **transação** foi desfeita (“**rolled back**”).
- Uma **transação** que complete sua execução com **sucesso** é chamada de **compromissada** (**committed**).

# ACID

- A sigla **ACID** significa:
- - *Atomicity*,
- - *Consistency*,
- - *Isolation* e
- - *Durability*.

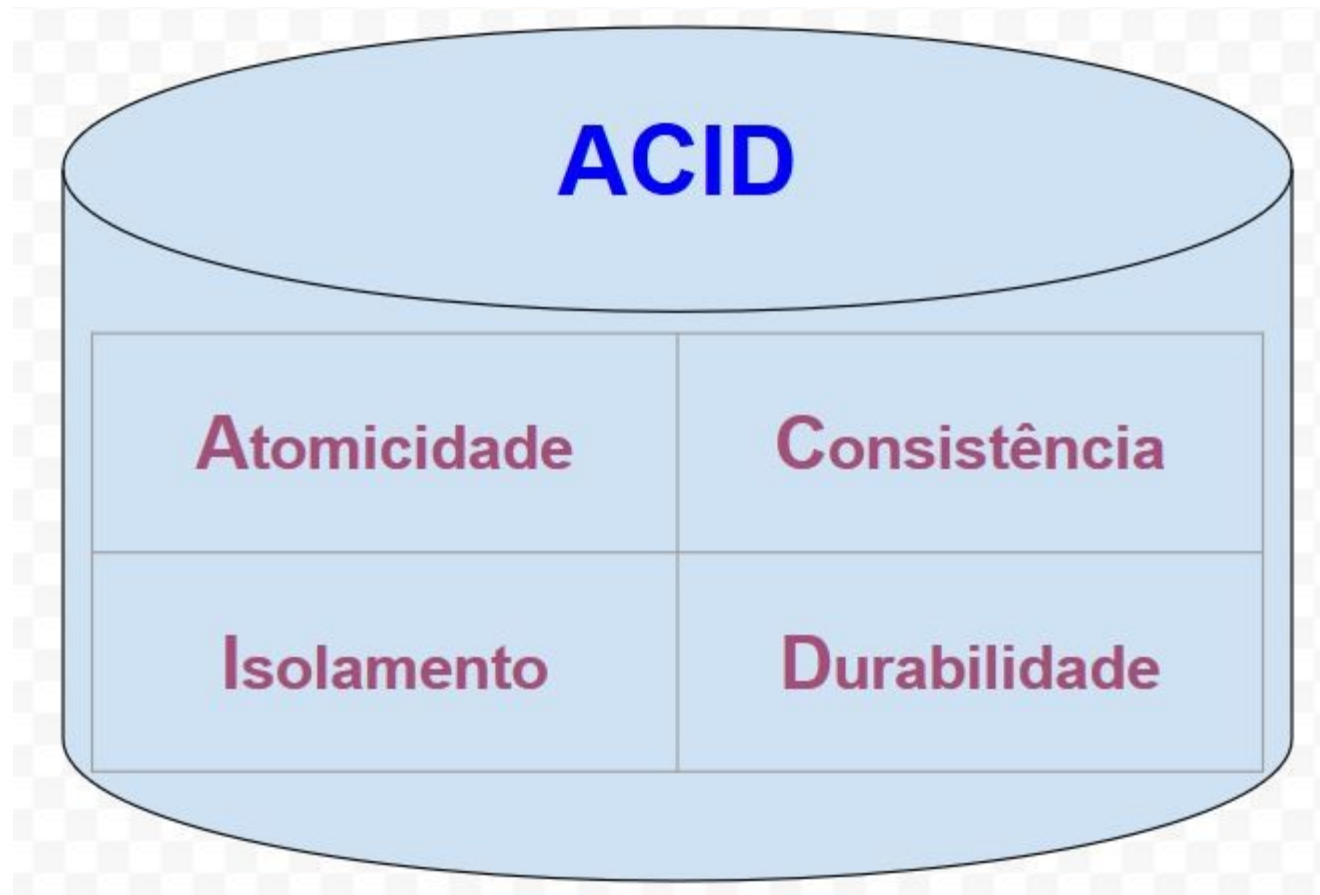
# ACID

- Em português:
- - Atomicidade,
- - Consistência,
- - Isolamento e
- - Durabilidade.

# ACID

- **ACID** diz respeito a um conjunto de propriedades em transações de bancos de dados que são importantes para garantir a validade dos dados mesmo quando ocorrem erros ou falhas.

# ACID



# ACID

- **Atomicidade:**

- - As transações são, geralmente, compostas de várias declarações (comandos / operações).
- - A atomicidade é uma propriedade que garante que cada transação seja tratada como uma entidade única, a qual deve ser executada por completo ou falhar completamente.

# Atomicidade (Novamente)

- Desta forma, todas as operações da transação devem ser executadas com sucesso para que a transação tenha sucesso.
- Se uma única operação que seja do bloco da transação falhar, toda a transação deverá ser cancelada – as transações são aplicadas de uma forma “tudo ou nada”.

# Consistência

- **Consistência:** A propriedade da consistência permite assegurar que uma transação somente leve o banco de dados de um estado válido a outro também válido, mantendo a estabilidade do banco.
- Os dados que são gravados devem sempre ser válidos.



# Isolamento

- **Isolamento**: É muito comum que **transações** sejam **executadas de forma concorrente**, ou seja, de forma que várias tabelas sejam lidas ou alteradas por vários usuários simultaneamente.
- Com a propriedade do **isolamento** a **execução concorrente** permite deixar o **banco de dados** no mesmo estado em que ele estaria caso as **transações** fossem executadas em sequência.

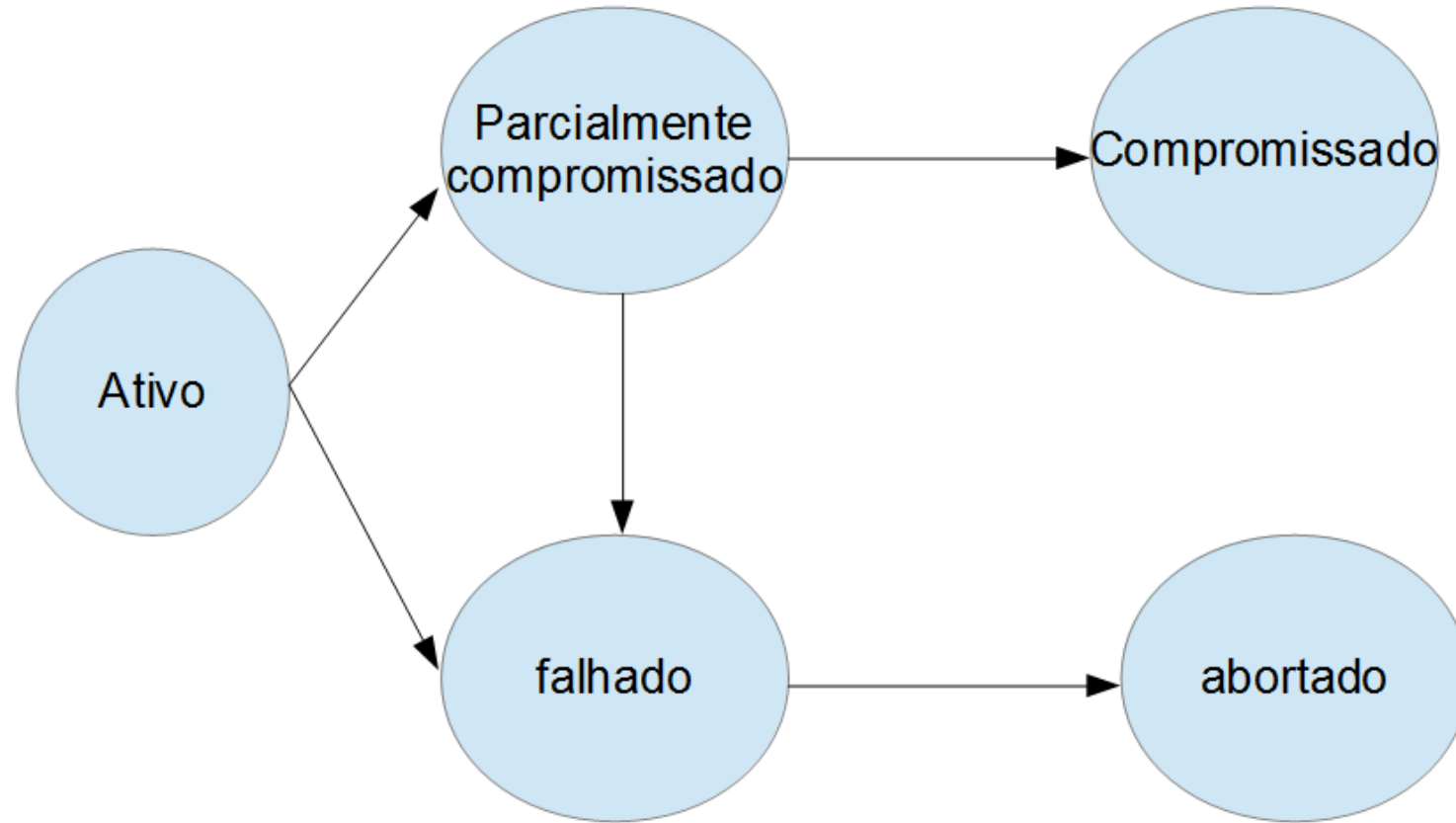
# Durabilidade

- **Durabilidade:** A propriedade da **durabilidade** garante que uma **transação**, uma vez executada (efetivada), **permanecerá neste estado mesmo que haja um problema grave no sistema, como travamento de sistema ou falta de energia elétrica no servidor.**
- Para isso, as **transações finalizadas** são gravadas em **dispositivos de memória permanente (não-volátil)**, como **discos rígidos**, de modo que os dados estejam **sempre disponíveis**, mesmo que a **instância do BD** seja reiniciada.

