**SYSTEM REQUIREMENTS SPECIFICATION DOCUMENT**

Document Number: BBT 3104-SyRS-2020/04

Version: 1.0

Review Date:

Submission Date:

Case Study: Hannah Clifford’s Nairobi Garage, off Ngong Road

Group Number: 4

Team Members:

|  |  |  |
| --- | --- | --- |
|  | **Student Details** | **Mark Awarded** |
|  | Samuel Mariwa – 106221 – Group A – SU Email: samuel.mariwa@strathmore.edu |  |
|  | Angela Mate – 110201 – Group A – SU Email: angela.mate@strathmore.edu |  |
|  | Nelly Ongwenyi – 106221 – Group A – SU Email: nelly.ongwenyi@strathmore.edu |  |

|  |  |  |
| --- | --- | --- |
| **Version Change** | **Date of Change** | **Changes Made** |
| Version 1.0 to 1.2 | 20th April 2020 | Added system objectives |
| Version 1.2 to 1.6 | 29th April 2020 | Added flowchart and DFD diagram |
| Version 1.6 to 3.0 | 15th May 2020 | Added system capabilities, conditions and constraints |
| Version 3.0 to 3.1 | 20th May 2020 | Added assumptions |
| Version 3.2 to 4.0 | 9th June 2020 | Added ERD diagram, database schema and system architecture |
| Version 4.0 to 4.2 | 2nd July 2020 | Added functional dependencies and normal forms |
| Version 4.2 to 4.5 | 13th May 2020 | Added DDL statements |

# Executive Summary

Nairobi Garage is an organization that provides co-working space for small businesses. This is done by giving allowing small companies to use the desks alongside other facilities that they need to run their operations. This small organizations can do their operations without having to pay so much for their needs for example rent for an office, electricity bills, water bills and internet connectivity bills. Our goal is to a computerized way they do their business to help them be a more effective company. By analyzing their work process, we will be able to make a structure of a database which will improve their daily process and record keeping. The database will utilize all the data and organize it in a way that will be easy to use and record. It will also allow data that is shared by different department to be share and record with easy.

**Table of Contents**

[Executive Summary 2](#_Toc45495440)

[List of Figures 5](#_Toc45495441)

[Chapter 1. Introduction 6](#_Toc45495442)

[1.1 The Mission of the Business 6](#_Toc45495443)

[1.2 The Day to Day Operations of the Business 6](#_Toc45495444)

[1.3 Statement of Objectives 8](#_Toc45495446)

[Chapter 2. System Capabilities 9](#_Toc45495447)

[A. Client Data 9](#_Toc45495448)

[B. Branch Data 9](#_Toc45495449)

[C. Support Data 9](#_Toc45495450)

[D. Operation Key Data 9](#_Toc45495451)

[E. Office Data 10](#_Toc45495452)

[F. Employee Data 10](#_Toc45495453)

[Chapter 3. System Conditions 11](#_Toc45495454)

[A. Client Data 11](#_Toc45495455)

[B. Branch Data 11](#_Toc45495456)

[C. Support Data 11](#_Toc45495457)

[D. Operation Key Data 11](#_Toc45495458)

[E. Office Data 11](#_Toc45495459)

[F. Employee Data 12](#_Toc45495460)

[Chapter 4. System Constraints 13](#_Toc45495461)

[A. Client Data 13](#_Toc45495462)

[B. Branch Data 13](#_Toc45495463)

[C. Support Data 13](#_Toc45495464)

[D. Operation Key Data 14](#_Toc45495465)

[E. Office Data 14](#_Toc45495466)

[F. Employee Data 14](#_Toc45495467)

[Chapter 5. Assumptions 15](#_Toc45495468)

[Chapter 6. Database Design 16](#_Toc45495469)

[a. Entity Relationship Diagram Based on the Chen Notation 16](#_Toc45495470)

[b. Database Schema Based on the UML Notation 17](#_Toc45495471)

[c. System Architecture 18](#_Toc45495472)

[Chapter 7. Normalization 19](#_Toc45495473)

[a. List of Database Dependencies before Normalization 19](#_Toc45495474)

[b. Normal Forms 20](#_Toc45495475)

[Chapter 8. Advanced Database Objects 23](#_Toc45495476)

[a. Storage Engines 23](#_Toc45495477)

[b. Indexes 23](#_Toc45495478)

[c. Normal Triggers 23](#_Toc45495479)

[d. Temporal Triggers 23](#_Toc45495480)

[e. Procedures 23](#_Toc45495481)

[f. Functions 24](#_Toc45495482)

[g. Views 24](#_Toc45495483)

[Chapter 9. DDL Statements 25](#_Toc45495484)

# List of Figures

[Figure 1.1 Flowchart 7](#_Toc45492211)

