SQL queries used:

? = To be substitute with the user prompt data

“+ x + ”= x is a argument get from the user

Note: **Even though the queries have been optimized separately and verified by using them on the database, some of the optimized queries are not implemented in the final design as we were unable to measure their performance upon large set of data due to time limitations.**

Directly on Account Table

1. Getting the Descending order of accounts ID to calculate the next ID

   Query: Select accId from account order by 1 desc limit 1;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: Yes*

2. Creating a New Bank Account for a user

   Query: Insert into account values(?,?,?,?);
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

3. Updating the Normal Balance of an Account

   Query: Update account set norBalance = ? where accId = ? ;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: Yes*

4. Updating the Compulsory Balance

   Query: Update account set comBalance = ? where accId = ? ;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: Yes*

5. Getting the current Normal Balance for an account

   Query: Select norBalance from account where memId = ?;
   Indexes: Index for the column 'memId'
   CREATE INDEX idxMemId ON account(memId);
   Optimizations: None
   Transactions used: Yes*

6. Getting the current Compulsory Balance for an account

   Query: Select comBalance from account where memId = ?;
   Indexes: Index for the column 'memId' - idxMemID
   Optimizations: None
   Transactions used: Yes*

7. Finding the Current Worth of the Bank

   Query: select sum(norBalance), sum(comBalance) from account;
   select sum(amount) from fixed;
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

Directly on Attendance Table

1. Getting the Monthly attendance of the members

   Query: select month(meetDate), sum(status) from attendance where year(meetDate)= ? group by month(meetDate);
   Indexes: Index for the column 'meetDate'
   CREATE INDEX idxMeetDate ON attendance (meetDate);
   Optimizations: None
   Transactions used: Yes*

Directly on Detail Table

1. Getting the derail regarding transactions using detailID

   Query: Select detail from detail where detailId = ?;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: None

Directly on Fixed Table

1. Getting the Descending order of Fixed deposit ID to determine the next ID

   Query: Select fixId from fixed order by 1 desc limit 1;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: Yes*

2. Opening a new Fixed Deposit

   Query: Insert into fixed values(?,?,?,?,?,?,?,?,?,?);
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

3. Convert the fixed deposit to a Bond

Query: Update fixed set isBond = true where fixId = ? ;
Indexes: Primary Key Index
Optimizations: None
Transactions used: Yes*

4. Updating the Compulsory Balance

Query: select * from fixed natural join account
where
memId = ? and
isWithdraw = false and
isBond = false;
Indexes: Index for the column 'memId' - idxMemID
Index for the columns 'isWithdraw' & 'isBond'
CREATE INDEX idxIsWithdBond ON fixed (isWithdraw, isBond);

Optimizations:

```
select * from
    (select * from fixed
    Where
        isWithdraw = false AND
        isBond = false) as s natural join
    (select * from account
    Where
        memId = ?) as t;
```

Transactions used: Yes*

5. Getting the current Number of fixed Deposits owned by a user

Query:
```
select count(fixId)
    from fixed f, account a
where
    f.accId = a.accId and
    isWithdraw = false and
    memId = ?;
```

Indexes: Primary key index on ‘accId’ on account
Indexes on ‘fixId’ , ‘accId’ & ‘isWithdraw’

```
CREATE INDEX idxFixidAccidIsWithd ON fixed (fixId, accId, isWithdraw);
```

Optimizations:
```
select count(fixId) from
(select fixId, accId from fixed
Where
    isWithdraw = false) as s inner join
(select accId from account
Where
    memId = 1) as t
where
    s.accId = t.accId;
```

Transactions used: Yes*

6. Getting the current Compulsory Balance for an account

Query:
```
select a.memId, name, fixDate, amount, period
        from fixed f, account a, member m
    where
            f.accId = a.accId and
            a.memId = m.memId and
            isMember = ?
            isWithdraw = ?
    order by a.memId;
```

Indexes:
Index for the columns ‘name’, ‘memId’ & ‘isMember’
```
CREATE INDEX idxNaMeIsMem ON member (name, memId, isMember);
```
Primary Key Index for ‘memId’
Indexes on ‘fixId’, ‘accId’ & ‘isWithdraw’ - idxFixidAccidIsWithd

