7. Databases and JDBC

3. JDBC Additional Tasks

1. Pay List

- You should create a pay list for merchants accordingly their parameters: period of payment and minimal payment sum
- Please, create a DB table for saving pay list

1. Pay List – DB Structure

```
create table transMoney (
   id int not null generated always as identity,
   merchantId int constraint merchmoney_fk
references merchant,
   sumSent decimal(19,2),
   sentDate timestamp,
   status char(1),
   primary key (id)
);
```

1. Pay List – periodEnum

public enum periodEnum {*UNKNOWN, WEEKLY, TENDAYS, MONTHLY};

1. Pay List – getMerchantInfo Method

```
public ArrayList<MerchantInfo> getMerchantInfo(Connection
    conn) throws SQLException{
  Statement stmt = conn.createStatement();
  ArrayList<MerchantInfo> list = new ArrayList<MerchantInfo>();
  String sql = "select id, period, needToSend, lastSent, minSum
       from merchant";
  ResultSet rs = stmt.executeQuery(sql);
```

1. Pay List – MerchantInfo Inner Class

```
class MerchantInfo{
    private int id;
    private java.sql.Date lastSent;
    private double sum;
    private periodEnum period;
    private double minSum;
    public MerchantInfo(){}
        // accessors
}
```

1. Pay List – getMerchantInfo Method

```
    while (rs.next()){
        MerchantInfo info = new MerchantInfo();
        info.setId(rs.getInt("id"));
        info.setLastSent(rs.getDate("lastSent"));
        info.setPeriod(periodEnum.values()[rs.getInt("period")]);
        info.setSum(rs.getDouble("needToSend"));
        info.setMinSum(rs.getDouble("minSum"));
        list.add(info);
    }
    return list;
}
```

1. Pay List – filterList Method

```
public ArrayList<MerchantInfo> filterList(ArrayList<MerchantInfo>
                                          list){
   ArrayList<MerchantInfo> listRet = new ArrayList<MerchantInfo>();
   for (MerchantInfo info: list){
      if (info.getMinSum() > info.getSum()) continue;
         Instant instant = Instant.ofEpochMilli(info.getLastSent().getTime());
         LocalDate dt = LocalDateTime.ofInstant(instant,
          ZoneId.systemDefault()).toLocalDate();
         LocalDate current = LocalDate.now();
```

1. Pay List – filterList Method

```
switch(info.getPeriod()){
case WEEKLY:
   if (dt.until(current, ChronoUnit.WEEKS) < 1) continue;
   break;
case TENDAYS:
   if (dt.until(current, ChronoUnit.DAYS) < 10) continue;
   break;
case MONTHLY:
   if (dt.until(current, ChronoUnit.MONTHS) < 1) continue;
   break;
```

1. Pay List – filterList Method

```
        default:
            break;
        }
        listRet.add(info);
    }
    return listRet;
}
```

1. Pay List – addToTrans Method

```
public void addToTrans(Connection conn, ArrayList<MerchantInfo>
                                    list) throws SQLException{
  String sql = "INSERT INTO transMoney(merchantId, sumSent,
sentDate, status) values(?,?,?, '0')";
  PreparedStatement stmt = conn.prepareStatement(sql);
  for(MerchantInfo info: list){
    stmt.setInt(1, info.getId());
    stmt.setDouble(2, info.getSum());
    java.sql.Timestamp dt = new java.sql.Timestamp(new
java.util.Date().getTime());
    stmt.setTimestamp(3, dt);
    stmt.executeUpdate();
  }}
```

1. Pay List – main Method

```
public static void main(String[] args) {
   try{
      Connection conn = getConnection();
      TransMoney t = new TransMoney();
      ArrayList<MerchantInfo> list = t.getMerchantInfo(conn);
      list = t.filterList(list);
      t.addToTrans(conn, list);
      conn.close();
   } catch(Exception ex){
      System.out.println("Error " + ex.getMessage());
}}
```

1. Pay List

- See 729TransMoney project for the full text

2. Money Transfer

- Create a method that gets an accessible transfer sum as a parameter and sends money to merchants accordingly to the pay list under condition that general transfer sum should not grater then accessible transfer sum.

