Islington College

Information System

CC4002NA

Coursework 3

Submitted By:

Rajat Shrestha

17030954

Group: L1C5

Date: 22nd mar 2018

Submitted To:

Mr. Sukrit Shakya

Module Leader

Information System

Proposal:

The goal of this section is to provide overall detail on the third coursework assigned by Information System module. The assignment was given out as an individual task in week 17 and requires to be submitted within week 20. This project involves detailed documentation and SQL dump file itself maintained to manage a database model of a remittance.

1. Purpose:

This project requires student to not only develop the database but to document the procedure. Documentation is required so that it becomes more useful for future use or for any other programmer, student or association. The documentation contains detailed information on how the database was developed including the very important relational diagram and ER diagram to represent the model of the database so that it can be easily understood.

2. Problem Statement:

This project involves a development of a database which is developed to store a remittance's data. The database contains the details about the types of account, employees, customer transactions, and the transaction must be between two account holders (Senders and receivers) and must be authorized by an employee. This procedure must be properly documented and the database should be simple and should be normalized to 3rd degree. The dump file of database should load and run properly without and errors and the process should be documented properly.

3. Objective:

The main objective is to develop a database which helps the remittance to manage the various data in the database. The aim is to develop the database which stores data efficiently and interacts with the user in non-confusing manner. To get started with this project a valid ER diagram or relational diagram should be produced for better understanding of the database. The dump file must contain all the required database entries. The report is being prepared on the development process which includes all the further details about the program. Features integrated in python is also described briefly which is included in the program.

4. Target Audience:

This project is aimed for developers, companies and others who are working on developing similar projects and database management in MySQL. This project can also be helpful to students to observe how to complete certain tasks and as an example for similar projects. This project also has a good scope when it comes to developing similar program transaction monitoring.

5. Requirements for the program:

As the database is very light weight and doesn’t involve any graphical aspects, the hardware and software requirements for running this program is very basic. This program can run on any computer which can run latest version of XAMPP. I recommend using a computer with at least 2GHz of processing power 1GB of ram and 10GB free HDD space.

6. Purposed approach:

To complete the given coursework various steps were taken:

- To understand the assignment
- Gathering information
- Creating models and plans
- Coding and appending data into database
- Documenting
- Testing

7. Progression of the project:

The given task to develop and document the working database system in MySQL XAMPP required various knowledge and skills for understanding the problem, developing the solution, crafting codes, testing and solving the existing problems. This resulted the final product that had been created.

The project was started on 10th Mar 2018 and was completed and submitted on 23rd Mar 2018. This timeline shows my active engagement in the process of creating, managing and appending data into the database and documentation process too.

Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13
Model	🟥	🟥											
Database		🟨	🟨	🟨	🟨								
Report			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Queries									🟦	🟦			

Contents:

Table of tables:

Table of Figures:

1. Introduction:

As this project involves MySQL database, it contains several keywords derived from various programming languages and their technical jargons used in fields of database. For the viewers to understand this report easily I have included some important keywords used throughout the project. So, to understand this report one must first understand the meanings of these key terms used throughout the report. Some of the essential key words are as given below most of them are discussed throughout the project.

Some keywords:

Normalization, ER diagram, Relational Diagrams, Constraints, Primary Keys, Foreign Keys, Data dictionary, Debugging, Testing.

This project is based on developing a database for a remittance to manage information regarding its branches, services, employees, customers and mainly involves the transactions between the customers. The database includes 5 separate tables which was created so that the database is normalized to the 3rd normalization form. The database must show detailed information about remittance's branches, employees, account types, customers and transactions. The main point of this database is to track the transactions so the transaction table depends on branches, employee, and customer table. The transaction table takes two values from the customer's tables, one for the sender and another from the receiver which is account numbers of the respective people. The transaction must also have a timestamp and should be authorized by a staff existing in the database this value must not be null at any cost. So to start the project certain goals were set which is mentioned in the list below.

Goals:

- To find a suitable example on which database can be implemented
- To design how the database will work
- Create the database and fill it with data
- Test the usability of the database
- Document the process
- Get deeper understanding on the respective module.

So as the goals were set it needs to be accomplished with great determination. For creating and documenting the project. Doing this will ensure the practicality of the project and to complete the set goals following objectives were made:

Objectives:

- To construct a data model for a remittance company.
- Represent the model in form of diagrams and tables.
- Create the database with tables and data.
- Run some queries to ensure the proper functioning.
- Learn and implement skills in the project.
- Document the process.

The database so created

2. Discussion and analysis:

Database is a collection of data which is used to collect inter-related information and store it so that it can be easily accessed (Kedar, 2009). Database can also be defined as structured set of data with a collection of tables, queries, reports and views. Databases are widely used due to its simplicity as anyone can access it if they have permission, cost effectiveness and reduction of useless data. DBMS is simply Database Management Software which is required to create, manage and modify a database Some examples are: ORACLE, Microsoft SQL Server, MySQL etc.

Before developing the main database, a basic model sketch was designed to fulfill the desired expectations of the database. Then the sketch was converted into a rough relational diagram pointing the Primary key and Foreign Keys. Then a prototype database was created to observe the mechanics so that the original database will not lack any aspect. After the models that were made is satisfactory it was implemented in the main database. The tables were made according to the model constraints, datatypes and keys were carefully assigned to each attributes of the table. After the main framework of interconnected tables were created the database was then populated with their respective data and datatypes.

After the database was complete and filled with test data a dump file is made of the database and then various queries are done in the same database. The9 queries involves modifying the database elements such as entities, attributes or database itself and also modifying, adding, removing and viewing various entered data. This is to prove the flexibility of a database and also demonstrates the functionality of a Database Management System.

To accomplish the goals set by this project, a computer equipped following tools and software were used:

i. Microsoft Word 2016

   Word processing software to prepare the report.

ii. XAMPP

   Control panel to host MySQL database

iii. Sublime Text 3

   Text editor to edit texts and code

iv. Draw.io

   Online diagram creating platform

v. Microsoft Visio

   Professional diagram making software to develop flowcharts

More about XAMPP:

XAMPP stands for Cross platform, Apache, MariaDB, PHP and Pearl. Where MariaDB is a community edition of MySQL and it provides the required framework to run database servers in a simple and lightweight program. As this project heavily involves on this software and the dump file was created using it, It is highly recommended that the dump file so created be used using XAMPP.

3. Model

As this project is all about databases so a visual representation of the ideas that involves development of a database is quite important. There are many ways we can make solutions from visual perspective so the UML was created. UML stands for Unified modelling language, it is the universal or standardized way of representing solutions in forms of pictures and graphics (Jacobson, et al., 2000). The ERD used in this project has a similar approach to the UML, it is a subset of “semantic models” and is one of the best known way to express the concept of a database (Bagui & Earp, 2011).

The database is developed to run without errors and made so that it can be easily modified as it is normalized for ease of changing the data which might occur in future. Normalization also eliminates the data redundancy problem which makes the readability of a database very confusing and inaccurate. Normalization is basically a method of database handling to reduce the errors by reducing data redundancy and making similar data easier to update in future (Pratt & Last, 2014). There are four different types of normalization (Studytonight, 2018):

1st Normalization form

2nd Normalization form

3rd Normalization form

In the 3rd normalization form all the data redundancy and conflicting data are well managed. The attributes are related using keys which is much more efficient than typing whole repeating information in a single table and it also doesn’t have transitive dependencies. So this is the reason the database is split into forms of various tables instead of appending all the data into a single table. Normalization makes the functionality of database simple and it is lot easier to change the referred data which is used throughout the tables.

3.1. Tables:

To meet the desired expectation a model for the database is vital to easily visualize the inner mechanisms of how the database is going to process. So for this particular reason a model for remittance company was developed which is used to store the branch location, employee details, Senders, Receivers and most importantly the transaction part. All of these tables are inter connected to each other in one to many or one to one order, so The brief description of the table are as follows:

1. **Branches:**
   This table contains all the important details about the remittance company's overseas branches like it's location, branch code and contact.

2. **Employees**
   This table contains the information about the employees like the employees' ID, employees' name, contact, workplace and post.

3. **Senders**
   This table contains all the information of senders like Sender no, name, contact, and address.

4. **Receivers**
   This table contains all the information of receivers like Sender no, name, contact, and address.

5. **Transactions**
   This table is the most important part of the database and it must contain the transaction number, sender, receiver, date, authorizer of the transaction and the location of authorization(branch).

3.2. Relations:

As the database must be normalized it heavily depends on the use of primary keys and foreign keys which make the database to link the various tables. There are 5 relations between the 5 tables and which are:

1. **Sender – transactions relation:**
   One sender can send multiple transactions,
   But one transaction has only one sender

2. **Transaction – receiver relation:**
   One receiver can receiver multiple transactions,
   But one transaction has only one receiver

3. **Employee – transaction relation**
   One employee can authorize multiple transactions,
   But one transaction can only be authorized by one employee

4. **Branch – transaction relation**
   One branch can handle multiple transactions,
   But one transaction happens at only one branch

5. **Employees – branch relation**
   One branch can have multiple employees,
   But one employee works at only one branch.

3.3. Constraints

Constraints are the rules that controls the mode of data in a table (tutorialspoint, 2018). It includes following:

1. UNIQUE:

   Rule that dictates that the entry must be unique than other.
2. NOT NULL:

   The field must not be empty
3. AUTO_INCREMENT:

   The field should be left empty for auto generation of unique characters.

In this database various different constraints are used in various different tables which are as follows:

1. **Branches:**

   Location in branches must have a unique value

2. **Employees**

   Employees' name in employees must have a unique value and default value for post if not specified is "Teller"

3. **Senders**

   Sender's name must not be null

4. **Receivers**

   Receiver's name must not be null

5. **Transactions**

   The transactionCode (Primary Key) is auto increased so the field should be null.

3.4. Keys:

Keys are the constraints which is used to identify each record in a database. Each table has its own unique primary keys to ensure unique data entries and reduce conflicting data which might confuse the users. There are two types of keys in a database, Primary Key and Foreign key. The primary keys are referenced to other tables to create a link which makes the referencing easier in other tables by the use of foreign keys (Wenzel, 2018). The list of Primary keys and foreign keys are listed below:

Primary Keys:

Primary keys must be unique and isn't null able in any given record

1. Employees (empID)
2. Branches (branchCode)
3. Senders (SNo)
4. Receivers (RNo)
5. Transactions (transactionCode)

Foreign Keys:

Foreign keys must refer to the Primary Keys of another table

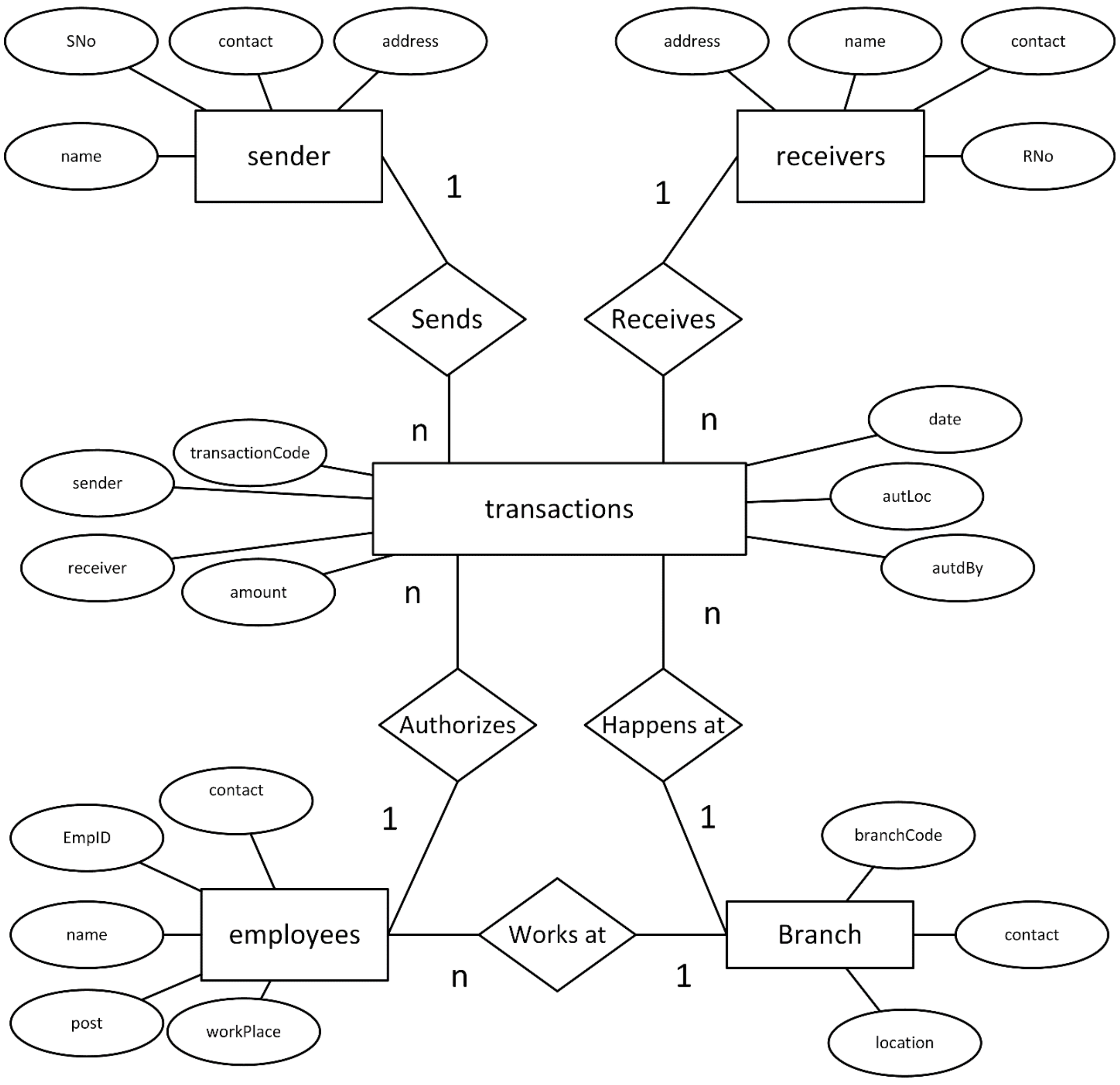
1. Employees:
    a. Workplace refers to branchCode in table branches

2. Transactions:
    a. Sender refers to SNo in senders table
    b. Receiver refers to RNo in receivers table
    c. autdBy refers to empID in employees table
    d. autLoc refers to branchCode in table branches

3.5. ER diagram

ER diagram stands for Entity – Relationship Diagram also referred to as ERD. The “Chen” like model is used in this project for representing the data model to be used in the database. This diagram simply shows the relation between the entities and its attributes in a functional database.

Figure 1: ER Diagram



3.6. Relational Diagram

It is simply another form of ERDs in which contains mode detailed information about the attributes Primary keys and Foreign Keys. It also shows the direct relationship between the entities bounded by the keys (lucid chart, 2018). This is also another variant of the Chen like model. The reason that type of model is called Chen like model is because Chen (1976) introduced the idea of ER diagrams.

