BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam sub bab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan perangkat lunak ini adalah sebagai berikut :

1. Intel® Core™ 2 CPU P8600 @ 2,40GHz
2. Memori 4 GB
3. Harddisk 320 GB
4. Monitor 12”

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan perangkat lunak ini adalah sebagai berikut :

1. Sistem Operasi Windows 7 Ultimate 32 bit
2. Xampp 1.3.2
3. Microsoft Visual C# 2010 Express
4. Notepad++

4.2 Implementasi Program

Berdasarkan analisa dan perancangan proses yang terdapat pada bab 3, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Frequent Sequence

Proses ini bertujuan untuk menemukan semua frequent sequence, mulai 1-sequence hingga k-sequence.

4.2.1.1 Frequent 1-sequence

Fungsi ini bertujuan untuk mencari frequent 1-sequence, karena untuk mendapatkan 1-sequence tidak perlu melakukan proses join, maka untuk mencari

1-sequence langsung dilakukan scan pada database untuk mencari 1-sequence yang memenuhi nilai minimum support. Sourcecode untuk fungsi ini dapat dilihat pada Sourcecode 4.1

Frequent 1-sequence

```
public void createTable1Sequence(int min_sup)
        {
            dbSpade.executeNonQuery("TRUNCATE TABLE `Frequent_Sequence`;"+
                                    "INSERT INTO `Frequent_Sequence` " +
                                    "SELECT TRIM(LEADING '0' FROM
`Retail`.`Transactions`.`Product_Code`) AS `Sequence`,
COUNT(DISTINCT(`Retail`.`Transactions`.`Card_id`)), 1 " +
                                    "FROM `Retail`.`Transactions` " +
                                    "GROUP BY `Sequence` " +
                                    "HAVING
COUNT(DISTINCT(`Retail`.`Transactions`.`Card_id`)) >= " + min_sup +
                                    " ORDER BY `Sequence`;");
            dbSpade.executeNonQuery("TRUNCATE TABLE `Sequence_Rules`;");

            db.openDB();
            MySqlDataReader reader = db.executeReader("SELECT
`Retail`.`Transactions`.`Product_Code` AS `Sequence`, " +

"COUNT(DISTINCT(`Retail`.`Transactions`.`Card_id`)) AS `Count` FROM
`Retail`.`Transactions`" +
                                    " GROUP BY `Sequence` HAVING
COUNT(DISTINCT(`Retail`.`Transactions`.`Card_id`)) >= " + min_sup +
                                    " ORDER BY `Sequence`;");
            StringBuilder sb = new StringBuilder();

            int idx = 0;
            while (reader.Read())
            {
                idx++;

                sb.Append("CREATE TABLE IF NOT EXISTS `");
                sb.Append(reader.GetInt32(0));
                sb.Append("` (`SID` char(16), `EID` int(2));");

                sb.Append("TRUNCATE TABLE `");
                sb.Append(reader.GetInt32(0));
                sb.Append("`;");

                sb.Append("INSERT INTO `");
                sb.Append(reader.GetInt32(0));
                sb.Append("` SELECT `Retail`.`Transactions`.`Card_ID` AS
`SID`, MONTH(`Retail`.`Transactions`.`Timestamp`) AS `EID` ");
                sb.Append("FROM `Retail`.`Transactions` ");
                sb.Append("WHERE `Retail`.`Transactions`.`Product_Code` =
'");
                sb.Append(reader.GetString(0));
                sb.Append("' GROUP BY `SID`, `EID` ");
                sb.Append("ORDER BY `SID`, `EID`;");

                if (idx % 30 == 0)
                {
```

```
                dbSpade.executeNonQuery(sb.ToString());
                sb.Clear();
                idx = 0;
            }
        }
        reader.Close();

        dbSpade.executeNonQuery(sb.ToString());
        sb.Clear();

        db.closeDB();
    }
```