# Estados de uma Transação

- Uma transação precisa estar em um dos seguintes estados:
  - **Ativo:** estado inicial.
  - **Parcialmente Compromissado:** depois que a última instrução foi executada.
  - **Falhado:** depois da descoberta de que uma execução normal não pode mais prosseguir.
  - **Abortado:** depois que a transação foi desfeita e o BD foi restaurado ao seu estado anterior ao início da transação.
  - **Compromissado:** depois de a transação ser completada “com sucesso”.

# Diagrama de Estados de uma Transação



# Recuperação Baseada em LOG

- Considere novamente nosso **sistema bancário simplificado** e a **transação T** que transfere \$50 da conta A para a conta B com os **valores iniciais \$1.000 e \$2.000** para A e B, respectivamente.

# Recuperação Baseada em LOG

- A fim de atingir nossa meta de **atomicidade**, precisamos primeiro **gravar informações descrevendo as modificações no armazenamento estável** sem modificar o banco de dados em si.
- A estrutura mais largamente usada para registrar modificações no BD é o **log**.

# O LOG do Banco de Dados

- Cada registro do log descreve uma única gravação no banco de dados e tem os seguintes campos:
  - **Nome da transação**: nome único da transação que executou a operação write.
  - **Nome do item de dado**: o nome único do item de dado gravado.
  - **Valor antigo**: o valor do item de dado anterior à gravação.
  - **Novo Valor**: o valor que o item de dado terá depois da gravação.

# O LOG do Banco de Dados

- Representamos os vários tipos de registros de log como se segue:
  - **<Ti start>**. A transação Ti iniciou.
  - **<Ti, Xj, V1, V2>**. A transação Ti executou uma gravação num item de dado Xj. Xj tinha o valor V1 antes da gravação e terá o valor V2 depois.
  - **<Ti commit>**. A transação Ti foi compromissada.



# O LOG do Banco de Dados

- Toda vez que uma **transação** executa uma **gravação**, é essencial que o **registro de log** para essa gravação seja **criado antes que o BD seja modificado**.

# Modificação de Banco de Dados Adiada

- A técnica de adiar a modificação assegura a **atomicidade** da **transação** gravando todas as modificações do BD no **log**, mas adiando a execução de todas as operações **write** de uma transação até que a **transação** seja **parcialmente comprometida**.

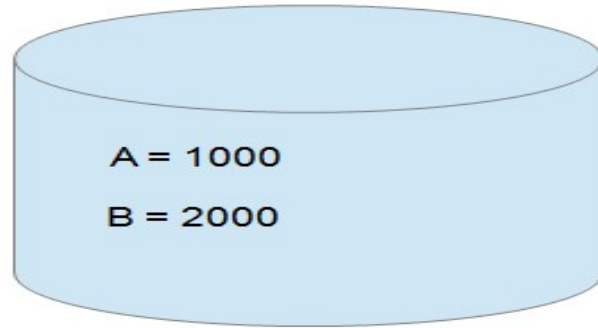
# Modificação de BD Adiada

- Quando uma **transação** torna-se **parcialmente comprometida**, a informação no **log** associada à **transação** é usada na **execução de gravações adiadas**.

# Modificação de BD Adiada

- A execução da transação T0 procede da seguinte maneira:
  - Um registro **<T0 start>** é escrito no **log**.
  - Uma operação **write** executada por **T0** resulta na gravação de um novo registro no **log**.
  - Quando T0 torna-se parcialmente compromissada, um registro **<T0 commit>** é escrito no **log**.

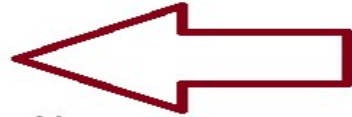
# Modificação de BD Adiada



Disco



Memória Principal

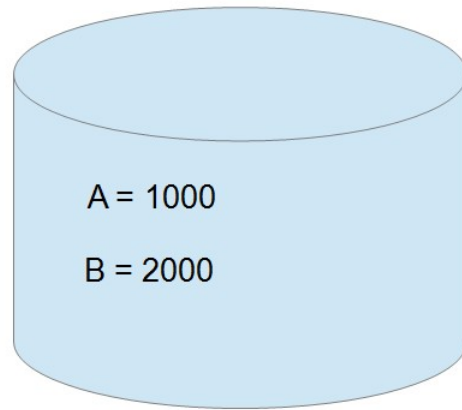


```
T: read(A, saldoA)
saldoA = saldoA - 50
write(A, saldoA)
read(B, saldoB)
saldoB = saldoB + 50
write(B, saldoB)
```

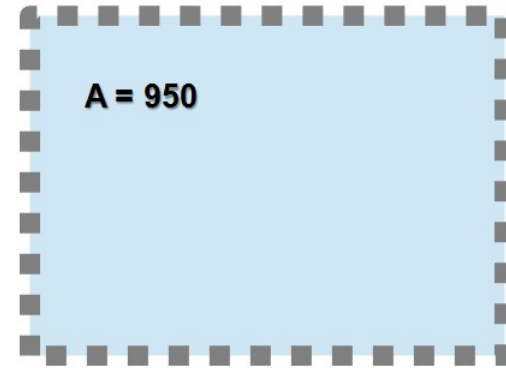
**Log:**

**<T0 start>**

# Modificação de BD Adiada



Disco



Memória Principal

T: **read**(A, saldoA)  
saldoA = saldoA - 50  
**write**(A, saldoA)  
**read**(B, saldoB)  
saldoB = saldoB + 50  
**write**(B, saldoB)

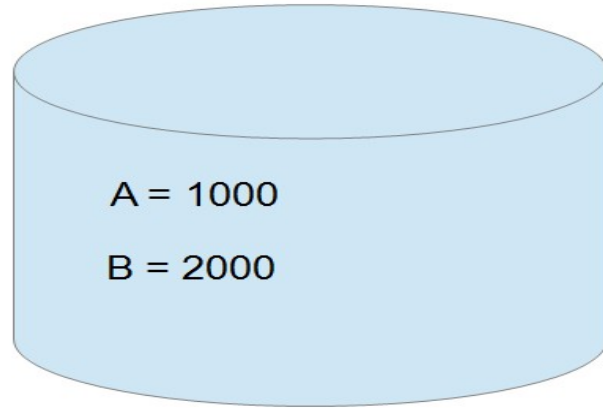


**Log:**

<T0 start>

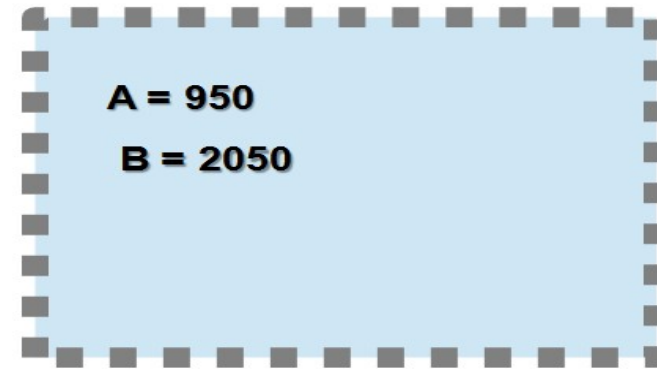
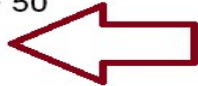
<T0, A, 950>

# Modificação de BD Adiada



Disco

T: **read**(A, saldoA)  
saldoA = saldoA - 50  
**write**(A, saldoA)  
**read**(B, saldoB)  
saldoB = saldoB + 50  
**write**(B, saldoB)



Memória Principal

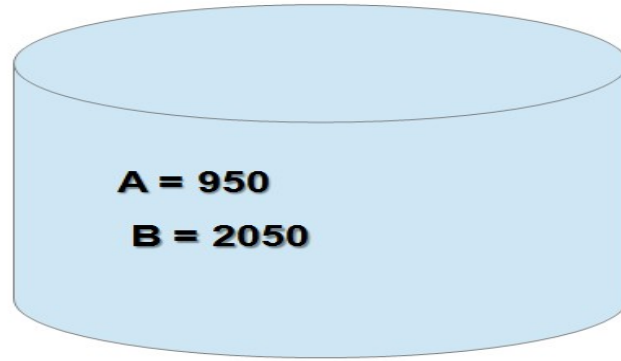
**Log:**

**<T0 start>**

**<T0, A, 950>**

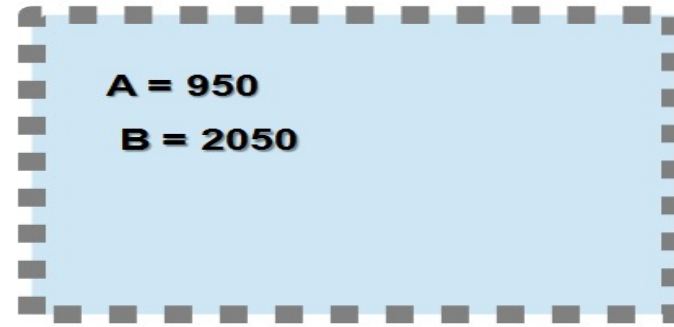
**<T0, B, 2050>**

# Modificação de BD Adiada



Disco

```
T: read(A, saldoA)
saldoA = saldoA - 50
write(A, saldoA)
read(B, saldoB)
saldoB = saldoB + 50
write(B, saldoB)
```



Memória Principal

**Log:**

<T0 start>

<T0, A, 950>

<T0, B, 2050>

<T0 commit>



# Modificação de BD Adiada

- Quando a transação T0 torna-se parcialmente comprometida, os registros associados a ela no log são usados na execução de gravações adiadas.
- Falhas podem acontecer enquanto essa atualização estiver ocorrendo.
- Os registros do log estão gravados em dispositivo de armazenamento estável.