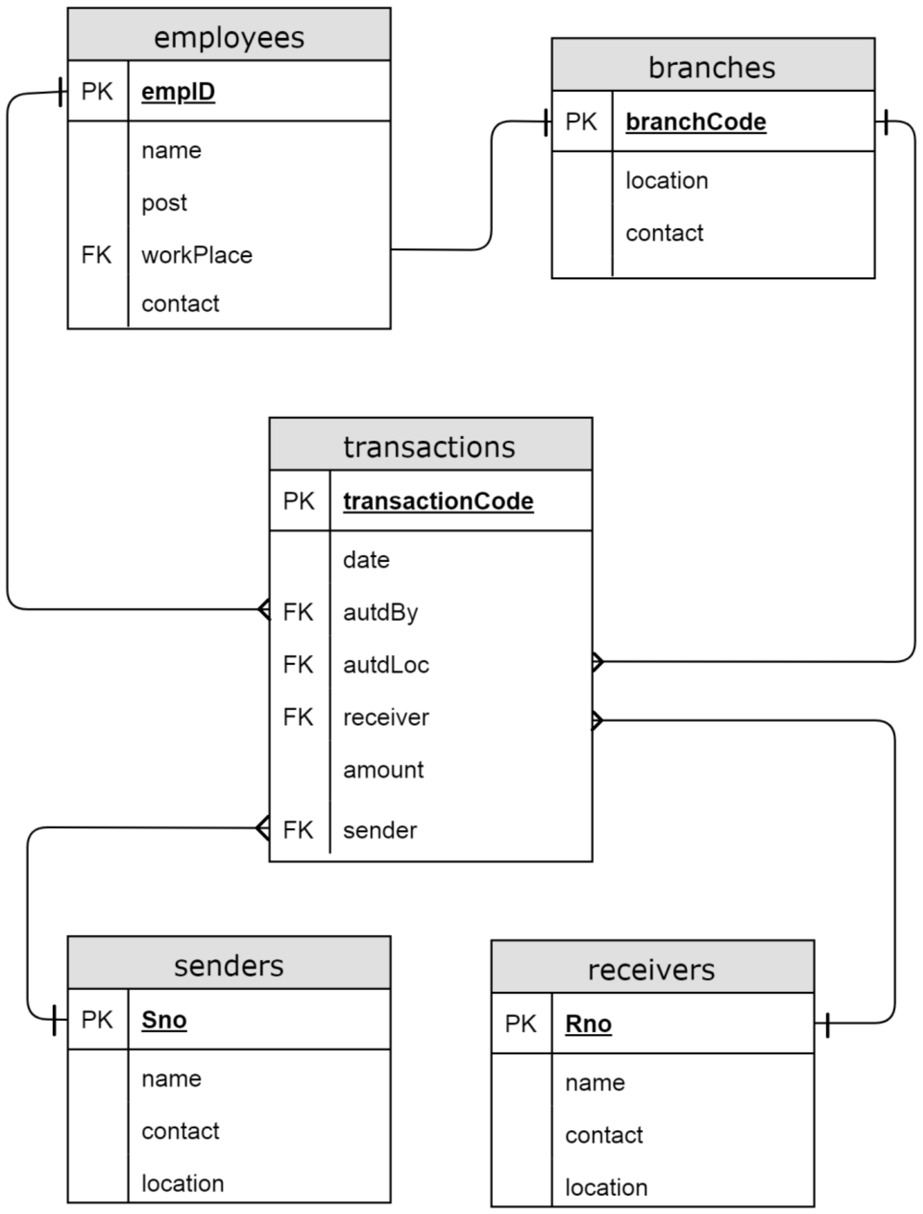
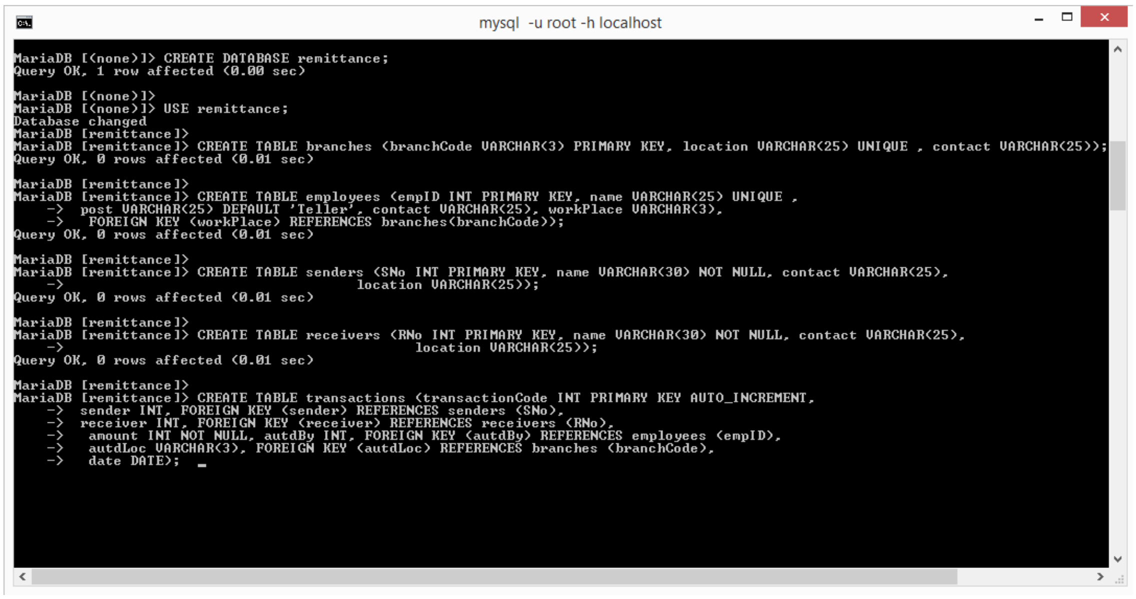


Figure 2: Relational Diagram

By the following models the database was created the evidence is as follows:

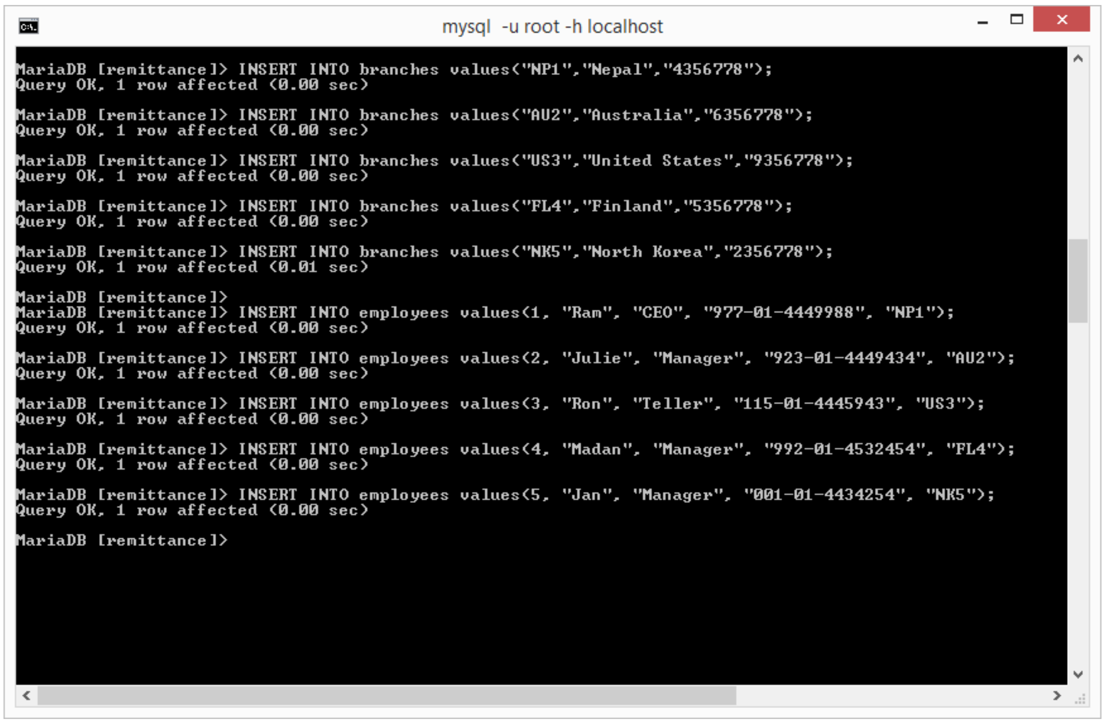
1. Creating the database and creating all the tables:



```
mysql  -u root -h localhost

MariaDB [(none)]> CREATE DATABASE remittance;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]>
MariaDB [(none)]> USE remittance;
Database changed
MariaDB [remittance]>
MariaDB [remittance]> CREATE TABLE branches (branchCode VARCHAR(3) PRIMARY KEY, location VARCHAR(25) UNIQUE , contact VARCHAR(25));
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]>
MariaDB [remittance]> CREATE TABLE employees (empID INT PRIMARY KEY, name VARCHAR(25) UNIQUE ,
    ->  post VARCHAR(25) DEFAULT 'Teller', contact VARCHAR(25), workPlace VARCHAR(3),
    ->   FOREIGN KEY (workPlace) REFERENCES branches(branchCode));
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]>
MariaDB [remittance]> CREATE TABLE senders (SNo INT PRIMARY KEY, name VARCHAR(30) NOT NULL, contact VARCHAR(25),
    ->                                  location VARCHAR(25));
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]>
MariaDB [remittance]> CREATE TABLE receivers (RNo INT PRIMARY KEY, name VARCHAR(30) NOT NULL, contact VARCHAR(25),
    ->                                         location VARCHAR(25));
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]>
MariaDB [remittance]> CREATE TABLE transactions (transactionCode INT PRIMARY KEY AUTO_INCREMENT,
    ->  sender INT, FOREIGN KEY (sender) REFERENCES senders (SNo),
    ->  receiver INT, FOREIGN KEY (receiver) REFERENCES receivers (RNo),
    ->   amount INT NOT NULL, autdBy INT, FOREIGN KEY (autdBy) REFERENCES employees (empID),
    ->   autdLoc VARCHAR(3), FOREIGN KEY (autdLoc) REFERENCES branches (branchCode),
    ->   date DATE);  _
```

Figure 3: Screenshot of creating tables

2. Adding values to the branches and employees tables:



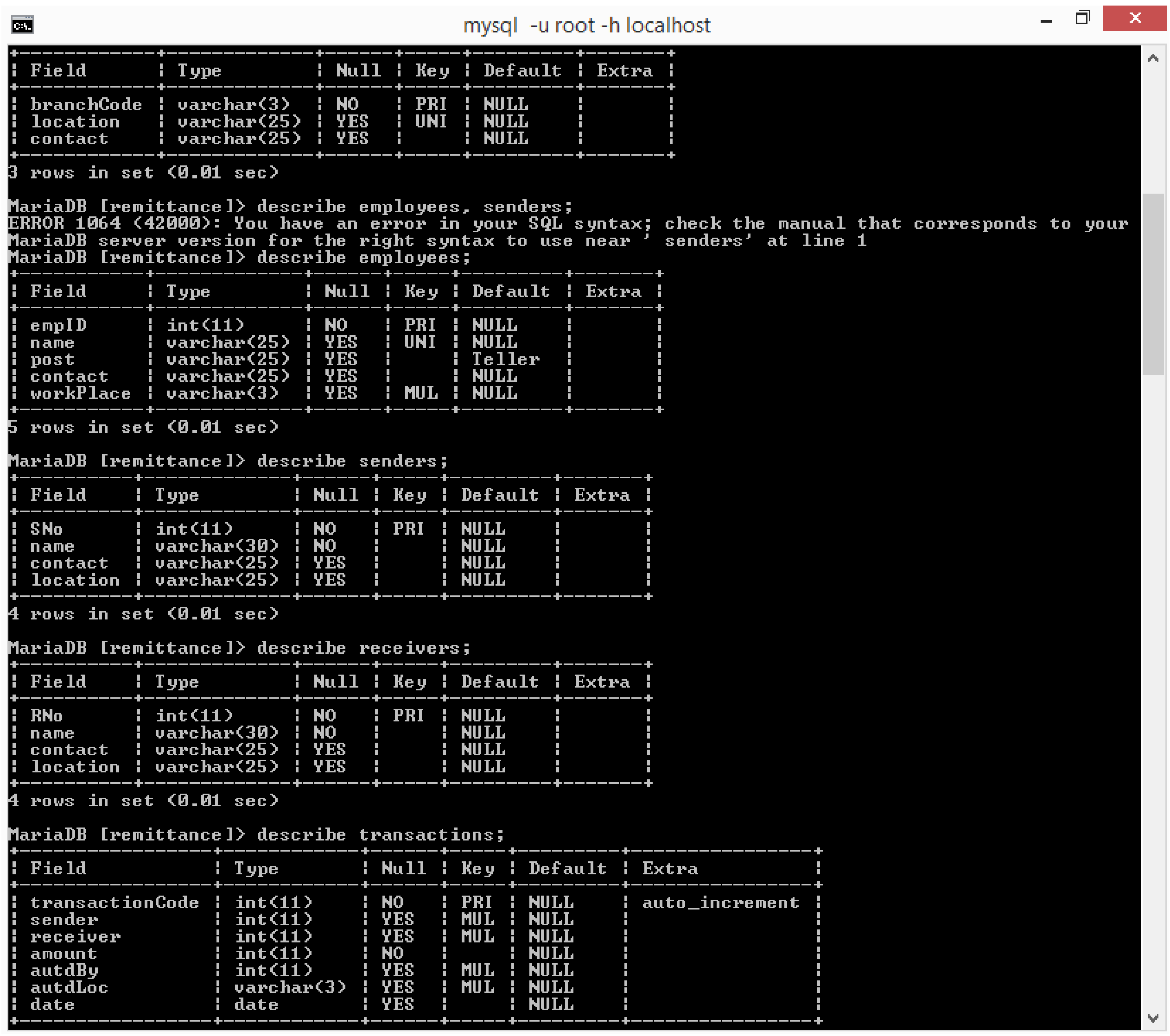
```
mysql  -u root -h localhost

MariaDB [remittance]> INSERT INTO branches values("NP1","Nepal","4356778");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO branches values("AU2","Australia","6356778");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO branches values("US3","United States","9356778");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO branches values("FL4","Finland","5356778");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO branches values("NK5","North Korea","2356778");
Query OK, 1 row affected (0.01 sec)

MariaDB [remittance]>
MariaDB [remittance]> INSERT INTO employees values(1, "Ram", "CEO", "977-01-4449988", "NP1");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO employees values(2, "Julie", "Manager", "923-01-4449434", "AU2");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO employees values(3, "Ron", "Teller", "115-01-4445943", "US3");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO employees values(4, "Madan", "Manager", "992-01-4532454", "FL4");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO employees values(5, "Jan", "Manager", "001-01-4434254", "NK5");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]>
```

Figure 4:Adding values to tables

3. Describing the tables:

All the tables were described in this screenshot which includes:

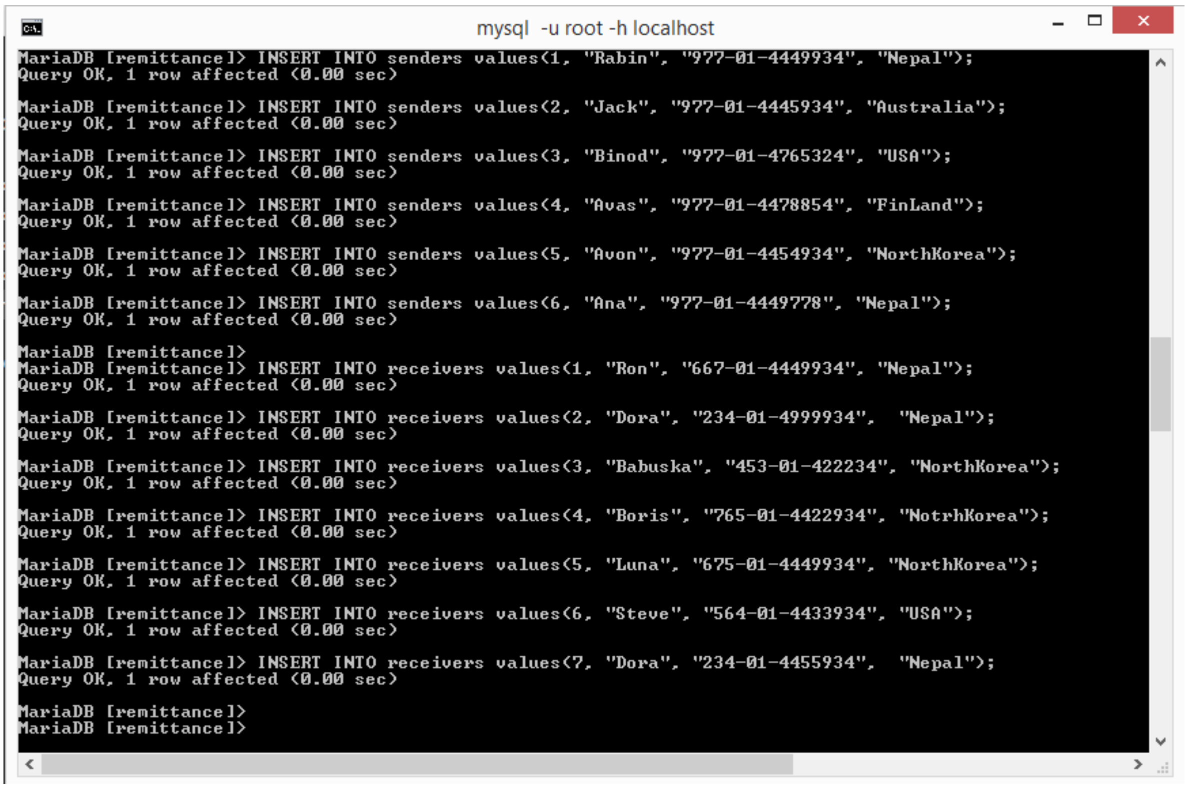
- Sender
- Receiver
- Employees
- Branches
- transactions



```
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| branchCode | varchar(3)  | NO   | PRI | NULL    |       |
| location   | varchar(25) | YES  | UNI | NULL    |       |
| contact    | varchar(25) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

MariaDB [remittance]> describe employees, senders;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near ' senders' at line 1
MariaDB [remittance]> describe employees;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| empID     | int(11)     | NO   | PRI | NULL    |       |
| name      | varchar(25) | YES  | UNI | NULL    |       |
| post      | varchar(25) | YES  |     | Teller  |       |
| contact   | varchar(25) | YES  |     | NULL    |       |
| workPlace | varchar(3)  | YES  | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

MariaDB [remittance]> describe senders;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| SNo      | int(11)     | NO   | PRI | NULL    |       |
| name     | varchar(30) | NO   |     | NULL    |       |
| contact  | varchar(25) | YES  |     | NULL    |       |
| location | varchar(25) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

MariaDB [remittance]> describe receivers;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| RNo      | int(11)     | NO   | PRI | NULL    |       |
| name     | varchar(30) | NO   |     | NULL    |       |
| contact  | varchar(25) | YES  |     | NULL    |       |
| location | varchar(25) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

MariaDB [remittance]> describe transactions;
+-----------------+------------+------+-----+---------+----------------+
| Field           | Type       | Null | Key | Default | Extra          |
+-----------------+------------+------+-----+---------+----------------+
| transactionCode | int(11)    | NO   | PRI | NULL    | auto_increment |
| sender          | int(11)    | YES  | MUL | NULL    |                |
| receiver        | int(11)    | YES  | MUL | NULL    |                |
| amount          | int(11)    | NO   |     | NULL    |                |
| autdBy          | int(11)    | YES  | MUL | NULL    |                |
| autdLoc         | varchar(3) | YES  | MUL | NULL    |                |
| date            | date       | YES  |     | NULL    |                |
+-----------------+------------+------+-----+---------+----------------+
```