Sourcecode 4.1 Fungsi frequent 1-itemset

4.2.1.2 Fungsi Join Sequence

Fungsi ini terdiri dari 2 bagian, fungsi getsequence dan join2sequence. Getsequence berfungsi untuk menggabungkan 2 sequence, awalanya dilakukan pengecekan terlebih dahulu panjang dari sequence yang akan digabungkan, kemudian dilakukan pengecekan apakah sequence yang akan digabungkan memiliki prefix yang sama. Kemudian hasil dari getsequence ini dijadikan input dalam fungsi join2sequence, dimana dalam fungsi ini hasil dari getsequence dimasukkan dalam tabel. Dalam proses join2sequence juga dilakukan pengecekan untuk setiap kemungkinan kandidat sequence yang terbentuk. Dalam fungsi ini juga langsung dibentuk rule dengan mengecek subsequence dari frequent sequnce yang terbentuk. Kemudian dihitung nilai confidence dan nilai lift rasionya. Rule ini kemudian ditampilkan dalam tabel sequence rule. Fungsi – fungsi ini dapat dilihat pada Sourcecode 4.2 dan 4.3 berikut.

GetSequence

```
    public string[] getSequence(string s1, string s2)
    {
        string[] result = new string[0];
        string[] sq1 = s1.Split('-');
        string[] sq2 = s2.Split('-');

        if (sq1.Length < sq2.Length)
        {
            int idx = -1;
            for (int i = 0; i < sq1.Length; i++)
            {
```

```
                    if (sq1[i] == sq2[i]) idx = i;
                    else break;
                }
                if (idx == sq1.Length - 2 && sq1[idx+1].IndexOf(sq2[idx+1])
> -1)
                {
                    result = new string[] { s1 + "-" + sq2[sq2.Length - 1]
}; //1 right
                }
            }
            else if (sq1.Length == sq2.Length)
            {
                string s = "";
                int idx = -1;

                for (int i = 0; i < sq1.Length; i++)
                {
                    if (sq1[i] == sq2[i]) { idx = i; s += sq1[i] + "-"; }
                    else break;
                }

                if (idx == sq1.Length - 2)
                {
                    string[] atom1 = sq1[sq1.Length - 1].Split(',');
                    string[] atom2 = sq2[sq2.Length - 1].Split(',');

                    if (atom1.Length > 1 || atom2.Length > 1)
                    {
                        string tmp = "";
                        int x = 0, y = 0;
                        while (true)
                        {
                            if (x == atom1.Length && y == atom2.Length)
break;
                            else if (x < atom1.Length && y < atom2.Length
&& atom1[x].CompareTo(atom2[y]) <= 0)
                            {
                                if (tmp.IndexOf(atom1[x]) == -1)
                                {
                                    tmp += atom1[x] + ",";
                                }
                                x++;
                            }
                            else if (x < atom1.Length && y < atom2.Length
&& atom1[x].CompareTo(atom2[y]) > 0)
                            {
                                if (tmp.IndexOf(atom2[y]) == -1)
                                {
                                    tmp += atom2[y] + ",";
                                }
                                y++;
                            }
                            else if (x < atom1.Length && y == atom2.Length)
                            {
                                for (; x < atom1.Length; x++)
                                {
                                    if (tmp.IndexOf(atom1[x]) == -1)
                                    {
                                        tmp += atom1[x] + ",";
```

```
                                }
                            }
                        }
                        else if (x == atom1.Length && y < atom2.Length)
                        {
                            for (; y < atom2.Length; y++)
                            {
                                if (tmp.IndexOf(atom2[y]) == -1)
                                {
                                    tmp += atom2[y] + ",";
                                }
                            }
                        }
                    }
                    tmp = tmp.Remove(tmp.Length - 1);

                    result = new string[] { s + tmp, "" }; //1 and
                }
                else if(atom1.Length == 1 && atom2.Length == 1 &&
atom1[0] != atom2[0])
                {
                    string a1 = atom1[0];
                    string a2 = atom2[0];
                    if (atom1[0].CompareTo(atom2[0]) > 0)
                    {
                        a1 = atom2[0];
                        a2 = atom1[0];
                    }
                    //3 and, right, left
                    result = new string[] { s + a1 + "," + a2, s +
atom1[0] + "-" + atom2[0], s + atom2[0] + "-" + atom1[0] };
                }
            }
            else if (idx == sq1.Length - 1)
            {
                result = new string[] { s + sq1[sq1.Length - 1] }; //1
right (same atom)
            }
        }
        else
        {
            int idx = -1;
            for (int i = 0; i < sq2.Length; i++)
            {
                if (sq1[i] == sq2[i]) idx = i;
                else break;
            }
            if (idx == sq2.Length - 2 && sq2[idx + 1].IndexOf(sq1[idx +
1]) > -1)
            {
                result = new string[] { s2 + "-" + sq1[sq1.Length - 1]
}; //1 right
            }
        }

        return result;
    }
```