# Modificações de BD Adiada

- Observe que apenas o novo valor do item de dado é requerido pela técnica de modificação adiada.
- Usando o log, o sistema pode manipular qualquer falha que resultar na perda de informações em dispositivo de armazenamento volátil.

# Modificação de BD Adiada

- O **esquema de recuperação** usa o seguinte procedimento de recuperação: **redo(T0)**.
- Ele **ajusta o valor de todos os itens de dados atualizados pela transação T0 para os novos valores**.
- A operação **redo** precisa ser **idempotente**: **executá-la diversas vezes precisa ser equivalente a executá-la apenas uma vez**.

# Modificação de BD Adiada

- Depois que uma falha tenha ocorrido:
  - O **subsistema de recuperação** consulta o **log** para determinar que transações precisam ser refeitas.
  - A **transação** precisa ser **refeita** se e somente se o **log** contiver o registro **<Ti start>** e o registro **<Ti commit>**.
  - Se o sistema cair depois que a **transação** completar sua execução, a informação no **log** será usada para restaurar o sistema para um estado consistente anterior.

# Modificação de BD Adiada

**Cenário 1:** conteúdo do log no momento de falha do sistema:

<T0 start>

<T0,A,950>

<T0,B,2050>

O sistema caiu **antes** que a **transação T0** tenha sido **completamente executada**. Assuma que a queda logo depois que o registro de log para **write(B,b1)** da transação **T0** tenha sido **gravado em meio estável**.

# Modificação de BD Adiada

- Quando o sistema volta, nenhuma ação de recuperação precisa ser feita.
- Afinal, nenhum registro executado aparece no **log**.
- Os valores das **contas A e B** permanecem **\$1.000** e **\$2.000**, respectivamente.

# Modificação de BD Adiada

**Cenário 2:** conteúdo do log no momento de falha do sistema:

<T0 start>

<T0, A, 950>

<T0, B, 2050>

<T0 commit>

<T1 start>

<T1, C, 600>

# Modificação de BD Adiada

- Agora assumimos que a queda ocorreu logo após o passo **write(C,c1)** da transação **T1** ter sido **gravado no meio estável**.
- A transação **T0** transfere \$50 da conta A para a conta B.
- A transação **T1** saca \$100 da conta C.



# Modificação de BD Adiada

- Quando o sistema voltar, a **operação redo(T0)** será executada uma vez que o **registro <T0 commit>** aparece no **log do disco**.
- Depois desta operação, os valores das contas A e B serão \$950 e \$2.050, respectivamente.
- O valor da conta C permanecerá \$700.

# Modificação de BD Adiada

Cenário 3: conteúdo do log no momento de falha do sistema:

<T0 start>

<T0, A, 950>

<T0, B, 2050>

<T0 commit>

<T1 start>

<T1, C, 600>

<T1 commit>

# Modificação de BD Adiada

- Assuma que a queda ocorreu logo após o registro de log **<T1 commit>** ser gravado no meio estável.
- Quando o sistema voltar, a existência de **<T1 commit>** e **<T0 commit>** tornarão necessária a execução das operações **redo(T0)** e **redo(T1)**.

# Modificação de BD Adiada

- Com isso, as **contas A, B e C** terão como saldo **\$950, \$2.050 e \$600**, respectivamente.
- **Caso ocorra uma segunda queda do sistema – durante a recuperação da primeira queda** – algumas mudanças podem ter sido feitas no **BD** como resultado das operações **redo()**, mas talvez nem todas as mudanças tenham sido realizadas.

# Modificação de BD Adiada

- Quando o sistema voltar depois da segunda queda, a recuperação procederá exatamente como nos exemplos anteriores.
- As ações de recuperação serão reiniciadas desde o início.

# Modificação de BD Imediata

- A técnica da **atualização imediata** permite a gravação de **modificações no BD** enquanto a **transação** está ainda no estado **ativo**.
- As modificações gravadas por transações ativas são chamadas **modificações descompromissadas** (***uncommitted***).

# Modificação de BD Imediata

- Se ocorrer uma queda ou falha da **transação**, o campo com **valor antigo** dos registros do **log** deve ser usado para restaurar os itens de dados modificados com os valores que tinham antes do início da transação.
- Isto é feito por meio da operação **undo**.

# Modificação de BD Imediata

- Antes que uma transação **Ti** inicie sua execução, o registro **<Ti start>** é gravado no **log**.
- Durante a execução, qualquer operação **write(X, xi)** de **Ti** é precedida pela gravação do novo registro apropriado no **log**.
- Quando torna-se parcialmente comprometida, o registro **<Ti commit>** é escrito no **log**.



# Modificação de BD Imediata

- Uma vez que a informação no log é usada na reconstrução do estado do banco de dados (BD), **não podemos permitir a atualização do BD antes do registro do log** correspondente ser escrito no armazenamento estável.

# Modificação de BD Imediata

- <T0 start>
- <T0, A, 1.000, 950>
- <T0, B, 2.000, 2.050>
- <T0 commit>
- <T1 start>
- <T1, C, 700, 600>
- <T1 commit>

# Modificação de BD Imediata

- O esquema de recuperação utiliza dois procedimentos:
  - **Undo(Ti)**, que restaura o valor de todos os itens de dados atualizados pela transação Ti aos **valores antigos**.
  - **Redo(Ti)**, que ajusta o valor de todos os itens de dados atualizados pela transação Ti aos **novos valores**.

# Modificação de BD Imediata

LOG

Banco de Dados

<T0 start>

<T0, A, 1.000, 950>

A = 950

<T0, B, 2.000, 2.050>

B = 2.050

<T0 commit>

<T1 start>

<T1, C, 700, 600>

C = 600

<T1 commit>

# Transações no PostgreSQL

- No **PostgreSQL** existem três comandos de suma importância para iniciar (**begin**) e finalizar (**commit** e **rollback**) uma transação.
- O comando **BEGIN** inicia uma transação.
- Em seguida promovemos uma **atualização** no **BD** por meio de um **INSERT**.

# Transações no PostgreSQL

```
Banco/postgres@PostgreSQL 12
Query Editor
1 CREATE TABLE CLIENTE (
2     NR_CLIENTE INTEGER NOT NULL PRIMARY KEY,
3     NOME VARCHAR(40) NOT NULL,
4     SEXO CHAR(1) NOT NULL
5 );
6
7 INSERT INTO CLIENTE VALUES (1035, 'JAMBIRA ETELVINA SOUZA', 'F'),
8                             (1040, 'ASDRUBAL SOARES', 'M'),
9                             (1086, 'DESIDÉRIO RIVIERA', 'M');
10
11 SELECT * FROM CLIENTE;
12
13
14
```

Banco/postgres@PostgreSQL 12			
Data Output			
	nr_cliente [PK] integer	nome character varying (40)	sexo character (1)
1	1035	JAMBIRA ETELVINA SOUZA	F
2	1040	ASDRUBAL SOARES	M
3	1086	DESIDÉRIO RIVIERA	M

# Transações no PostgreSQL



Banco/postgres@PostgreSQL 12

Query Editor

```
12
13 BEGIN; -- INÍCIO DA TRANSAÇÃO
14 INSERT INTO CLIENTE VALUES (1099, 'URRACA SANCHES', 'F');
15
16 SELECT * FROM CLIENTE; -- NOTE QUE A CLIENTE 'URRACA' FOI INSERIDA NA TABELA
17                        -- ANTES MESMO DA FINALIZAÇÃO DA TRANSAÇÃO
18                        -- MODIFICAÇÃO IMEDIATA DO BD ???
19
20
21
```



Banco/postgres@PostgreSQL 12

Data Output

	nr_cliente [PK] integer	nome character varying (40)	sexo character (1)
1	1035	JAMBIRA ETELVINA SOUZA	F
2	1040	ASDRUBAL SOARES	M
3	1086	DESIDÉRIO RIVIERA	M
4	1099	URRACA SANCHES	F

# Transações no PostgreSQL

```
Banco/postgres@PostgreSQL 12
Query Editor
12
13 BEGIN; -- INÍCIO DA TRANSAÇÃO
14     INSERT INTO CLIENTE VALUES (1099, 'URRACA SANCHES', 'F');
15
16     SELECT * FROM CLIENTE; -- NOTE QUE A CLIENTE 'URRACA' FOI INSERIDA NA TABELA
17                             -- ANTES MESMO DA FINALIZAÇÃO DA TRANSAÇÃO
18                             -- MODIFICAÇÃO IMEDIATA DO BD ???
19
20 ROLLBACK; -- FIM DA TRANSAÇÃO (A MESMA NÃO FOI COMPROMISSADA. TUDO FOIDESFEITO).
21
22 SELECT * FROM CLIENTE; -- OBSERVE QUE URRACA DESAPARECEU.
23
24
```

Banco/postgres@PostgreSQL 12			
Data Output			
	nr_cliente [PK] integer	nome character varying (40)	sexo character (1)
1	1035	JAMBIRA ETELVINA SOUZA	F
2	1040	ASDRUBAL SOARES	M
3	1086	DESIDÉRIO RIVIERA	M



# Transações no PostgreSQL



Banco/postgres@PostgreSQL 12

Query Editor

```
23
24 BEGIN; -- INÍCIO DA TRANSAÇÃO
25     INSERT INTO CLIENTE VALUES (1099, 'URRACA SANCHES', 'F'); -- REINSERÇÃO DE URRACA
26 COMMIT; -- FIM DA TRANSAÇÃO (COMPROMISSADA. URRACA FOI CONFIRMADA).
27
28 SELECT * FROM CLIENTE;
29
```



Banco/postgres@PostgreSQL 12

Data Output

	nr_cliente [PK] integer	nome character varying (40)	sexo character (1)
1	1035	JAMBIRA ETELVINA SOUZA	F
2	1040	ASDRUBAL SOARES	M
3	1086	DESIDÉRIO RIVIERA	M
4	1099	URRACA SANCHES	F