Optimizations:
```
select m.memId, m.name, f.fixDate, f.amount, f.period from
    (select memId, name from member
        Where
            isMember = true) as m inner join
    (select accId , memId from account) as a inner join
    (select accId fixDate , amount,  period
        Where
            isWithdraw = false ) as f
where
            m.memId = a.memId and
            f.accId = a.accId;
```

Transactions used: Yes*

Directly on Gurantor Table

1. Getting the Descending order of gurantor ID to calculate the next guratnor

   Query: Select guaId from guarantor order by 1 desc limit 1;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: Yes*

2. Creating a New Bank Account for a user

   Query: Insert into guarantor values(?,?,?,?);
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

Directly on Loan Table

1. Getting the Descending order of Loan Modal ID to calculate the next loanId

   Query: Select loanId from loan order by 1 desc limit 1;
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: Yes*

2. Adding a New Bank Loan for a user

   Query: Insert into loan values(?,?,?,?,?,?,?,?,?,?,?);
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

3. View currently available Loan Modals

   Query: select * from loan;
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

4. Find the number of currently available Loan Modals

   Query: select count(loanId) from loan;
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

5. Delete a Loan Modal

   Query: Delete from loan where loanId = ?;
   Indexes: None
   Optimizations: None
   Transactions used: Yes*

Directly on LoanTaken Table

1. Get the next laon taken ID

   Query: Select takenId from loanTaken order by 1 desc limit 1
   Indexes: Primary Key Index
   Optimizations: None
   Transactions used: None*

2. Add a new loan

   Query: Insert into loanTaken values(?,?,?,?,?,?,?,?,?,?,?,?,?)
   Indexes: None
   Optimizations: None
   Transactions used: None*

3. list taken loans

   Query: select loanId, t.loanName
   from loanTaken t inner join loan l
   on t.loanName = l.loanName
   group by loanId;
   Indexes: none
   Optimizations: yes
   Transactions used: None*

4. list debtors of the loan

   Query: select memId from loanTaken group by memId
   Indexes: -memId
   Optimizations: None
   Transactions used: None*

5. list loans of the client

Query: select loanId, balance
from loanTaken t, loan l
where t.loanName = l.loanName and memId = ?
Indexes: None
Optimizations:
select loanId, balance
from (Select loanName ,memId,balance from loanTaken t
where memId = memId + ") t inner join loan l
ON t.loanName = l.loanName;
Transactions used: None*

6. Get loan shedule

Query: select m.memId, name, takenDate, period, amount, balanc
from member m, loanTaken t
where m.memId = t.memId and
loanName = \""+selectedItem+"\"";
Indexes: None
Optimizations: None
Transactions used: None*

7. Update loan balance of the members

Query: Update loanTaken set balance = ? where takenId = ?
Indexes: None
Optimizations: None
Transactions used: None*

8. Get amount of all the loan models

Query: select loanName, sum(amount)
from loanTaken group by loanName
Indexes: None
Optimizations: None
Transactions used: None*

Directly on Member Table

1. Get next member Id

   Query: Select memId from member order by 1 desc limit 1
   Indexes: None
   Optimizations: None
   Transactions used: None*

2. Add new member

   Query: Insert into member values(?,?,?,?,?,?,?,?,?,?,?)
   Indexes: None
   Optimizations: None
   Transactions used: None*

3. Search member from member Id

   Query: Select * from member where memId = ?
   Indexes: None
   Optimizations: None
   Transactions used: None*

4. Search member from member Id

   Query: Select * from member where memId = ?
   Indexes: None
   Optimizations: None
   Transactions used: None*

5. Search member id from text

   Query: Select memId, name
   from member
   where name like '%" + text + "%' and
   isMember = true and
   isActive = true;
   Indexes: None
   Optimizations: None
   Transactions used: None*

6. Search member from text

Query: Select memId, name
from member
where name like '%" + text + "%' and
isActive = true;
Indexes: None
Optimizations: None
Transactions used: None*

7. list all the client with relevant account type

Query: select * from member natural join account where =type
Indexes: None
Optimizations: None
Transactions used: None*

