Insy - Transactions

Isolationsebenen:

Isolation bei Datenbanken bezieht sich auf die Trennung von Transaktionen. Die Isolation ist eine der vier ACID-Eigenschaften. Man unterscheidet dabei zwischen verschiedenen Isolationsebenen, welche entsprechend verschiedenen Schutz vor *"Errorklassen"* ermöglichen.

ISOLATIONSEBENE	DIRTY READ	LOST UPDATES	NON-REPEATABLE READ	PHANTOM READ
Read Committed	unmöglich	möglich	möglich	möglich
Repeatable Read	unmöglich	unmöglich	unmöglich	möglich
Serializable	unmöglich	unmöglich	unmöglich	unmöglich
Read Uncommitted	möglich	möglich	möglich	möglich

READ COMMITTED

A statement can only see rows committed before it began. This is the default.

REPEATABLE READ

All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

SERIALIZABLE

All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If a pattern of reads and writes among concurrent serializable transactions would create a situation which could not have occurred for any serial (one-at-a-time) execution of those transactions, one of them will be rolled back with a serialization_failure error.

Setzen des Transaction-Levels

```
SET TRANSACTION ISOLATION LEVEL { SERIALIZABLE |
REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED };
```

Aufgaben

Ich habe eine Datenbank erstellet, welch personen mit geld speichert. Gaz einfallsreich...

```
CREATE TABLE "personen" (
  "id" int NOT NULL,
  "name" varchar NOT NULL,
  "geldsack" int NOT NULL,
  PRIMARY KEY ("id")
);
INSERT INTO "personen" ("id", "name", "geldsack") VALUES
  ('1', 'Gerog', '200'),
  ('2', 'James', '400');
```

ID	NAME	GELDSACK
1	Georg	200
2	James	400

Isolationsebenen Beispiele

Beispiel zu "read uncomitted"

In Console 1 wird ein Update verrichtet und in Console 2 während dem Update ausgelesen. Erwartetes ergebnis: Inhalt sollte mit uncomittetem Update gezeigt werden (dirty read).

CONSOLE 1

```
insy_sem8=# begin;
BEGIN
insy_sem8=# select * from personen where id=1;
 id | name  | geldsack
----+-------+----------
  1 | Gerog |      200
(1 row)

insy_sem8=# update personen set geldsack = 300 where id = 1;
UPDATE 1
insy_sem8=#
```

CONSOLE 2

```
insy_sem8=# begin;
BEGIN
insy_sem8=# select * from personen where id = 1;
 id | name  | geldsack
----+-------+----------
  1 | Gerog |      200
(1 row)

insy_sem8=#
```

Wir sehen es passiert trotzdem nix, wegen der Sicherheit von PostgreSQL.

Beispiel zu "read comitted"

In Console 1 wird ein Update verrichtet während dem update wird in Console 2 versucht der Wert auszulesen und selbst ein Update auszuführen. Erwartetes ergebnis: Nach den transaktionen sollte nurmehr das 2. update existieren (lost-update).

CONSOLE 1

```
insy_sem8=# begin;
BEGIN
insy_sem8=# select * from personen
insy_sem8-# ;
 id | name  | geldsack
----+-------+----------
  2 | James |      400
  1 | Gerog |      300
(2 rows)

insy_sem8=# update personen set geldsack = 300 where id = 1;
UPDATE 1
```

CONSOLE 2

```
insy_sem8=# begin
insy_sem8-# ;
BEGIN
insy_sem8=# select * from personen
insy_sem8-# ;
 id | name  | geldsack
----+-------+----------
  2 | James |      400
  1 | Gerog |      300
(2 rows)

insy_sem8=# update personen set geldsack = 350 where id = 1;
```

Es gilt nur der 2. Update., Aber er wartet bis der erste Commit fertig ist.

Beispiel zu "repeatable read"

Zwei Transaktionen starten Zeitgleich. Die eine Zählt, die andere fügt hinzu. Erwartetes Ergebnis: Er liest immer den neuesten Wert.

CONSOLE 1

```
insy_sem8=# begin;
BEGIN
insy_sem8=# select count(*) from personen
;
 count
-------
     2
(1 row)

insy_sem8=# select count(*) from personen
;
 count
-------
     2
(1 row)

insy_sem8=#
```

CONSOLE 2

```
insy_sem8=# begin;
BEGIN
insy_sem8=# insert into personen values(3, 'Sam', 250);
INSERT 0 1
insy_sem8=# insert into personen values(4, 'Chris', 300);
INSERT 0 1
insy_sem8=#
```

In Postgresql nicht.

Beispiel zu "serializeable"

Beide Transaktionen versuchen was zu inserten. Erwartetetes Ergebnis: Fehler bei Console 2 nach dem Commit der Console 1.

CONSOLE 1

```
insy_sem8=# begin;
BEGIN
insy_sem8=# insert into personen values(5, 'Bob', 200);
INSERT 0 1
insy_sem8=# commit;
COMMIT
```

CONSOLE 2

```
insy_sem8=# begin;
BEGIN
insy_sem8=#  insert into personen values(5, 'Gob', 2000);
ERROR:  duplicate key value violates unique constraint "personen_pkey"
DETAIL:  Key (id)=(5) already exists.
insy_sem8=#
```

Errorklassen

Dirty Read

Wenn Transaktionen nicht festgeschriebene Änderungen lesen, die von anderen Transaktionen vorgenommen wurden.

Lost update

Wenn ein Update von einem anderen Überschrieben wird, welches mit den selben ausgangsdaten Arbeitet.

Non-Repeatable Read

Wenn Transaktionen eine Abfrage erneut ausführen und feststellt, dass sich der Satz von Zeilen aufgrund von zugeschriebenen Änderungen durch andere Transaktionen ändert.

Phantom Read

Anomalie, wenn das Ergebnis des erfolgreichen Commits einer Gruppe von Transaktionen mit allen möglichen Reihenfolgen der Ausführung dieser Transaktionen nacheinander unvereinbar ist.