Sourcecode 4.2 Fungsi getsequence

Join2sequence

```
public void join2Sequence(string s1, float f1, string s2, float f2, int
min_sup, int k)
        {
            string[] result = getSequence(s1, s2);

            if (result.Length == 1) //1 right
            {
                object value = dbSpade.executeScalar("SELECT
COUNT(DISTINCT(`TMP`.`SID`)) " +
                                                      "FROM `" + s1 + "`, `"
+ s2 + "` AS `TMP` " +
                                                      "WHERE `" + s1 +
"`.`SID` = `TMP`.`SID` AND `" + s1 + "`.`EID` < `TMP`.`EID`;");
                object count = dbSpade.executeScalar("SELECT
COUNT(`Sequence`) FROM `Frequent_Sequence` WHERE `Sequence` = '" +
result[0] + "';");
                if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) == 0)
                {
                    dbSpade.executeNonQuery("CREATE TABLE IF NOT EXISTS `"
+ result[0] + "` (`SID` char(16), `EID` int(2));" +
                                            "TRUNCATE TABLE `" + result[0]
+ "`;" +
                                            "INSERT INTO `" + result[0] +
"` SELECT `TMP`.`SID`, `TMP`.`EID` " +
                                            "FROM `" + s1 + "`, `" + s2 +
"` AS `TMP` " +
                                            "WHERE `" + s1 + "`.`SID` =
`TMP`.`SID` AND `" + s1 + "`.`EID` < `TMP`.`EID`;");
                    dbSpade.executeNonQuery("INSERT INTO
`Frequent_Sequence` VALUES ('" + result[0] + "', " + value.ToString() + ",
" + k.ToString() + ");");

                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100+ ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
                else if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) > 0)
                {
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
```

```
                }
            }
            else if (result.Length == 2) //1 and
            {
                object value = dbSpade.executeScalar("SELECT
COUNT(DISTINCT(`TMP`.`SID`)) " +
                                                     "FROM `" + s1 + "`, `"
+ s2 + "` AS `TMP` " +
                                                     "WHERE `" + s1 +
"`.`SID` = `TMP`.`SID` AND `" + s1 + "`.`EID` = `TMP`.`EID`;");
                object count = dbSpade.executeScalar("SELECT
COUNT(`Sequence`) FROM `Frequent_Sequence` WHERE `Sequence` = '" +
result[0] + "';");
                if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) == 0)
                {
                    dbSpade.executeNonQuery("CREATE TABLE IF NOT EXISTS `"
+ result[0] + "` (`SID` char(16), `EID` int(2));" +
                                            "TRUNCATE TABLE `" + result[0]
+ "`;" +
                                            "INSERT INTO `" + result[0] +
"` SELECT `TMP`.`SID`, `TMP`.`EID` " +
                                            "FROM `" + s1 + "`, `" + s2 +
"` AS `TMP` " +
                                            "WHERE `" + s1 + "`.`SID` =
`TMP`.`SID` AND `" + s1 + "`.`EID` = `TMP`.`EID`;");
                    dbSpade.executeNonQuery("INSERT INTO
`Frequent_Sequence` VALUES ('" + result[0] + "', " + value.ToString() + ",
" + k.ToString() + ");");

                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
                else if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) > 0)
                {
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
            }
            else if (result.Length == 3) //3 and, right, left
            {
                object value = dbSpade.executeScalar("SELECT
```

```
COUNT(DISTINCT(`TMP`.`SID`)) " +
                                                "FROM `" + s1 + "`, `"
+ s2 + "` AS `TMP` " +
                                                "WHERE `" + s1 +
"`.`SID` = `TMP`.`SID` AND `" + s1 + "`.`EID` = `TMP`.`EID`;");
                object count = dbSpade.executeScalar("SELECT
COUNT(`Sequence`) FROM `Frequent_Sequence` WHERE `Sequence` = '" +
result[0] + "';");
                if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) == 0)
                {
                    dbSpade.executeNonQuery("CREATE TABLE IF NOT EXISTS `"
+ result[0] + "` (`SID` char(16), `EID` int(2));" +
                                            "TRUNCATE TABLE `" + result[0]
+ "`;" +
                                            "INSERT INTO `" + result[0] +
"` SELECT `TMP`.`SID`, `TMP`.`EID` " +
                                            "FROM `" + s1 + "`, `" + s2 +