# Transações no PostgreSQL



Banco/postgres@PostgreSQL 12

Query Editor

```
29
30 CREATE TABLE CONTA (
31     NR_CONTA INTEGER NOT NULL PRIMARY KEY,
32     NR_CLIENTE INTEGER NOT NULL,
33     SALDO NUMERIC(11,2) NOT NULL,
34
35     FOREIGN KEY (NR_CLIENTE) REFERENCES CLIENTE (NR_CLIENTE) ON DELETE RESTRICT
36 );
37
38 INSERT INTO CONTA VALUES (12650, 1099, 2000), (25099, 1099, 150), (37000, 1086, 4000);
39
40 SELECT * FROM CONTA;
41
42
```



Banco/postgres@PostgreSQL 12

Data Output

	nr_conta [PK] integer	nr_cliente integer	saldo numeric (11,2)
1	12650	1099	2000.00
2	25099	1099	150.00
3	37000	1086	4000.00

# Transações no PostgreSQL

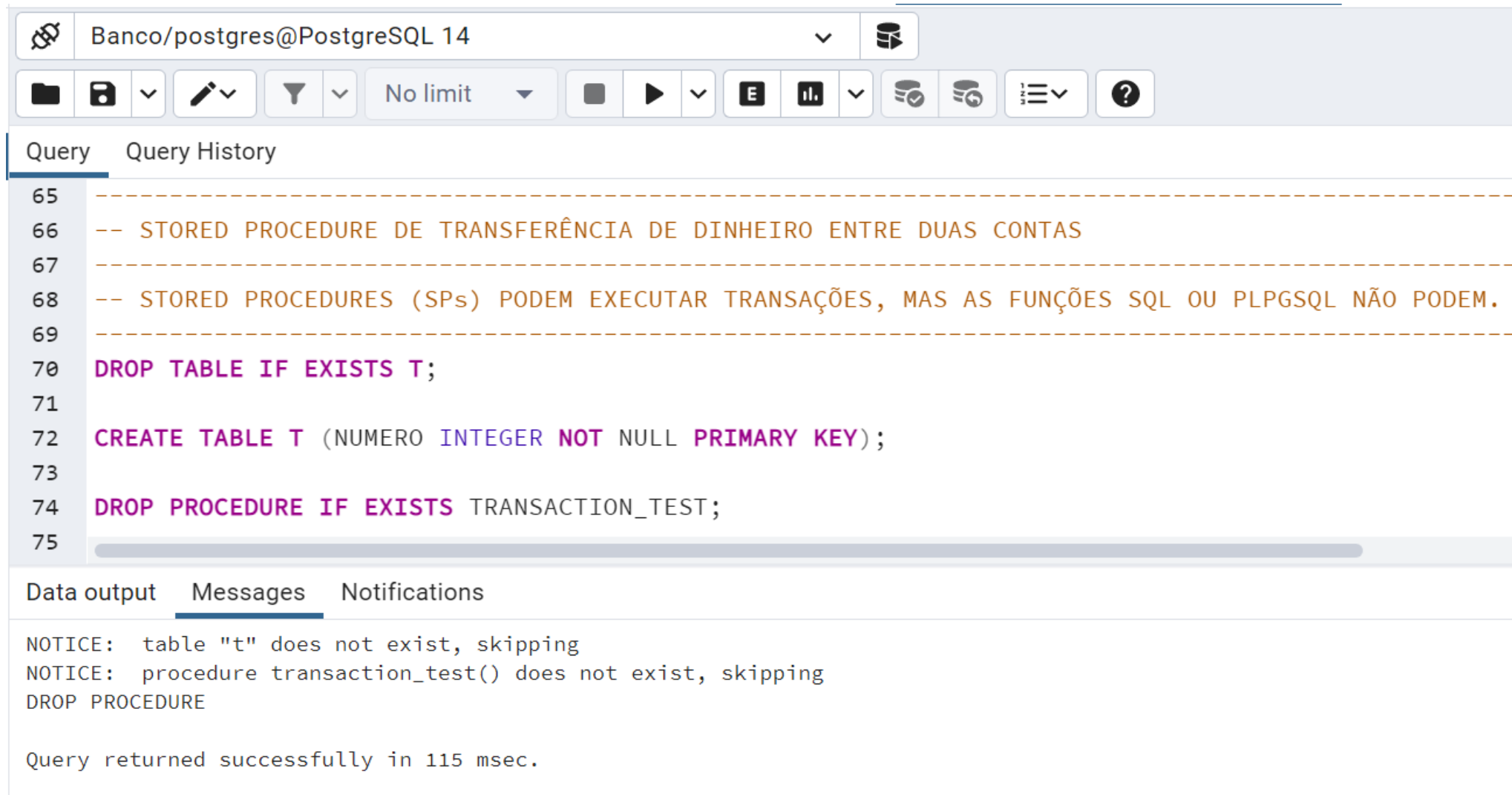


Banco/postgres@PostgreSQL 12

Query Editor

```
42 DROP TABLE IF EXISTS MOVIMENTACAO;
43
44 CREATE TABLE MOVIMENTACAO (
45     NR_MOVIMENTACAO SERIAL NOT NULL PRIMARY KEY,
46     NR_CONTA INTEGER NOT NULL,
47     DT_MOVIMENTACAO DATE NOT NULL CHECK (DT_MOVIMENTACAO = CURRENT_DATE),
48     OPERACAO CHAR(1) NOT NULL CHECK (OPERACAO = 'D' OR OPERACAO = 'C'),
49     VALOR NUMERIC(11,2) NOT NULL,
50     LOCAL VARCHAR(50) NOT NULL,
51
52     FOREIGN KEY (NR_CONTA) REFERENCES CONTA (NR_CONTA) ON DELETE CASCADE
53 );
54
55
```

# Transações no PostgreSQL



The screenshot displays the PostgreSQL pgAdmin interface. At the top, the connection is labeled 'Banco/postgres@PostgreSQL 14'. Below this is a toolbar with various icons for file operations, query execution, and settings. The main area is divided into two tabs: 'Query' and 'Query History'. The 'Query' tab is active, showing a SQL script with line numbers 65 through 75. The script contains comments in Portuguese and SQL commands to drop a table and a procedure, and create a table. Below the query editor, there are three tabs: 'Data output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the output of the query execution, which includes two notices about non-existent table and procedure, and a confirmation that the query was successful.

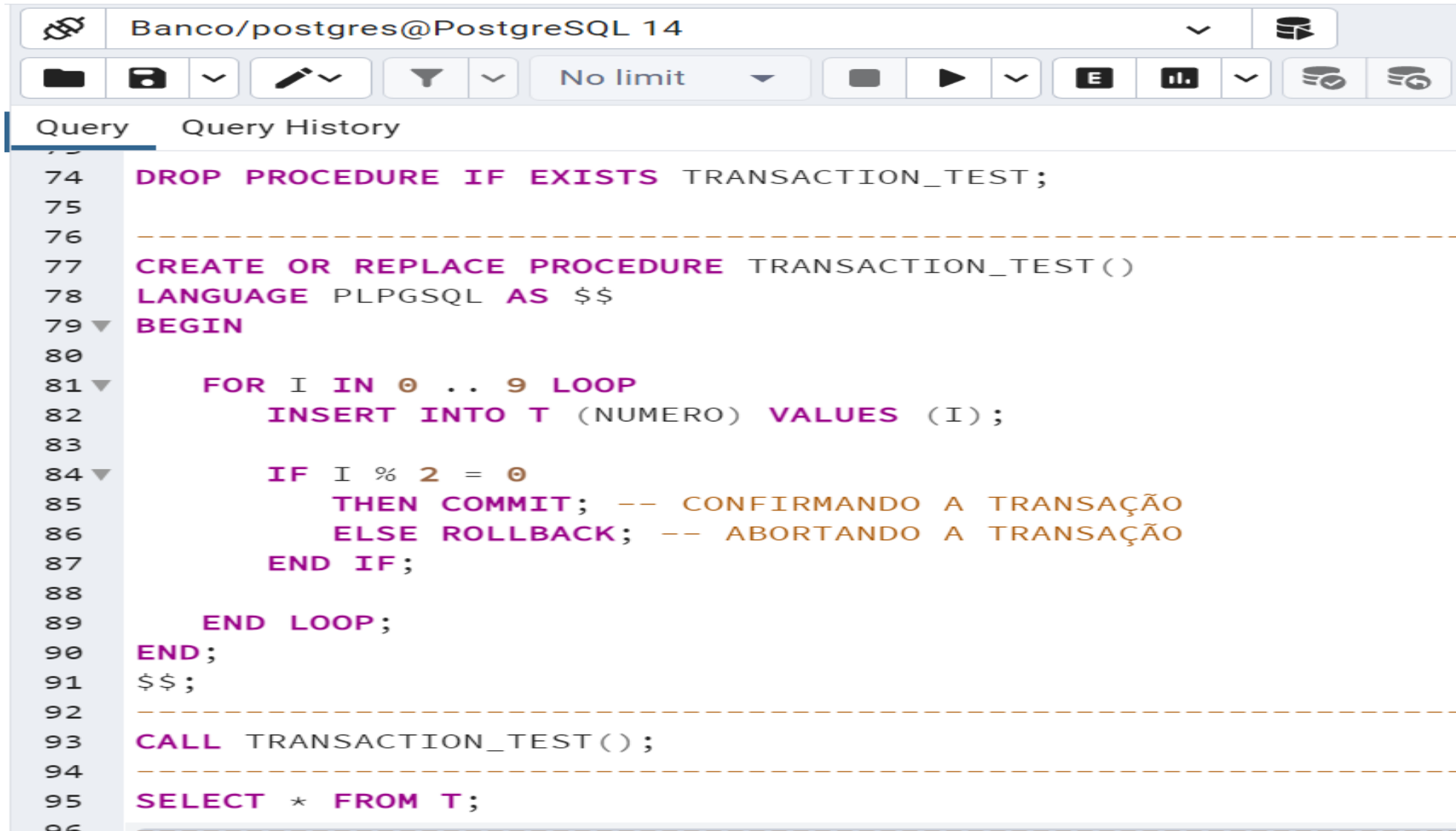
```
65 -----  
66 -- STORED PROCEDURE DE TRANSFERÊNCIA DE DINHEIRO ENTRE DUAS CONTAS  
67 -----  
68 -- STORED PROCEDURES (SPs) PODEM EXECUTAR TRANSAÇÕES, MAS AS FUNÇÕES SQL OU PLPGSQL NÃO PODEM.  
69 -----  
70 DROP TABLE IF EXISTS T;  
71  
72 CREATE TABLE T (NUMERO INTEGER NOT NULL PRIMARY KEY);  
73  
74 DROP PROCEDURE IF EXISTS TRANSACTION_TEST;  
75
```

Data output   **Messages**   Notifications

NOTICE: table "t" does not exist, skipping  
NOTICE: procedure transaction\_test() does not exist, skipping  
DROP PROCEDURE

Query returned successfully in 115 msec.

