

Concurrency Control Practice

VLDB Lab.

Professor Sangwon Lee

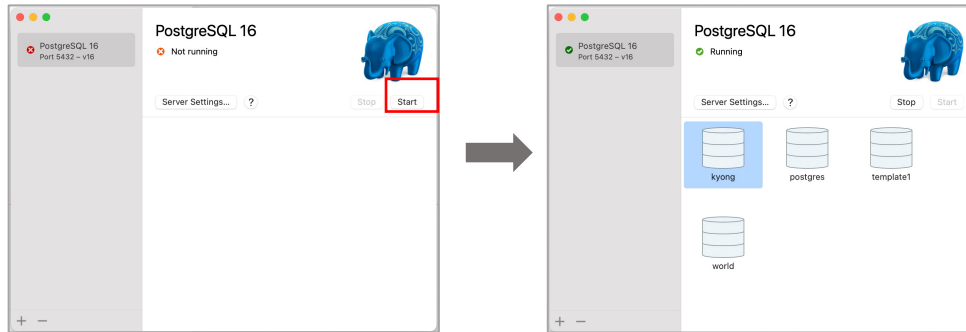
Contents

- **ACID Properties**
- **Isolation Levels and Locks (Concurrency Control)**

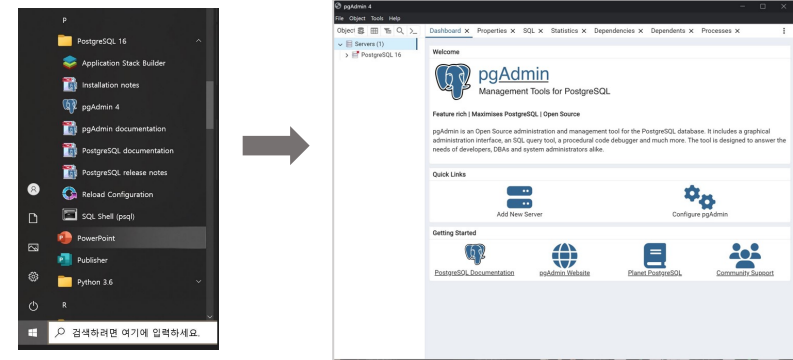
Start Postgres

1. Start Postgres.

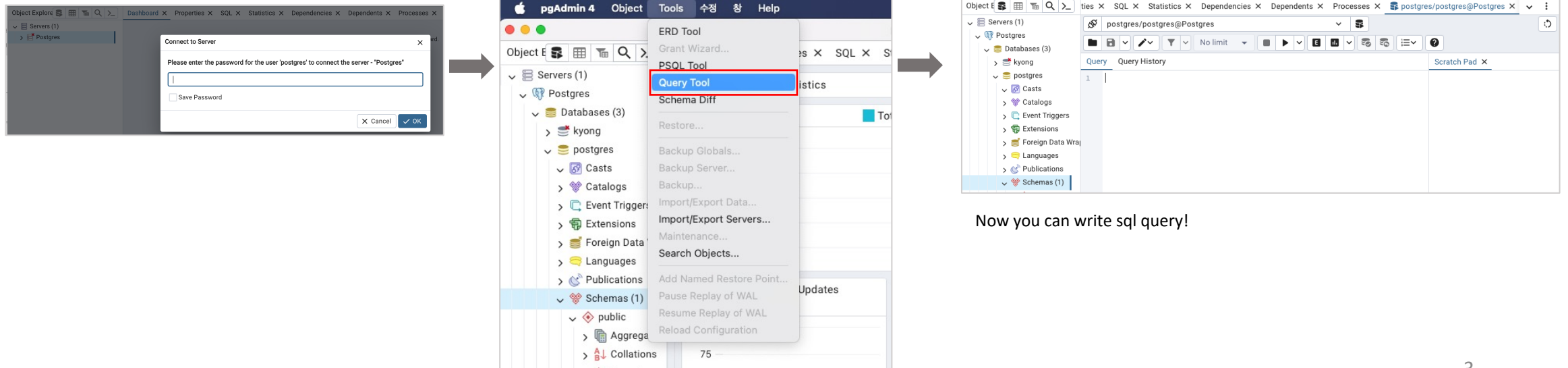
[Mac]