Figure 5: Description of tables

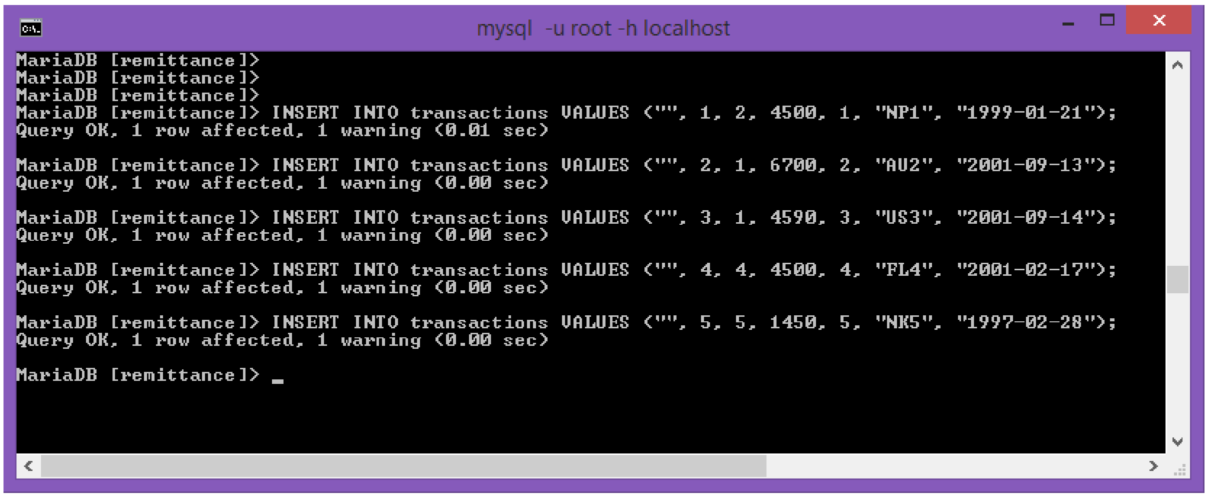
4. Inserting values of senders and receivers:



```
MariaDB [remittance]> INSERT INTO senders values(1, "Rabin", "977-01-4449934", "Nepal");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO senders values(2, "Jack", "977-01-4445934", "Australia");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO senders values(3, "Binod", "977-01-4765324", "USA");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO senders values(4, "Avas", "977-01-4478854", "FinLand");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO senders values(5, "Avon", "977-01-4454934", "NorthKorea");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO senders values(6, "Ana", "977-01-4449778", "Nepal");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]>
MariaDB [remittance]> INSERT INTO receivers values(1, "Ron", "667-01-4449934", "Nepal");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO receivers values(2, "Dora", "234-01-4999934",  "Nepal");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO receivers values(3, "Babuska", "453-01-422234", "NorthKorea");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO receivers values(4, "Boris", "765-01-4422934", "NotrhKorea");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO receivers values(5, "Luna", "675-01-4449934", "NorthKorea");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO receivers values(6, "Steve", "564-01-4433934", "USA");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]> INSERT INTO receivers values(7, "Dora", "234-01-4455934",  "Nepal");
Query OK, 1 row affected (0.00 sec)

MariaDB [remittance]>
MariaDB [remittance]>
```

Figure 6: Inserting values in senders and receivers

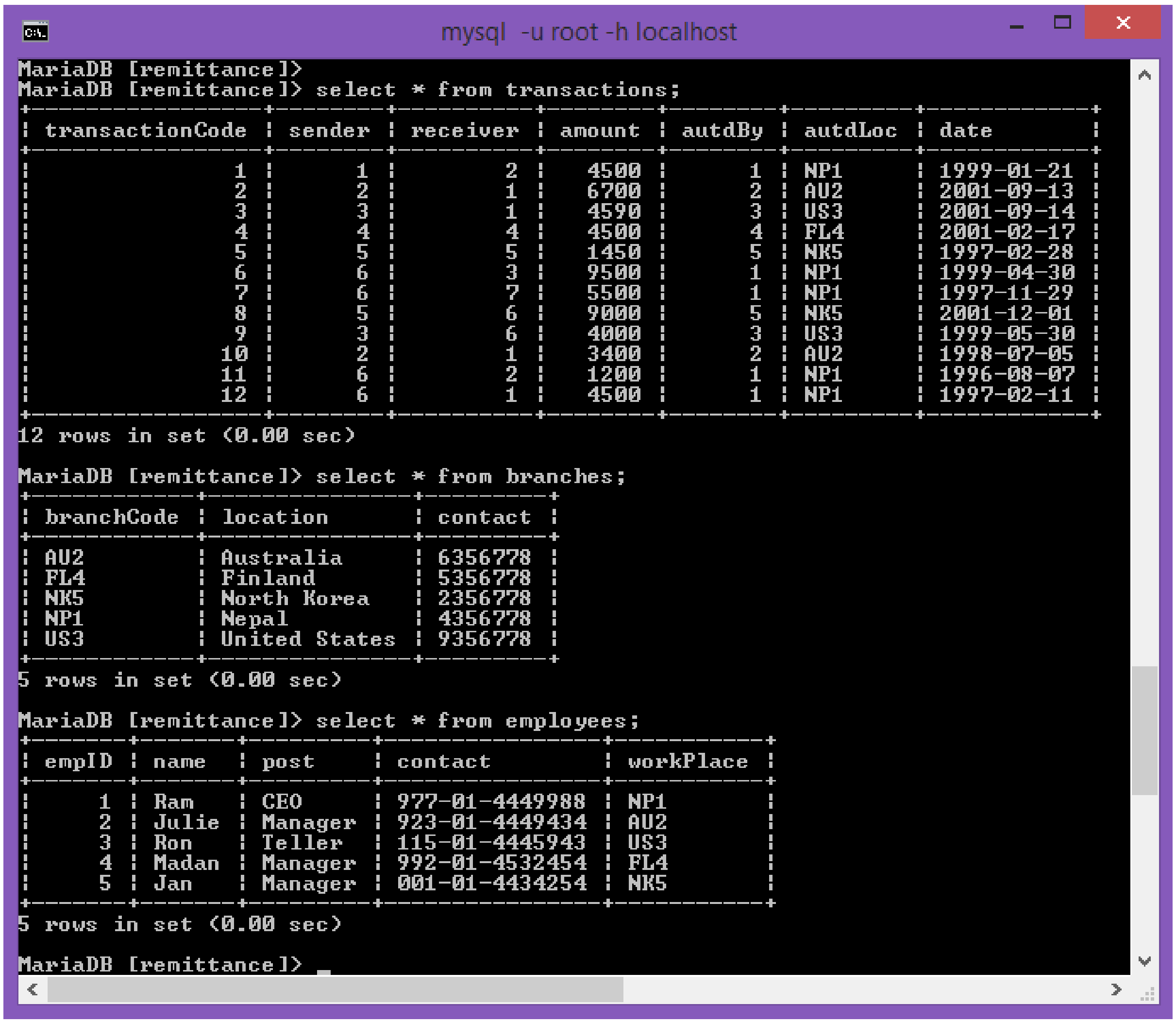
5. Inserting values for transactions:



```
MariaDB [remittance]>
MariaDB [remittance]>
MariaDB [remittance]>
MariaDB [remittance]> INSERT INTO transactions VALUES ("", 1, 2, 4500, 1, "NP1", "1999-01-21");
Query OK, 1 row affected, 1 warning (0.01 sec)

MariaDB [remittance]> INSERT INTO transactions VALUES ("", 2, 1, 6700, 2, "AU2", "2001-09-13");
Query OK, 1 row affected, 1 warning (0.00 sec)

MariaDB [remittance]> INSERT INTO transactions VALUES ("", 3, 1, 4590, 3, "US3", "2001-09-14");
Query OK, 1 row affected, 1 warning (0.00 sec)

MariaDB [remittance]> INSERT INTO transactions VALUES ("", 4, 4, 4500, 4, "FL4", "2001-02-17");
Query OK, 1 row affected, 1 warning (0.00 sec)

MariaDB [remittance]> INSERT INTO transactions VALUES ("", 5, 5, 1450, 5, "NK5", "1997-02-28");
Query OK, 1 row affected, 1 warning (0.00 sec)

MariaDB [remittance]> _
```

Figure 7: Inserting transaction values

6. Viewing the data in various tables:



```
MariaDB [remittance]>
MariaDB [remittance]> select * from transactions;
+-----------------+--------+----------+--------+--------+---------+------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | date       |
+-----------------+--------+----------+--------+--------+---------+------------+
|               1 |      1 |        2 |   4500 |      1 | NP1     | 1999-01-21 |
|               2 |      2 |        1 |   6700 |      2 | AU2     | 2001-09-13 |
|               3 |      3 |        1 |   4590 |      3 | US3     | 2001-09-14 |
|               4 |      4 |        4 |   4500 |      4 | FL4     | 2001-02-17 |
|               5 |      5 |        5 |   1450 |      5 | NK5     | 1997-02-28 |
|               6 |      6 |        3 |   9500 |      1 | NP1     | 1999-04-30 |
|               7 |      6 |        7 |   5500 |      1 | NP1     | 1997-11-29 |
|               8 |      5 |        6 |   9000 |      5 | NK5     | 2001-12-01 |
|               9 |      3 |        6 |   4000 |      3 | US3     | 1999-05-30 |
|              10 |      2 |        1 |   3400 |      2 | AU2     | 1998-07-05 |
|              11 |      6 |        2 |   1200 |      1 | NP1     | 1996-08-07 |
|              12 |      6 |        1 |   4500 |      1 | NP1     | 1997-02-11 |
+-----------------+--------+----------+--------+--------+---------+------------+
12 rows in set (0.00 sec)

MariaDB [remittance]> select * from branches;
+------------+---------------+---------+
| branchCode | location      | contact |
+------------+---------------+---------+
| AU2        | Australia     | 6356778 |
| FL4        | Finland       | 5356778 |
| NK5        | North Korea   | 2356778 |
| NP1        | Nepal         | 4356778 |
| US3        | United States | 9356778 |
+------------+---------------+---------+
5 rows in set (0.00 sec)

MariaDB [remittance]> select * from employees;
+-------+-------+---------+-----------------+-----------+
| empID | name  | post    | contact         | workPlace |
+-------+-------+---------+-----------------+-----------+
|     1 | Ram   | CEO     | 977-01-4449988  | NP1       |
|     2 | Julie | Manager | 923-01-4449434  | AU2       |
|     3 | Ron   | Teller  | 115-01-4445943  | US3       |
|     4 | Madan | Manager | 992-01-4532454  | FL4       |
|     5 | Jan   | Manager | 001-01-4434254  | NK5       |
+-------+-------+---------+-----------------+-----------+
5 rows in set (0.00 sec)

MariaDB [remittance]> _
```

Figure 8: Viewing data in tables

4. Data Dictionary

Data dictionary describes the structure of the whole database which is represented in a table. It includes the detailed information about the table(s) in a database and acts like a database about a database which provides all the required information about a database. Data dictionary simply put is collection of tables which gives us detailed information of the inner workings of a database (Kreines, 2003). It provides the details of all the attributes of the entity like the entity name, entity description, column name column description, data types, length of data type. The data dictionary also describes and mentions the constraints used in that particular entity like Primary Key, Foreign Key, Null able value, Unique and notes. The notes part might contain various details about that particular attributes like if it is auto incremented, or it has a default value or if the attribute is liked as a foreign key it specifies the column of the table to which it is linked. The data dictionaries of the five tables of the database are as follows:

4.1. Branches:

Branches table has 3 columns, which includes the Branch Code, location of the branch and contact number of the branch. The BranchNo is the primary key and should be unique and not be null. While the other fields are null able if the data is to be entered later.

Entity Name	Entity Description	Column Name	Column Description	Data type	Length	Primary Key	Foreign Key	Nullable	Unique	Notes
Branches	Branch details overseas	BranchNo	Each branch has a unique code	VARCHAR	3	True	False	False	True	
		location	Name of location	VARCHAR	25	False	False	True	True	
		contact	Phone no	VARCHAR	10	False	False	True	False	

Table 1: Branches

4.2. Employees:

The employees table contains all the information about all the employees working on the remittance company. It includes the employee's unique ID number, the employees' name, post, contact, and workplace. The EmpID is the primary key of this table and must always be unique and not empty. The name must be Unique for the employee, and the default value for the post of the employee if not specified is "Teller". The foreign key workplace refers to the branchCode of branches table and has the same datatype specified as of the branch ID

Entity Name	Entity Description	Column Name	Column Description	Data type	Length	Primary Key	Foreign Key	Nullable	Unique	Notes
Employees	Details of employees	EmpID	Each employe has a unique code	INT	11	True	False	False	True	
		name	Name of location	VARCHAR	25	False	False	True	True	
		post	Rank of worker	VARCHAR	25	False	False	True	False	Default = "Teller"
		contact	Phone no	25	10	False	False	True	False	
		workPlace	Address of work	VARCHAR	3	False	True	True	False	Refers to Branch-Code

Table 2: Employees

4.3. Senders:

Senders has the details about all the senders, Its primary key is SNo. The name of the sender must be unique and must not be null.

Entity Name	Entity Description	Column Name	Column Description	Data type	Length	Primary Key	Foreign Key	Nullable	Unique	Notes
Senders	Details of senders	SNo	Sender's ID	INT	11	True	False	False	True	
		name	Name of location	VARCHAR	30	False	False	False	True	
		contact	Phone no	VARCHAR	25	False	False	True	False	
		address	Address of sender	VARCHAR	25	False	False	True	False	

Table 3: Sender

4.4. Receivers:

Receivers has the details about all the senders, its primary key is RNo. The name of the sender must be unique and must not be null.

Entity Name	Entity Description	Column Name	Column Description	Data type	Length	Primary Key	Foreign Key	Nullable	Unique	Notes
Receivers	Details of receivers	RNo	Ref no of receiver	INT	11	True	False	False	True	
		name	Name of location	VARCHAR	30	False	False	False	True	
		contact	Phone no	VARCHAR	25	False	False	True	False	
		address	Address of sender	VARCHAR	25	False	False	True	False	

Table 4: Receivers

4.5. Transactions:

Transactions contains 7 attributes. Its primary key is TransactionCode as every transaction has an own unique code to distinguish it from others. The transaction code is also auto incremented i.e. the field for the primary key is to be left null without a value assigned so that the database assigns a unique value to it by itself. The sender refers to SNo of the senders table, similarly the receiver refers to RNo of receivers table, the amount must not be empty. The autdBy refers to the EmpID of the employees table, the autLoc refers to the location of the branch where the transaction happens and finally date which includes the date value of the day of transaction

Entity Name	Entity Description	Column Name	Column Description	Data type	Length	Primary Key	Foreign Key	Nullable	Unique	Notes
Employees	Details of employees	Transaction-Code	Each transaction has a unique code	INT	11	True	False	False	True	Auto increment
		sender	Sender of the amount	INT	11	False	True	True	False	Refers to SNo
		receiver	Receiver of the amount	INT	11	False	True	True	False	Refers to RNo
		amount	The value of money sent	INT	11	False	False	False	False	
		autdBy	Employee who authorized the transaction	INT	11	False	True	True	False	Refers to empID
		autLoc	The location where the transaction took place	VARCHAR	3	False	True	True	False	Refers to Branch-Code
		Date	The date of transaction	DATE		False	False	True	False	

Table 5: Transactions

5. Queries:

Queries are basically commands that are introduced to the databases later. Queries can have many functions like to view, search, delete, modify or edit the entries. Queries can also be used to modify the table attributes like to add, remove or modify the columns. Since queries are used to modify the database later it is quite useful to learn related skills.

