LAB 9

LIBIN N GEORGE
111501015

PART A

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ javac simpledb/server/Startup.java
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java simpledb.server.Startup studentdb
new transaction: 1
creating new database
transaction 1 committed
database server ready
```

README.txt

```
1 package simpledb.server;
2
3 import simpledb.remote.*;
4 import java.rmi.registry.*;
5
6 public class Startup {
7    public static void main(String args[]) throws Exception {
8       // configure and initialize the database
9       SimpleDB.init(args[0]);
10
11      // create a registry specific for the server on the default port
12      Registry reg = LocateRegistry.createRegistry(1099);
13
14      // and post the server entry in it
15      RemoteDriver d = new RemoteDriverImpl();
16      reg.rebind("simpledb", d);
17
18      System.out.println("database server ready");
19   }
20 }
```

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ javac -Xlint studentClient/simpledb/CreateStudentDB.java
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java -cp .:studentClient/simpledb CreateStudentDB
Table STUDENT created.
STUDENT records inserted.
Table DEPT created.
DEPT records inserted.
Table COURSE created.
COURSE records inserted.
Table SECTION created.
SECTION records inserted.
Table ENROLL created.
ENROLL records inserted.
```

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java simpledb.server.Startup studentdb
new transaction: 1
creating new database
transaction 1 committed
database server ready
new transaction: 2
transaction 2 committed
new transaction: 3
transaction 3 committed
new transaction: 4
transaction 4 committed
new transaction: 5
transaction 5 committed
new transaction: 6
transaction 6 committed
new transaction: 7
transaction 7 committed
new transaction: 8
transaction 8 committed
new transaction: 9
transaction 9 committed
new transaction: 10
transaction 10 committed
new transaction: 11
transaction 11 committed
new transaction: 12
transaction 12 committed
new transaction: 13
transaction 13 committed
new transaction: 14
transaction 14 committed
new transaction: 15
transaction 15 committed
new transaction: 16
transaction 16 committed
new transaction: 17
transaction 17 committed
new transaction: 18
transaction 18 committed
new transaction: 19
transaction 19 committed
new transaction: 20
transaction 20 committed
new transaction: 21
transaction 21 committed
new transaction: 22
transaction 22 committed
new transaction: 23
transaction 23 committed
new transaction: 24
transaction 24 committed
new transaction: 25
transaction 25 committed
new transaction: 26
transaction 26 committed
new transaction: 27
transaction 27 committed
```

Each insertion in for creating records to Studentdb is executed as a transaction. Hence we get transaction for each statement.

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java -cp .:studentClient/simpledb:simpledb/remote/ StudentMajor
Name    Major
joe     compsci
max     compsci
lee     compsci
amy     math
sue     math
kim     math
pat     math
bob     drama
art     drama
```

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ javac studentClient/simpledb/FindMajors.java
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java -cp .:studentClient/simpledb:simpledb/remote/ FindMajors math
Here are the math majors
Name    GradYear
amy     2004
sue     2005
kim     2001
pat     2001
```

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ gedit studentClient/simpledb/FindMajors.java
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ javac studentClient/simpledb/SQLInterpreter.java
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java -cp .:studentClient/simpledb:simpledb/remote/ SQLInterpreter

SQL> select SName from STUDENT where GradYear=2004

     sname
----------
       joe
       amy
       art
       lee

SQL>
```

```java
public class SQLInterpreter {
    private static Connection conn = null;

    public static void main(String[] args) {
            try {
                        Driver d = new SimpleDriver();
                        conn = d.connect("jdbc:simpledb://localhost", null);

                        Reader rdr = new InputStreamReader(System.in);
                        BufferedReader br = new BufferedReader(rdr);

                        while (true) {
                                // process one line of input
                                System.out.print("\nIITPKD> ");
                                String cmd = br.readLine().trim();
                                System.out.println();
                                if (cmd.startsWith("exit"))
                                        break;
                                else if (cmd.startsWith("select"))
                                        doQuery(cmd);
                                else
                                        doUpdate(cmd);
                        }
             }
             catch (Exception e) {
                        e.printStackTrace();
                }
                finally {
                        try {
                                if (conn != null)
                                        conn.close();
                        }
                        catch (Exception e) {
                                e.printStackTrace();
                        }
```

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ java -cp .:studentClient/simpledb:simpledb/remote/ SQLInterpreter

IITPKD>
```

Below files show the files(functions) which is used to run select query.