# Transações no PostgreSQL



The screenshot shows a PostgreSQL query editor window. The title bar indicates the connection is to 'Banco/postgres@PostgreSQL 14'. The interface includes a toolbar with icons for file operations, query execution, and settings. The 'Query' tab is active, displaying a PL/SQL procedure named 'TRANSACTION\_TEST'. The procedure is designed to insert numbers 0 through 9 into a table 'T'. For each number, it checks if the number is even (I % 2 = 0). If even, it commits the transaction; if odd, it rolls back the transaction. After the loop, it calls the 'TRANSACTION\_TEST' procedure and then selects all data from table 'T'.

```
74 DROP PROCEDURE IF EXISTS TRANSACTION_TEST;  
75  
76 -----  
77 CREATE OR REPLACE PROCEDURE TRANSACTION_TEST()  
78 LANGUAGE PLPGSQL AS $$  
79 BEGIN  
80  
81     FOR I IN 0 .. 9 LOOP  
82         INSERT INTO T (NUMERO) VALUES (I);  
83  
84         IF I % 2 = 0  
85             THEN COMMIT; -- CONFIRMANDO A TRANSAÇÃO  
86             ELSE ROLLBACK; -- ABORTANDO A TRANSAÇÃO  
87         END IF;  
88  
89     END LOOP;  
90 END;  
91 $$;  
92 -----  
93 CALL TRANSACTION_TEST();  
94 -----  
95 SELECT * FROM T;  
96
```

# Transações no PostgreSQL

The screenshot shows a PostgreSQL client interface. At the top, the connection string is 'Banco/postgres@PostgreSQL 14'. Below this is a toolbar with icons for file operations, query execution, and filters. The 'Query' tab is active, showing a SQL query with line numbers 92 to 96. The query consists of a function call followed by a SELECT statement. The 'Data output' tab is also visible, showing a table with two columns: 'numero' (integer, primary key) and an unnamed column. The table contains five rows of data.

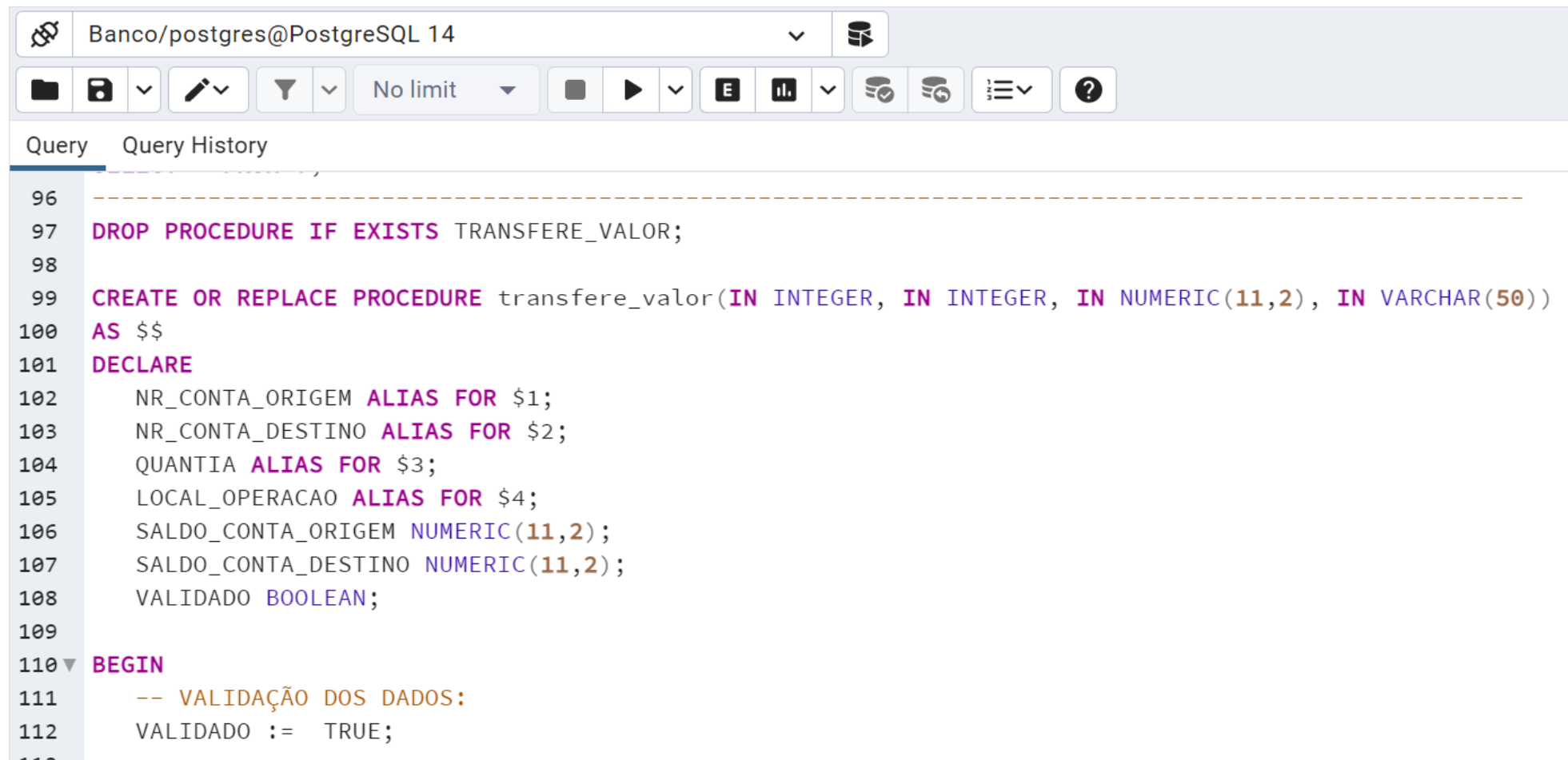
Query

```
92  
93 CALL TRANSACTION_TEST();  
94  
95 SELECT * FROM T;  
96
```