[Figure 1.2 Data Flow Concept Diagram 8](#_Toc45492212)

[Figure 1.1 Data Flow Diagram Level 1 8](#_Toc45492213)

[Figure 6.1 Entity Relationship Diagram 17](#_Toc45492214)

[Figure 6.2 UML Notation 18](#_Toc45492215)

[Figure 6.3 System Architecture 19](#_Toc45492216)

# Introduction

## The Mission of the Business

Nairobi Garage is an organization that provides co-working space for small businesses. This is done by giving allowing small companies to use the desks alongside other facilities that they need to run their operations. Their mission is therefore ensuring that young organizations can do their operations without having to pay so much for their needs for example rent for an office, electricity bills, water bills, internet connectivity bills etc.

## The Day to Day Operations of the Business

A screenshot of a cell phone

Description automatically generated

Figure 1.1 Flowchart

A close up of a logo

Description automatically generated

Figure 1.2 Data Flow Concept Diagram

# 

Figure 1.1 Data Flow Diagram Level 1

## Statement of Objectives

1. To have a computerized way of customers asking for help. The alerts are stored in a database based on the time of alert such that it is a first come first serve based help service. This is contrary to what they have, people who have to be walking around checking out for people who need help with something. This is all because the process of the staff walking all round looking at times in areas of the office where no help is needed and not looking in areas that help is needed is a non-value adding activity.
2. To have a system in the organization for clients where the clients are given keys that expire monthly but are renewed on payment. This system should be used for the daily interactivity of the clients with the office. Activities like booking an office can be done online. In the case of booking an office, the system automatically assigns the clients an office based on the number of people in the meeting, time of the meeting etc. This will make the processes that the clients much faster and efficient. This will reduce the urge of the clients having to physically interact with the managers of the business which is a non-value adding activity.
3. To have a joined client database around all branches of Nairobi Garage such that anybody who has membership with the organization can work in any branch of Nairobi Garage for their convenience. This will reduce the urge of clients having to go to the actual location of the branch that they registered in. This could be considered a non-value adding activity since the small business can operate at any location of their convenience.

# System Capabilities

## Client Data

The database should be capable of storing data about the clients and for each client, the following data should be stored:

|  |  |  |
| --- | --- | --- |
| **1.** | A client’s identification number |  |
| **2.** | The client’s key expiration date |  |
| **3.** | The client’s name/ company |  |
| **4.** | Client’s operation key number |  |
| **5.** | The client’s membership number |  |
| **6.** | Client’s contacts |  |

## Branch Data

The database should be capable of storing data about the rental properties and for each rental property, the following data should be stored:

|  |  |  |
| --- | --- | --- |
| **1.** | The branch ID |  |
| **2.** | Branch name |  |
| **3.** | Branch manager |  |
| **4.** | The branch’s client capacity |  |
| **5.** | Branch contact |  |
| **6.** | The number of employees |  |

## Support Data

The database should be capable of storing data about the support provided and for each support task, the following data should be stored:

|  |  |  |
| --- | --- | --- |
| **1.** | The support ID |  |
| **2.** | The client’s ID |  |
| **3.** | Branch ID |  |
| **4.** | The client’s issue |  |
| **5.** | Was the issue resolved or not |  |
| **6.** | Issued timestamp |  |

## 

## Operation Key Data

The database should be capable of storing data about the operation key and for each key, the following data should be stored:

|  |  |
| --- | --- |
| **1.** | Key Number |
| **2.** | Key’s status: active or inactive |
| **3.** | Operation key expiry date |

## Office Data

The database should be capable of storing data about the office details and for each office, the following data should be stored:

|  |  |
| --- | --- |
| 1. | Office ID |
| 2. | Branch ID |
| 3. | Office Capacity |
|  | Office contact |

## Employee Data

The database should be capable of storing data about the employees and for each employee, the following data should be stored:

|  |  |
| --- | --- |
| 1. | Employee ID |
| 2. | Branch ID |
| 3. | Employee name |
| 4. | Employee contact |
| 5. | Employee salary |
| 6. | Employee shift |

# System Conditions

## Client Data

The database should contain the following measurable characteristics for each capability specified in Chapter 2.

|  |  |
| --- | --- |
| **2.A.1.** | A client identification number shall be a national identity number or a company PIN. If the clients are in a group, then 1 of them shall take the lead ­role and make the reservation on behalf of the whole group. Therefore, only 1 national identity number or 1 company PIN shall be recorded per reservation. |
| **2.A.2.** | A client’s key expiration date shall be a date stating the end of the client’s usage of the office space. |
| **2.A.3.** | A client’s name or company shall be a first and last name. If the client registers as a company, it shall be the company’s full name. |
| **2.A.4.** | A client’s key operation number shall be the registration number of the key. |
| **2.A.5.** | A client’s membership number shall be a number |
| **2.A.6.** | A client’s contacts shall include the client’s email and telephone number(s) |