```
// Step 1: connect to database server
Driver d = new SimpleDriver();
conn = d.connect("jdbc:simpledb://localhost", null);

// Step 2: execute the query
Statement stmt = conn.createStatement();
String qry = "select sname, gradyear "
           + "from student, dept "
           + "where did = majorid "
           + "and dname = '" + major + "'";
ResultSet rs = stmt.executeQuery(qry);
```

```
public class SimpleDriver extends DriverAdapter {

   /**
    * Connects to the SimpleDB server on the specified host.
    * The method retrieves the RemoteDriver stub from
    * the RMI registry on the specified host.
    * It then calls the connect method on that stub,
    * which in turn creates a new connection and
    * returns the RemoteConnection stub for it.
    * This stub is wrapped in a SimpleConnection object
    * and is returned.
    * <P>
    * The current implementation of this method ignores the
    * properties argument.
    * @see java.sql.Driver#connect(java.lang.String, Properties)
    */
   public Connection connect(String url, Properties prop) throws SQLException {
      try {
         String host = url.replace("jdbc:simpledb://", "");  //assumes no port specified
         Registry reg = LocateRegistry.getRegistry(host);
         RemoteDriver rdvr = (RemoteDriver) reg.lookup("simpledb");
         RemoteConnection rconn = rdvr.connect();
         return new SimpleConnection(rconn);
      }
      catch (Exception e) {
         throw new SQLException(e);
      }
   }
}
```

```java
public class SimpleConnection extends ConnectionAdapter {
   private RemoteConnection rconn;

   public SimpleConnection(RemoteConnection c) {
      rconn = c;
   }

   public Statement createStatement() throws SQLException {
      try {
         RemoteStatement rstmt = rconn.createStatement();
         return new SimpleStatement(rstmt);
      }
      catch(Exception e) {
         throw new SQLException(e);
      }
   }

   public void close() throws SQLException {
      try {
         rconn.close();
      }
      catch(Exception e) {
         throw new SQLException(e);
      }
   }
}
```

```java
/**
 * Executes the specified SQL query string.
 * The method calls the query planner to create a plan
 * for the query. It then sends the plan to the
 * RemoteResultSetImpl constructor for processing.
 * @see simpledb.remote.RemoteStatement#executeQuery(java.lang.String)
 */
public RemoteResultSet executeQuery(String qry) throws RemoteException {
   try {
      Transaction tx = rconn.getTransaction();
      Plan pln = SimpleDB.planner().createQueryPlan(qry, tx);
      return new RemoteResultSetImpl(pln, rconn);
   }
   catch(RuntimeException e) {
      rconn.rollback();
      throw e;
   }
}
```