2. Money Transfer – transMoney Table

```
create table transMoney (
    id int not null generated always as identity,
    merchantId int constraint merchmoney_fk references
merchant,
    sumSent decimal(19,2),
    sentDate timestamp,
    status char(1),
    primary key (id)
);
```

TransferInfo Inner Class

```
class TransferInfo{
    private int id;
    private int merchantId;
    private double sumSent;
    private java.sql.Date sentDate;
    private String status;

    public TransferInfo(){}
    // accessors
}
```

2. Money Transfer - getUnpayed

```
public ArrayList<TransferInfo> getUnpayed(Connection conn)
    throws SQLException{
  Statement stmt = conn.createStatement();
  ArrayList<TransferInfo> list = new ArrayList<TransferInfo>();
  String sql = "select id, merchantId, sumSent, sentDate, status
    from transMoney where status='0' order by sentDate,
sumSent";
  ResultSet rs = stmt.executeQuery(sql);
```

2. Money Transfer - getUnpayed

```
while (rs.next()){
   TransferInfo info = new TransferInfo();
   info.setId(rs.getInt("id"));
   info.setMerchantId(rs.getInt("merchantId"));
   info.setSumSent(rs.getDouble("sumSent"));
   info.setSentDate(rs.getDate("sentDate"));
   info.setStatus(rs.getString("status"));
   list.add(info);
   }
   return list;
}
```

2. Money Transfer - procUnpayed

```
public void procUnpayed(Connection conn,
   ArrayList<TransferInfo> list, double sum) throws SQLException{
      double sentSum = 0.0;
      for(TransferInfo info: list){
     if (sentSum + info.getSumSent() > sum) continue;
     sentSum += info.getSumSent();
     try{
            conn.setAutoCommit(false);
            sendPayment(conn, info);
            updateMerchant(conn, info);
           conn.commit();
```

2. Money Transfer - procUnpayed

```
    } catch (Exception ex){
            ex.printStackTrace();
           conn.rollback();
    }
    }
}
```

2. Money Transfer - sendPayment

```
public void sendPayment(Connection conn, TransferInfo info)
     throws SQLException{
    String sql = "UPDATE transMoney set sentDate=?, status='1'
where id=?";
    PreparedStatement stmt = conn.prepareStatement(sql);
    java.sql.Timestamp dt = new java.sql.Timestamp(new
java.util.Date().getTime());
    stmt.setTimestamp(1, dt);
    stmt.setInt(2, info.getId());
    stmt.executeUpdate();
}
```

2. Money Transfer - updateMerchant

```
public void updateMerchant(Connection conn, TransferInfo info)
        throws SQLException{
   String sql = "SELECT needToSend, sent FROM merchant where
     id=?";
   PreparedStatement stmtRead = conn.prepareStatement(sql);
   stmtRead.setInt(1, info.getMerchantId());
   ResultSet rs = stmtRead.executeQuery();
   rs.next();
   double needToSend = rs.getDouble("needToSend");
   double sent = rs.getDouble("sent");
```

2. Money Transfer - updateMerchant

```
    sql = "UPDATE merchant set lastSent=?, needToSend=?, sent=?
      where id=?";
    PreparedStatement stmt = conn.prepareStatement(sql);
    java.sql.Timestamp dt = new java.sql.Timestamp(new
      java.util.Date().getTime());
    stmt.setTimestamp(1, dt);
    stmt.setDouble(2, needToSend - info.getSumSent());
    stmt.setDouble(3, sent + info.getSumSent());
    stmt.setInt(4, info.merchantId);
    stmt.executeUpdate();
}
```

2. Money Transfer - main

```
public static void main(String[] args) throws SQLException{
    Connection conn = null;
  try{
    double sum1 = Double.valueOf(args[0]);
    conn = getConnection();
    MainTrans t = new MainTrans();
    ArrayList<TransferInfo> list = t.getUnpayed(conn);
    t.procUnpayed(conn, list, sum1);
  }
```

2. Money Transfer - main

```
    catch(SQLException ex){
     System.out.println("Error " + ex.getMessage());
    }
    catch(Exception ex){
     System.out.println("Error " + ex.getMessage());
    }
    finally{
     if (conn!= null) conn.close();
    }
}
```

See 729aTransMoney project for the full text