Data output

	numero [PK] integer	
1		0
2		2
3		4
4		6
5		8

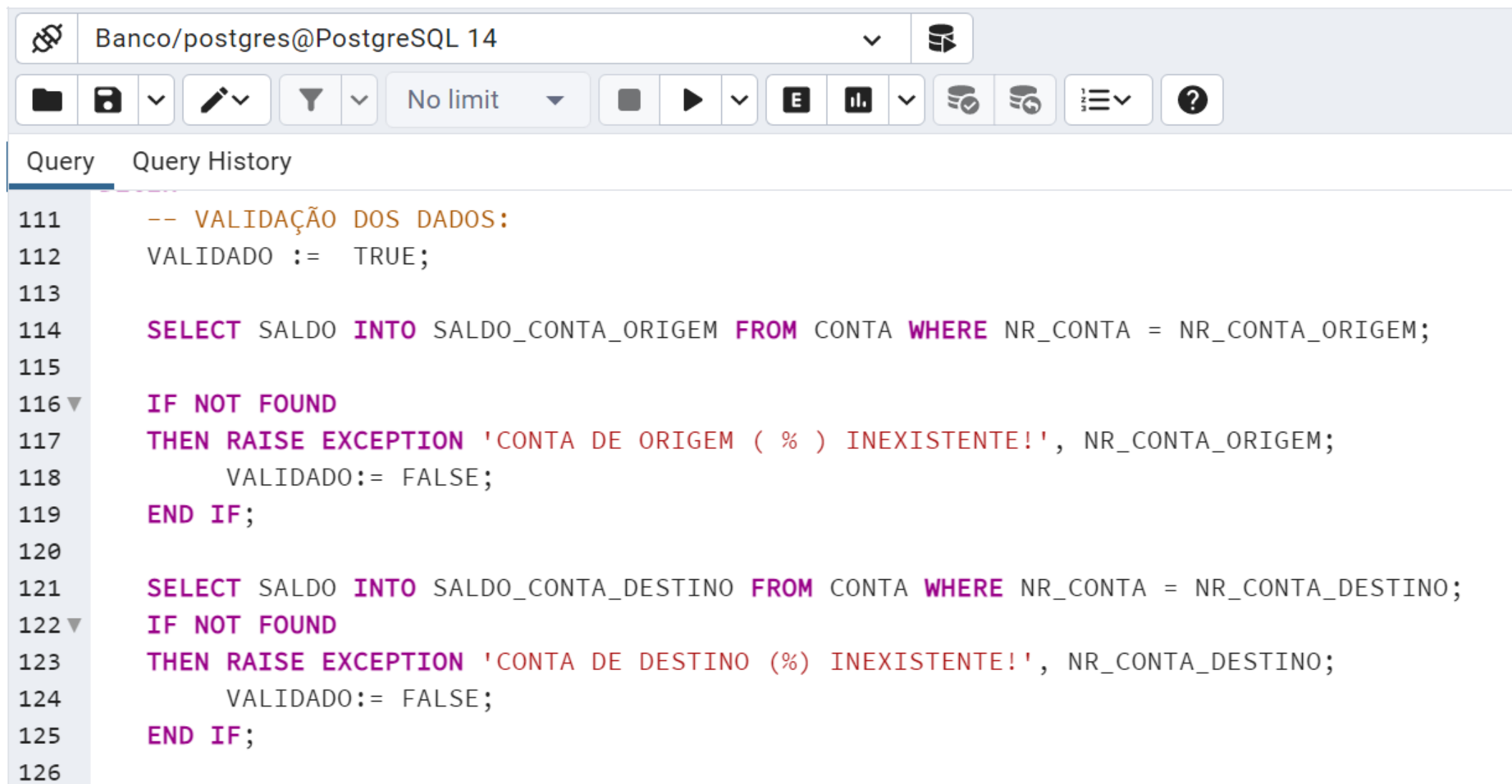
# Transações no PostgreSQL



The screenshot shows a PostgreSQL client interface with a toolbar at the top containing icons for file operations, query execution, and database management. The main area displays a SQL query editor with a line number column on the left. The query is a PL/pgSQL procedure definition for 'transfere\_valor'.

```
96 -----
97 DROP PROCEDURE IF EXISTS TRANSFERE_VALOR;
98
99 CREATE OR REPLACE PROCEDURE transfere_valor(IN INTEGER, IN INTEGER, IN NUMERIC(11,2), IN VARCHAR(50))
100 AS $$
101 DECLARE
102     NR_CONTA_ORIGEM ALIAS FOR $1;
103     NR_CONTA_DESTINO ALIAS FOR $2;
104     QUANTIA ALIAS FOR $3;
105     LOCAL_OPERACAO ALIAS FOR $4;
106     SALDO_CONTA_ORIGEM NUMERIC(11,2);
107     SALDO_CONTA_DESTINO NUMERIC(11,2);
108     VALIDADO BOOLEAN;
109
110 BEGIN
111     -- VALIDAÇÃO DOS DADOS:
112     VALIDADO := TRUE;
```

# Transações no PostgreSQL

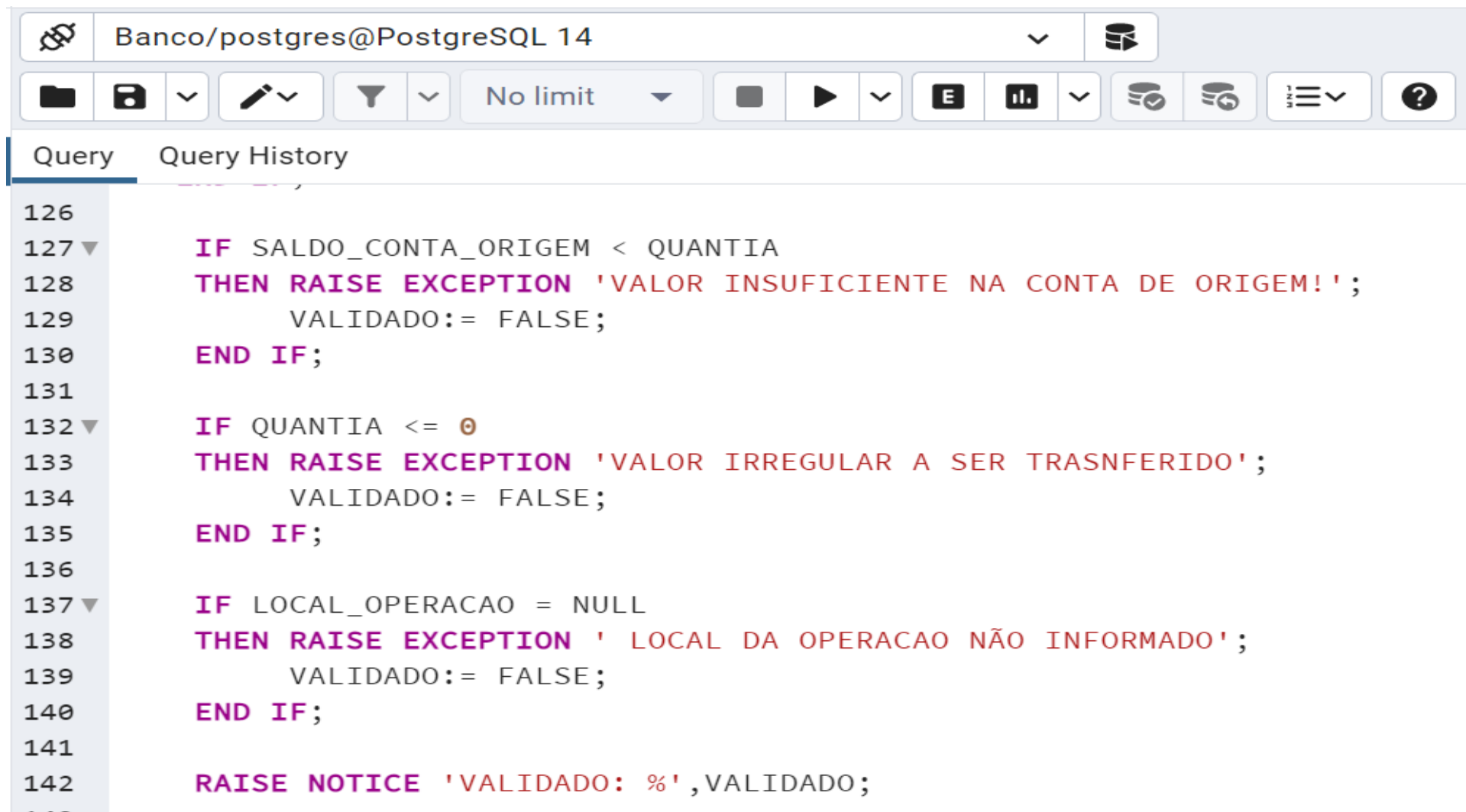


The screenshot shows a PostgreSQL client window titled 'Banco/postgres@PostgreSQL 14'. The interface includes a toolbar with icons for file operations, query execution, and settings. Below the toolbar, there are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL script with line numbers 111 through 126. The script performs data validation by checking for the existence of source and destination accounts in a 'CONTA' table. It uses 'SELECT INTO' to store results and 'RAISE EXCEPTION' to handle 'IF NOT FOUND' scenarios.

```
111  -- VALIDAÇÃO DOS DADOS:
112  VALIDADO := TRUE;
113
114  SELECT SALDO INTO SALDO_CONTA_ORIGEM FROM CONTA WHERE NR_CONTA = NR_CONTA_ORIGEM;
115
116  IF NOT FOUND
117  THEN RAISE EXCEPTION 'CONTA DE ORIGEM ( % ) INEXISTENTE!', NR_CONTA_ORIGEM;
118       VALIDADO:= FALSE;
119  END IF;
120
121  SELECT SALDO INTO SALDO_CONTA_DESTINO FROM CONTA WHERE NR_CONTA = NR_CONTA_DESTINO;
122  IF NOT FOUND
123  THEN RAISE EXCEPTION 'CONTA DE DESTINO ( % ) INEXISTENTE!', NR_CONTA_DESTINO;
124       VALIDADO:= FALSE;
125  END IF;
126
```