8. Initialize an image of a member

Query: Insert into memImage values(?,?)
Indexes: None
Optimizations: None
Transactions used: None*

9. Add an image

Query: Update memImage set image = ? where memId = ?
Indexes: None
Optimizations: None
Transactions used: None*

10. get the image

Query: Select image from memImage where memId = "+memId
Indexes: None
Optimizations: None
Transactions used: None*

11. Update a member profile

Query: Update member set " + text + " where memId = " + memId
Indexes: None
Optimizations: None
Transactions used: None*

Directly on Normal Table

1. Get next normal member Id

Query: Select norId from normal order by 1 desc limit 1
Indexes: None
Optimizations: None
Transactions used: None*

2. Get normal member deposit details

Query:

```
select norDate, norTime, norType, amount, n.balance
from normal n, account a
where n.accId = a.accId and memId = " + memberId +
order by norDate desc, norTime desc;
```

Indexes: Primary key index on 'memId'
Optimizations:

```
select norDate, norTime, norType, amount, n.balance
from
    (Select * from account a
        where
            memId =" + memberId + "  ) a
            inner join normal n
ON n.accId = a.accId
order by norDate desc, norTime desc;
```

Transactions used: None*

3. Get last interest day of a normal member

Query:

```
select min(datediff(curdate(), norDate))
from normal n, account a
where n.accId = a.accId and
        norType = 1 and
        memId = "+memberId;
```

Indexes: None

Optimizations:

```
select min(datediff(curdate(), norDate))
      from
            (Select memID, accID from account a
            where memId =" + memberId + " ) a inner join
            (Select    norDate ,accID ,norType from normal n
            where norType ='1') n
     ON  n.accId = a.accId;
```

Transactions used: None*

4. Get next interest details of a normal member

Query:

```
select
a.accId, min(datediff(curdate(), norDate)), sum(interest), norBalance
from
      (Select norBalance ,accId from account a
       where memId = " + memberId + ") a inner join normal n
ON
      n.accId = a.accId
order by norDate asc;
```

Indexes: None

Optimizations:

```
select a.accId, min(datediff(curdate(), norDate)), sum(interest),
norBalance\n" +
from (Select norBalance ,accId from account a
        where
                memId = " + memberId + ") a inner join normal n
ON n.accId = a.accId
order by norDate asc;
```

Transactions used: None*

5. Add new deposit for a normal member

Query: Insert into normal values(?,?,?,?,?,?,?,?)
Indexes: None
Optimizations: None
Transactions used: yes

6. Get details of a deposit schedule of a normal member

Query:
```
select a.accId, name, balance
        from normal n, account a, member m
        where
              n.accId = a.accId and
               a.memId = m.memId "+clientType+" and norId in
              (select max(norId)  from norma
                        where norDate <= '" + date + "
                        group by accId\n
               )order by accId;
```
Indexes: Index on column accId
```
CREATE INDEX IdxNor
   ON normal(accId)
```
Optimizations: None
Transactions used: No

Directly on Payment Table

1. Get next payment id

   Query: Select payId from payment order by 1 desc limit 1
   Indexes: None
   Optimizations: None
   Transactions used: None

2. Add a new payment

   Query: Insert into payment values(?,?,?,?,?,?,?)
   Indexes: None
   Optimizations: None
   Transactions used: None

3. Get next payment details

   Query:

```
select t.takenID, loanName, rate, method, rateType, fineType,
period, t.amount, balance,
max(datediff(curdate(), payDate)),
min(datediff(curdate(), payDate))

from payment p, loanTaken t
        where      p.takenId = t.takenID and
                   balance != -1 and memId = " + memberId + "
group by t.takenID;
```

   Indexes: None

Optimizations:

```
select t.takenID, loanName, rate, method, rateType, fineType,
period, t.amount, balance,
max(datediff(curdate(), payDate)),
min(datediff(curdate(), payDate))

from
        (select * from  loanTaken t
                Where     memId = "+ memberId + " and
                          balance != -1 ) t inner join payment p
on  p.takenId = t.takenID
group by t.takenID;
```