```java
    /**
     * Creates a plan for an SQL select statement, using the supplied planner.
     * @param qry the SQL query string
     * @param tx the transaction
     * @return the scan corresponding to the query plan
     */
    public Plan createQueryPlan(String qry, Transaction tx) {
       Parser parser = new Parser(qry);
       QueryData data = parser.query();
       return qplanner.createPlan(data, tx);
    }
    |
    /**
```

Calles parser from transaction class .

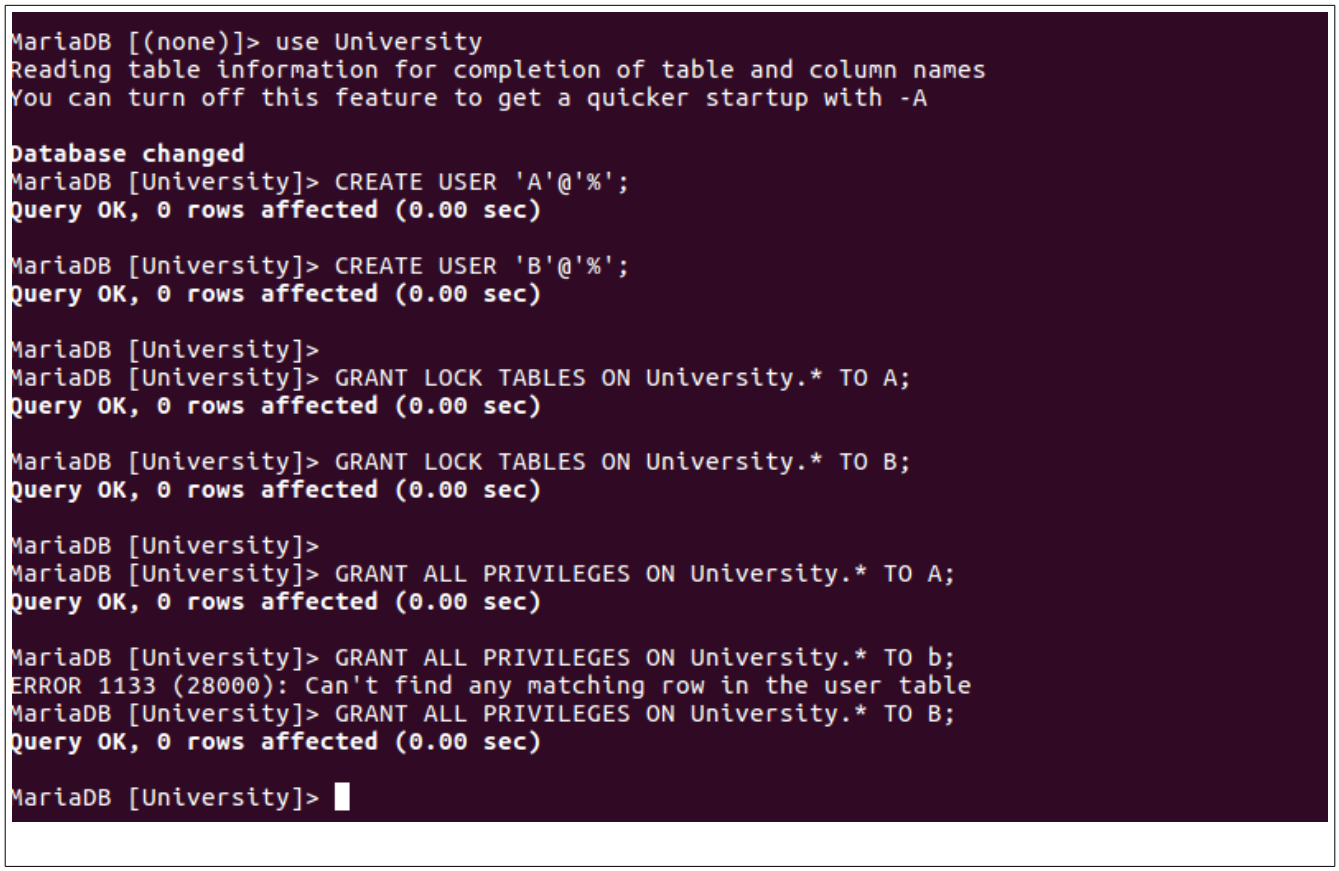
```java
package simpledb.file;

import static simpledb.file.Page.BLOCK_SIZE;
import java.io.*;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.util.*;

/**
 * The SimpleDB file manager.
 * The database system stores its data as files within a specified directory.
 * The file manager provides methods for reading the contents of
 * a file block to a Java byte buffer,
 * writing the contents of a byte buffer to a file block,
 * and appending the contents of a byte buffer to the end of a file.
 * These methods are called exclusively by the class {@link simpledb.file.Page Page},
 * and are thus package-private.
 * The class also contains two public methods:
 * Method {@link #isNew() isNew} is called during system initialization by {@link simpledb.server.SimpleDB#init}.
 * Method {@link #size(String) size} is called by the log manager and transaction manager to
 * determine the end of the file.
 * @author Edward Sciore
 */
public class FileMgr {
   private File dbDirectory;
   private boolean isNew;
   private Map<String,FileChannel> openFiles = new HashMap<String,FileChannel>();

   /**
    * Creates a file manager for the specified database.
    * The database will be stored in a folder of that name
    * in the user's home directory.
    * If the folder does not exist, then a folder containing
    * an empty database is created automatically.
    * Files for all temporary tables (i.e. tables beginning with "temp") are deleted.
    * @param dbname the name of the directory that holds the database
    */
   public FileMgr(String dbname) {
      String homedir = System.getProperty("user.home.test");
      dbDirectory = new File(homedir, dbname);
      isNew = !dbDirectory.exists();

