Tecnologie e Architetture per la Gestione dei Dati – Homework 2

Su Postgres è stata provata l'esecuzione concorrente del seguente codice:

```
start transaction
  isolation level serializable;
select saldo into app1
from conti where num=1;
```

```
start transaction
  isolation level serializable;
select saldo into app2
from conti where num=1;
```

```
update conti set saldo =
  (select saldo + 100 from app1)
     where num = 1;
commit
```

```
update conti set saldo =
  (select saldo + 1 from app2)
     where num = 1;
commit
```

```
drop table if exists app1
```

```
drop table if exists app2
```

Al momento dell'esecuzione dell'update sul secondo client:

```
ERROR:  could not serialize access due to concurrent update
SQL state: 40001
```

Cio è dovuto al livello di isolamento **serializable** usato per entrambe le transazioni, qualora i commit fossero posticipati la seconda aspetterebbe il rilascio del lock della prima e darebbe di nuovo lo stesso errore.

Usando il livello di isolamento **read committed** la seconda transazione viene eseguita ma il valore inserito dentro a saldo non è corretto. Si verifica quindi una **perdita di aggiornamento** in cui il valore di saldo viene sovrascritto e non tiene conto del valore aggiornato dalla prima transazione.

Provando a eseguire di nuovo le stesse transazioni usando come livello di isolamento **repeatable read**:

```
start transaction isolation level
                repeatable read;
select saldo into app1
from conti where num=1;

update conti set saldo =
  (select saldo + 100 from app1)
     where num = 1;

commit
```

```
start transaction isolation level
                repeatable read;
select saldo into app2
from conti where num=1;

update conti set saldo =
  (select saldo + 1 from app2)
     where num = 1;

commit
```

La prima transazione riesce a effettuare l'update, la seconda viene bloccata in quanto il livello di isolamento scelto impedisce **l'aggiornamento fantasma**.

Usando invece repeatable read per la prima e read committed per la seconda si può notare che vengono concesse tutte le anomalie tranne la lettura sporca. In questo caso partiamo da un valore di saldo = 200 e alla fine delle transazioni il valore diventa:

	num [PK] integer	saldo integer
1	2	1000
2	3	1000
3	1	201

Facendo 2 transazioni di questo tipo:

```
start transaction isolation level
repeatable read;
select saldo into app1
from conti where num=1;

select sum(saldo)
from conti;
```

```
start transaction isolation level
repeatable read;
select saldo into app2
from conti where num=1;

update conti set saldo =
(select saldo + 1 from app2)
where num = 1;
```

Notiamo che la transazione 1 dopo il commit della seconda legge un valore sbagliato del valore di sum(saldo) e porta il database in uno stato inconsistente.