"` AS `TMP` " +
                                            "WHERE `" + s1 + "`.`SID` =
`TMP`.`SID` AND `" + s1 + "`.`EID` = `TMP`.`EID`;");
                    dbSpade.executeNonQuery("INSERT INTO
`Frequent_Sequence` VALUES ('" + result[0] + "', " + value.ToString() + ",
" + k.ToString() + ");");

                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
                else if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) > 0)
                {
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[0] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }

                ////
                value = dbSpade.executeScalar("SELECT
COUNT(DISTINCT(`TMP`.`SID`)) " +
                                            "FROM `" + s1 + "`, `" + s2 +
"` AS `TMP` " +
                                            "WHERE `" + s1 + "`.`SID` =
`TMP`.`SID` AND `" + s1 + "`.`EID` < `TMP`.`EID`;");
                count = dbSpade.executeScalar("SELECT COUNT(`Sequence`)
```

```
FROM `Frequent_Sequence` WHERE `Sequence` = '" + result[1] + "';");
                if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) == 0)
                {
                    dbSpade.executeNonQuery("CREATE TABLE IF NOT EXISTS `"
+ result[1] + "` (`SID` char(16), `EID` int(2));" +
                                            "TRUNCATE TABLE `" + result[1]
+ "`;" +
                                            "INSERT INTO `" + result[1] +
"` SELECT `TMP`.`SID`, `TMP`.`EID` " +
                                            "FROM `" + s1 + "`, `" + s2 +
"` AS `TMP` " +
                                            "WHERE `" + s1 + "`.`SID` =
`TMP`.`SID` AND `" + s1 + "`.`EID` < `TMP`.`EID`;");
                    dbSpade.executeNonQuery("INSERT INTO
`Frequent_Sequence` VALUES ('" + result[1] + "', " + value.ToString() + ",
" + k.ToString() + ");");

                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[1] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[1] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
                else if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) > 0)
                {
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[1] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[1] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }

                ////
                value = dbSpade.executeScalar("SELECT
COUNT(DISTINCT(`TMP`.`SID`)) " +
                                            "FROM `" + s2 + "`, `" + s1 +
"` AS `TMP` " +
                                            "WHERE `" + s2 + "`.`SID` =
`TMP`.`SID` AND `" + s2 + "`.`EID` < `TMP`.`EID`;");
                count = dbSpade.executeScalar("SELECT COUNT(`Sequence`)
FROM `Frequent_Sequence` WHERE `Sequence` = '" + result[2] + "';");
                if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) == 0)
                {
                    dbSpade.executeNonQuery("CREATE TABLE IF NOT EXISTS `"
+ result[2] + "` (`SID` char(16), `EID` int(2));" +
                                            "TRUNCATE TABLE `" + result[2]
```

```
+ "`;" +
                                                "INSERT INTO `" + result[2] +
"` SELECT `TMP`.`SID`, `TMP`.`EID` " +
                                                "FROM `" + s2 + "`, `" + s1 +
"` AS `TMP` " +
                                                "WHERE `" + s2 + "`.`SID` =
`TMP`.`SID` AND `" + s2 + "`.`EID` < `TMP`.`EID`;");
                    dbSpade.executeNonQuery("INSERT INTO
`Frequent_Sequence` VALUES ('" + result[2] + "', " + value.ToString() + ",
" + k.ToString() + ");");

                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[2] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[2] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
                else if (Int32.Parse(value.ToString()) >= min_sup &&
Int32.Parse(count.ToString()) == 0)
                {
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s1 + ") - (" + result[2] + ")'," +
float.Parse(value.ToString()) / f1 * 100 + ", " +
(float.Parse(value.ToString()) / f1) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                    dbSpade.executeNonQuery("INSERT IGNORE INTO
`Sequence_Rules` VALUES ('(" + s2 + ") - (" + result[2] + ")'," +
float.Parse(value.ToString()) / f2 * 100 + ", " +
(float.Parse(value.ToString()) / f2) / (float.Parse(value.ToString()) /
all_support) / 100 + ", " + k.ToString() + ");");
                }
            }
        }
```