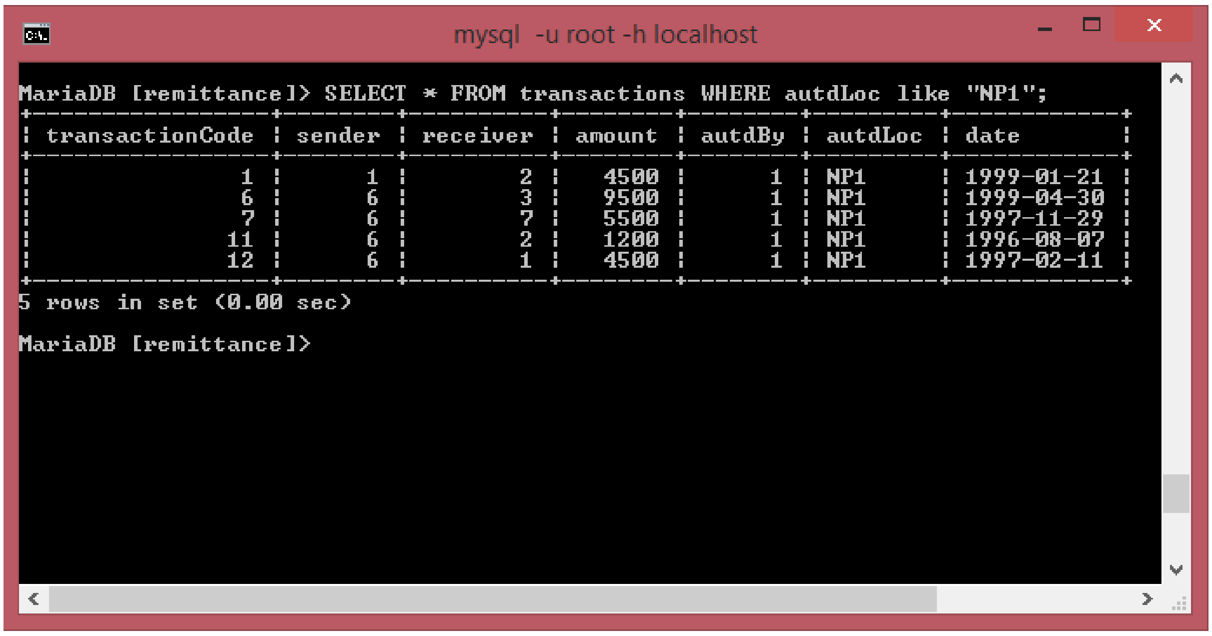
The Queries that have been entered in the database are as follows:

1. To view the entries in transaction which got authorized in “NP1”
2. To show all the branches and employee details
3. To show records in transactions which was done before year 2000
4. To add and remove a column in branches table
5. To add a table and remove it.
6. Inner joining two tables (transactions with receivers)
7. Joining branches and employees
8. Right joining transactions with receivers:
9. Removing date from transactions and contacts from senders and receivers
10. Joining 3 tables
11. Joining Four tables
12. Joining Five tables

5.1. To view the entries in transaction which got authorized in “NP1”

This query is to view all the transactions that took place in the Nepal branch “NP1”

```
SELECT * FROM transactions WHERE name LIKE “NP01”;
```



```
MariaDB [remittance]> SELECT * FROM transactions WHERE autdLoc like "NP1";
+-----------------+--------+----------+--------+--------+---------+------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | date       |
+-----------------+--------+----------+--------+--------+---------+------------+
|               1 |      1 |        2 |   4500 |      1 | NP1     | 1999-01-21 |
|               6 |      6 |        3 |   9500 |      1 | NP1     | 1999-04-30 |
|               7 |      6 |        7 |   5500 |      1 | NP1     | 1997-11-29 |
|              11 |      6 |        2 |   1200 |      1 | NP1     | 1996-08-07 |
|              12 |      6 |        1 |   4500 |      1 | NP1     | 1997-02-11 |
+-----------------+--------+----------+--------+--------+---------+------------+
5 rows in set (0.00 sec)

MariaDB [remittance]>
```

Figure 9: Query 1 use of "LIKE"

The query did exactly what it was intended to do. All the transactions that was authorized in Nepal “NP1” was viewed in as an output. The “LIKE” keyword is used with “WHERE” clause for a specific target. It can also be used with “%<string>%” to find a target with certain string as its character. For example:

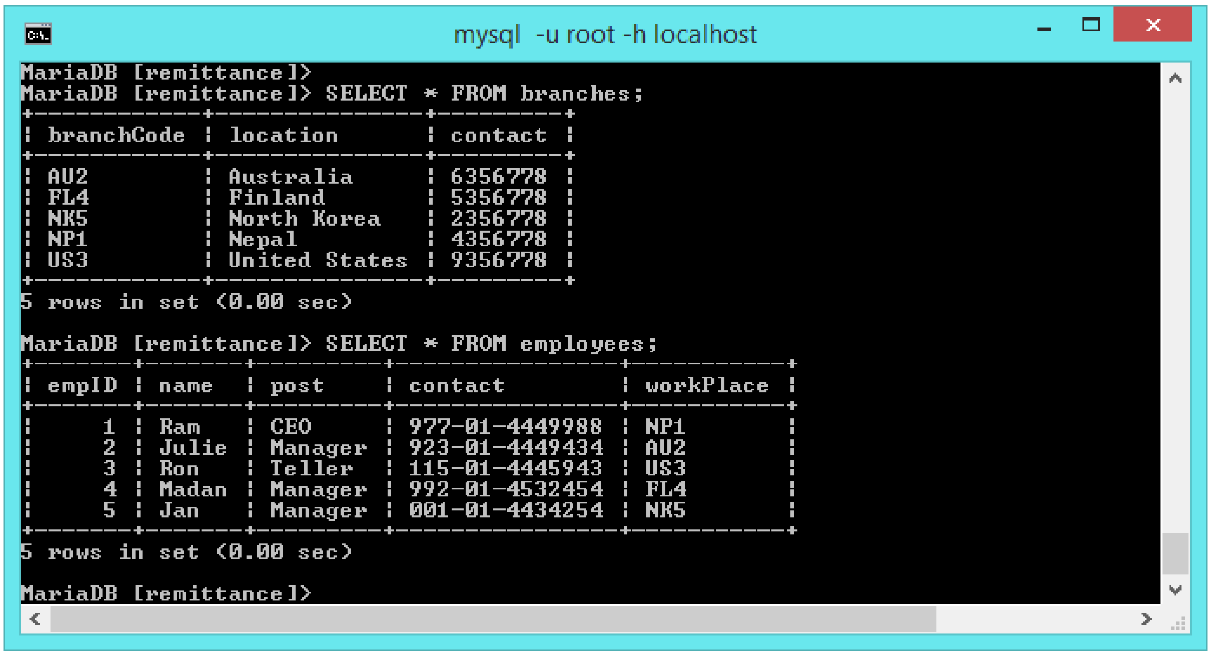
```
SELECT * FROM senders WHERE name like "%a%";
```

Will show all the senders with “a” as an character in its name.

5.2. To show all the branches and employee details

“SELECT *” can be used to select and view all the records in the respective tables.

```
SELECT * FROM branches;

SELECT * FROM employees;
```



```
MariaDB [remittance]>
MariaDB [remittance]> SELECT * FROM branches;
+------------+---------------+---------+
| branchCode | location      | contact |
+------------+---------------+---------+
| AU2        | Australia     | 6356778 |
| FL4        | Finland       | 5356778 |
| NK5        | North Korea   | 2356778 |
| NP1        | Nepal         | 4356778 |
| US3        | United States | 9356778 |
+------------+---------------+---------+
5 rows in set (0.00 sec)

MariaDB [remittance]> SELECT * FROM employees;
+-------+-------+---------+-----------------+-----------+
| empID | name  | post    | contact         | workPlace |
+-------+-------+---------+-----------------+-----------+
|     1 | Ram   | CEO     | 977-01-4449988  | NP1       |
|     2 | Julie | Manager | 923-01-4449434  | AU2       |
|     3 | Ron   | Teller  | 115-01-4445943  | US3       |
|     4 | Madan | Manager | 992-01-4532454  | FL4       |
|     5 | Jan   | Manager | 001-01-4434254  | NK5       |
+-------+-------+---------+-----------------+-----------+
5 rows in set (0.00 sec)

MariaDB [remittance]>
```

Figure 10: Query 2 Viewing all the elements in tables using ""*

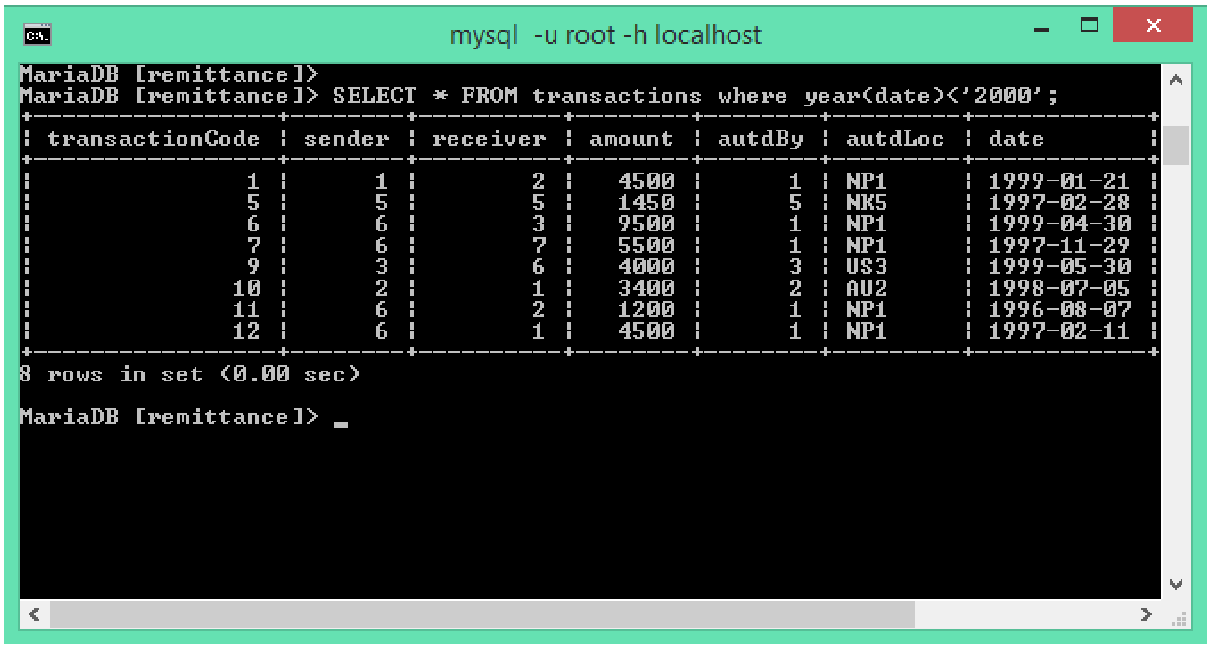
The “SELECT” keyword is usually used to view data and entries and is used often with other keywords.

“*” Refers to all records in the table in programming terms if we want to include everything it is usually followed by “*” to denote that it includes everything.

5.3. To show records in transactions which was done before year 2000

“WHERE” clause can also be used with comparison operators to filter through date values:

```
SELECT * FROM transactions where year(date)<'2000';
```



```
mysql  -u root -h localhost
MariaDB [remittance]>
MariaDB [remittance]> SELECT * FROM transactions where year(date)<'2000';
+-----------------+--------+----------+--------+--------+---------+------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | date       |
+-----------------+--------+----------+--------+--------+---------+------------+
|               1 |      1 |        2 |   4500 |      1 | NP1     | 1999-01-21 |
|               5 |      5 |        5 |   1450 |      5 | NK5     | 1997-02-28 |
|               6 |      6 |        3 |   9500 |      1 | NP1     | 1999-04-30 |
|               7 |      6 |        7 |   5500 |      1 | NP1     | 1997-11-29 |
|               9 |      3 |        6 |   4000 |      3 | US3     | 1999-05-30 |
|              10 |      2 |        1 |   3400 |      2 | AU2     | 1998-07-05 |
|              11 |      6 |        2 |   1200 |      1 | NP1     | 1996-08-07 |
|              12 |      6 |        1 |   4500 |      1 | NP1     | 1997-02-11 |
+-----------------+--------+----------+--------+--------+---------+------------+
8 rows in set (0.00 sec)

MariaDB [remittance]> _
```

Figure 11: Query 3 by using "WHERE" with date

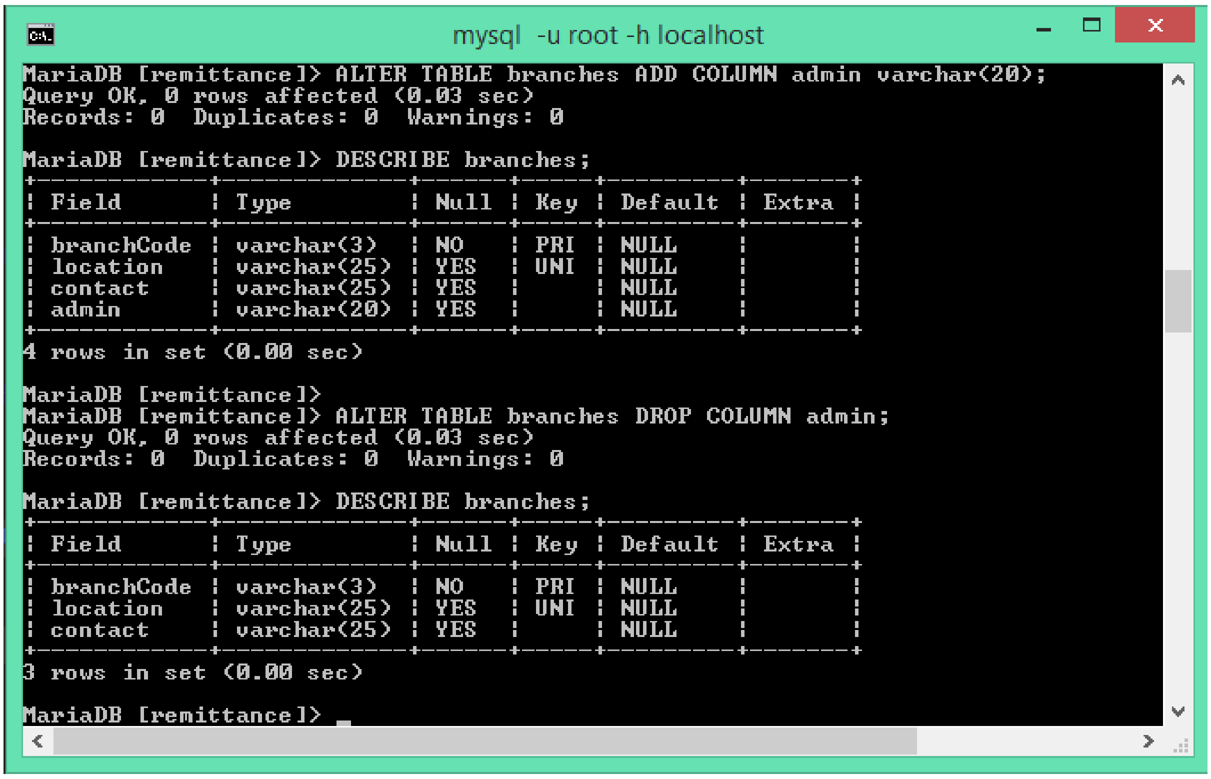
The “WHERE” clause can also be used with months and days like:

```
SELECT * FROM transactions where day(date)='01';
SELECT * FROM transactions where month(date)=<'06';
```

5.4. To add and remove a column in branches table

The ”ALTER TABLE” and “ADD COLUMN” can be used to add new columns to an existing table. Then “DROP COLUMN” can be used to drop or remove any given table.

```
ALTER TABLE branches ADD COLUMN admin VARCHAR(20);

DESCRIBE branches;

ALTER TABLE branches DROP COLUMN admin;

DESCRIBE branches;
```



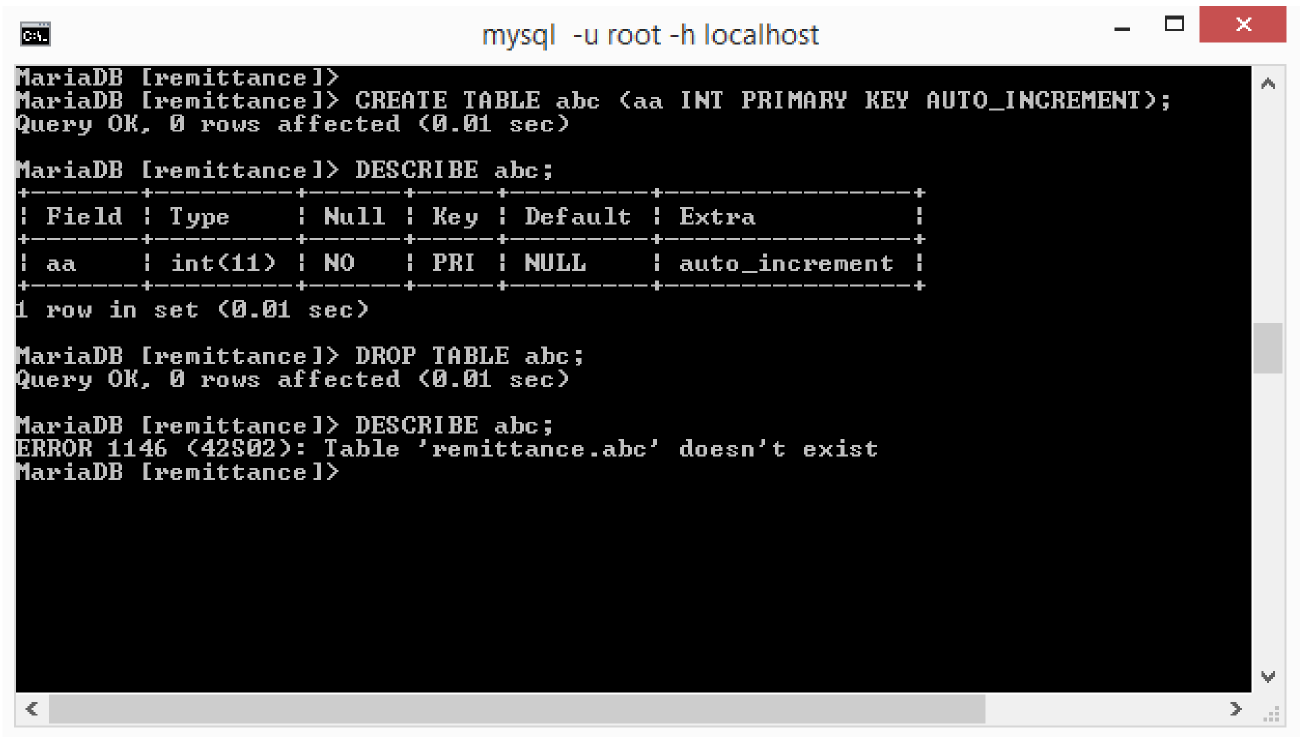
```
MariaDB [remittance]> ALTER TABLE branches ADD COLUMN admin varchar(20);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [remittance]> DESCRIBE branches;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| branchCode | varchar(3)  | NO   | PRI | NULL    |       |
| location   | varchar(25) | YES  | UNI | NULL    |       |
| contact    | varchar(25) | YES  |     | NULL    |       |
| admin      | varchar(20) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

MariaDB [remittance]>
MariaDB [remittance]> ALTER TABLE branches DROP COLUMN admin;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [remittance]> DESCRIBE branches;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| branchCode | varchar(3)  | NO   | PRI | NULL    |       |
| location   | varchar(25) | YES  | UNI | NULL    |       |
| contact    | varchar(25) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

MariaDB [remittance]> _
```

Figure 12: Query 4 adding and then removing a column in a table.

