Transaktionen

Inhanlt

Plan

Wir wollen anhand der verschiedenen Isolation Levels

Die Datenbank ist **"MySQL"** welche auf einer Ubuntu-VM realisiert wird.

unter Datenbank aufsetzen ist beschrieben wie die Ausgangslage nachzumachen ist.

Dummy Table mit dummy Daten:

```
CREATE DATABASE isolation_demo;
USE isolation_demo;

CREATE TABLE konto (
  id INT PRIMARY KEY,
  kontostand DECIMAL(10,2)
);

INSERT INTO konto VALUES (1, 100.00);
```

mit diesen Table und Daten wird gearbeitet!

- ## Dirty read

  - ### READ UNCOMMITTED - isolation level

    Session A:

    ```
    USE isolation_demo;

    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
    START TRANSACTION;

    UPDATE konto SET kontostand = 200.00 WHERE id = 1;
    -- Noch kein COMMIT!
    ```

Session B:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
START TRANSACTION;

SELECT * FROM konto WHERE id = 1;
-- → Gibt 200.00 zurück, obwohl A noch nicht committet hat!
```

- ### READ COMMITTED - isolation level

Session A:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
UPDATE konto SET kontostand = 300.00 WHERE id = 1;
-- Kein COMMIT
```

Session B:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT * FROM konto WHERE id = 1;
-- → Gibt 100.00 zurück (alte, committete Version)
```

- ### REPEATABLE READ - isolation level

Session A:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
UPDATE konto SET kontostand = 400.00 WHERE id = 1;
-- Kein COMMIT
```

Session B:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT * FROM konto WHERE id = 1;
-- → 100.00 (keine uncommitteten Werte sichtbar)
```

Serializable - isolation level

Session A:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
UPDATE konto SET kontostand = 500.00 WHERE id = 1;
-- Kein COMMIT
```

Session B:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
SELECT * FROM konto WHERE id = 1;

-- wartet bis andere laufende transaktionen fertig sind bevor
daten gelsensen werden
```

Nonrepeatable read

read uncommitet wird weggelassen.

READ COMMITTED - isolation level

Session A *(reader)*:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;

-- Erste Leseoperation
SELECT kontostand FROM konto WHERE id = 1;
-- → Ergebnis: 100.00
```

Session B *(writer)*:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;

UPDATE konto SET kontostand = 300.00 WHERE id = 1;
COMMIT;
```

Session A (selbe transaktion):

```
-- Zweite Leseoperation
SELECT kontostand FROM konto WHERE id = 1;
-- → Ergebnis: 300.00
```

beim zweiten read ist der wert anders als beim ersten Obwohl Session A immernoch in der selben transaktion ist.

REPEATABLE READ - isolation level

Session A *(reader)*:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

-- Erste Leseoperation
SELECT kontostand FROM konto WHERE id = 1;
-- → Ergebnis: 100.00
```

Session B *(writer)*:

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

UPDATE konto SET kontostand = 400.00 WHERE id = 1;
COMMIT;
```

Session A (selbe transaktion):

```
SELECT kontostand FROM konto WHERE id = 1;
-- → 100.00 (immer noch alter Wert!)
```

Serializable - isolation level

hier wartet session B bis Session A commitet oder ein rollbaack macht -> kein repeatable read aber auch keine parralellen transaktionen.

Phantom read

Ein Phantom Read tritt auf, wenn:

Eine Transaktion mehrere Zeilen liest, die einem bestimmten Kriterium entsprechen, und eine andere Transaktion fügt neue Zeilen hinzu oder löscht welche, sodass beim zweiten Lesen mehr oder weniger Zeilen erscheinen -> also „Phantome“.

- ### READ COMMITTED - isolation level

  Session A *(reader)*:

  ```
  USE isolation_demo;

  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  START TRANSACTION;

  SELECT * FROM konto WHERE kontostand >= 100;
  -- → Zeigt 1 Zeile
  ```

  Session B *(writer*):

  ```
  USE isolation_demo;

  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  START TRANSACTION;

  INSERT INTO konto VALUES (2, 100.00);
  COMMIT;
  ```

  Session A (selbe transaktion):

  ```
  SELECT * FROM konto WHERE kontostand >= 100;
  -- → Jetzt 2 Zeilen (1 und 2)
  COMMIT;
  ```

- ### REPEATABLE READ - isolation level

  Session A *(reader)*:

  ```
  USE isolation_demo;

  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
  START TRANSACTION;

  SELECT * FROM konto WHERE kontostand >= 100;
  -- → Zeigt 1 Zeile
  ```

  Session B *(writer*):

```
USE isolation_demo;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

INSERT INTO konto VALUES (2, 100.00);
COMMIT;
```

Session A (selbe transaktion):

```
SELECT * FROM konto WHERE kontostand >= 100;
-- → Immer noch 1 Zeile1
COMMIT;
```

ACHTUNG!: es kann bei Repeatable read (in mysql) zu phantom reads kommen! Auch wenn obiges anderes aussagt. Bei kompleyeren abfragen kann es durchaus zu phantoms kommen mehr info findest du hier

---

- ### Serializable - isolation level

  hier wartet session B wieder bis Session A commitet oder ein rollbaack macht

Datenbank aufsetzen

wir verwenden eine Ubuntu VM

- MySQL instalieren
- remote login zulassen

ich verwende dann die ip von meiner Ubuntu vm und das DB passwort und user um mich via datagrip remote einzuloggen