# Lab Note 2: Transaction in SQL

Subject database

## Preamble

Follow these steps to get your database ready :

1. Toggle your auto-commit mode to 0 : `set autocommit = 0;`
2. You can check if the auto-commit is in the right mode : `select @@autocommit;`

# Tasks

## 1. Commit and rollback

I inserted a new dept `INSERT INTO dept VALUES ('50', 'COMPUTING', 'PARIS');` and when i did `SELECT * FROM dept` in the same window the new dept appeared. However in an other window i couldn't see the new department.
When i did a `ROLLBACK` the new department was no longer visible.

## 2. Client failure

I started a new transaction by inserting a new dept `INSERT INTO dept VALUES ('50', 'COMPUTING', 'PARIS');` . I then closed the window and when i reopened it, my dept was not visible.

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
+-----+------------+----------+
```

The same happened when i closed my MySQL Workbench using the task manager, i got the same output.

## 3. Transaction isolation

I executed the following command `show variables like '%isolation%'` and i got this output :

```
+-----------------------+-----------------+
| Variable_name         | Value           |
+-----------------------+-----------------+
| transaction_isolation | REPEATABLE-READ |
| tx_isolation          | REPEATABLE-READ |
+-----------------------+-----------------+
```

In order to prove that the modifications a transaction makes are only visible to that transaction I've opened two MySql Workbench :
In the first one, in transaction mode, I'm making some changes by adding a computing department : `INSERT INTO dept VALUES ('50', 'COMPUTING', 'PARIS');`
If i want to see all the department : `SELECT * FROM dept;` from that Workbench :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
|  50 | COMPUTING  | PARIS    |
+-----+------------+----------+
```

We can see that the new department appear.
However now I'm making the same query from the second Workbench :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
+-----+------------+----------+
```

And now the department doesn't appear.
Now I'm removing this department from the first Workbench : `DELETE FROM dept WHERE did = 50;` and the output is now the same in both Workbench :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
+-----+------------+----------+
```

## 4. Isolation levels

I repeated the previous experiment but i changed the transaction isolation : `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`
I once again opened two Workbench and i added a new department in the first one: `INSERT INTO dept VALUES ('50', 'COMPUTING', 'PARIS');`
In the first Workbench the new department appear as expected `SELECT * FROM dept` :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
|  50 | COMPUTING  | PARIS    |
+-----+------------+----------+
```

In the second Workbench the new department also appeared `SELECT * FROM dept` :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
```

```
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
|  50 | COMPUTING  | PARIS    |
+-----+------------+----------+
```

So we can conclude that this isolation level allow uncommitted changed to be read by other transactions.
We can see all the different isolation level here.

## 5. Isolation levels - Continued

I repeated the previous experiment but i changed the transaction isolation : `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`
I once again opened two Workbench and i added a new department in the first one: `INSERT INTO dept VALUES ('50', 'COMPUTING', 'PARIS');`
If i want to see all the department : `SELECT * FROM dept;` from that Workbench :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
|  50 | COMPUTING  | PARIS    |
+-----+------------+----------+
```

We can see that the new department appear.
However now I'm making the same query from the second Workbench :

```
+-----+------------+----------+
| DID | DNAME      | DLOC     |
+-----+------------+----------+
|  10 | ACCOUNTING | NEW-YORK |
|  20 | RESEARCH   | DALLAS   |
|  30 | SALES      | CHICAGO  |
|  40 | OPERATIONS | BOSTON   |
+-----+------------+----------+
```

Serializable is the highest isolation level. It avoid to get errors like Dirty Reads, Non-repeatable read and Phantoms. It's the same source as above Source

## 6. JDBC code

Code to display the default value of the autocommit mode :

```
public int getCommitMode()
{
  try {
    if(conn.getAutoCommit()){
      System.out.println(1);
      return 1;
    }
    else {
      System.out.println(0);
      return 0;
    }
  }catch(SQLException ex) {
    ex.printStackTrace();
  }
```