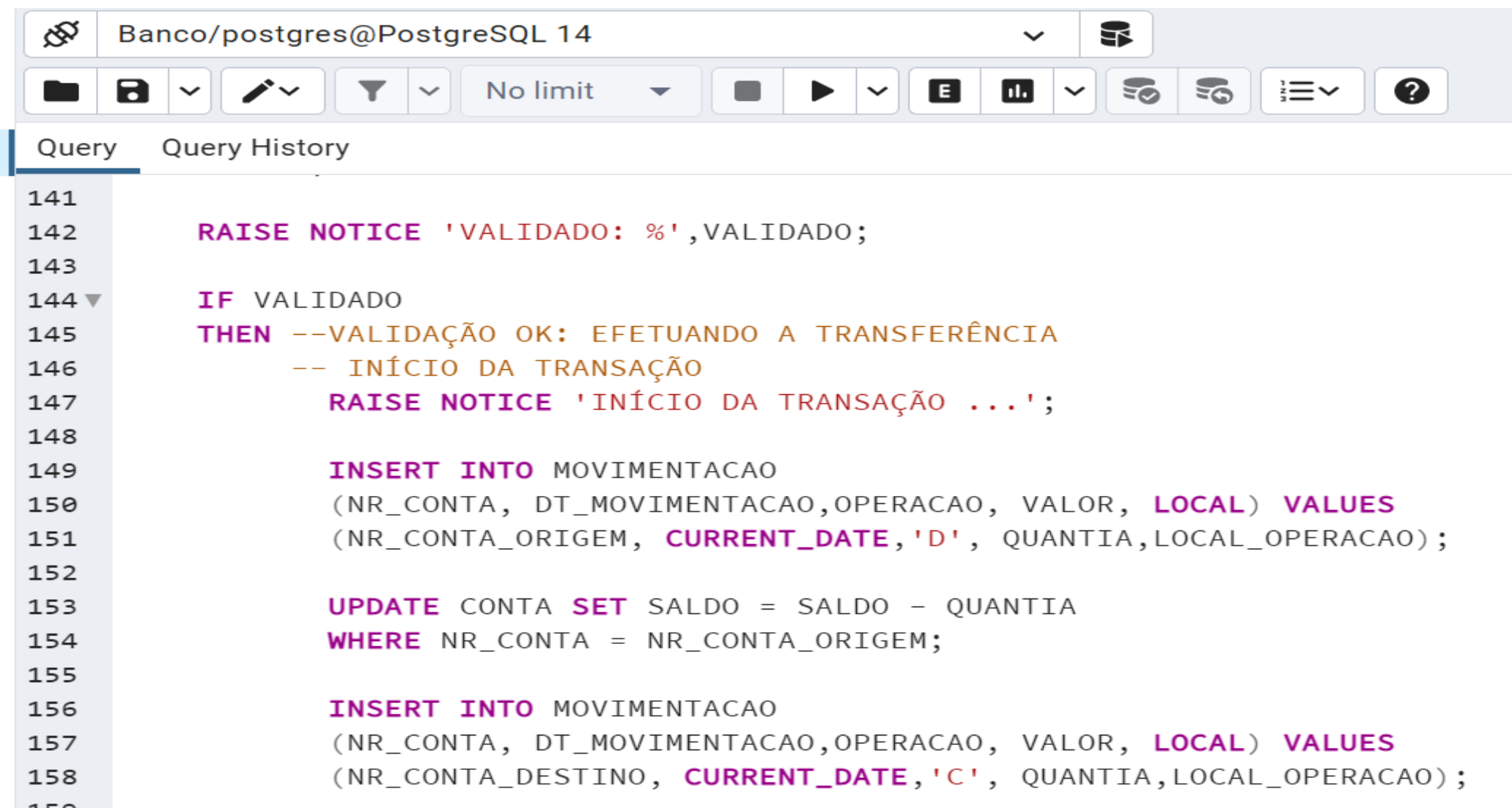
# Transações no PostgreSQL



The image shows a screenshot of a PostgreSQL query editor interface. At the top, there is a toolbar with various icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is selected, displaying a PL/pgSQL function snippet. The code is as follows:

```
126
127 ▼ IF SALDO_CONTA_ORIGEM < QUANTIA
128 THEN RAISE EXCEPTION 'VALOR INSUFICIENTE NA CONTA DE ORIGEM!';
129     VALIDADO:= FALSE;
130 END IF;
131
132 ▼ IF QUANTIA <= 0
133 THEN RAISE EXCEPTION 'VALOR IRREGULAR A SER TRASNFERIDO';
134     VALIDADO:= FALSE;
135 END IF;
136
137 ▼ IF LOCAL_OPERACAO = NULL
138 THEN RAISE EXCEPTION ' LOCAL DA OPERACAO NÃO INFORMADO';
139     VALIDADO:= FALSE;
140 END IF;
141
142 RAISE NOTICE 'VALIDADO: %',VALIDADO;
```

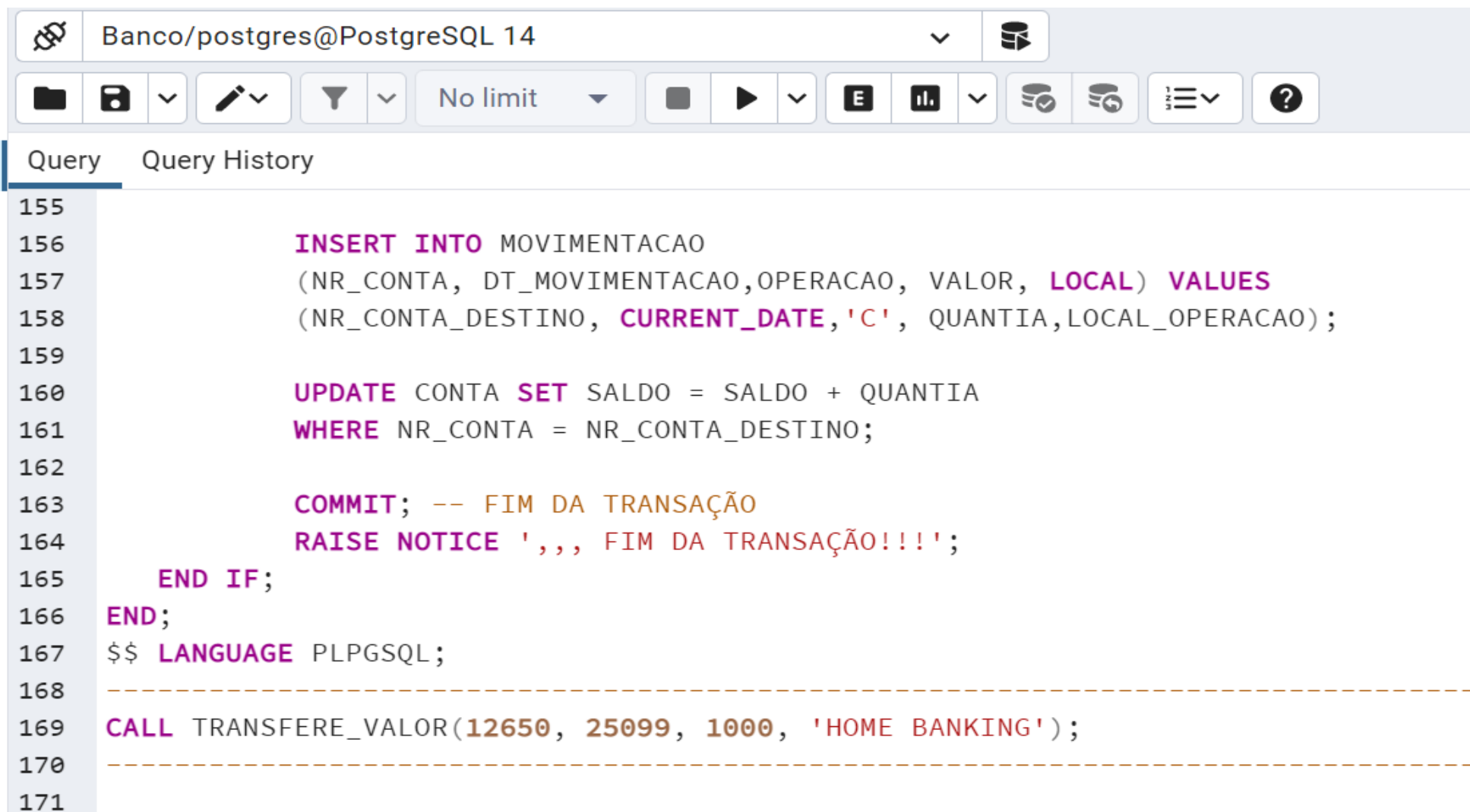
# Transações no PostgreSQL



The screenshot shows a PostgreSQL query editor window. The title bar indicates the connection is to 'Banco/postgres@PostgreSQL 14'. The interface includes a toolbar with icons for file operations, query execution, and settings. Below the toolbar, there are tabs for 'Query' and 'Query History'. The main area displays a SQL script with line numbers on the left. The script is a PL/pgSQL function that checks a 'VALIDADO' flag. If it is valid, it performs a transaction: it inserts a record into the 'MOVIMENTACAO' table, updates the 'CONTA' table to subtract the 'QUANTIA' from the 'SALDO', and then inserts another record into 'MOVIMENTACAO' for the destination account. The script uses 'RAISE NOTICE' to provide feedback at various stages.

```
141
142     RAISE NOTICE 'VALIDADO: %',VALIDADO;
143
144 IF VALIDADO
145 THEN --VALIDAÇÃO OK: EFETUANDO A TRANSFERÊNCIA
146     -- INÍCIO DA TRANSAÇÃO
147     RAISE NOTICE 'INÍCIO DA TRANSAÇÃO ...';
148
149     INSERT INTO MOVIMENTACAO
150     (NR_CONTA, DT_MOVIMENTACAO,OPERACAO, VALOR, LOCAL) VALUES
151     (NR_CONTA_ORIGEM, CURRENT_DATE,'D', QUANTIA,LOCAL_OPERACAO);
152
153     UPDATE CONTA SET SALDO = SALDO - QUANTIA
154     WHERE NR_CONTA = NR_CONTA_ORIGEM;
155
156     INSERT INTO MOVIMENTACAO
157     (NR_CONTA, DT_MOVIMENTACAO,OPERACAO, VALOR, LOCAL) VALUES
158     (NR_CONTA_DESTINO, CURRENT_DATE,'C', QUANTIA,LOCAL_OPERACAO);
159
```