Sourcecode 4.3 Fungsi Join2sequence

4.2.1.3 Frequent 2-sequence

Fungsi ini menampilan frequent 2-sequence dengan memanggil fungsi join2sequence, dengan menginputkan nilai k = 2, dimana dalam hal ini 2 menunjukkan panjang sequence yang dipanggil. Sourcecode untuk fungsi ini dapat dilihat pada Sourcecode 4.4.

Frequent 2-sequence

```
public void createTable2Sequence(int min_sup)
        {
            dbSpade.executeNonQuery("DELETE FROM `Frequent_Sequence` WHERE
k = 2;");
```

```
            dbSpade.executeNonQuery("DELETE FROM `Sequence_Rules` WHERE k =
2;");
            List<string>[] curSeq = dbSpade.getReader("SELECT `Sequence`,
`Frequent` FROM `Frequent_Sequence` "+
                                    "WHERE k = 1 AND `Frequent` >= " + min_sup
+ ";");

            for (int i = 0; i < curSeq[0].Count-1; i++)
            {
                for (int j = i+1; j < curSeq[0].Count; j++)
                {
                    join2Sequence(curSeq[0][i], float.Parse(curSeq[1][i]),
curSeq[0][j], float.Parse(curSeq[1][j]), min_sup, 2);
                }
            }
        }
```

Sourcecode 4.4 Fungsi Frequent 2-sequence

4.2.1.3 Frequent k-sequence

Fungsi ini menampilkan semua frequent k-sequence dengan memanggil fungsi join2sequence. Perbedaanya pada fungsi ini nilai k tidak ditentukan. Sehingga sequence akan dicari hingga sequence maksimal dimana tidak ada sequence lagi yang dapat dibentuk. Sourcecode untuk fungsi ini dapat dilihat pada Sourcecode 4.5