## Branch Data

|  |  |
| --- | --- |
| **2.B.1.** | The branch ID shall be a number. |
| **2.B.2.** | A branch name shall include the branch name and the location of the branch. |
| **2.B.3.** | A branch manager shall include the manager’s first and last name only. |
| **2..B.4.** | The branch’s client capacity shall list the maximum capacity the branch can accommodate at one time. |
| **2.B.5.** | A branch’s contacts shall include the branch’s email and telephone number(s). |
| **2.B.6.** | The number of employees shall include the total number of staff within the branch. |

## Support Data

|  |  |
| --- | --- |
| **2.C.1.** | The support ID shall be a number. |
| **2.C.2.** | The client’s ID shall be a number. |
| **2.C.3.** | A branch’s ID shall be a number. |
| **2.C.4.** | The client’s name shall include a first and last name. |
| **2.C.5.** | An issue resolvent shall be a yes or no Boolean. |
| **2.C.6.** | An issued timestamp shall be the date and time of the issue resolvent. |

## Operation Key Data

|  |  |
| --- | --- |
| **2.D.1.** | The key’s ID shall be a number. |
| **2.D.2.** | A key’s issue date shall be the date. |
| **2.D.3.** | A key’s status shall be a yes or no Boolean. |

## Office Data

|  |  |
| --- | --- |
| **2.E.1.** | An office ID shall be a number. |
| **2.E.2.** | A branch’s ID shall be a number. |
| **2.E.3.** | The office capacity shall include the total number of clients an office can accommodate at one time.. |
| **2.E.4.** | The office contact shall include the branch’s email and telephone number(s). |

## Employee Data

|  |  |
| --- | --- |
| **2.F.1.** | An employee’s ID shall be a number. |
| **2.F.2.** | A branch’s ID shall be a number. |
| **2.F.3.** | An employee’s name shall include a first and last name. |
| **2.F.4.** | An employee’s contacts shall include the client’s email and telephone number(s). |
| **2.F.5.** | An employee’s salary shall include the gross salary figure of an employee. |
| **2.F.6.** | An employee’s shift shall include the time of an employee’s work shift. |

# System Constraints

The database must satisfy the following constraints:

## Client Data

|  |  |
| --- | --- |
| **2.A.1.** | The client’s National Identity Number or company PIN shall together have a key constraint. This will be in the form of an interrelational, static, composite key constraint to form the primary key. |
| **2.A.2.** | The data in the client’s key expiration date must conform to the interrelational, static, semantic, DATE constraint so as to be a valid date entry. |
| **2.A.3.** | The data in the client’s name/ company must conform to the interrelational, static VARCHAR domain constraint to be in the form of a valid name. |
| **2.A.4.** | The data in the client’s operation key number shall have key constraint. It will be in the form of an interrelational, static INT domain constraint to form the foreign key. |
| **2.A.5.** | The data in the client’s membership number must conform to the intrarelational, static INT domain constraint to form a valid membership number. |
| **2.A.6.** | The data in the Client’s contacts must conform to the intrarelational, static VARCHAR domain constraint to form a valid email address. |

## Branch Data

|  |  |  |
| --- | --- | --- |
| **2.B.1.** | The branch ID shall have a key constraint. It will be in the form of an interrelational, static, composite key constraint to form the primary key. |  |
| **2.B.2.** | The data in the branch name must conform to the intrarelational, static VARCHAR domain constraint to be in the form of a valid name. |  |
| **2.B.3.** | The data in the branch manager must conform to the intrarelational, static VARCHAR domain constraint to be in the form of a valid name. |  |
| **2.B.4.** | The data in the branch’s client capacity must conform to the interrelational, static INT domain constraint to be in the form of a valid capacity. |  |
| **2.B.5.** | The data in the branch contact must conform to the intrarelational, static VARCHAR domain constraint to be in the form of a valid email address. |  |
| **2.B.6.** | The data in the number of employees must conform to the intrarelational, static INT domain constraint to be in the form of a valid employee number. |  |

## Support Data

|  |  |  |
| --- | --- | --- |
| **2.C.1.** | The support ID shall have a key constraint. It will be in the form of an intrarelational, static, composite key constraint to form the primary key. |  |
| **2.C.2.** | The client’s ID shall have a key constraint. It will be in the form of an interrelational, static INT domain constraint to form the foreign key. |  |
| **2.C.3.** | Branch ID shall have a key constraint. It will be in the form of an interrelational, static INT domain constraint to form the foreign key. |  |
| **2.C.4.** | The client’s issue must conform to the interrelational, static VARCHAR domain constraint to be in the form of a valid issue. |  |
| **2.C.5.** | Was the issue resolved or not must conform to the intrarelational, static BOOLEAN domain constraint to be in the form of a valid option on whether the issue was resolved or not. |  |
| **2.C.6.** | Issued timestamp must conform to the interrelational, static, semantic, DATE constraint so as to be a valid date entry. |  |