5.5. To add a table and remove it.

In MySQL a table can be removed if it is useless. We should always remove tables carefully as it removes the data inside the table too.

```
CREATE TABLE abc (aa INT PRIMARY KEY AUTO_INCREMENT);
DESCRIBE abc;
DROP TABLE abc;
DESCRIBE abc;
```



```
MariaDB [remittance]>
MariaDB [remittance]> CREATE TABLE abc (aa INT PRIMARY KEY AUTO_INCREMENT);
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]> DESCRIBE abc;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| aa    | int(11) | NO   | PRI | NULL    | auto_increment |
+-------+---------+------+-----+---------+----------------+
1 row in set (0.01 sec)

MariaDB [remittance]> DROP TABLE abc;
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]> DESCRIBE abc;
ERROR 1146 (42S02): Table 'remittance.abc' doesn't exist
MariaDB [remittance]>
```

Figure 13: Query 4 adding and then removing a table

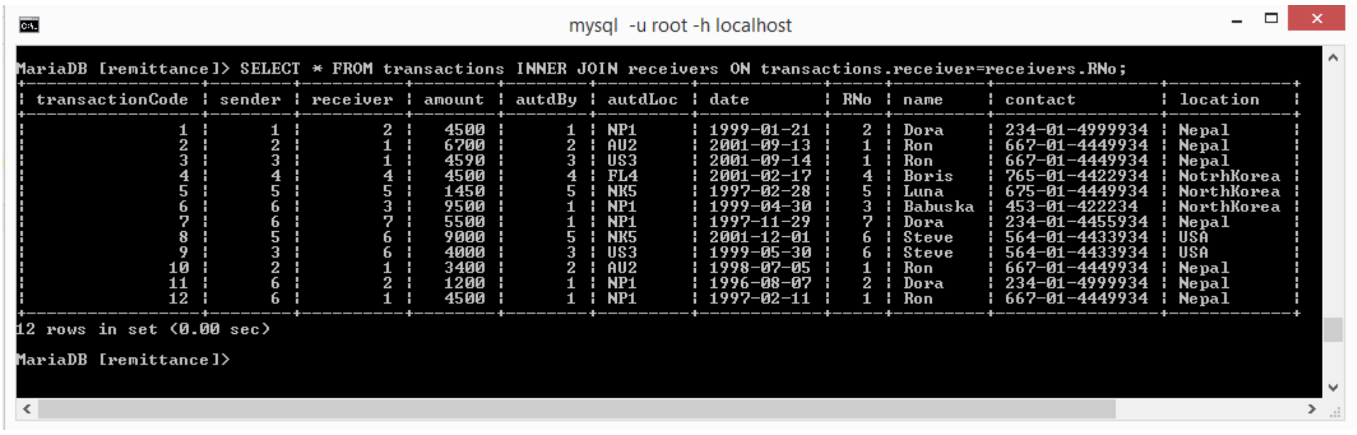
The “DROP” keyword can also be used to remove an entire database by using following syntax:

```
DROP DATABASE remittance;
```

5.6. Inner joining two tables (transactions with receivers):

The tables can be joined in my sql by “JOIN” or “INNER JOIN” keyword.

```
SELECT * FROM transactions JOIN receivers ON
 transactions.receiver = receivers.RNo;
```



```
MariaDB [remittance]> SELECT * FROM transactions INNER JOIN receivers ON transactions.receiver=receivers.RNo;
+-----------------+--------+----------+--------+--------+-----------+------------+-----+---------+-----------------+-------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc   | date       | RNo | name    | contact         | location    |
+-----------------+--------+----------+--------+--------+-----------+------------+-----+---------+-----------------+-------------+
|               1 |      1 |        2 |   4500 |      1 | NP1       | 1999-01-21 |   2 | Dora    | 234-01-4999934  | Nepal       |
|               2 |      2 |        1 |   6700 |      2 | AU2       | 2001-09-13 |   1 | Ron     | 667-01-4449934  | Nepal       |
|               3 |      3 |        1 |   4590 |      3 | US3       | 2001-09-14 |   1 | Ron     | 667-01-4449934  | Nepal       |
|               4 |      4 |        4 |   4500 |      4 | FL4       | 2001-02-17 |   4 | Boris   | 765-01-4422934  | NotrhKorea  |
|               5 |      5 |        5 |   1450 |      5 | NK5       | 1997-02-28 |   5 | Luna    | 675-01-4449934  | NorthKorea  |
|               6 |      6 |        3 |   9500 |      1 | NP1       | 1999-04-30 |   3 | Babuska | 453-01-422234   | NorthKorea  |
|               7 |      6 |        7 |   5500 |      1 | NP1       | 1997-11-29 |   7 | Dora    | 234-01-4455934  | Nepal       |
|               8 |      5 |        6 |   9000 |      5 | NK5       | 2001-12-01 |   6 | Steve   | 564-01-4433934  | USA         |
|               9 |      3 |        6 |   4000 |      3 | US3       | 1999-05-30 |   6 | Steve   | 564-01-4433934  | USA         |
|              10 |      2 |        1 |   3400 |      2 | AU2       | 1998-07-05 |   1 | Ron     | 667-01-4449934  | Nepal       |
|              11 |      6 |        2 |   1200 |      1 | NP1       | 1996-08-07 |   2 | Dora    | 234-01-4999934  | Nepal       |
|              12 |      6 |        1 |   4500 |      1 | NP1       | 1997-02-11 |   1 | Ron     | 667-01-4449934  | Nepal       |
+-----------------+--------+----------+--------+--------+-----------+------------+-----+---------+-----------------+-------------+
12 rows in set (0.00 sec)

MariaDB [remittance]>
```

Figure 14: Query 6 Joining two tables

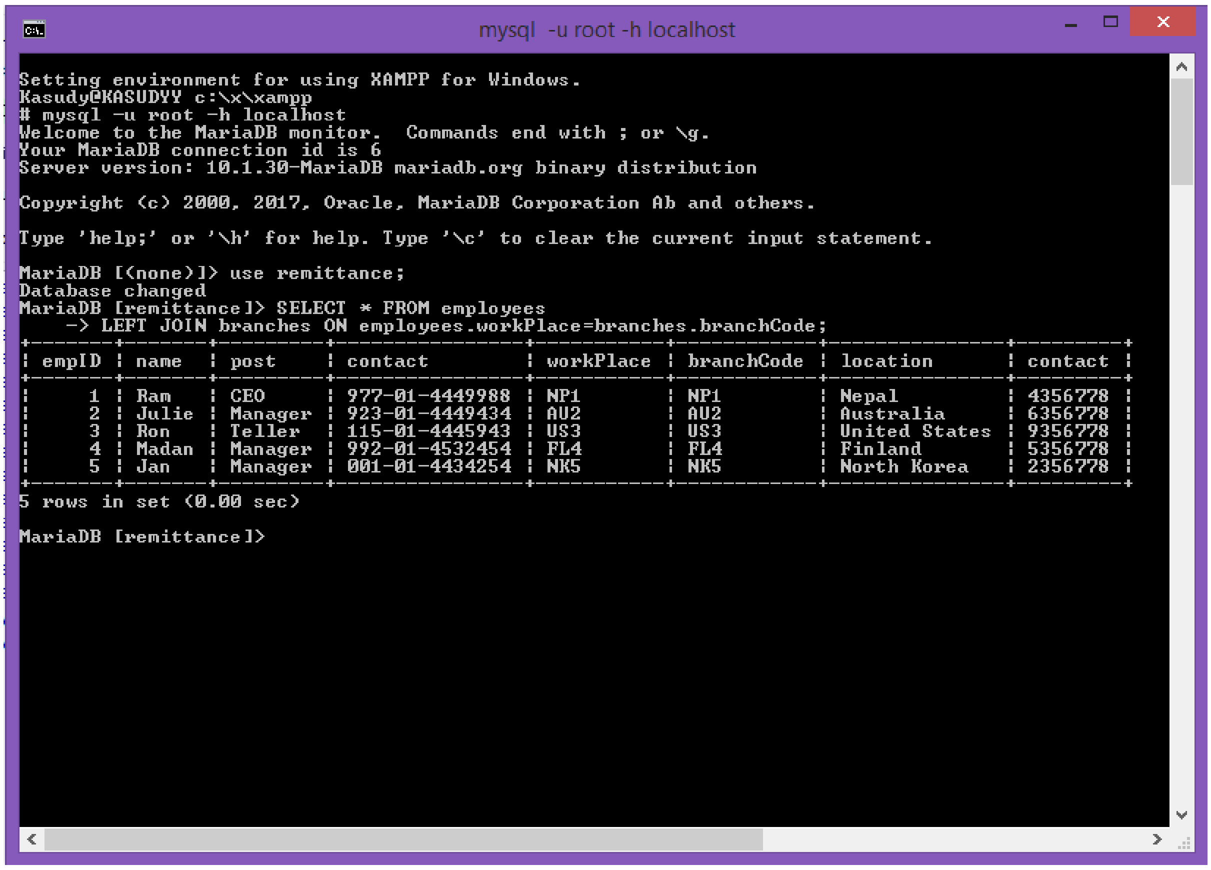
The syntax can also be:

```
SELECT * FROM transactions INNER JOIN receivers ON
 transactions.receiver=receivers.RNo;
```

5.7. Joining branches and employees

The tables branches and employees are joined

```
SELECT * FROM employees
LEFT JOIN branches ON employees.workPlace=branches.branchCode;
```



```
Setting environment for using XAMPP for Windows.
Kasudy@KASUDYY c:\x\xampp
# mysql -u root -h localhost
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 10.1.30-MariaDB 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 remittance;
Database changed
MariaDB [remittance]> SELECT * FROM employees
    -> LEFT JOIN branches ON employees.workPlace=branches.branchCode;
+-------+-------+---------+----------------+-----------+------------+---------------+---------+
| empID | name  | post    | contact        | workPlace | branchCode | location      | contact |
+-------+-------+---------+----------------+-----------+------------+---------------+---------+
|     1 | Ram   | CEO     | 977-01-4449988 | NP1       | NP1        | Nepal         | 4356778 |
|     2 | Julie | Manager | 923-01-4449434 | AU2       | AU2        | Australia     | 6356778 |
|     3 | Ron   | Teller  | 115-01-4445943 | US3       | US3        | United States | 9356778 |
|     4 | Madan | Manager | 992-01-4532454 | FL4       | FL4        | Finland       | 5356778 |
|     5 | Jan   | Manager | 001-01-4434254 | NK5       | NK5        | North Korea   | 2356778 |
+-------+-------+---------+----------------+-----------+------------+---------------+---------+
5 rows in set (0.00 sec)

MariaDB [remittance]>
```

Figure 15: Query 7 LEFT Joining two tables

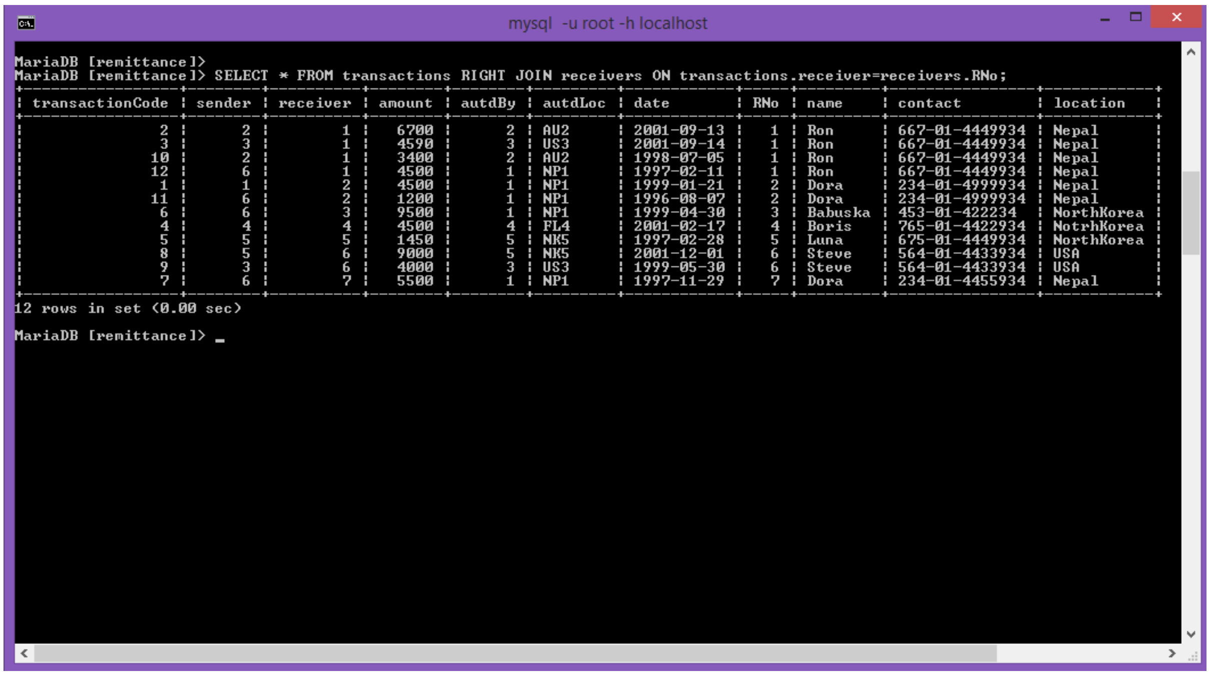
The result will be same if only “JOIN” keyword were used as both the tables have 1 employees each :

```
SELECT * FROM employees
JOIN branches ON employees.workPlace=branches.branchCode;
```

5.8 Right joining transactions with receivers:

This query is almost same as 5.6. Which was joining receivers with transactions but this is a right join which means the transactions are sorted in form of the receivers.

```
SELECT * FROM transactions RIGHT JOIN
 receivers ON transactions.receiver=receivers.RNo;
```



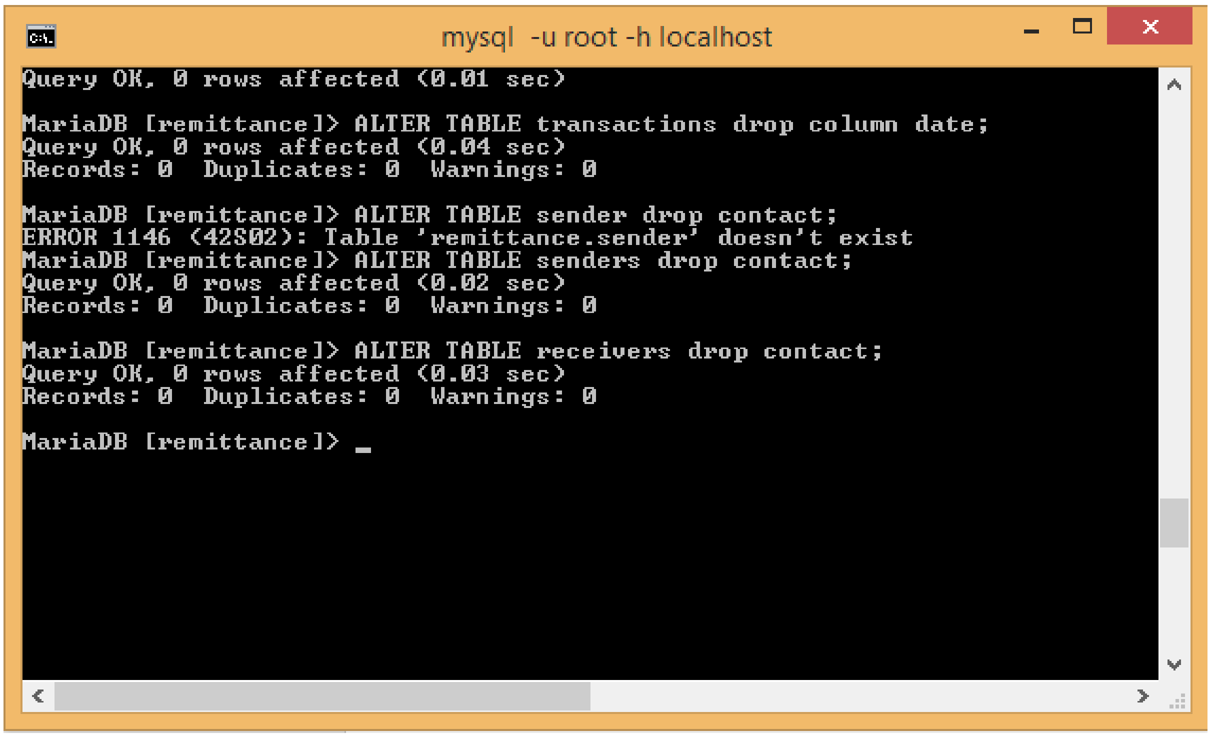
```
MariaDB [remittance]>
MariaDB [remittance]> SELECT * FROM transactions RIGHT JOIN receivers ON transactions.receiver=receivers.RNo;
+-----------------+--------+----------+--------+--------+---------+------------+-----+---------+-----------------+------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | date       | RNo | name    | contact         | location   |
+-----------------+--------+----------+--------+--------+---------+------------+-----+---------+-----------------+------------+
|               2 |      2 |        1 |   6700 |      2 | AU2     | 2001-09-13 |   1 | Ron     | 667-01-4449934  | Nepal      |
|               3 |      3 |        1 |   4590 |      3 | US3     | 2001-09-14 |   1 | Ron     | 667-01-4449934  | Nepal      |
|              10 |      2 |        1 |   3400 |      2 | AU2     | 1998-07-05 |   1 | Ron     | 667-01-4449934  | Nepal      |
|              12 |      6 |        1 |   4500 |      1 | NP1     | 1997-02-11 |   1 | Ron     | 667-01-4449934  | Nepal      |
|               1 |      1 |        2 |   4500 |      1 | NP1     | 1999-01-21 |   2 | Dora    | 234-01-4999934  | Nepal      |
|              11 |      6 |        2 |   1200 |      1 | NP1     | 1996-08-07 |   2 | Dora    | 234-01-4999934  | Nepal      |
|               6 |      6 |        3 |   9500 |      1 | NP1     | 1999-04-30 |   3 | Babuska | 453-01-422234   | NorthKorea |
|               4 |      4 |        4 |   4500 |      4 | FL4     | 2001-02-17 |   4 | Boris   | 765-01-4422934  | NotrhKorea |
|               5 |      5 |        5 |   1450 |      5 | NK5     | 1997-02-28 |   5 | Luna    | 675-01-4449934  | NorthKorea |
|               8 |      5 |        6 |   9000 |      5 | NK5     | 2001-12-01 |   6 | Steve   | 564-01-4433934  | USA        |
|               9 |      3 |        6 |   4000 |      3 | US3     | 1999-05-30 |   6 | Steve   | 564-01-4433934  | USA        |
|               7 |      6 |        7 |   5500 |      1 | NP1     | 1997-11-29 |   7 | Dora    | 234-01-4455934  | Nepal      |
+-----------------+--------+----------+--------+--------+---------+------------+-----+---------+-----------------+------------+
12 rows in set (0.00 sec)

MariaDB [remittance]> _
```