Frequent k-sequence

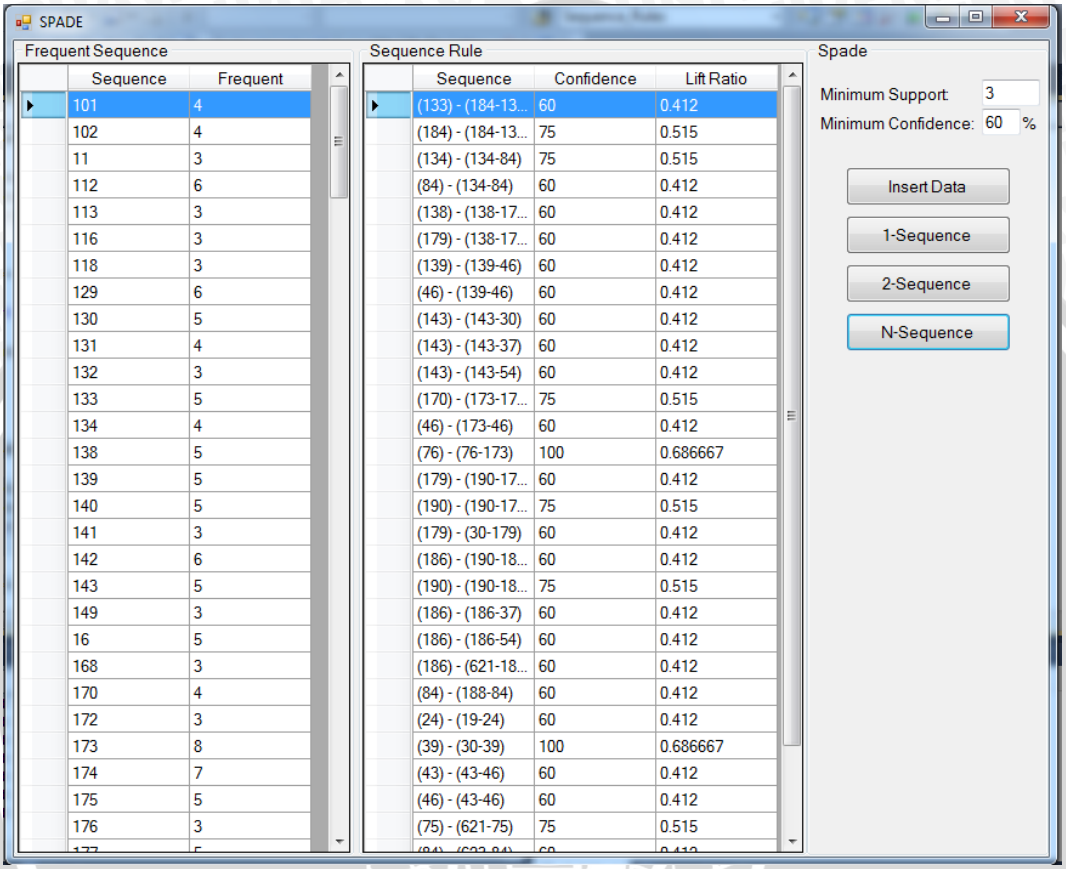
```
    public void createTableNSequence(int k, int min_sup)
        {
            dbSpade.executeNonQuery("DELETE FROM `Frequent_Sequence`
WHERE k = " + k.ToString() + ";");
            dbSpade.executeNonQuery("DELETE FROM `Sequence_Rules` WHERE
k = " + k.ToString() + ";");
            List<string>[] curSeq = dbSpade.getReader("SELECT
`Sequence`, `Frequent` FROM `Frequent_Sequence` " +
                                    "WHERE k = " + (k - 1).ToString() + "
AND `Frequent` >= " + min_sup + ";");

            for (int i = 0; i < curSeq[0].Count; i++)
            {
                for (int j = 0; j < curSeq[0].Count; j++)
                {
                    if (i != j) join2Sequence(curSeq[0][i],
float.Parse(curSeq[1][i]), curSeq[0][j], float.Parse(curSeq[1][j]),
min_sup, k);
                }
            }
        }
```

.Sourcecode 4.5 Frequent k-sequence

4.3 Implementasi Antarmuka

Form utama sistem ini terdiri dari minimum support dan minimum confidence yang diinputkan oleh user, serta insert data yang dipilih user. Tampilan sistem ini dapat dilihat pada Gambar 4.1 berikut.



Gambar 4.1 Antarmuka Sistem

Pada Gambar 4.1 setelah menentukan minimum support dan minimum confidence serta melakukan insert data. Button ditekan berurutan mulai 1-sequence, 2-sequence kemudian n-sequence. Setiap button yang ditekan akan menampilkan semua frequent sequence dan sequence rule yang ditemukan.