[Windows]

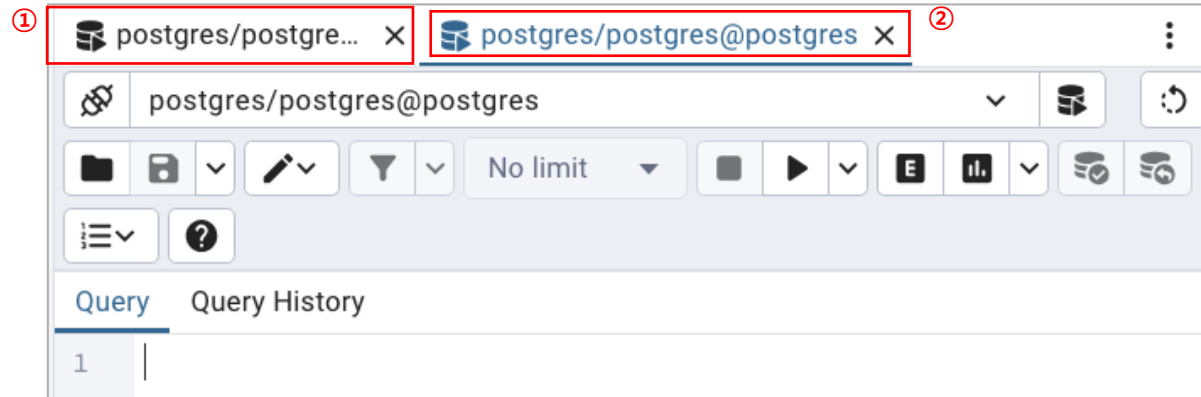


2. Start pgAdmin and connect postgres server.



ACID Properties

1. 실습을 위해 Query Tool을 2개 열어놓습니다



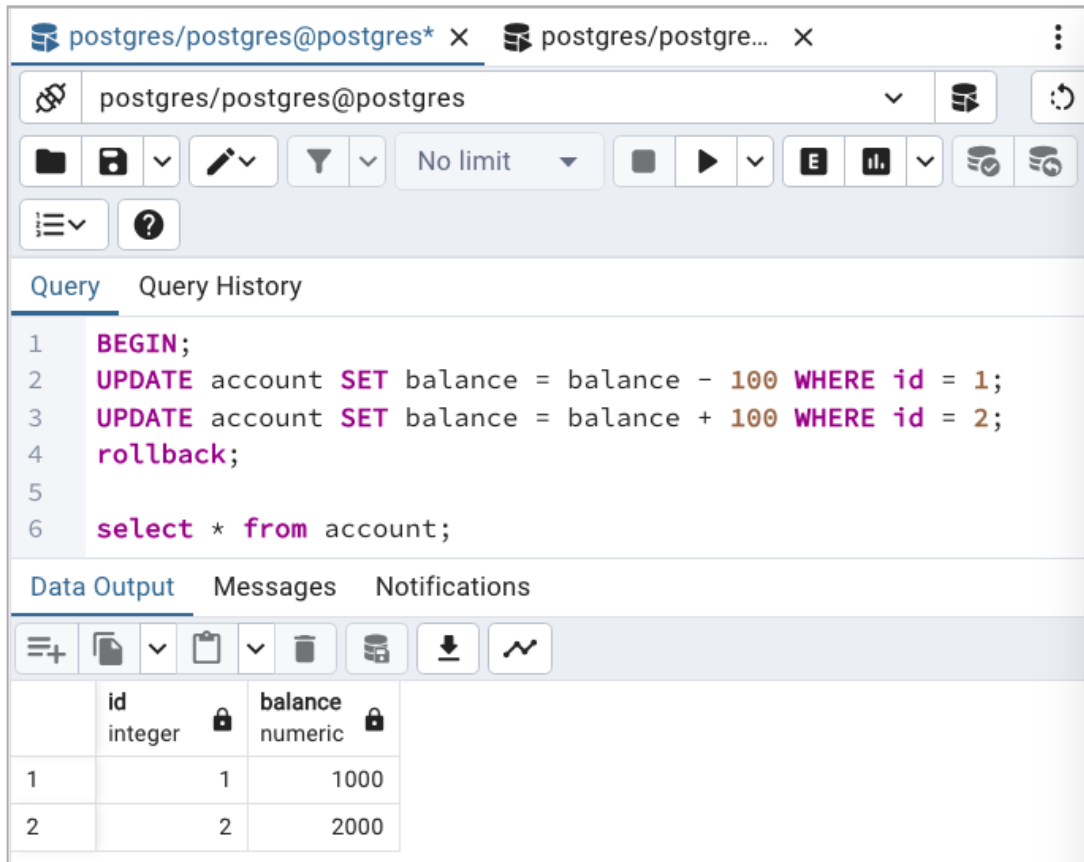
2. 첫번째 query tool 창에서 다음 sql문을 실행합니다.

```
-- Drop and create table
DROP TABLE IF EXISTS account;
CREATE TABLE account (id INT, balance NUMERIC);

-- Insert initial data
INSERT INTO account (id, balance) VALUES (1, 1000);
INSERT INTO account (id, balance) VALUES (2, 2000);
COMMIT;
```