Figure 16: Query 8 RIGHT Joining two tables

5.9 Removing date from transactions and contacts from senders and receivers:

If the column is not quite useful it can also be removed easily by drop column keyword. However, this will delete the data entered in the column.

```
ALTER TABLE receivers DROP COLUMN contact;

ALTER TABLE senders DROP COLUMN contact;

ALTER TABLE transactions DROP COLUMN date;
```



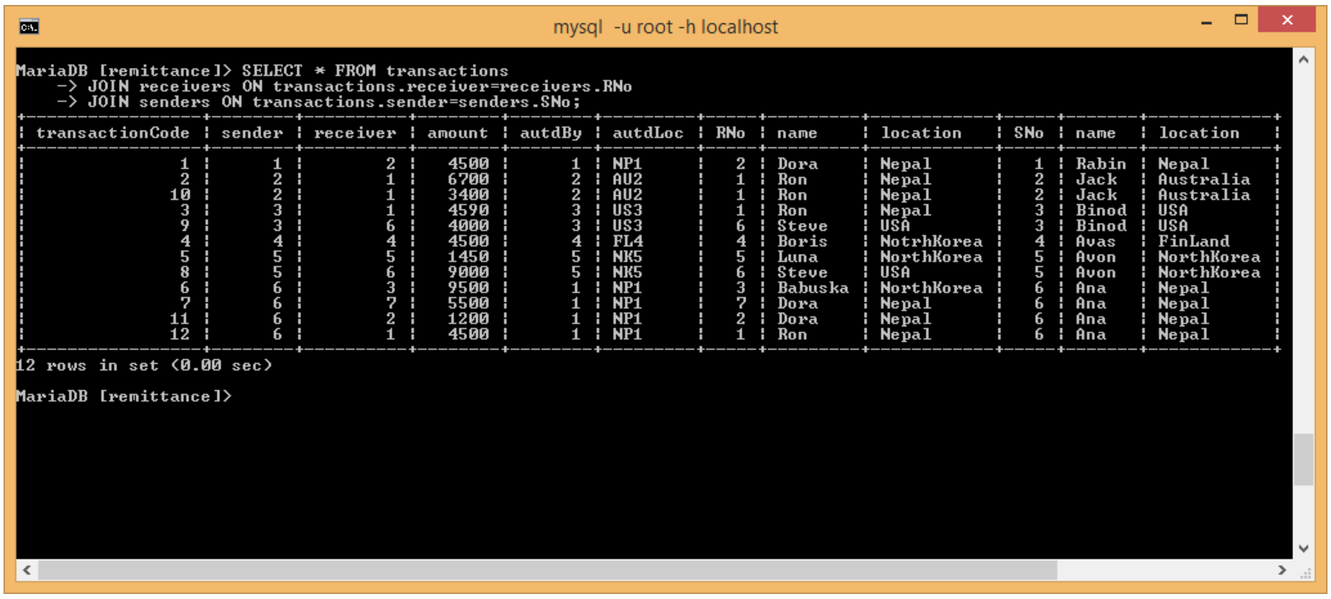
```
Query OK, 0 rows affected (0.01 sec)

MariaDB [remittance]> ALTER TABLE transactions drop column date;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [remittance]> ALTER TABLE sender drop contact;
ERROR 1146 (42S02): Table 'remittance.sender' doesn't exist
MariaDB [remittance]> ALTER TABLE senders drop contact;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [remittance]> ALTER TABLE receivers drop contact;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [remittance]> _
```

Figure 17: Query 9 removing some columns with data from tables

5.10 Joining 3 tables:

Tables can be joined in multiple times it depends on the foreign key of the table. In this database the maximum no of table joins is 5.

```
SELECT * FROM transactions
JOIN receivers ON transactions.receiver=receivers.RNo
JOIN senders ON transactions.sender=senders.SNo;
```



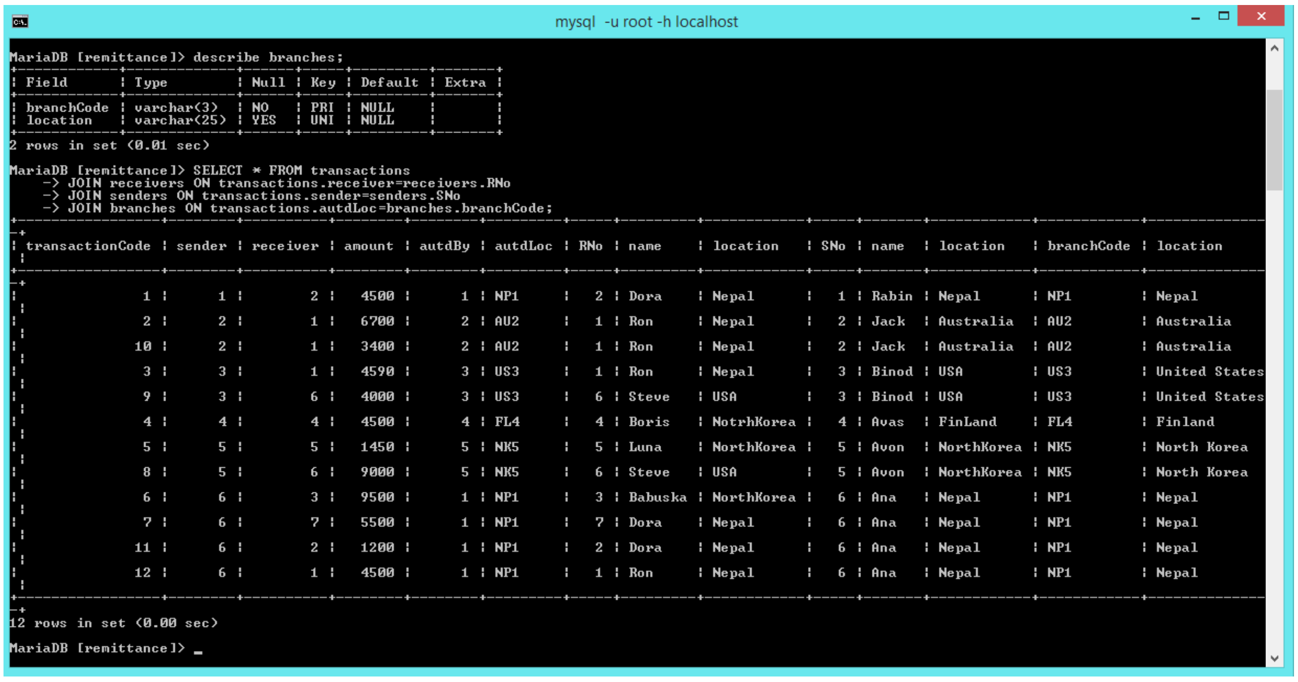
```
MariaDB [remittance]> SELECT * FROM transactions
    -> JOIN receivers ON transactions.receiver=receivers.RNo
    -> JOIN senders ON transactions.sender=senders.SNo;
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | RNo | name    | location   | SNo | name  | location   |
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+
|               1 |      1 |        2 |   4500 |      1 | NP1     |   2 | Dora    | Nepal      |   1 | Rabin | Nepal      |
|               2 |      2 |        1 |   6700 |      2 | AU2     |   1 | Ron     | Nepal      |   2 | Jack  | Australia  |
|              10 |      2 |        1 |   3400 |      2 | AU2     |   1 | Ron     | Nepal      |   2 | Jack  | Australia  |
|               3 |      3 |        1 |   4590 |      3 | US3     |   1 | Ron     | Nepal      |   3 | Binod | USA        |
|               9 |      3 |        6 |   4000 |      3 | US3     |   6 | Steve   | USA        |   3 | Binod | USA        |
|               4 |      4 |        4 |   4500 |      4 | FL4     |   4 | Boris   | NotrhKorea |   4 | Avas  | FinLand    |
|               5 |      5 |        5 |   1450 |      5 | NK5     |   5 | Luna    | NorthKorea |   5 | Avon  | NorthKorea |
|               8 |      5 |        6 |   9000 |      5 | NK5     |   6 | Steve   | USA        |   5 | Avon  | NorthKorea |
|               6 |      6 |        3 |   9500 |      1 | NP1     |   3 | Babuska | NorthKorea |   6 | Ana   | Nepal      |
|               7 |      6 |        7 |   5500 |      1 | NP1     |   7 | Dora    | Nepal      |   6 | Ana   | Nepal      |
|              11 |      6 |        2 |   1200 |      1 | NP1     |   2 | Dora    | Nepal      |   6 | Ana   | Nepal      |
|              12 |      6 |        1 |   4500 |      1 | NP1     |   1 | Ron     | Nepal      |   6 | Ana   | Nepal      |
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+
12 rows in set (0.00 sec)

MariaDB [remittance]>
```

Figure 18: Query 10 Joining three tables

5.11 Joining Four tables:

As mentioned earlier there can be number of tables which can be joined together so in this query four tables are joined.

```
SELECT * FROM transactions
JOIN receivers ON transactions.receiver=receivers.RNo
JOIN senders ON transactions.sender=senders.SNo
JOIN branches ON transactions.autdLoc=branches.branchCode;
```



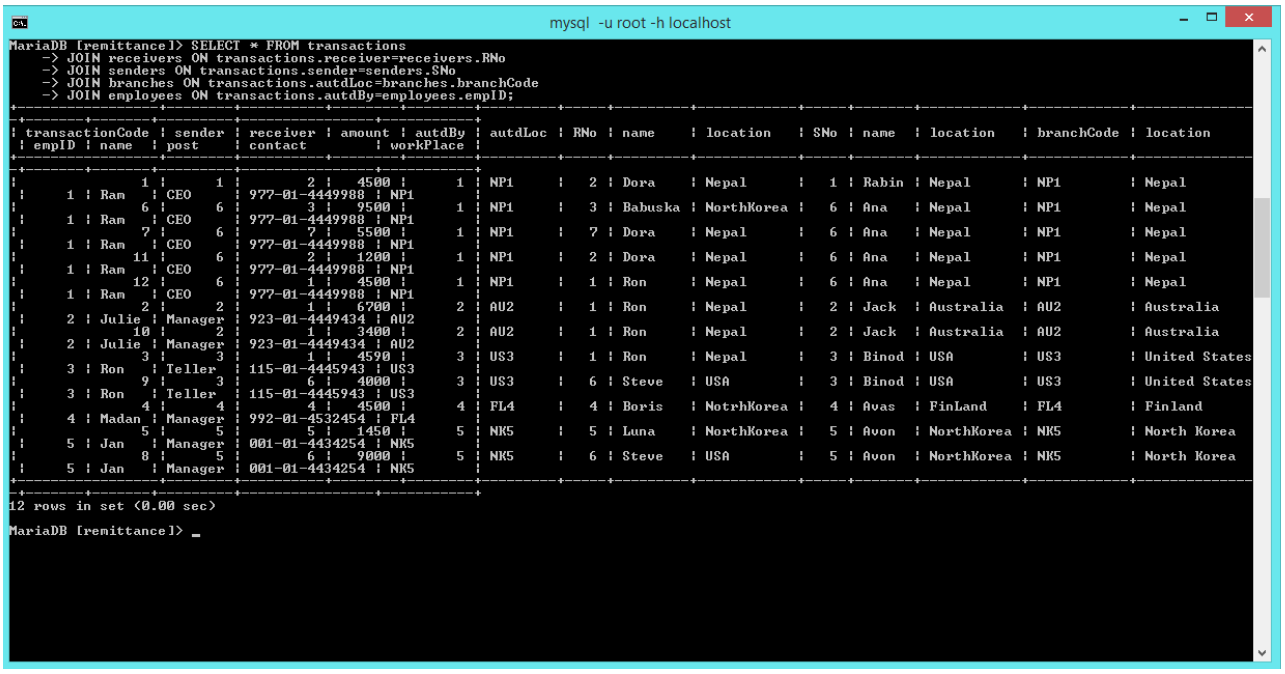
```
MariaDB [remittance]> describe branches;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| branchCode | varchar(3)  | NO   | PRI | NULL    |       |
| location   | varchar(25) | YES  | UNI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

MariaDB [remittance]> SELECT * FROM transactions
    -> JOIN receivers ON transactions.receiver=receivers.RNo
    -> JOIN senders ON transactions.sender=senders.SNo
    -> JOIN branches ON transactions.autdLoc=branches.branchCode;
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+------------+---------------
-+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | RNo | name    | location   | SNo | name  | location   | branchCode | location
 |
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+------------+---------------
-+
|               1 |      1 |        2 |   4500 |      1 | NP1     |   2 | Dora    | Nepal      |   1 | Rabin | Nepal      | NP1        | Nepal
 |
|               2 |      2 |        1 |   6700 |      2 | AU2     |   1 | Ron     | Nepal      |   2 | Jack  | Australia  | AU2        | Australia
 |
|              10 |      2 |        1 |   3400 |      2 | AU2     |   1 | Ron     | Nepal      |   2 | Jack  | Australia  | AU2        | Australia
 |
|               3 |      3 |        1 |   4590 |      3 | US3     |   1 | Ron     | Nepal      |   3 | Binod | USA        | US3        | United States
 |
|               9 |      3 |        6 |   4000 |      3 | US3     |   6 | Steve   | USA        |   3 | Binod | USA        | US3        | United States
 |
|               4 |      4 |        4 |   4500 |      4 | FL4     |   4 | Boris   | NotrhKorea |   4 | Avas  | FinLand    | FL4        | Finland
 |
|               5 |      5 |        5 |   1450 |      5 | NK5     |   5 | Luna    | NorthKorea |   5 | Avon  | NorthKorea | NK5        | North Korea
 |
|               8 |      5 |        6 |   9000 |      5 | NK5     |   6 | Steve   | USA        |   5 | Avon  | NorthKorea | NK5        | North Korea
 |
|               6 |      6 |        3 |   9500 |      1 | NP1     |   3 | Babuska | NorthKorea |   6 | Ana   | Nepal      | NP1        | Nepal
 |
|               7 |      6 |        7 |   5500 |      1 | NP1     |   7 | Dora    | Nepal      |   6 | Ana   | Nepal      | NP1        | Nepal
 |
|              11 |      6 |        2 |   1200 |      1 | NP1     |   2 | Dora    | Nepal      |   6 | Ana   | Nepal      | NP1        | Nepal
 |
|              12 |      6 |        1 |   4500 |      1 | NP1     |   1 | Ron     | Nepal      |   6 | Ana   | Nepal      | NP1        | Nepal
 |
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+------------+---------------
-+
12 rows in set (0.00 sec)

MariaDB [remittance]> _
```