## Operation Key Data

|  |  |  |
| --- | --- | --- |
| **2.D.1.** | The key’s ID shall have a key constraint. It will be in the form of an interrelational, static, composite key constraint to form the primary key. |  |
| **2.D.2.** | Date the key was issued must conform to the interrelational, static, semantic, DATE constraint so as to be a valid date entry. |  |
| **2.D.3.** | Key’s status: active or inactive must conform to the intrarelational, static BOOLEAN domain constraint to be in the form of a valid option on whether the key is active or inactive. |  |

## Office Data

|  |  |  |
| --- | --- | --- |
| **2.E.1.** | Office ID shall have a key constraint. It will be in the form of an intrarelational, static, composite key constraint to form the primary key. |  |
| **2.E.2.** | Branch ID shall have a key constraint. It will be in the form of an interrelational, static, composite key constraint to form the foreign key. |  |
| **2.E.3.** | Office Capacity must conform to the interrelational, static, semantic, INT constraint so as to be a valid capacity entry. |  |
| **2.E.4.** | Office contact must conform to the intrarelational, static VARCHAR domain constraint to form a valid email address. |  |

## Employee Data

|  |  |  |
| --- | --- | --- |
| **2.F.1.** | Employee ID shall have a key constraint. It will be in the form of an intrarelational, static, composite key constraint to form the primary key. |  |
| **2.F.2.** | Branch ID shall have a key constraint. It will be in the form of an interrelational, static, composite key constraint to form the foreign key. |  |
| **2.F.3.** | Employee name must conform to the intrarelational, static, VARCHAR domain constraint so as to be a valid name. |  |
| **2.F.4.** | Employee contact must conform to the intrarelational, static VARCHAR domain constraint to form a valid email address. |  |
| **2.F.5.** | Employee salary must conform to the intrarelational, static, semantic INT domain constraint to form a valid salary entry. |  |
| **2.F.6.** | Employee shift must conform to the intrarelational, static, semantic DATE domain constraint to form a valid shift entry. |  |

# Assumptions

The Nairobi Garage might or might not have branches. However, we have designed the database, to cater for the availability of branches or future expansion of the Nairobi Garage.

Another assumption is that we have a steady inflow of clients all through year and abundant amount of workspace. This is because there might be lack of workspace if the clients occupy the workspace of a long time without leaving. This will lead to a low inflow of clients. We have made a way to work around that by allocating an operation key expire date. This will prevent clients from staying for too long.

We are working under the assumption that each Nairobi Garage’s workspace has a limited capacity that can hold a limited number of clients at once. We have allocated a way to record the number of the client employee and office capacity to avoid any inconvenience and to allow better planning.

We also under the assumption that Nairobi Garage is a 24-hour business which allows it to provide its services despite the hour. We allocated employ shifts to keep track of what time the works come and go for their shifts. This allows the branch managers to keep track of the employees at a specific shift.

# Database Design

## Entity Relationship Diagram Based on the Chen Notation

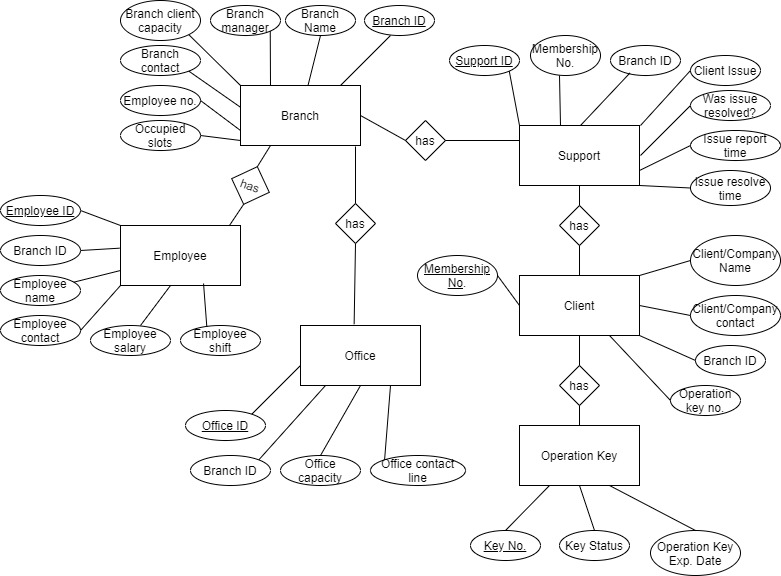


Figure 6.1 Entity Relationship Diagram

## Database Schema Based on the UML Notation

A screenshot of a cell phone

Description automatically generated

Figure 6.2 UML Notation

## System Architecture

Branch Manager PC

Database

Branch Client PC

Branch Client PC

Branch Client PC

Branch Manager PC

Branch Y Server

Branch X Server

Overall Manager PC

Business Logic (Server)