```
  return 2;
}
```

Code to display the default level of transaction, i did two different versions with different return/output :

1. Return an int

```
public int getTransactionLevelInt()
{
  try {
    System.out.println(conn.getTransactionIsolation());
    return conn.getTransactionIsolation();
  }catch(SQLException ex) {
    ex.printStackTrace();
  }
  return 10;
}
```

2. Return a String

```
public String getTransactionLevelString()
    {
        try {
            System.out.println(conn.getTransactionIsolation());
            switch(conn.getTransactionIsolation()) {
            case 0:
            {
                System.out.println("TRANSACTION_NONE");
                return("TRANSACTION_NONE");
            }
            case 1:
            {
                System.out.println("TRANSACTION_READ_UNCOMMITTED");
                return("TRANSACTION_READ_UNCOMMITTED");
            }
            case 2:
            {
                System.out.println("TRANSACTION_READ_COMMITTED");
                return("TRANSACTION_READ_COMMITTED");
            }
            case 4:
            {
                System.out.println("TRANSACTION_REPEATABLE_READ");
                return("TRANSACTION_REPEATABLE_READ");
            }
            case 8:
            {
                System.out.println("TRANSACTION_SERIALIZABLE");
                return("TRANSACTION_SERIALIZABLE");
            }
            default:
            {
                return("Error");
            }
            }
        }catch(SQLException ex) {
            ex.printStackTrace();
        }
        return "Error";
    }
```

Now i've modified my code so that every Java method executes a transaction and the transaction abort if an `SQLException` is thrown.

```
package model;

import java.sql.*;
import java.util.*;

public class DataAccess {

    Connection conn = null;

    public DataAccess(String url, String login, String password) throws
    SQLException {
        try{
            conn = DriverManager.getConnection(url, login, password);
            System.out.println("connected to " + url);
            conn.setAutoCommit(false);
        }
        catch(SQLException ex) {
            ex.printStackTrace();
        }
    }

    public void close(){
        try {
            conn.close();
        }
        catch(SQLException ex) {
            ex.printStackTrace();
        }
    }

    public void autoCommitOff()
    {
        try {
            conn.setAutoCommit(false);
        }catch(SQLException ex) {
            ex.printStackTrace();
        }
    }

    public void autoCommitOn()
    {
        try {
            conn.setAutoCommit(true);
        }catch(SQLException ex) {
            ex.printStackTrace();
        }
    }


    public int getCommitMode()
    {
        try {
            if(conn.getAutoCommit()){
                System.out.println(1);
                return 1;
            }
            else {
                System.out.println(0);
                return 0;
            }
        }catch(SQLException ex) {
            ex.printStackTrace();
        }
        return 2;
    }

    public int getTransactionLevelInt()
    {
        try {
            System.out.println(conn.getTransactionIsolation());
            return conn.getTransactionIsolation();
        }catch(SQLException ex) {
            ex.printStackTrace();
```

```
        }
        return 10;
    }

    public String getTransactionLevelString()
    {
        try {
            System.out.println(conn.getTransactionIsolation());
            switch(conn.getTransactionIsolation()) {
            case 0:
            {
                System.out.println("TRANSACTION_NONE");
                return("TRANSACTION_NONE");
            }
            case 1:
            {
                System.out.println("TRANSACTION_READ_UNCOMMITTED");
                return("TRANSACTION_READ_UNCOMMITTED");
            }
            case 2:
            {
                System.out.println("TRANSACTION_READ_COMMITTED");
                return("TRANSACTION_READ_COMMITTED");
            }
            case 4:
            {
                System.out.println("TRANSACTION_REPEATABLE_READ");
                return("TRANSACTION_REPEATABLE_READ");
            }
            case 8:
            {
                System.out.println("TRANSACTION_SERIALIZABLE");
                return("TRANSACTION_SERIALIZABLE");
            }
            default:
            {
                return("Error");
            }
            }
        }catch(SQLException ex) {
            ex.printStackTrace();
        }
        return "Error";
    }


    public List<EmployeeInfo> getEmployee()
    {
        List<EmployeeInfo> returnList = new ArrayList<>();
        try {
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM emp");
            while(rs.next()) {
                returnList.add(new EmployeeInfo(rs.getInt(1),rs.getString(2), rs.getFloat(6)));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnList;
    }

    boolean raiseSalary(String job, float amount)
    {
        boolean returnRaise = false;
        try {
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            stmt.executeQuery("UPDATE emp SET salary = salary + "+amount+" WHERE job = "+job);
```