Figure 19: Query 11 Joining 4 Tables

5.12 Joining Five tables:

As mentioned earlier there can be number of tables which can be joined together so in this query five tables are joined.

```
SELECT * FROM transactions
JOIN receivers ON transactions.receiver=receivers.RNo
JOIN senders ON transactions.sender=senders.SNo
JOIN branches ON transactions.autdLoc=branches.branchCode
JOIN employees ON transactions.autdBy=employees.empID;
```



```
MariaDB [remittance]> SELECT * FROM transactions
    -> JOIN receivers ON transactions.receiver=receivers.RNo
    -> JOIN senders ON transactions.sender=senders.SNo
    -> JOIN branches ON transactions.autdLoc=branches.branchCode
    -> JOIN employees ON transactions.autdBy=employees.empID;
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+------------+--------------
-+-------+--------+---------+----------------+------------+
| transactionCode | sender | receiver | amount | autdBy | autdLoc | RNo | name    | location   | SNo | name  | location   | branchCode | location
 | empID | name   | post    | contact        | workPlace  |
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+------------+--------------
-+-------+--------+---------+----------------+------------+
|               1 |      1 |        2 |   4500 |      1 | NP1     |   2 | Dora    | Nepal      |   1 | Rabin | Nepal      | NP1        | Nepal
 |     1 | Ram    | CEO     | 977-01-4449988 | NP1        |
|               6 |      6 |        3 |   9500 |      1 | NP1     |   3 | Babuska | NorthKorea |   6 | Ana   | Nepal      | NP1        | Nepal
 |     1 | Ram    | CEO     | 977-01-4449988 | NP1        |
|               7 |      6 |        7 |   5500 |      1 | NP1     |   7 | Dora    | Nepal      |   6 | Ana   | Nepal      | NP1        | Nepal
 |     1 | Ram    | CEO     | 977-01-4449988 | NP1        |
|              11 |      6 |        2 |   1200 |      1 | NP1     |   2 | Dora    | Nepal      |   6 | Ana   | Nepal      | NP1        | Nepal
 |     1 | Ram    | CEO     | 977-01-4449988 | NP1        |
|              12 |      6 |        1 |   4500 |      1 | NP1     |   1 | Ron     | Nepal      |   6 | Ana   | Nepal      | NP1        | Nepal
 |     1 | Ram    | CEO     | 977-01-4449988 | NP1        |
|               2 |      2 |        1 |   6700 |      2 | AU2     |   1 | Ron     | Nepal      |   2 | Jack  | Australia  | AU2        | Australia
 |     2 | Julie  | Manager | 923-01-4449434 | AU2        |
|              10 |      2 |        1 |   3400 |      2 | AU2     |   1 | Ron     | Nepal      |   2 | Jack  | Australia  | AU2        | Australia
 |     2 | Julie  | Manager | 923-01-4449434 | AU2        |
|               3 |      3 |        1 |   4590 |      3 | US3     |   1 | Ron     | Nepal      |   3 | Binod | USA        | US3        | United States
 |     3 | Ron    | Teller  | 115-01-4445943 | US3        |
|               9 |      3 |        6 |   4000 |      3 | US3     |   6 | Steve   | USA        |   3 | Binod | USA        | US3        | United States
 |     3 | Ron    | Teller  | 115-01-4445943 | US3        |
|               4 |      4 |        4 |   4500 |      4 | FL4     |   4 | Boris   | NotrhKorea |   4 | Avas  | FinLand    | FL4        | Finland
 |     4 | Madan  | Manager | 992-01-4532454 | FL4        |
|               5 |      5 |        5 |   1450 |      5 | NK5     |   5 | Luna    | NorthKorea |   5 | Avon  | NorthKorea | NK5        | North Korea
 |     5 | Jan    | Manager | 001-01-4434254 | NK5        |
|               8 |      5 |        6 |   9000 |      5 | NK5     |   6 | Steve   | USA        |   5 | Avon  | NorthKorea | NK5        | North Korea
 |     5 | Jan    | Manager | 001-01-4434254 | NK5        |
+-----------------+--------+----------+--------+--------+---------+-----+---------+------------+-----+-------+------------+------------+--------------
-+-------+--------+---------+----------------+------------+
12 rows in set (0.00 sec)

MariaDB [remittance]> _
```

Figure 20: Query 12 joining 12 tables

Here, due to lack of space the information provided collides so the join query must be used carefully.

6. **Research:**

This project would be impossible to complete without the help of books, websites and journals containing vital information about the respective subjects. Several books and websites contained information that was required to design and code the database and journals was also quite helpful. Google Scholar and Google Books were used to provide legit information provided by professionals or experts related to the subject supporting this project. Providing right information is quite hard nowadays due to the redundancy of information. As people can express their feelings on the internet it is quite difficult to find reasonable works supported by evidence so the search engines were strictly limited to google scholar and books. The lecture slides were also very helpful for quickly retrieving useful information about the models, techniques and keywords as the content is well made it also helped a lot while documenting and coding the project. The effort of the creators and people involved in the creation of books, journals, websites and the lecture slides are well appreciated and has helped a lot in the creation of this project. This project would be impossible to complete without the information gathered by the help of research.

The books, websites and journals are properly referenced at the end of the project to honor the authors for their work which was very helpful to create this project. Every idea and works extracted from the referenced materials are properly cited to their respective topics. The previous course works and lecture slides was also quite useful as a reference material for the formatting of the documentation and methods to express the database.

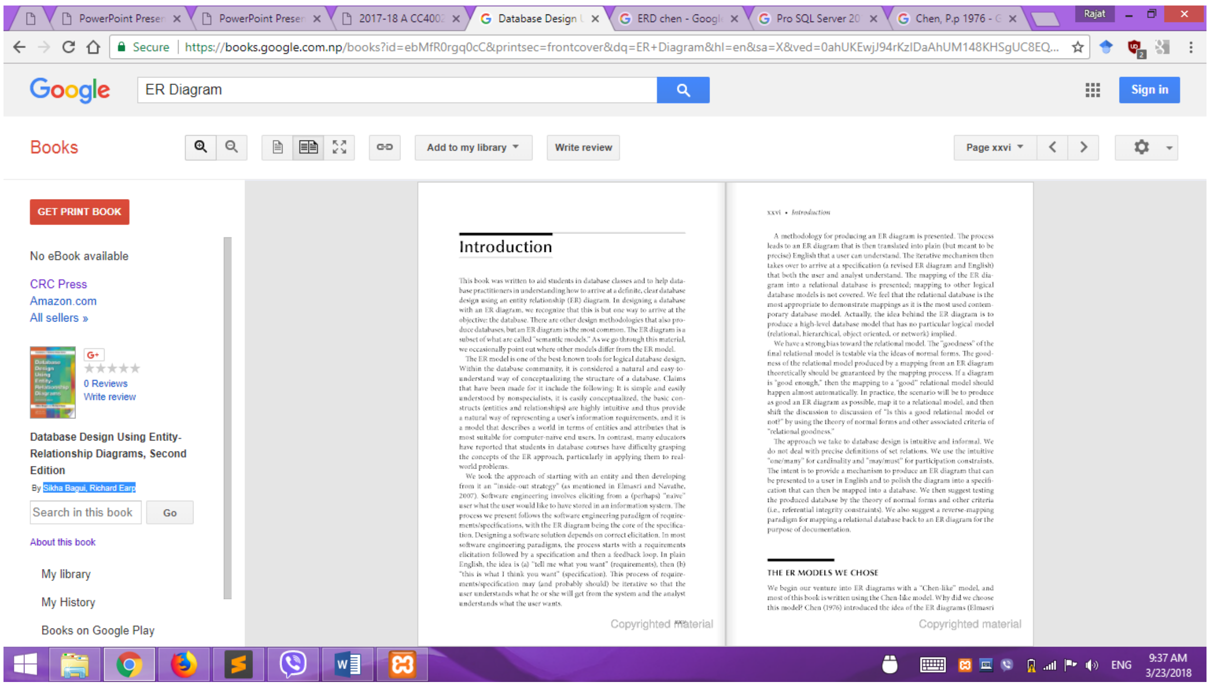
Books:

I have gone through several books online but only referenced the books in which I had found useful information were cited and included in this project. The list of books I have gone through in this process are as follows:

1. **Database Design Using Entity-Relationship Diagrams, Second Edition**

By Sikha Bagui and Richard Earp

This book contains all the required information about ER diagrams so this was an easy pick to understand more about the database modelling diagrams. (Bagui & Earp, 2011)



Introduction

This book was written to aid students in database classes and to help database practitioners in understanding how to arrive at a definite, clear database design using an entity relationship (ER) diagram. In designing a database with an ER diagram, we recognize that this is but one way to arrive at the objective: the database. There are other design methodologies that also produce databases, but an ER diagram is the most common. The ER diagram is a subset of what are called "semantic models." As we go through this material, we occasionally point out where other models differ from the ER model.

The ER model is one of the best-known tools for logical database design. Within the database community, it is considered a natural and easy-to-understand way of conceptualizing the structure of a database. Claims that have been made for it include the following: It is simple and easily understood by nonspecialists, it is easily conceptualized, the basic constructs (entities and relationships) are highly intuitive and thus provide a natural way of representing a user's information requirements, and it is a model that describes a world in terms of entities and attributes that is most suitable for computer-naïve end users. In contrast, many educators have reported that students in database courses have difficulty grasping the concepts of the ER approach, particularly in applying them to real-world problems.

We took the approach of starting with an entity and then developing from it an "inside-out strategy" (as mentioned in Elmasri and Navathe, 2007). Software engineering involves eliciting from a (perhaps) "naïve" user what the user would like to have stored in an information system. The process we present follows the software engineering paradigm of requirements/specifications, with the ER diagram being the core of the specification. Designing a software solution depends on correct elicitation. In most software engineering paradigms, the process starts with a requirements elicitation followed by a specification and then a feedback loop. In plain English, the idea is (a) "tell me what you want" (requirements), then (b) "this is what I think you want" (specification). This process of requirements/specification may (and probably should) be iterative so that the user understands what he or she will get from the system and the analyst understands what the user wants.

xxvi • *Introduction*

A methodology for producing an ER diagram is presented. The process leads to an ER diagram that is then translated into plain (but meant to be precise) English that a user can understand. The iterative mechanism then takes over to arrive at a specification (a revised ER diagram and English) that both the user and analyst understand. The mapping of the ER diagram into a relational database is presented; mapping to other logical database models is not covered. We feel that the relational database is the most appropriate to demonstrate mappings as it is the most used contemporary database model. Actually, the idea behind the ER diagram is to produce a high-level database model that has no particular logical model (relational, hierarchical, object oriented, or network) implied.

We have a strong bias toward the relational model. The "goodness" of the final relational model is testable via the ideas of normal forms. The goodness of the relational model produced by a mapping from an ER diagram theoretically should be guaranteed by the mapping process. If a diagram is "good enough," then the mapping to a "good" relational model should happen almost automatically. In practice, the scenario will be to produce as good an ER diagram as possible, map it to a relational model, and then shift the discussion to discussion of "Is this a good relational model or not?" by using the theory of normal forms and other associated criteria of "relational goodness."

The approach we take to database design is intuitive and informal. We do not deal with precise definitions of set relations. We use the intuitive "one/many" for cardinality and "may/must" for participation constraints. The intent is to provide a mechanism to produce an ER diagram that can be presented to a user in English and to polish the diagram into a specification that can then be mapped into a database. We then suggest testing the produced database by the theory of normal forms and other criteria (i.e., referential integrity constraints). We also suggest a reverse-mapping paradigm for mapping a relational database back to an ER diagram for the purpose of documentation.

THE ER MODELS WE CHOSE

We begin our venture into ER diagrams with a "Chen-like" model, and most of this book is written using the Chen-like model. Why did we choose this model? Chen (1976) introduced the idea of the ER diagrams (Elmasri

Figure 21: Database Design Using Entity-Relationship Diagrams, Second Edition

2. Database Management Systems

By Kedar, Seema

This book was quite helpful for understanding what a database was. It gave out detailed information on the familiar topics and the book discussed about the fundamentals of database and DBMS. This book gave a clear definition on what a database is and also mentioned how a database can be implemented to solve real world problems.

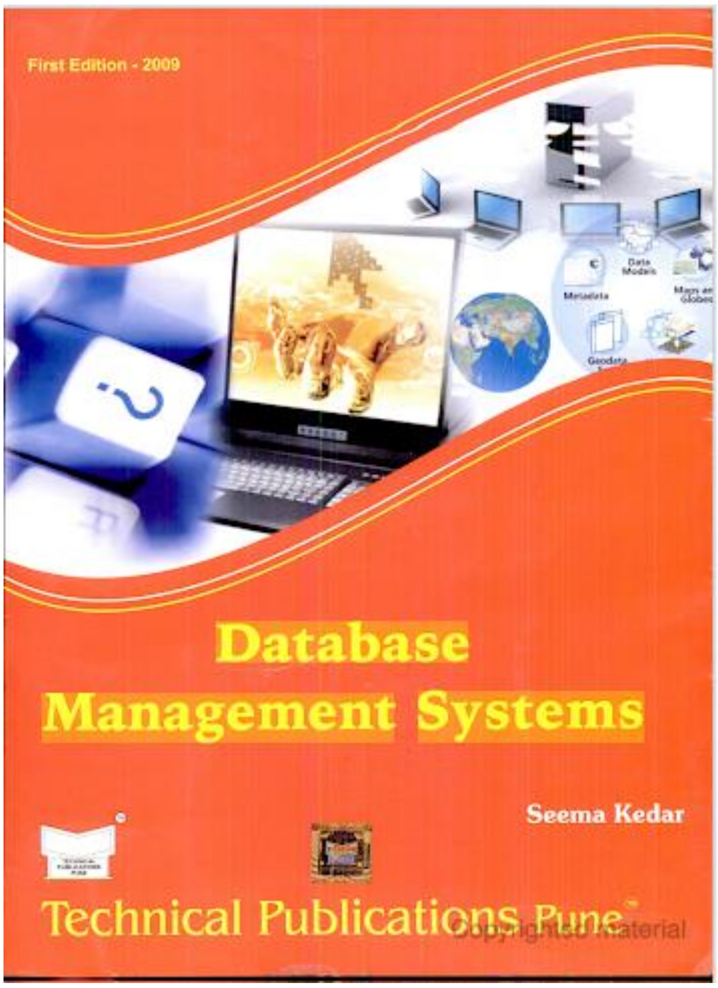
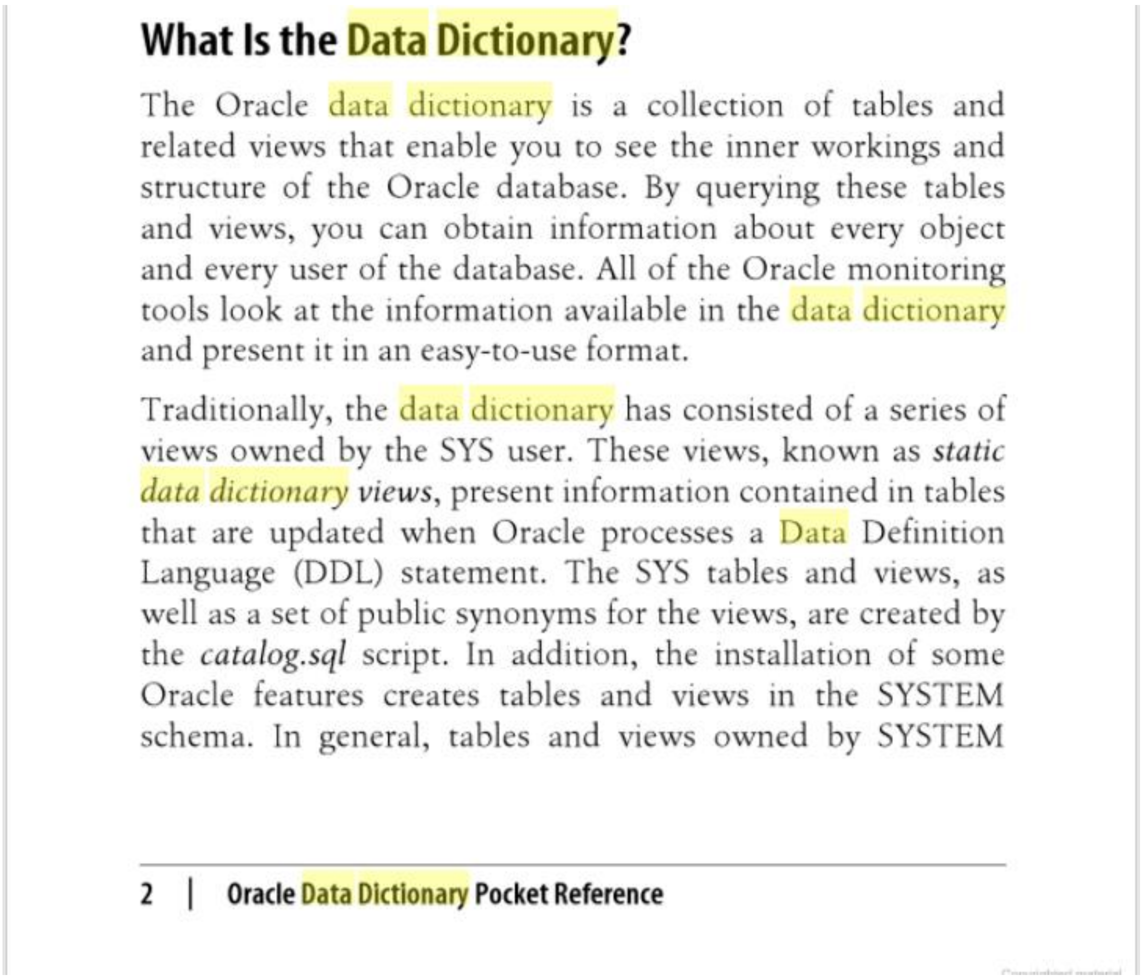


Figure 22: Book by kedar seema about DBMS

3. Oracle Data Dictionary Pocket Reference

By David Kreines

Although this book is for referencing oracle database management system, this book had the best definition of a data dictionary. It made it clear how a data dictionary is a table and other information which conveys the idea of inner mechanism of a database.



What Is the Data Dictionary?

The Oracle data dictionary is a collection of tables and related views that enable you to see the inner workings and structure of the Oracle database. By querying these tables and views, you can obtain information about every object and every user of the database. All of the Oracle monitoring tools look at the information available in the data dictionary and present it in an easy-to-use format.

