**Petar Petrov**

**h11729352**

# 1A

## Repeatable Read Isolation Level - Possible

This level is different from Read Committed in that a query in a repeatable read transaction sees a snapshot as of the start of the **transaction**, not as of the start of the current query within the transaction. Thus, successive SELECT commands within a **single** transaction see the same data, i.e., they do not see changes made by other transactions that committed after their own transaction started.

## Read Committed Isolation Level – NOT Possible

*Read Committed* is the default isolation level in PostgreSQL. When a transaction uses this isolation level, a SELECT query (without a FOR UPDATE/SHARE clause) sees only data committed before the query began; it never sees either uncommitted data or changes committed during query execution by concurrent transactions. In effect, a SELECT query sees a snapshot of the database as of the instant the query begins to run.
However, SELECT does see the effects of previous updates executed within its own transaction, even though they are not yet committed. **Also note that two successive SELECT commands can see different data, even though they are within a single transaction, if other transactions commit changes during execution of the first SELECT.**

So here it wouldn't be possible, to have this isolation level, because other transaction has already committed changes, which cannot be seen by the uncommitted and still running transaction of User 1.

### Serializable Isolation Level – NOT Possible

```
ERROR:  could not serialize access due to
read/write dependencies among transactions
```

User 2 would not be able to read/write, because T1 has not been committed yet.

# 1B

### Repeatable Read Isolation Level - Possible

The *Repeatable Read* isolation level only sees data committed before the transaction began; it never sees either uncommitted data or changes committed during transaction execution by concurrent transactions.

### Read Committed Isolation Level – Possible

*Read Committed* is the default isolation level in PostgreSQL. When a transaction uses this isolation level, a SELECT query (without a FOR UPDATE/SHARE clause) sees only data committed before the query began; it never sees either uncommitted data or changes committed during query execution by concurrent transactions. In effect, a SELECT query sees a snapshot of the database as of the instant the query begins to run.

### Read uncommitted – same as read committed

## 2A

M->CT

C->T

## 2B

MCT

MC

MT

## 2C

MC; MT – because only 1 course type can be in 1 menu, therefore it is unique. Also each menu has only 1 course.

## 2D

No it is not, we could remove the course type (T) column from the table and create a new table consisting of all the courses with the corresponding course type. R1(MC) M2(CT) – with foreign key C;

## 2E

No, it is not, because the Course Type(T) can be derived from the Course (C), therefore it is transitive dependent. 3NF doesn't allow transitive dependent columns to stay in the same table. The columns aren't completely dependent on the table's primary key, it stands to reason they belong elsewhere. In both cases, the columns are moved to new tables.

## 3

**BCNF:**

| **Nickname** | **Real Name** |
|---|---|
| Lights Out | James Toney |
| Iron Junior | Vincent Feigenbutz |
| Captain | Marco Huck |
| What the Heck | Owen Beck |

| **Nickname** | **Age** |
|---|---|
| Lights Out | 47 |
| Iron Junior | 18 |
| Captain | 30 |
| What the Heck | 39 |

| **Nickname** | **Weight** |
|---|---|
| Lights Out | 99 |
| Iron Junior | 76 |
| Captain | 90 |
| What the Heck | 105 |

| **Weight** | **Age** | **Category** |
|---|---|---|
| 99 | 47 | HW |
| 76 | 18 | JMW |

| | | |
|---|---|---|
| 90 | 30 | HW |
| 105 | 39 | HW |

| Place | City |
|---|---|
| Olimpiahalle | Muenchen |
| Zenith | Muenchen |
| Europahalle | Karlsruhe |
| Dm-Arena | Karlsruhe |

| Date | Place |
|---|---|
| 22-10-2015 | Olimpiahalle |
| 05-09-2015 | Zenith |
| 01-09-2015 | Europahalle |
| 15-09-2015 | Dm-Arena |

| Nickname | Date |
|---|---|
| Lights Out | 22-10-2015 |
| Iron Junior | 05-09-2015 |
| Captain | 22-10-2015 |
| Lights Out | 01-09-2015 |
| Captain | 05-09-2015 |
| Iron Junior | 15-09-2015 |
| What the Heck | 01-09-2015 |