```
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnRaise;
    }

    public List<EmployeeInfo> getEmployeePS()
    {
        List<EmployeeInfo> returnList = new ArrayList<>();
        try {
            conn.setAutoCommit(false);
            PreparedStatement prep = conn.prepareStatement("SELECT * FROM emp");
            ResultSet rs = prep.executeQuery();
            while(rs.next()) {
                returnList.add(new EmployeeInfo(rs.getInt(1),rs.getString(2), rs.getFloat(6)));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnList;
    }

    boolean raiseSalaryPS(String job, float amount)
    {
        boolean returnRaise = false;
        try {
            PreparedStatement prep = conn.prepareStatement("UPDATE emp SET salary = ? + WHERE job = ?");
            prep.setString(1, job);
            prep.setFloat(1,amount);
            conn.commit();
            returnRaise = true;
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnRaise;
    }

    public List<DepartmentInfo> getDepartments(Integer id, String name, String location)
    {
        List<DepartmentInfo> returnList = new ArrayList<>();
        try {
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT *"
                    + " FROM    dept"
                    + " WHERE   (DID ="+id+" OR "+id+" IS NULL)"
                    + " AND     (DNAME ="+name+" OR "+name+" IS NULL)"
                    + " AND     (DLOC ="+location+" OR "+location+" IS NULL)");
            while(rs.next()) {
                returnList.add(new DepartmentInfo(rs.getInt(1),rs.getString(2), rs.getString(3)));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnList;
```

```
    }

    public List<DepartmentInfo> getDepartmentsPS(Integer id, String name, String location)
    {
        List<DepartmentInfo> returnList = new ArrayList<>();
        try {
            conn.setAutoCommit(false);
            PreparedStatement prep = conn.prepareStatement("SELECT *"
                    + " FROM    dept"
                    + " WHERE   (DID = ? OR ? IS NULL)"
                    + " AND     (DNAME = ? OR ? IS NULL)"
                    + " AND     (DLOC = ? OR ? IS NULL)");
            ResultSet rs = prep.executeQuery();
            prep.setInt(1, id);
            prep.setInt(2, id);
            prep.setString(1, name);
            prep.setString(2, name);
            prep.setString(3, location);
            prep.setString(4, location);
            while(rs.next()) {
                returnList.add(new DepartmentInfo(rs.getInt(1),rs.getString(2), rs.getString(3)));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnList;
    }

    List<String> executeQuery(String query){
        List<String> returnList = new ArrayList<>();
        try {
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            while(rs.next()) {
                returnList.add(rs.getString(1));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnList;
    }

    List<String> executeStatement(String statement){
        List<String> returnList = new ArrayList<>();
        try {
            conn.setAutoCommit(false);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(statement);
            while(rs.next()) {
                returnList.add(rs.getString(1));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
        return returnList;
    }

    public void printEmployeePS()
```

```
    {
        try {
            conn.setAutoCommit(false);
            PreparedStatement prep = conn.prepareStatement("SELECT * FROM emp");
            ResultSet rs = prep.executeQuery();
            while(rs.next()) {
                System.out.print("ID = ");
                System.out.println(rs.getInt(1));
                System.out.println("Name = " + rs.getString(2) + "/nSalary = " + rs.getFloat(6));
            }
            conn.commit();
        }catch(SQLException e) {
            try {
                conn.rollback();
            }catch(SQLException ex) {
                ex.printStackTrace();
            }
        }
    }

}
```