Figure 6.3 System Architecture

# Normalization

## List of Database Dependencies before Normalization

**Full Dependencies**

**Employee Relation**

{Employee ID} 🡪 {Branch ID, Employee Name, Employee Contact, Employee Salary, Employee shift}

**Office Relation**

{Office ID, Branch ID} 🡪 {Office Capacity, Office Contact}

**Key Relation**

{Key Number} 🡪 {Key Status}

**Transitive Dependencies**

**Client Support Relation**

{Support ID} 🡪 {Client ID, Client Issue, Branch ID} 🡪 {Was issue resolved or not} 🡪 {Issue Timestamp}

The fourth dependency was formed on the basis that the attribute ‘was issues resolved or not’ determines the timestamp ‘updated’ since the very time the issue is resolved or not is the time the ‘update timestamp ’ changes thus the dependency.

**Partial Dependencies**

**Branch Relation**

{Branch ID, Branch Name} 🡪 {Branch Manager, Branch Client Capacity, Branch Contact, Branch Number of Employees}

**Multi-valued Dependencies**

**Client Relation**

{Client ID, Client Name, Client Membership} 🡪 {Clients Key Number, Clients Key Expiration Date}

{Client ID, Client Name, Client Membership} 🡪 {Client Contact}

This dependency is formed on the basis that a client’s contact can change at any given time thus can’t be used as a primary key of the relation. The Client ID, Client Name and Client Membership separately determine the two sets of attributes above that have no relation to one another.

## Normal Forms

**UNF**

We will remove the repeating groups so that we can make it 1NF. Here repeating groups will apply in ‘employee’ and ‘client’ relations.

EMPLOYEE

An employee can have more than one contact and we do not want one cell in the relation to have more than one value. We will thus separate the ‘Employee contacts’ from the ‘Employees table’. ‘Employee contacts’ will be linked to the ‘Employees Details’ with ‘Employee ID’ being a foreign key in the ‘Employees Contacts’ relation.

CLIENT

A Client can have more than one contacts and we will thus separate the contacts to have ‘Clients contacts’ table and ‘Clients’ table. ‘Client contacts’ will be linked to the ‘Client Details’ with ‘Membership No.’ being a foreign key in the ‘Client Contacts’ relation.

**1NF**

We need to remove the partial dependencies to move to the 2NF.

BRANCH

We will remove the branch id and make the branch name a primary key since it's also a good candidate key to make the primary key. After that we will have 2 partial dependencies ‘Branch Name’ and ‘Branch Contact’. But there's one missing detail, ‘Branch Location’ which is also a determinant that can stand alone. To move to first NF we will split the 3 partial dependencies and put them in their own separate tables and link them to the determinants.

The three tables will be linked by having ‘Branch Name’(New Primary Key) as the foreign key in ‘Branch Location’ and ‘Branch Contacts’ relations.

Resultant tables -  Branch Contacts, Branch Locations and Branch Details.

‘Branch Name’ will be retained under branch details together with the other details.

NB: We had an agreement that Branch Manager is a foreign key from ‘employees’ relation so that we can have full details of the branch manager instead of the name only. We will add ‘Slots Occupied’ in the Branch Relation.

EMPLOYEE

Here we will replace ‘Branch Id’ with ‘Branch Name’ as a foreign key since we removed ‘Branch Id’.

Since we already separated the ‘employees contacts’ from the ‘employees details’ we have no partial dependencies hence will still have ‘employee details’ and ‘employee contacts’ relations.

OFFICE

Again we will replace ‘Branch id’ with ‘Branch Name’ as a foreign key. Here we have 2 partial dependencies, ‘Office Id’ and ‘Office contact lines’. This will result in 2 tables, ‘Office Contact Lines’ and  ‘Office Details' under which ‘Office Id’ falls.

SUPPORT

Support table will remain as it is since it has no partial dependencies but the ‘Branch Id’ will change to ‘Branch Name’ and ‘client id’ will change to ‘membership number’ as foreign key as will be indicated in the next relation below.

