Transaction Isolation Levels

The transaction isolation levels in **JDBC** help us to determine whether the concurrently running transactions in a **DB** can affect each other or not. If there are 2 or more transactions concurrently accessing the same **DataBase,** then we need to prevent the actions of the transactions from interfering with each other.

The above is achieved **using** the isolation levels in **JDBC.**

```
Some of the common problems that might occur when transactions are running concurrently are:
```

Dirty Reads :

This is also called as uncommitted dependency problem. An example to explain to problem :

Consider a Student making changes to a project report document, When changes are being made, another student takes a copy of the document which includes the changes done so far, and distributes it to the intended audience. The 1st student then further modifies the previously made changes and saves the document. Now, the distributed document contains information that no longer exists.

Solution: The solution to this problem will be that no one should be able to read the changed document until the first editor determines that the changes are final.

Non-repeatable Reads :

This problem is know as the inconsistent analysis problem. Here is an example to explain this problem :

An employer reads a particular document twice. Between each reading by the employee, the writer rewrites the original document. When the first employee reads the document for the second time, it has completely changed. Hence, the original read was not repeatable, leading to confusion.

Solution : To ensure that this does not happen, the employee should be able to read the document only after the writer has completely finished writing it.

Phantom Reads :

The problem occurs when one of the concurrently running transactions tries to perform an action, such as insertion and deletion on the database rows that are being used by another transaction.

Example : The Supervisor reads and makes suggestions to change a document submitted by an employee, When the suggested changes are being incorporated into the master copy of the document, the other employees find the new content has been added to the document by the employee, leading to confusion and problems.

Solution : No one should be able to add new material to the document until the editor finishes working with the original document.

ISOLATION FIELDS :

In Java these isolation levels are provided by the Connection **interface** of the **JDBC API.** The below are the following fields which needs to set as **int** values to set isolation levels :

TRANSACTION_READ_UNCOMMITTED : Provides the lowest level of isolation between concurrently running transactions. This level does not prevent the problem of dirty reads, non-repeatable reads and phantom reads. This only prevents reading of corrupted data.

TRANSACTION_READ_COMMITTED : Enables us to prevent the occurrence of dirty reads, but other problems can occur.

TRANSACTION_REPEATABLE_READ : Enables us to prevent dirty reads and non-repeatable reads. It also prevents updating of data used by the other concurrently running transaction.

TRANSACTION_SERIALIZABLE : Provides the highest level of isolation. If set then it helps us to prevent all problems like dirty reads, non-repeatable reads and phantom reads.

HOW TO APPLY?

The Connection **interface** contains **getTransactionIsolationLevel()** and **setTransactionIsolationLevel()** methods that help us to retrieve and set the value of the transaction isolation level for a database, respectively.

Below is the code snippet for **getTransactionIsolationLevel()** :

```
Connection conObj = DriverManager.getConnection

                    ("jdbc:odbc:MyDataSource","admin","");
int transLevel = getTransactionIsolationLevel();
```

Below is the code snippet for **setTransactionIsolationLevel()** :

```
Connection conObj = DriverManager.getConnection
                    ("jdbc:odbc:MyDataSource","admin","");
int transLevel = getTransactionIsolationLevel();
conObj.setTransactionIsolationLevel(TRANSACTION_SERIALIZABLE);
```