ACID Properties

Rollback



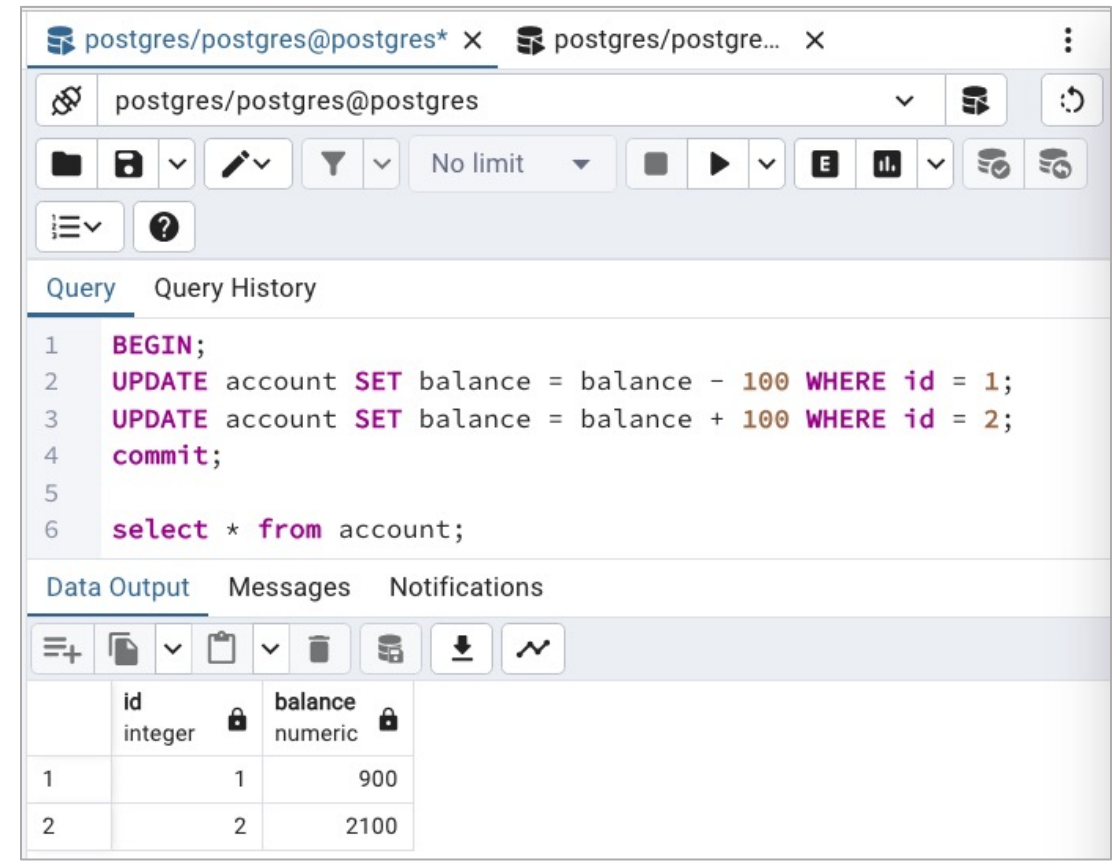
The screenshot shows a PostgreSQL client window with the following SQL query executed:

```
1 BEGIN;  
2 UPDATE account SET balance = balance - 100 WHERE id = 1;  
3 UPDATE account SET balance = balance + 100 WHERE id = 2;  
4 rollback;  
5  
6 select * from account;
```