CLIENT

We will get rid of the ‘Client Id’ and make the ‘membership number’ the primary key. This will result in 2 partial dependencies ‘Membership number’ and ‘Client Contact’. Since we already separated the ‘client contacts’ from the ‘client details’ we do not have any other partial dependencies thus will just remain with the ‘client details’ and ‘client contacts’ relations.We will add ‘Branch Name’ as a foreign Key in the ‘Client Details’ relation.

OPERATION KEY

Here we will remove the ‘Key Id’ and let the ‘Key No.’ be the primary key in the table. This will make it have only that one dependency, the ‘Key No.’. The table will therefore remain as is.

Tables - Branch Contacts, Branch Locations, Branch Details,Employees Contacts, Employees Details,Office Contact Lines, Office Details,Support,Client Contacts, Client Details, Operations Keys.

**2 NF**

Here we need to remove the transitive dependencies. We do not have any transitive dependencies thus our relations are already in 3NF.

**3 NF**

Ensure every determinant is a candidate key.  For our cases, all the determinants are candidate keys.

EMPLOYEE DETAILS

Determinant(s)- Employee ID

EMPLOYEE CONTACTS

Determinant(s)- Employee Contact

CLIENTDETAILS  
Determinant(s)- Membership Number

CLIENT CONTACTS

Determinant(s)- Client Contact

BRANCH CONTACTS

Determinant(s)- Branch Contact

BRANCH LOCATIONS

Determinant(s)- Branch Location

BRANCH DETAILS

Determinant(s)- Branch Name

OFFICE DETAILS

Determinant(s)- Office ID

OFFICE CONTACT LINES

Determinant(s)- Office Contact Line

SUPPORT

Determinant(s)-Support ID

OPERATION KEYS

Determinant(s)- Key Number.

The above determinants are all candidate keys in their respective relations.

**BOYCE-CODDE NF**

Here we ensure all multi-determinant in a multi-valued dependency is a candidate key.

We do not have any multivalued dependency in our relations thus move to the 4NF.

**4 NF**

Ensure each projection in a join dependency includes a candidate key of the original relation.

Our relations satisfy the above condition.

Our join dependencies include:

EMPLOYEE DETAILS-EMPLOYEE CONTACTS

‘Employee ID’ is the foreign key in ‘Employee Contacts’ relation and a candidate key in the ‘Employee Details’ relation which is the original relation.

CLIENT DETAILS-CLIENT CONTACTS

‘Membership Number’ is the foreign key in ‘Client Contacts’ relation and a candidate key in the ‘Client Details’ relation which is the original relation.

BRANCH DETAILS-BRANCH CONTACTS

‘Branch Name’ is the foreign key in ‘Branch Contacts’ relation and a candidate key in the ‘Branch Details’ relation which is the original relation.

BRANCH DETAILS-BRANCH LOCATIONS

‘Branch Location’ is the foreign key in ‘Branch Details’ relation and a candidate key in the ‘Branch Locations’ relation which is the original relation.

EMPLOYEE DETAILS-BRANCH DETAILS

‘Manager ID’ is the foreign key in ‘Branch Details’ relation and a candidate key in the ‘Employee Details’ relation which is the original relation.

OFFICE DETAILS-BRANCH DETAILS

‘Branch Name’ is the foreign key in ‘Office Details’ relation and a candidate key in the ‘Branch Details’ relation which is the original relation.

CLIENT DETAILS-SUPPORT

‘Membership Number’ is the foreign key in ‘Support’ relation and a candidate key in the ‘Client Details’ relation which is the original relation.

CLIENT DETAILS-OPERATION KEYS

‘Key No’ is the foreign key in ‘Client Details’ relation and a candidate key in the ‘Operation Keys’ relation which is the original relation.

SUPPORT-BRANCH DETAILS

‘Branch Name’ is the foreign key in ‘Support’ relation and a candidate key in the ‘Branch Details’ relation which is the original relation.

Given the above has been satisfied, our relations are in the **5 NF**.

# Advanced Database Objects

## Storage Engines

Branch – InnoDB

Support – InnoDB

Client – InnoDB

Operation key – InnoDB

Office – InnoDB

Employee – InnoDB

## Indexes

Branch – secondary index

Support – primary index

Client – primary index

Operation key – primary index

Office – primary index

Employee – primary index

## Normal Triggers

We will have a normal trigger that will reduce the number of slots occupied on Updating the ‘Client Relation’ to add the ‘Branch Name’ and ‘Operation Key’ once they have made their monthly payment to work in the office space.