      // create the directory if the database is new
      if (isNew && !dbDirectory.mkdir())
         throw new RuntimeException("cannot create " + dbname);

      // remove any leftover temporary tables
      for (String filename : dbDirectory.list())
         if (filename.startsWith("temp"))
         new File(dbDirectory, filename).delete();
   }
```

User.home is changed to change the location (here changed to user.home.test)

```
 2
 3 import simpledb.file.FileMgr;
 4 import simpledb.buffer.*;
 5 import simpledb.tx.Transaction;
 6 import simpledb.log.LogMgr;
 7 import simpledb.metadata.MetadataMgr;
 8 import simpledb.planner.*;
 9 import simpledb.opt.HeuristicQueryPlanner;
10 import simpledb.index.planner.IndexUpdatePlanner;
11
12 /**
13  * The class that provides system-wide static global values.
14  * These values must be initialized by the method
15  * {@link #init(String) init} before use.
16  * The methods {@link #initFileMgr(String) initFileMgr},
17  * {@link #initFileAndLogMgr(String) initFileAndLogMgr},
18  * {@link #initFileLogAndBufferMgr(String) initFileLogAndBufferMgr},
19  * and {@link #initMetadataMgr(boolean, Transaction) initMetadataMgr}
20  * provide limited initialization, and are useful for
21  * debugging purposes.
22  *
23  * @author Edward Sciore
24  */
25 public class SimpleDB {
26    public static int BUFFER_SIZE = 1024;
27    public static String LOG_FILE = "simpledb.log";
28
29    private static FileMgr     fm;
30    private static BufferMgr   bm;
31    private static LogMgr      logm;
32    private static MetadataMgr mdm;
33
34    /**
35     * Initializes the system.
36     * This method is called during system startup.
37     * @param dirname the name of the database directory
38     */
39    public static void init(String dirname) {
40       initFileLogAndBufferMgr(dirname);
41       Transaction tx = new Transaction();
42       boolean isnew = fm.isNew();
43       if (isnew)
44          System.out.println("creating new database");
45       else {
46          System.out.println("recovering existing database");
47          tx.recover();
48       }
49       initMetadataMgr(isnew, tx);
50       tx.commit();
51    }
52
53    // The following initialization methods are useful for
54    // testing the lower-level components of the system
55    // without having to initialize everything.
56
```

Buffer size is changed from 8 to 1024

B.


MariaDB [(none)]> use University
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed
MariaDB [University]> CREATE USER 'A'@'%';
Query OK, 0 rows affected (0.00 sec)
MariaDB [University]> CREATE USER 'B'@'%';
Query OK, 0 rows affected (0.00 sec)
MariaDB [University]>
MariaDB [University]> GRANT LOCK TABLES ON University.* TO A;
Query OK, 0 rows affected (0.00 sec)
MariaDB [University]> GRANT LOCK TABLES ON University.* TO B;
Query OK, 0 rows affected (0.00 sec)
MariaDB [University]>
MariaDB [University]> GRANT ALL PRIVILEGES ON University.* TO A;
Query OK, 0 rows affected (0.00 sec)
MariaDB [University]> GRANT ALL PRIVILEGES ON University.* TO b;
ERROR 1133 (28000): Can't find any matching row in the user table
MariaDB [University]> GRANT ALL PRIVILEGES ON University.* TO B;
Query OK, 0 rows affected (0.00 sec)
MariaDB [University]>

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ mysql -u A
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 13
Server version: 10.2.12-MariaDB-10.2.12+maria~xenial mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use University
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [University]> LOCK TABLE student WRITE ;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]>
```

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$  mysql -u B
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.2.12-MariaDB-10.2.12+maria~xenial mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use University
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [University]> lock table instructor WRITE;
Query OK, 0 rows affected (0.01 sec)

MariaDB [University]> describe student;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| ID        | varchar(5)  | NO   | PRI |         |       |
| name      | varchar(20) | NO   |     | NULL    |       |
| dept_name | varchar(20) | YES  | MUL | NULL    |       |
| tot_cred  | decimal(3,0)| YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.02 sec)