The "Data Output" tab shows the result of the query:

	id integer	balance numeric
1	1	1000
2	2	2000

Commit



The screenshot shows a PostgreSQL client window with the following SQL query executed:

```
1 BEGIN;  
2 UPDATE account SET balance = balance - 100 WHERE id = 1;  
3 UPDATE account SET balance = balance + 100 WHERE id = 2;  
4 commit;  
5  
6 select * from account;
```

The "Data Output" tab shows the result of the query:

	id integer	balance numeric
1	1	900
2	2	2100

Isolation Levels and Locks

- **트랜잭션 격리 수준:** 여러 트랜잭션이 동시에 처리될 때, 트랜잭션끼리 서로 얼마나 고립되어있는지를 나타내는 수준

	Dirty Read	Non-repeatable Read	Phantom Read
Read Uncommitted	O	O	O
Read Committed	X	O	O
Repeatable Read	X	X	O
Serializable	X	X	X



Isolation Levels and Locks

- **Account Table 생성**

```
DROP TABLE IF EXISTS account;  
CREATE TABLE account (  
    id SERIAL PRIMARY KEY,  
    balance NUMERIC  
);
```

```
INSERT INTO account (id, balance) VALUES (1, 100);
```

Query

Query History

1

select * from account;

Data Output

Messages

Notifications

<

Isolation Levels and Locks

1. T1: update

The screenshot shows a PostgreSQL client window with the following SQL queries executed:

```
1 BEGIN;  
2 UPDATE account SET balance = balance + 10 WHERE id = 1;  
3 select * from account;
```

The 'Data Output' tab shows the result of the query:

	id [PK] integer	balance numeric
1	1	110

T2: 확인

The screenshot shows a PostgreSQL client window with the following SQL query executed:

```
1 SELECT * FROM account WHERE id = 1;
```

The 'Data Output' tab shows the result of the query:

	id [PK] integer	balance numeric
1	1	100

No dirty read!

2. T1: commit

The screenshot shows a PostgreSQL client window with the following SQL queries executed:

```
1 COMMIT;  
2 select * from account;
```

The 'Data Output' tab shows the result of the query:

	id [PK] integer	balance numeric
1	1	110

T2: 확인

The screenshot shows a PostgreSQL client window with the following SQL query executed:

```
1 SELECT * FROM account WHERE id = 1;
```

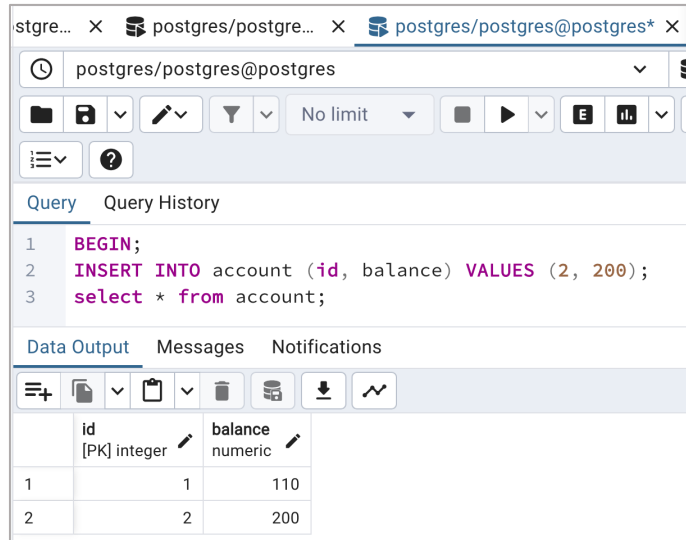
The 'Data Output' tab shows the result of the query:

	id [PK] integer	balance numeric
1	1	110

T1에서 commit한 후
T2에서 업데이트된 값이 보임

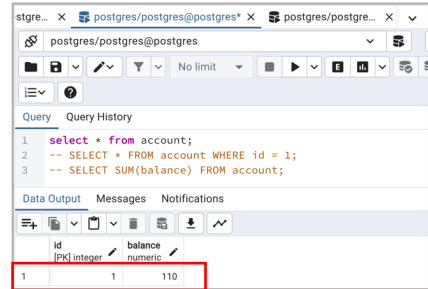
Isolation Levels and Locks

3. T3: update

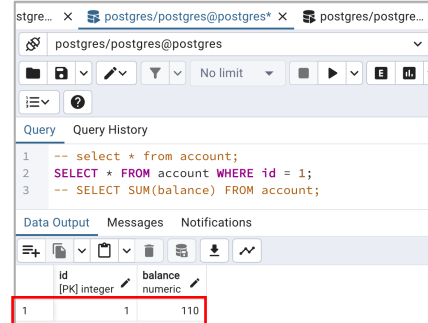


	id [PK] integer	balance numeric
1	1	110
2	2	200

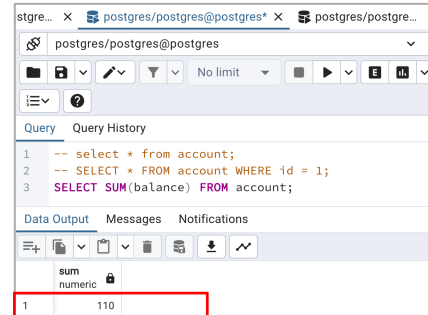
T2: 확인



	id [PK] integer	balance numeric
1	1	110



	id [PK] integer	balance numeric
1	1	110

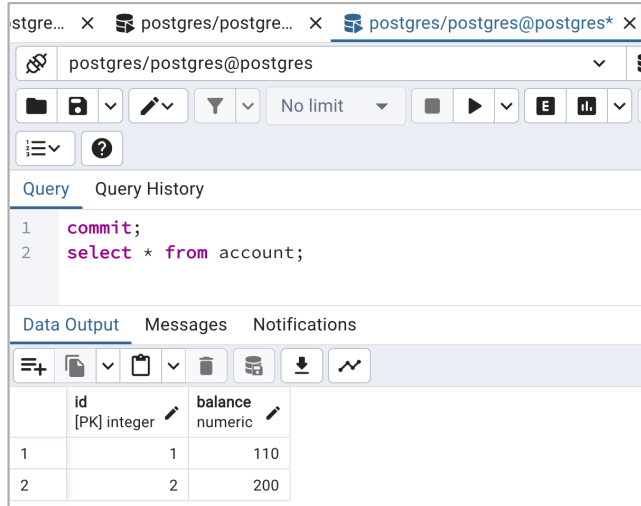


	sum numeric
1	110

T3에서 업데이트를 했음에도
T2에서는 해당 내용 관련이 바
뀌지 않음
(commit하지 않았기 때문)

Isolation Levels and Locks

4. T3: commit



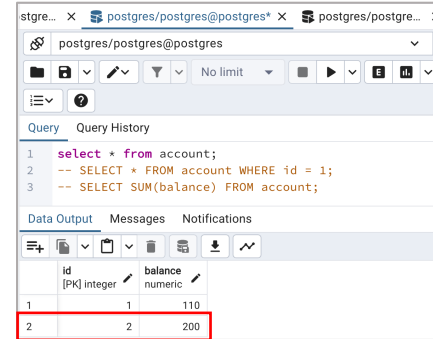
The screenshot shows a PostgreSQL client window with the following SQL queries entered:

```
1 commit;  
2 select * from account;
```