## Temporal Triggers

Under the client relation there will be a temporal trigger in operation key expiration date whereby the database will automatically check every day at midnight if the expiration date of the key equates to the current date to make the key inactive:

Select count(\*) from ‘client’ inner join ‘Branch’ on client.Branch Name = Branch.Branch Name where ‘Branch.Branch Name’ = ‘input’ and ‘client.Operation Key Exp. Date’ <= Date(Now()-30);

## Procedures

The procedures that will be used to retrieve data required for reports include:

* The number support issues that have been raised and noting how many were resolved and how many unresolved
* The number of clients that have worked in each branch

## Functions

The functions that will be used to retrieve data required for reports include:

* The number support issues that have been raised and noting how many were resolved and how many unresolved
* The number of clients that have worked in each branch

## Views

The views that will be used to provide various displays of the data for different users are the support issues that were resolved and those unresolved.

# DDL Statements

**DDL For creating the table ‘Branch Contacts’ and its respective index**

CREATE TABLE `branch contacts` (

`Branch Contact` varchar(200) NOT NULL DEFAULT '',

`Branch Id` bigint(11) unsigned DEFAULT NULL,

PRIMARY KEY (`Branch Contact`),

KEY `Branch Id Foreign` (`Branch Id`),

KEY `Branch Contact Index` (`Branch Contact`),

CONSTRAINT `Branch Id Foreign` FOREIGN KEY (`Branch Id`) REFERENCES `branch details` (`Branch Id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Branch Details’ and its respective index**

CREATE TABLE `branch details` (

`Branch Id` bigint(50) unsigned NOT NULL,

`Branch Name` varchar(255) NOT NULL,

`Branch Manager` bigint(20) unsigned NOT NULL,

`Branch Client Capacity` int(11) NOT NULL,

`Occupied Slots` int(11) NOT NULL,

PRIMARY KEY (`Branch Id`),

KEY `Branch Manger Foreign` (`Branch Manager`),

KEY `Branch Id` (`Branch Id`),

CONSTRAINT `Branch Manager Foreign` FOREIGN KEY (`Branch Manager`) REFERENCES `employee details` (`Employee Id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Branch Locations’ and its respective index**

CREATE TABLE `branch locations` (

`Branch Location` varchar(100) NOT NULL DEFAULT '',

`Branch Id` bigint(20) unsigned DEFAULT NULL,

PRIMARY KEY (`Branch Location`),

KEY `Branch Id Foreign5` (`Branch Id`),

KEY `Branch Location` (`Branch Location`),

CONSTRAINT `Branch Id Foreign5` FOREIGN KEY (`Branch Id`) REFERENCES `branch details` (`Branch Id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Client Contacts’ and its respective index**

CREATE TABLE `client contacts` (

`Client Contact` varchar(11) NOT NULL DEFAULT '',

`Client membership` bigint(20) unsigned NOT NULL,

PRIMARY KEY (`Client Contact`),

KEY `Client Membership Foreign` (`Client membership`),

KEY `Client Contact Index` (`Client Contact`),

CONSTRAINT `Client Membership Foreign` FOREIGN KEY (`Client membership`) REFERENCES `client details` (`Membership No.`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Client Details’ and its respective index**

CREATE TABLE `client details` (

`Membership No.` bigint(11) unsigned NOT NULL AUTO\_INCREMENT,

`Branch Id` bigint(11) unsigned NOT NULL,

`Operation Key` bigint(20) unsigned NOT NULL,

`Client Name` varchar(255) NOT NULL DEFAULT '',

PRIMARY KEY (`Membership No.`),

KEY `Operation Key Foreign` (`Operation Key`),

KEY `Branch Id Foreign3` (`Branch Id`),

KEY `Membership No. Index` (`Membership No.`),

CONSTRAINT `Branch Id Foreign3` FOREIGN KEY (`Branch Id`) REFERENCES `branch details` (`Branch Id`) ON DELETE CASCADE,

CONSTRAINT `Operation Key Foreign` FOREIGN KEY (`Operation Key`) REFERENCES `operation key` (`Key No.`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Employees Contacts’ and its respective index**

CREATE TABLE `employee contacts` (

`Employee Contact` varchar(14) NOT NULL DEFAULT '',

`Employee Id` bigint(11) unsigned DEFAULT NULL,

PRIMARY KEY (`Employee Contact`),

KEY `Employee Id Foreign` (`Employee Id`),

KEY `Employee Contact Index` (`Employee Contact`),

CONSTRAINT `Employee Id Foreign` FOREIGN KEY (`Employee Id`) REFERENCES `employee details` (`Employee Id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Employees Details’ and its respective index**

CREATE TABLE `employee details` (

`Employee Id` bigint(11) unsigned NOT NULL AUTO\_INCREMENT,

`Branch Id` bigint(100) unsigned NOT NULL,

`Employee Name` text NOT NULL,

`Employee Salary` double NOT NULL,

`Employee Shift` varchar(200) NOT NULL DEFAULT '',

PRIMARY KEY (`Employee Id`),

KEY `Branch Id Foreign2` (`Branch Id`),

KEY `Employee Id Index` (`Employee Id`),

CONSTRAINT `Branch Id Foreign2` FOREIGN KEY (`Branch Id`) REFERENCES `branch details` (`Branch Id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Office Contact Lines’ and its respective index**