MariaDB [University]> insert into student values ("CS1234", "qwe", "Biology", 2);
ERROR 1100 (HY000): Table 'student' was not locked with LOCK TABLES
MariaDB [University]>
```

We cannot insert data as the table is already LOCKED hence mariadb was unable to LOCK the table before insertion. Hence aborted

```
Database changed
MariaDB [University]> LOCK TABLE student WRITE ;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> insert into instructor values ("1", "1", "Accounting", 12)
    -> ;
ERROR 1100 (HY000): Table 'instructor' was not locked with LOCK TABLES
MariaDB [University]>
```

We cannot insert data as the table is already LOCKED hence mariadb was unable to LOCK the table before insertion. Hence aborted

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$ mysql -u A
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 15
Server version: 10.2.12-MariaDB-10.2.12+maria~xenial mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use University
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [University]> set global innodb_deadlock_detect=OFF;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> set global innodb_print_all_deadlocks=ON;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> set innodb_lock_wait_timeout =120;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> create table dl1(pk int primary key, data varchar(100));
Query OK, 0 rows affected (0.18 sec)

MariaDB [University]> create table dl2(pk int primary key, pk1 int not null, constraint
    -> dl2_fk foreign key(pk1) references dl1(pk), data varchar(100));
Query OK, 0 rows affected (0.12 sec)

MariaDB [University]> set autocommit=off;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> insert into dl1 values(1, 'a');
Query OK, 1 row affected (0.00 sec)

MariaDB [University]>
```

`innodb_deadlock_detect`

Description: By default, the InnoDB deadlock detector is enabled. If set to off, deadlock detection is disabled and MariaDB will rely on innodb_lock_wait_timeout instead. This may be more efficient in systems with high concurrency as deadlock detection can cause a bottleneck when a number of threads have to wait for the same lock.

`innodb_lock_wait_timeout`

Description: Time in seconds that an InnoDB transaction waits for an InnoDB row lock (not table lock) before giving up with the error `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction`. When this occurs, the statement (not transaction) is rolled back. The whole transaction can be rolled back if the innodb_rollback_on_timeout option is used. Increase this for data warehousing applications or where other long-running operations are common, or decrease for OLTP and other highly interactive applications. This setting does not apply to deadlocks, which InnoDB detects immediately, rolling back a deadlocked transaction.

`innodb_print_all_deadlocks`

If set to `1` (`0` is default), all XtraDB/InnoDB transaction deadlock information is written to the error log.

```
111501015@iitpkdd122:~/Downloads/SimpleDB_2.10/SimpleDB_2.10$  mysql -u B
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 16
Server version: 10.2.12-MariaDB-10.2.12+maria~xenial mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use University
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [University]> set autocommit=off;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> insert into dl2(pk, pk1, data) values(10, 1, 'a0');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
MariaDB [University]>
```

```
MariaDB [University]> set autocommit=off;
Query OK, 0 rows affected (0.00 sec)

MariaDB [University]> insert into dl1 values(1, 'a');
Query OK, 1 row affected (0.00 sec)

MariaDB [University]> insert into dl2(pk, pk1, data) values(10, 1, 'a0');
```

User A

USER A => Locks dl1 (because of insert)
USER B => insert to dl2 depends on foreign key of dl1. It locks dl2 but waits for lock in dl1.
USER A => tries to insert to dl2 but unable to get a lock on dl2. (dl2 lock with USER B)

```
ariaDB [University]> insert into dl2(pk, pk1, data) values(10, 1, 'a0');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
ariaDB [University]> insert into dl2(pk, pk1, data) values(10, 1, 'a0');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
ariaDB [University]> commit;
uery OK, 0 rows affected (0.04 sec)

ariaDB [University]>
```

As it is commited now User B can get the lock hence complete the transaction

```
MariaDB [University]> insert into dl2(pk, pk1, data) values(10, 1, 'a0');
Query OK, 1 row affected (49.42 sec)

MariaDB [University]>
```

USER B

```
MariaDB [University]> insert into dl2(pk, pk1, data) values(10, 1, 'a0');
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```

WHEN set global innodb_deadlock_detect=ON;, we get the above error when dead lock is created.