The 'Data Output' tab is active, displaying the following table:

	id [PK] integer	balance numeric
1	1	110
2	2	200

T2: 확인

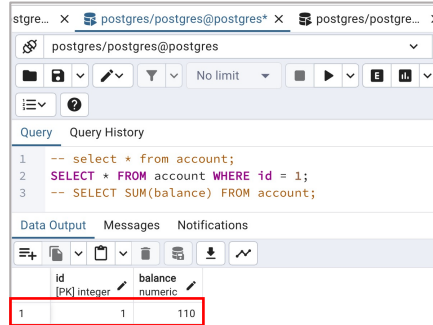


The screenshot shows a PostgreSQL client window with the following SQL queries entered:

```
1 select * from account;  
2 -- SELECT * FROM account WHERE id = 1;  
3 -- SELECT SUM(balance) FROM account;
```

The 'Data Output' tab is active, displaying the following table:

	id [PK] integer	balance numeric
1	1	110
2	2	200

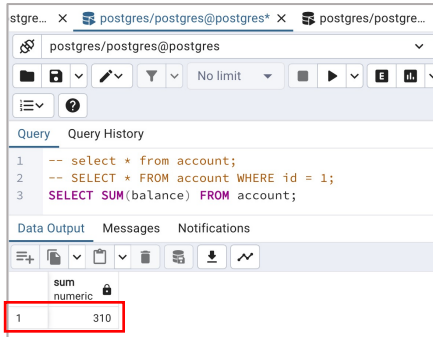


The screenshot shows a PostgreSQL client window with the following SQL queries entered:

```
1 -- select * from account;  
2 SELECT * FROM account WHERE id = 1;  
3 -- SELECT SUM(balance) FROM account;
```

The 'Data Output' tab is active, displaying the following table:

	id [PK] integer	balance numeric
1	1	110



The screenshot shows a PostgreSQL client window with the following SQL queries entered:

```
1 -- select * from account;  
2 -- SELECT * FROM account WHERE id = 1;  
3 SELECT SUM(balance) FROM account;
```

The 'Data Output' tab is active, displaying the following table:

	sum numeric
1	310

Isolation Levels and Locks

- How about “SERIALIZABLE” mode?

```
TRUNCATE TABLE account;  
INSERT INTO account (id, balance) VALUES (1, 100);  
COMMIT;
```

Query

Query History

1

select * from account;