CREATE TABLE `office contact lines` (

`Contact Line` varchar(11) NOT NULL DEFAULT '',

`Office Id` bigint(20) unsigned DEFAULT NULL,

PRIMARY KEY (`Contact Line`),

KEY `Office Id Foreign` (`Office Id`),

KEY `Contact Line Index` (`Contact Line`),

CONSTRAINT `Office Id Foreign` FOREIGN KEY (`Office Id`) REFERENCES `office details` (`Office Id`) ON DELETE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

DDL For creating the table ‘Office Details’ and its respective index

CREATE TABLE `office details` (

`Office Id` bigint(11) unsigned NOT NULL AUTO\_INCREMENT,

`Branch Name` varchar(200) DEFAULT NULL,

`Office Capacity` int(11) DEFAULT NULL,

PRIMARY KEY (`Office Id`),

KEY `Office Id Index` (`Office Id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Operation Key’ and its respective index**

CREATE TABLE `operation key` (

`Key No.` bigint(11) unsigned NOT NULL AUTO\_INCREMENT,

`Key Status` tinyint(1) NOT NULL DEFAULT 0,

`Expiry Date` date NOT NULL,

PRIMARY KEY (`Key No.`),

KEY `Key No Index` (`Key No.`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

**DDL For creating the table ‘Support’ and its respective index**

CREATE TABLE `support` (

`Support Id` bigint(11) unsigned NOT NULL AUTO\_INCREMENT,

`Client Membership` bigint(20) unsigned NOT NULL,

`Branch Id` bigint(11) unsigned NOT NULL,

`Client Issue` varchar(500) NOT NULL DEFAULT '',

`Issue Resolved?` tinyint(1) NOT NULL DEFAULT 0,

`Report Time` timestamp NULL DEFAULT current\_timestamp(),

`Resolve Time` timestamp NULL DEFAULT NULL,

PRIMARY KEY (`Support Id`),

KEY `Client Membership Foreign2` (`Client Membership`),

KEY `Branch Id Foreign4` (`Branch Id`),

KEY `Support Id Index` (`Support Id`),

CONSTRAINT `Branch Id Foreign4` FOREIGN KEY (`Branch Id`) REFERENCES `branch details` (`Branch Id`) ON DELETE CASCADE,

CONSTRAINT `Client Membership Foreign2` FOREIGN KEY (`Client Membership`) REFERENCES `client details` (`Membership No.`) ON DELETE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8mb4;

**DDL For creating the procedure**

CREATEPROCEDURE `Issue Resolve` ( IN input INT )

BEGINSELECT \* FROM Support inner join `Client Details` on `Support.Client Membership` = `Client Details.Membership No.` WHERE `support.Issue Resolved?` = @input;

END

**DDL For creating the function**

CREATE FUNCTION branchClientCount (in @input INT(5))

RETURN INT(5);

AS

BEGIN

Select count(\*) from ‘Client Details’ inner join ‘Branch Details’ on `Client Details.Branch id` = `Branch Details.Branch Id` inner join `Operation Key` on `Client Details.Operation Key`= `Operation Key.Key No.` where ‘Branch Details.Branch Id’ = @input and ‘Operation Key.Expiry Date’ <= Date(CURRENT\_DATE()-30);

END;

**DDL For creating the Temporal Trigger**

CREATE EVENT inactivate\_Key

ON SCHEDULE

EVERY 1 DAY

STARTS '2020-07-12 00:00:00'

DO

UPDATE `Operation Key`SET `Operation Key.Key Status` = '0' where DATE(`Expiry Date`) <= CURRENT\_DATE();

END;

**DDL For creating the Normal Trigger**

CREATETRIGGER TRG\_Update\_SlotsAFTER Update ON `Client Details`FOR EACH ROWUPDATE `Branch` SET Occupied\_Slots = ‘old.Occupied\_Slots-1’ where `Branch Details.Branch Id`= `Client Details.Branch Id `;

**DDL For creating the View(Resolved Issues)**

CREATE VIEW [Resolved Issues] AS

SELECT \*

FROM Support inner join `Client Details` on `Support.Client Membership` = `Client Details.Membership No.`

WHERE `support.Issue Resolved?` = "1";

**DDL For creating the View(Unresolved Issues)**

CREATE VIEW [Unresolved Issues] AS

SELECT \*

FROM Support inner join `Client Details` on `Support.Client Membership` = `Client Details.Membership No.`

WHERE `support.Issue Resolved?` = "0";