**3NF:**

| Nickname | Date | Place |
|---|---|---|
| Lights Out | 22-10-2015 | Olimpiahalle |
| Iron Junior | 05-09-2015 | Zenith |
| Captain | 22-10-2015 | Olimpiahalle |
| Lights Out | 01-09-2015 | Europahalle |
| Captain | 05-09-2015 | Zenith |
| Iron Junior | 15-09-2015 | Dm-Arena |
| What the Heck | 01-09-2015 | Europahalle |

| Nickname | Real Name | Weight | Age |
|---|---|---|---|
| Lights Out | James Toney | 99 | 47 |
| Iron Junior | Vincent Feigenbutz | 76 | 18 |
| Captain | Marco Huck | 90 | 30 |
| What the Heck | Owen Beck | 105 | 39 |

| Weight | Age | Category |
|---|---|---|
| 99 | 47 | HW |
| 76 | 18 | JMW |
| 90 | 30 | HW |
| 105 | 39 | HW |

| Place | City |
|---|---|
| Olimpiahalle | Muenchen |
| Zenith | Muenchen |
| Europahalle | Karlsruhe |
| Dm-Arena | Karlsruhe |

| Date | Place |
|---|---|
| 22-10-2015 | Olimpiahalle |
| 05-09-2015 | Zenith |
| 01-09-2015 | Europahalle |
| 15-09-2015 | Dm-Arena |

# 4

```
> SADD S:1 1
(integer) 1
> SADD S:1 3
(integer) 1
> SADD S:1 4
(integer) 1
> SADD S:1 6
(integer) 1
> SMEMBERS S:1
1) "1"
2) "3"
3) "4"
4) "6"
> SADD S:2 a
(integer) 1
> SADD S:2 4
```

```
(integer) 1
> SADD S:2 b
(integer) 1
> SADD S:2 3
(integer) 1
> SMEMBERS S:2
1) "3"
2) "a"
3) "b"
4) "4"
```

**B:**

```
> SINTERSTORE S:3 S:1 S:2
2
> SMEMBERS S:3
1) "3"
2) "4"
```

## C:

```
> SADD S:4 1
(integer) 1
> SREM S:4 1
1
> SMEMBERS S:4
(empty list or set)
```

**EXISTS – 0 => not possible to have empty set in redis**

**D:**

```
> RPUSH list i f a j i 3 b f a 2 i j a 3 a
(integer) 15
> lrange list -100 100
1) "i"
2) "f"
3) "a"
4) "j"
5) "i"
6) "3"
7) "b"
8) "f"
9) "a"
10) "2"
11) "i"
12) "j"
13) "a"
14) "3"
15) "a"
```

**ZADD**

## 5A

db.catalog.insertMany(

[

{

cid:1,

cost:120,

supplier: {sid:1, sname:"Best Red East", address: "Briggentau"},

part: {pid : 4, pname: "Sunpart", color :"Red"}

},

{

cid:2,

cost:223,

supplier: {sid:1, sname:"Best Red East", address: "Briggentau"},

part: {pid : 5, pname: "Firepart", color :"Red"}

},

{

cid:3,

cost : 523,

supplier: {sid:1, sname:"Best Red East", address: "Briggentau"},

part: {pid : 3, pname: "Grasspart", color :"Green"}

},

{

cid:4,

cost : 499,

supplier: {sid:2, sname:"Green West", address: "Hietzing"},

part: {pid :2 , pname: "Woodpart", color :"Green"}

},

{

cid:5,

cost : 320,

supplier: {sid:2, sname:"Green West", address: "Hietzing"},

part: {pid :3 , pname: "Grasspart", color :"Green"}

},

{

cid:6,

cost : 161,

supplier: {sid:2, sname:"Green West", address: "Hietzing"},

part: {pid :4 , pname: "Sunpart", color :"Red"}

},

{

cid:7,

cost : 356,

supplier: {sid:3, sname:"Nordparts", address: "Dobling"},

part: {pid :1 , pname: "Skypart", color :"Blue"}

},

{

cid:8,

cost : 650,

supplier: {sid:3, sname:"Nordparts", address: "Dobling"},

part: {pid :2 , pname: "Woodpart", color :"Green"}

},

{

cid:9,

cost : 586,

supplier: {sid:3, sname:"Nordparts", address: "Dobling"},

part: {pid :3 , pname: "Grasspart", color :"Green"}

},

{

cid:10,

cost : 184,

supplier: {sid:3, sname:"Nordparts", address: "Dobling"},

part: {pid :4 , pname: "Sunpart", color :"Red"}

}

])

## B

```
db.catalog.find( { "part.color": "Red" }, {"supplier.sname":1,_id:0} )


db.catalog.find ({$or: [{"part.color" : "Red"}, { "part.color":"Green"}]},{"supplier.sid":1,_id:0} )
```

db.catalog.find ({$or: [{"part.color" : "Red"}, { "supplier.address":"Dobling"}]},{"supplier.sid":1,_id:0} )

db.catalog.find ({$and: [{"part.color" : "Red"}, { "part.color":"Green"}]},{"supplier.sid":1,_id:0} )

## 6.1

ASK {

dbr:Bulgaria dbp:areaRank ?bg . dbr:Austria dbp:areaRank ?at.

FILTER (?bg > ?at).

}

## 6.2

SELECT (?usa - ?eu AS ?diff)

{

dbr:European_Union dbp:gdpNominalPerCapita ?eu . dbr:United_States dbp:gdpNominalPerCapita ?usa.

}