Data Output

Messages

Notifications

id

[PK] integer

balance

numeric

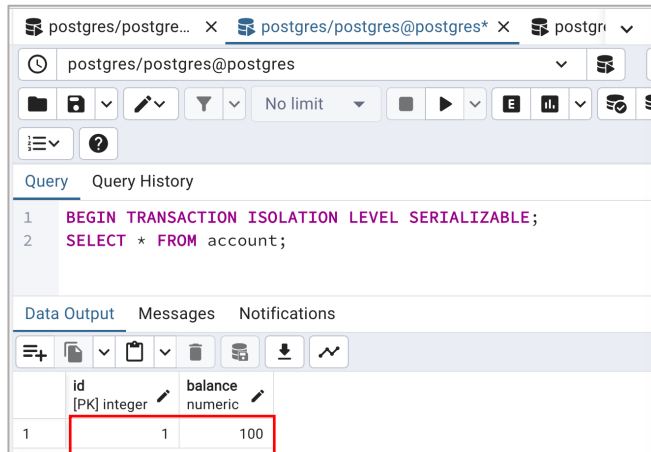
1

1

100

Isolation Levels and Locks

1. T2: SERIALIZABLE 설정, 테이블 확인



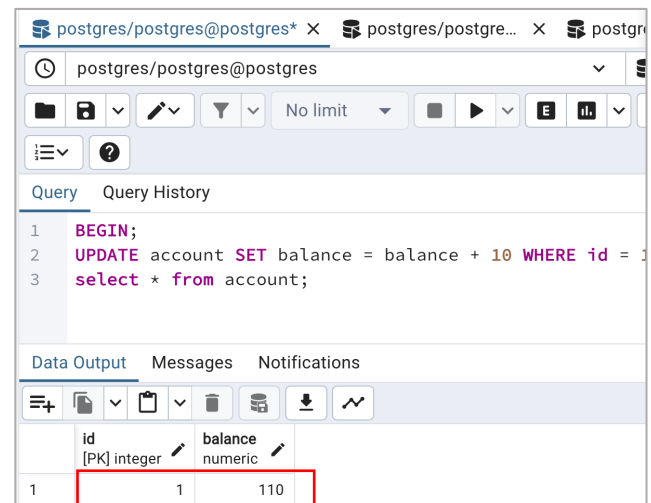
The screenshot shows a PostgreSQL client interface. The query editor contains the following SQL:

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2 SELECT * FROM account;
```

The 'Data Output' tab is selected, showing a table with two columns: 'id' (integer, primary key) and 'balance' (numeric). The table contains one row with id=1 and balance=100.

	id [PK] integer	balance numeric
1	1	100

2. T1: update



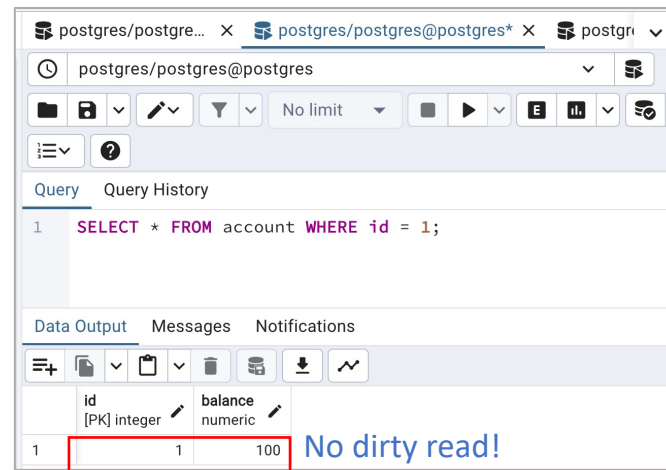
The screenshot shows a PostgreSQL client interface. The query editor contains the following SQL:

```
1 BEGIN;  
2 UPDATE account SET balance = balance + 10 WHERE id = 1;  
3 select * from account;
```

The 'Data Output' tab is selected, showing a table with two columns: 'id' (integer, primary key) and 'balance' (numeric). The table contains one row with id=1 and balance=110.

	id [PK] integer	balance numeric
1	1	110

T2: 확인



The screenshot shows a PostgreSQL client interface. The query editor contains the following SQL:

```
1 SELECT * FROM account WHERE id = 1;
```

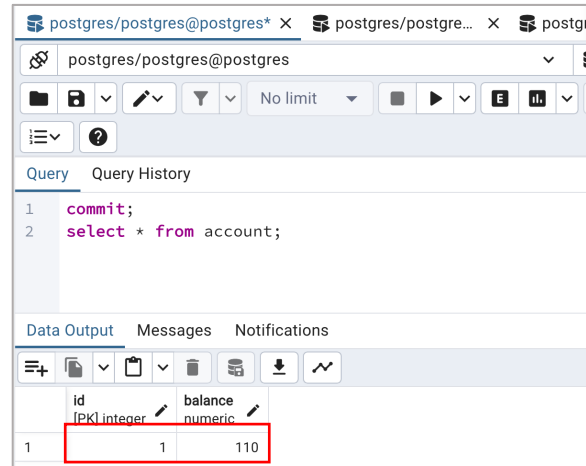
The 'Data Output' tab is selected, showing a table with two columns: 'id' (integer, primary key) and 'balance' (numeric). The table contains one row with id=1 and balance=100. The text 'No dirty read!' is displayed next to the table.

	id [PK] integer	balance numeric
1	1	100

No dirty read!

Isolation Levels and Locks

3. T1: commit



postgres/postgres@postgres* X postgres/postgre... X postgr

postgres/postgres@postgres

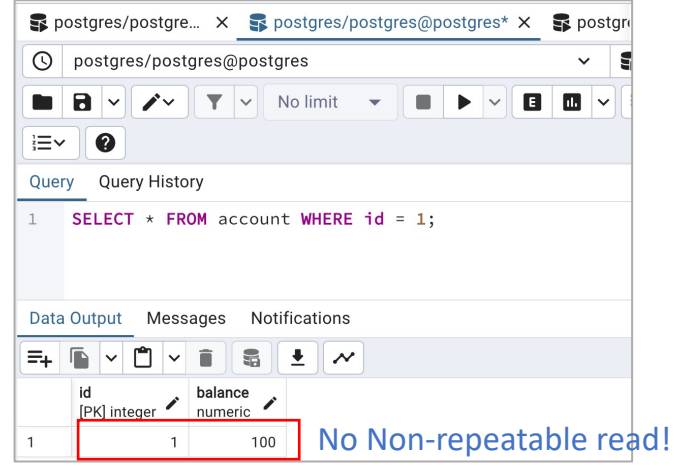
Query Query History