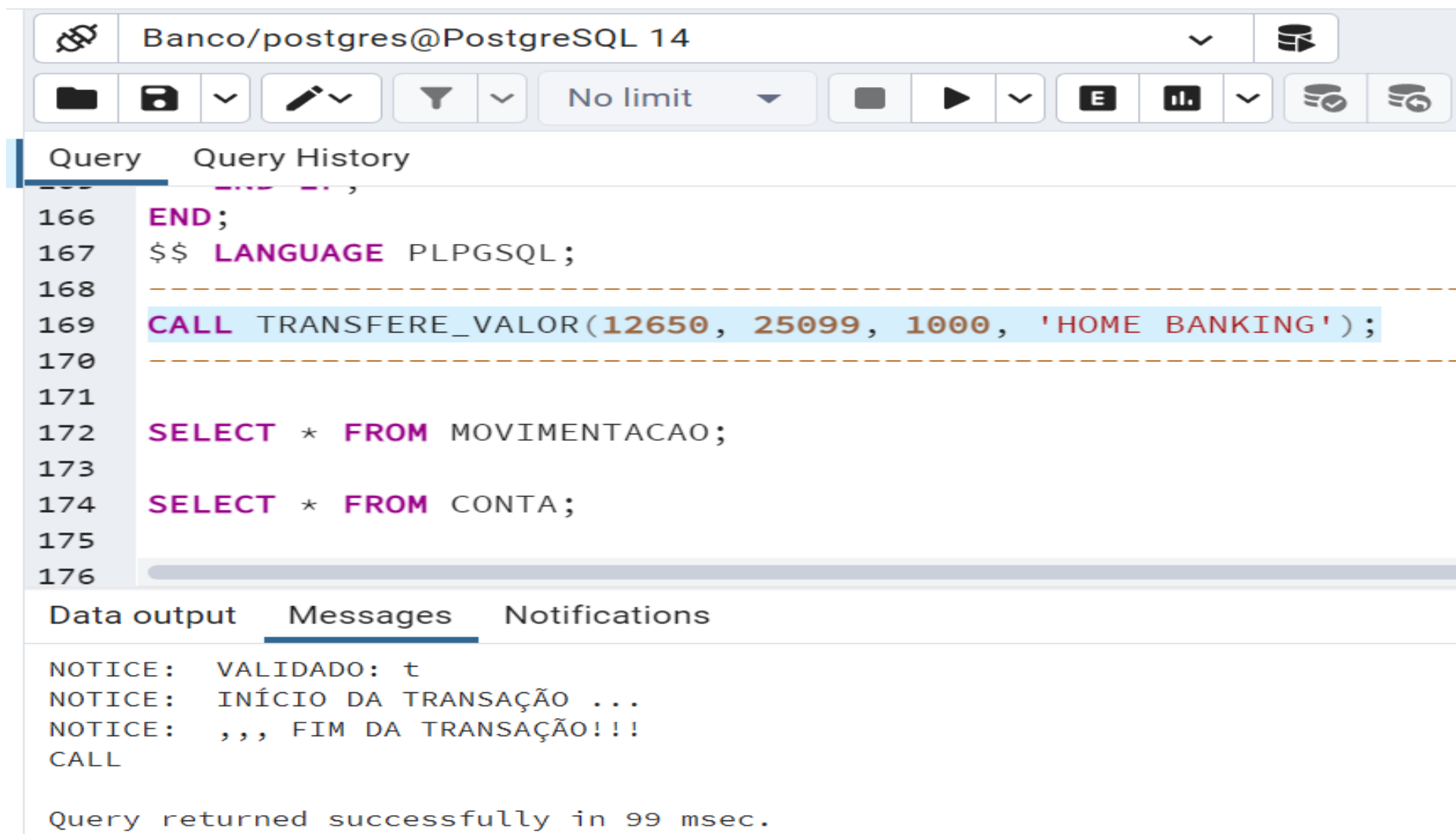
# Transações no PostgreSQL



The screenshot shows a PostgreSQL query editor window. The title bar indicates the connection is to 'Banco/postgres@PostgreSQL 14'. The interface includes a toolbar with icons for file operations, query execution, and settings. Below the toolbar, there are tabs for 'Query' and 'Query History'. The main area displays a SQL script with line numbers on the left. The script defines a function to handle a transaction for transferring value between accounts. It includes an INSERT statement for the movement, an UPDATE statement for the account balance, a COMMIT statement, and a RAISE NOTICE statement. The function is called with specific values at the bottom of the script.

```
155
156         INSERT INTO MOVIMENTACAO
157         (NR_CONTA, DT_MOVIMENTACAO, OPERACAO, VALOR, LOCAL) VALUES
158         (NR_CONTA_DESTINO, CURRENT_DATE, 'C', QUANTIA, LOCAL_OPERACAO);
159
160         UPDATE CONTA SET SALDO = SALDO + QUANTIA
161         WHERE NR_CONTA = NR_CONTA_DESTINO;
162
163         COMMIT; -- FIM DA TRANSAÇÃO
164         RAISE NOTICE ',,, FIM DA TRANSAÇÃO!!!';
165     END IF;
166 END;
167 $$ LANGUAGE PLPGSQL;
168 -----
169 CALL TRANSFERE_VALOR(12650, 25099, 1000, 'HOME BANKING');
170 -----
171
```

# Transações no PostgreSQL



The screenshot shows a PostgreSQL client interface with the following components:

- Top Bar:** Displays the connection name "Banco/postgres@PostgreSQL 14" and various icons for file operations, filters, and execution.
- Query Editor:** Contains a SQL script with line numbers 166 through 176. The script includes a transaction block, a call to a function, and two SELECT statements. The function call is highlighted in blue.
- Execution Results:** The "Messages" tab is active, showing the output of the query execution.

**SQL Script:**

```
166 END;  
167 $$ LANGUAGE PLPGSQL;  
168 -----  
169 CALL TRANSFERE_VALOR(12650, 25099, 1000, 'HOME BANKING');  
170 -----  
171  
172 SELECT * FROM MOVIMENTACAO;  
173  
174 SELECT * FROM CONTA;  
175  
176
```

**Execution Results (Messages):**

```
NOTICE:  VALIDADO: t  
NOTICE:  INÍCIO DA TRANSAÇÃO ...  
NOTICE: ,,, FIM DA TRANSAÇÃO!!!  
CALL  
  
Query returned successfully in 99 msec.
```

# Transações no PostgreSQL

Banco/postgres@PostgreSQL 14

No limit

Query Query History

171

172

173

174

175

176

SELECT \* FROM MOVIMENTACAO;

SELECT \* FROM CONTA;

Data output Messages Notifications

	nr_movimentacao [PK] integer	nr_conta integer	dt_movimentacao date	operacao character (1)	valor numeric (11,2)	local character varying (50)
1	1	12650	2022-08-16	D	1000.00	HOME BANKING
2	2	25099	2022-08-16	C	1000.00	HOME BANKING

# Transações no PostgreSQL

The screenshot shows a PostgreSQL client interface. At the top, the database connection is 'Banco/postgres@PostgreSQL 14'. Below the connection bar, there are icons for file operations and a 'No limit' dropdown. The main area is divided into 'Query' and 'Query History' tabs. The 'Query' tab is active, showing two SQL queries: 'SELECT \* FROM MOVIMENTACAO;' and 'SELECT \* FROM CONTA;'. The second query is highlighted in blue. Below the queries, there are tabs for 'Data output', 'Messages', and 'Notifications'. The 'Data output' tab is active, displaying a table with three rows of data. The table has four columns: 'nr\_conta' (integer, PK), 'nr\_cliente' (integer), and 'saldo' (numeric (11,2)). The data rows are: (1, 37000, 1086, 4000.00), (2, 12650, 1099, 1000.00), and (3, 25099, 1099, 1150.00).

Banco/postgres@PostgreSQL 14

Query Query History

```
172 SELECT * FROM MOVIMENTACAO;
173
174 SELECT * FROM CONTA;
175
176
177
```

Data output Messages Notifications

	nr_conta [PK] integer	nr_cliente integer	saldo numeric (11,2)
1	37000	1086	4000.00
2	12650	1099	1000.00
3	25099	1099	1150.00

# WAL

- No PostgreSQL, o WAL (Write-Ahead Log, ou Registro de Gravação Antecipada) é um mecanismo fundamental de integridade e recuperação de dados.
- É um **log sequencial** em disco que registra todas as alterações feitas nos dados do banco de dados **antes** de elas serem efetivamente gravadas nas tabelas.
- Em vez de gravar diretamente cada modificação nas páginas de dados do banco, o PostgreSQL primeiro grava no **WAL** um registro descrevendo a operação. Só depois, em segundo plano, os dados são efetivamente aplicados nos arquivos de tabelas.

# WAL

## Para que serve?

**Garantir consistência:** Se o servidor cair (pane elétrica, crash do SO, etc.), o PostgreSQL pode **reconstruir o estado exato** dos dados a partir do último checkpoint e dos WALs.

**Recuperação de falhas (Crash Recovery):** Após uma falha, o banco lê o WAL e "reaplica" as operações pendentes até deixar os dados consistentes.

**Replicação:** O WAL é usado na **replicação física** entre servidores (streaming replication). O nó primário envia os registros de WAL para os nós secundários, que os aplicam localmente.

**Ponto no tempo (PITR - Point In Time Recovery):** Com os WALs, é possível restaurar um backup até um momento exato (por exemplo, antes de uma exclusão acidental de dados).



# MVCC

- **MVCC (Multiversion Concurrency Control)** é o mecanismo de controle de concorrência usado pelo PostgreSQL para permitir que **várias transações leiam e escrevam dados ao mesmo tempo, sem bloqueios desnecessários** e mantendo a **consistência**.
- Em vez de sobrescrever um registro quando ele é atualizado, o PostgreSQL **cria uma nova versão** dele e marca a antiga como obsoleta, mas ainda visível para transações que começaram antes da mudança.

# MVCC

- Para que serve o MVCC?

## Leituras consistentes sem bloqueio

Cada transação "enxerga" o banco de dados como estava no momento em que ela começou, mesmo que outras transações estejam modificando os mesmos dados ao mesmo tempo.

Ou seja: um **SELECT** nunca fica travado esperando que outro **UPDATE** ou **INSERT**.

## Exceção:

Se o **SELECT** utiliza um **FOR UPDATE** ou **FOR SHARE**, ele **vai esperar** pelo fim da **transação** que possui o **bloqueio**, porque nesse caso a consulta está pedindo explicitamente um **bloqueio (lock)**.

# MVCC

O **DELETE** pode sim ficar bloqueado, porque ele precisa marcar a linha como removida e isso só pode acontecer se não houver outra transação segurando lock de escrita nessa linha.

## EXEMPLO:

*Transação A faz:*

**BEGIN;**  
**UPDATE cliente SET nome = 'Maria' WHERE id = 1;**

(mas ainda não concluiu a transação com um COMMIT ou ROLLBACK)

*Transação B tenta:*

***DELETE FROM cliente WHERE id = 1;***

(Aqui a transação B vai ficar esperando até a A liberar o lock.)

# MVCC

## UPDATE

Um **UPDATE** também pode esperar se outro **UPDATE** ou **DELETE** estiverem mexendo na mesma linha.

Isso ocorre porque só uma **transação** pode atualizar/remover uma linha por vez — senão haveria conflito sobre qual versão é a “válida”.

# MVCC

## Em resumo

**SELECT** comum → nunca trava em razão de um UPDATE/INSERT (MVCC garante leitura da versão correta).

**SELECT ... FOR UPDATE/SHARE** → pode esperar, pois exige lock.

**DELETE** → pode esperar se outro UPDATE ou DELETE estiver mexendo na mesma linha.

**UPDATE** → pode esperar pelos mesmos motivos do DELETE.