Transactions used: None

4. Get previous payment details

Query:

```
select payDate, loanName, interest, fine, p.amount-(interest+fine)
from payment p, loanTaken t
        where p.takenId = t.takenID and
                balance != -1 and
                memId = " + memberId + "
order by payDate desc;
```

Indexes: None

Optimizations:

```
select payDate, loanName, interest, fine, p.amount-(interest+fine)
from
        (select * from  loanTaken t
                where memId =" + memberId + "
                and balance != -1 ) t inner join payment p
on  p.takenId = t.takenID
order by payDate desc;
```

Transactions used: None

5. Get the latest payment details

Query:

```
select payDate, payTime, count(loanName), sum(interest),
sum(fine),   sum(p.amount-(interest+fine)), sum(p.amount)
from payment p, loanTaken t
where p.takenId = t.takenID and
          balance != -1 and
          memId = " + memberId + "
group by payDate
order by payDate desc limit 1;
```

Indexes: None

Optimizations:

```
select payDate, payTime, count(loanName), sum(interest),
sum(fine), sum(p.amount-      (interest+fine)), sum(p.amount)\
from
        (select * from  loanTaken t
                where       memId =" + memberId + " and
                            balance != -1 ) t inner join payment p\n
 on  p.takenId = t.takenID
 group by payDate
 order by payDate desc limit 1;
```

Transactions used: None

Directly on Shares Table

1. Get next Share id

   Query: Select shaId from shares order by 1 desc limit 1
   Indexes: None
   Optimizations: None
   Transactions used: None

2. Get all Share balance

   Query: select m.memId, name, sum(amount)
   from shares s, member m
   where s.memId = m.memId AND isMember = true
   group by m.memId";Indexes: None

   CREATE INDEX IdxSharesmemId
   ON shares(memId);
   Optimizations: None
   Transactions used: yes

3. Get list of members shares

   Query: select * from shares where memId= "+memberId;
   Indexes: None
   Optimizations: None
   Transactions used: None

Directly on Compulsory Table

1. Get next Compulsory id

   Query: Select comId from compulsory order by 1 desc limit 1"
   Indexes: None
   Optimizations: None
   Transactions used: None

2. Get Compulsory deposits list

   Query: select comDate, comTime, comType, amount, c.balance
   from compulsory c, account a
   where c.accId = a.accId and memId = " + memberId +
   order by comDate desc, comTime desc;
   Indexes: None
   Optimizations:
   select comDate, comTime, comType, amount, c.balance
   from
   (Select accId from account a
   where memId = " + memberId + ") a inner join compulsory c
   ON c.accId = a.accId
   order by comDate desc, comTime desc

   Transactions used: None

3. Get last interest day count

Query:

```
select min(datediff(curdate(), comDate))
from compulsory c, account a
where c.accId = a.accId and comType = 1 and memId = "+memberId;
```

Indexes: None

Optimizations:

```
select min(datediff(curdate(), comDate))
from
(Select accId from account a
        where memId = " + memberId + ") a inner join compulsory c
ON c.accId = a.accId;
```

Transactions used: None

4. Get next interest day details

Query:

```
select a.accId, min(datediff(curdate(), comDate)), sum(interest),
comBalance
from compulsory c, account a
where c.accId = a.accId and
          memId = " + memberId + "
order by comDate asc;
```

Indexes: None

Optimizations:

```
select a.accId, min(datediff(curdate(), comDate)), sum(interest),
comBalance \n" +
from
(Select accId,comBalance from account a
        where memId = " + memberId + ") a inner join compulsory c
ON c.accId = a.accId
order by comDate asc
```

Transactions used: None

5. Add new compulsory deposit

Query: Insert into compulsory values(?,?,?,?,?,?,?,?)
Indexes: None
Optimizations: None
Transactions used: Yes

6. Get compulsory shedule

Query:

```
 select a.accId, name, balance
from compulsory c, account a, member m
where c.accId = a.accId and
        a.memId = m.memId  and comId
in
       (select max(comId)
              from compulsory
              where comDate <= '" +date+ "'
group by accId
order by accId;
```

Indexes: None
Optimizations: None
Transactions used:None