```
1 commit;  
2 select * from account;
```

Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	110

T2: 확인



postgres/postgre... X postgres/postgres@postgres* X postgr

postgres/postgres@postgres

Query Query History

```
1 SELECT * FROM account WHERE id = 1;
```

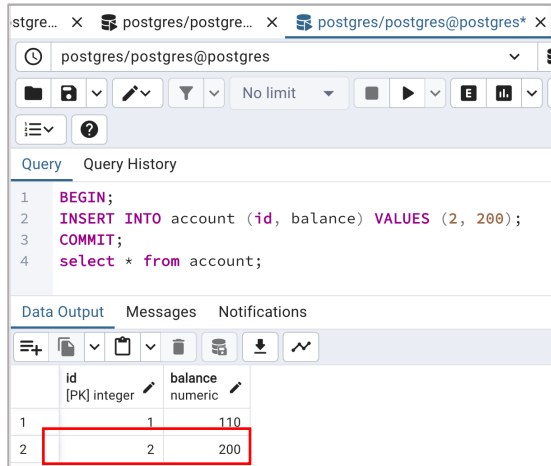
Data Output Messages Notifications

	id [PK] integer	balance numeric
1	1	100

No Non-repeatable read!

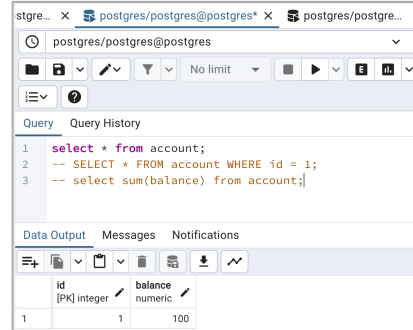
Isolation Levels and Locks

4. T3: insert value

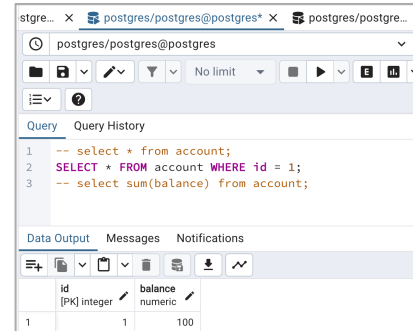


	id [PK] integer	balance numeric
1	1	110
2	2	200

T2: 확인

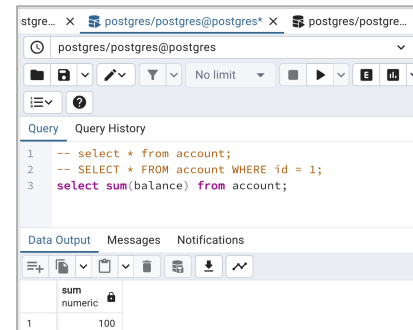


	id [PK] integer	balance numeric
1	1	100



	id [PK] integer	balance numeric
1	1	100

No Phantom Read



	sum numeric
1	100