Traditionally, the data dictionary has consisted of a series of views owned by the SYS user. These views, known as *static data dictionary views*, present information contained in tables that are updated when Oracle processes a Data Definition Language (DDL) statement. The SYS tables and views, as well as a set of public synonyms for the views, are created by the *catalog.sql* script. In addition, the installation of some Oracle features creates tables and views in the SYSTEM schema. In general, tables and views owned by SYSTEM

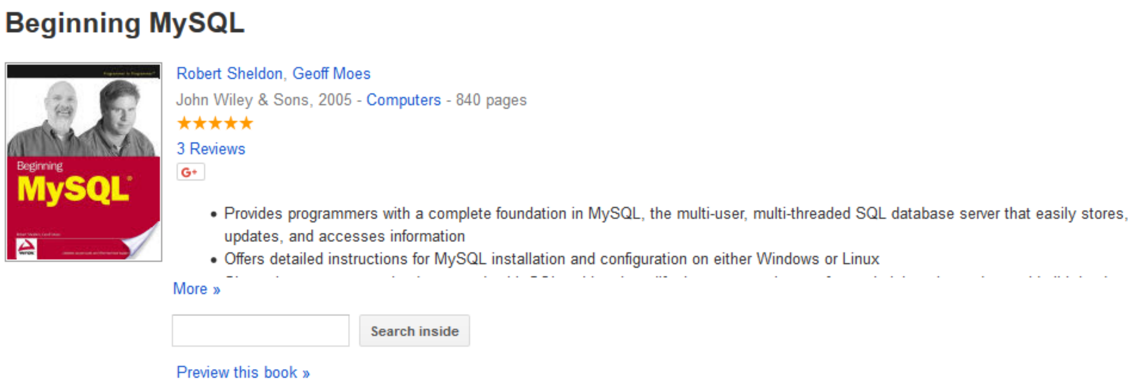
2 | **Oracle Data Dictionary Pocket Reference**

Figure 23: description of data dictionary in the book

4. Beginning MySQL

By Robert Sheldon and Geoff Moes

This book is a really great read if one is a beginner to database, DBMS and MySQL it gives out every important details and tips for beginners and is also a great reference material for intermediate or expert database managing people.


Beginning MySQL

Robert Sheldon, Geoff Moes
John Wiley & Sons, 2005 - Computers - 840 pages
★★★★★
3 Reviews

- Provides programmers with a complete foundation in MySQL, the multi-user, multi-threaded SQL database server that easily stores, updates, and accesses information
- Offers detailed instructions for MySQL installation and configuration on either Windows or Linux

More »

Figure 24: Details about beginning MySQL

5. Concepts of database management

By Phil Pratt and Mary Last

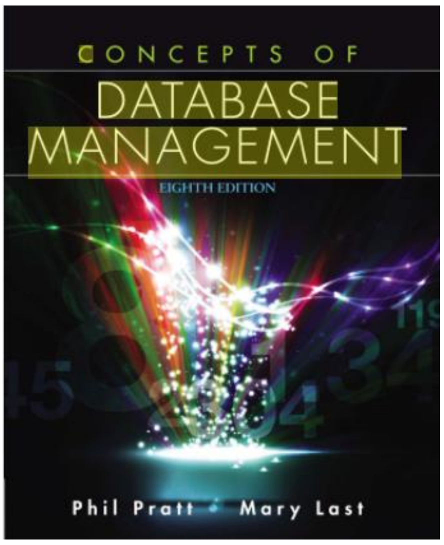


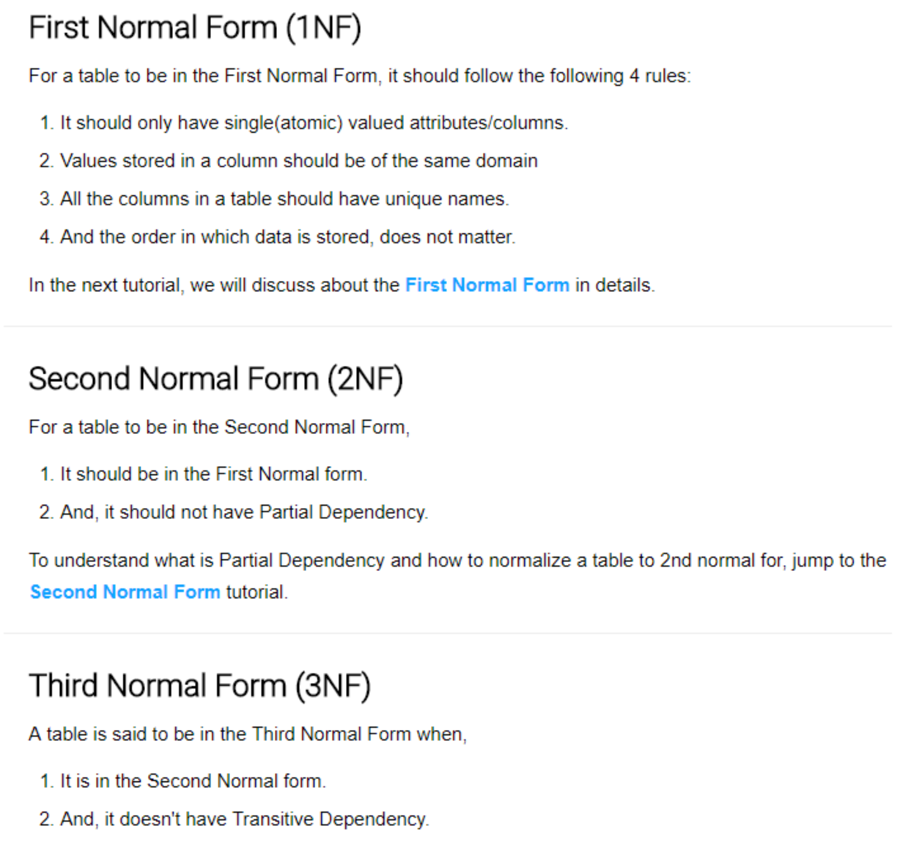
Figure 25:Concept of database management

Web-Sites:

Web sites provides vast knowledge and information in the era of computer and networks. If chosen correctly the web sites can be used to easily gain vital information on various topics. As many web sites are not quite suitable due to false information or informal methods of conveying information web sites were carefully selected and cited for this project. The websites that I have gone through for this project are as follows:

1. **Studytonight, 2018. 1NF, 2NF, 3NF and BCNF in Database Normalization**
   https://www.studytonight.com/dbms/database-normalization.php
   (Studytonight, 2018)

   this website provided simple and accurate description about the forms of normalization and helped a lot in developing the basic framework of the database to run on 3rd normal form.



First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

In the next tutorial, we will discuss about the First Normal Form in details.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

To understand what is Partial Dependency and how to normalize a table to 2nd normal for, jump to the Second Normal Form tutorial.

Third Normal Form (3NF)

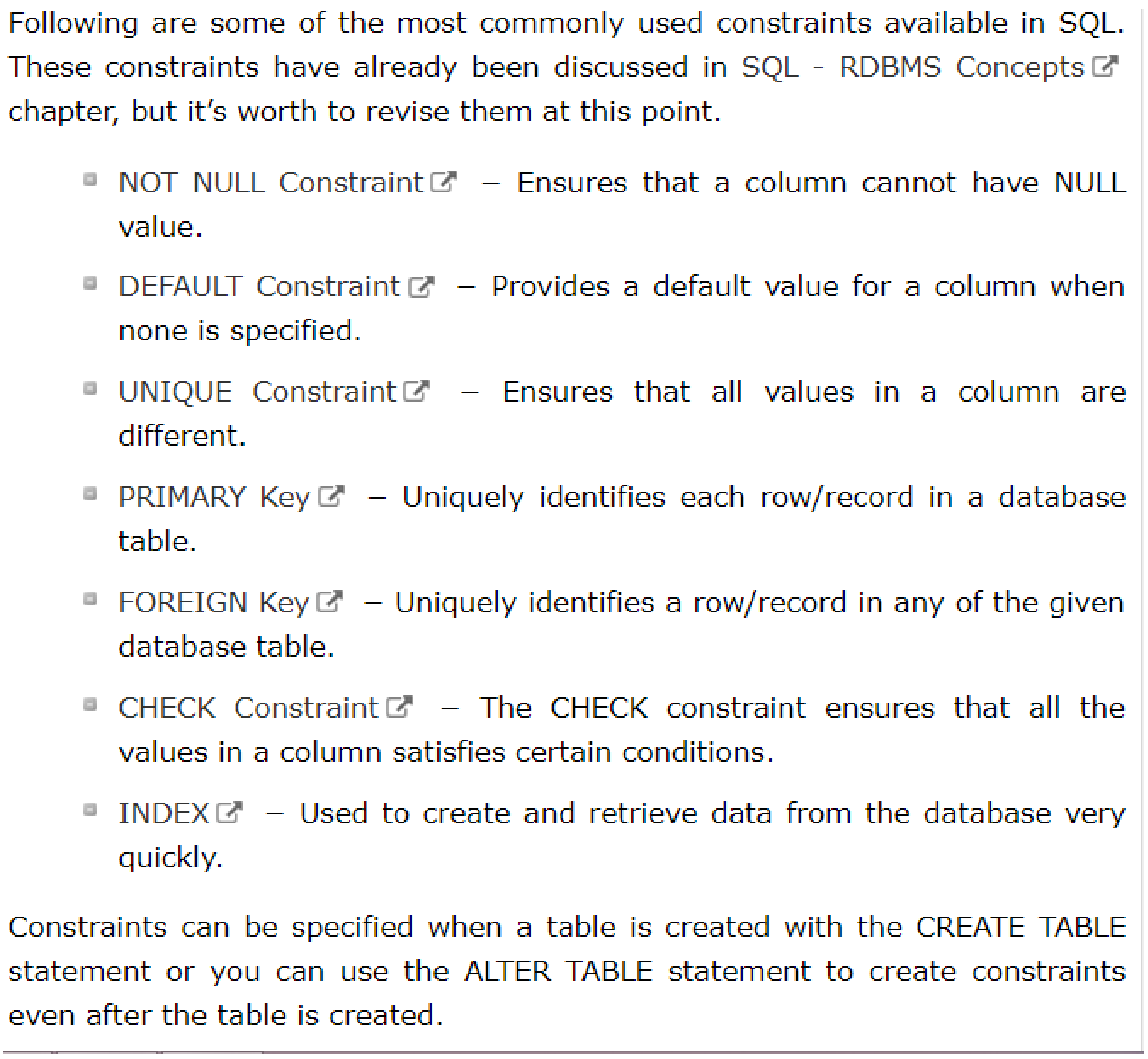
A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

Figure 26: modes of normalization in studytonight

2. Tutorials point (SQL – Constraints)

This website contained all the required information about constraints. There are many types of constraints implemented in the DBMS so knowing how to use them will result in greater efficiency.



Following are some of the most commonly used constraints available in SQL. These constraints have already been discussed in SQL - RDBMS Concepts chapter, but it's worth to revise them at this point.

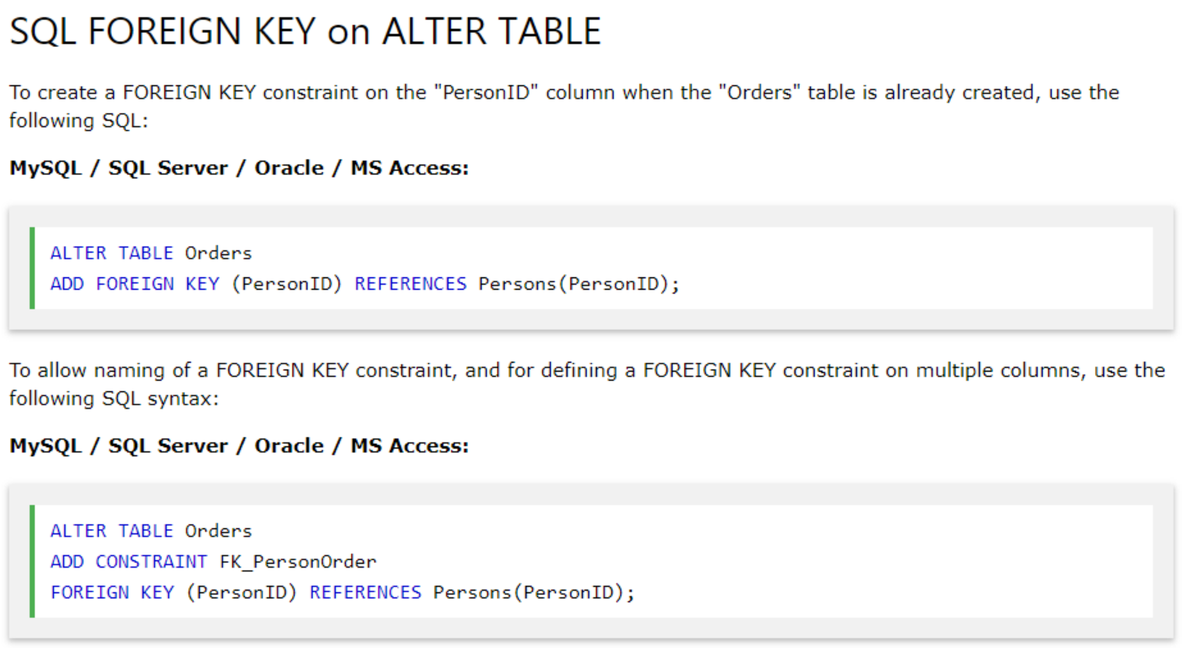
- NOT NULL Constraint – Ensures that a column cannot have NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX – Used to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

Figure 27:tutorialpoint website describing various constraints.

3. w3schools (SQL FOREIGN KEY Constraints)

This website contained all the required information about foreign keys. But not only that, it also contained various other MySQL tutorials and guides which was quite useful in terms of understanding the database approach of storing data and information. This website is very useful as a reference site to improve the database management skill.


SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

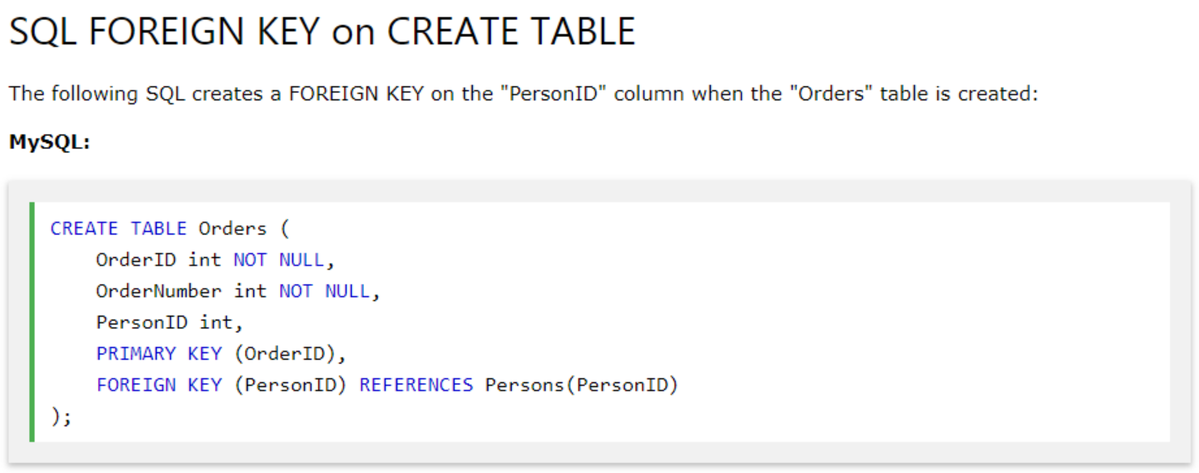
```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Figure 28: w3schools on foreign keys


SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

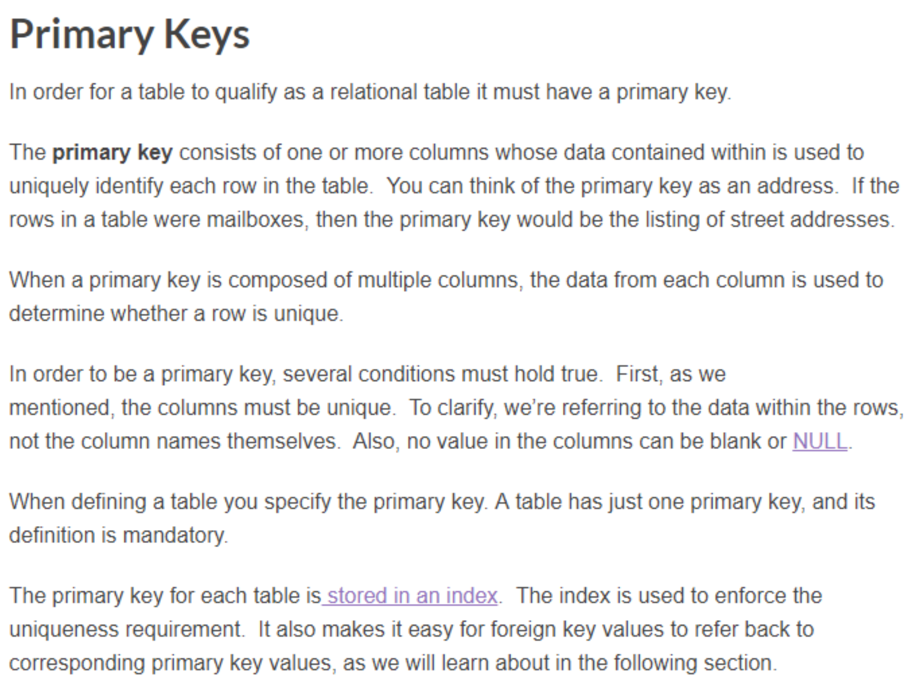
MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Figure 29: w3schools showing how to work with Foreign Key

4. Essential SQL (Differences between primary key and foreign key)

This website was very useful to understand what each constraint key did. It contained detailed information about the primary keys and foreign keys.



Primary Keys

In order for a table to qualify as a relational table it must have a primary key.

The **primary key** consists of one or more columns whose data contained within is used to uniquely identify each row in the table. You can think of the primary key as an address. If the rows in a table were mailboxes, then the primary key would be the listing of street addresses.

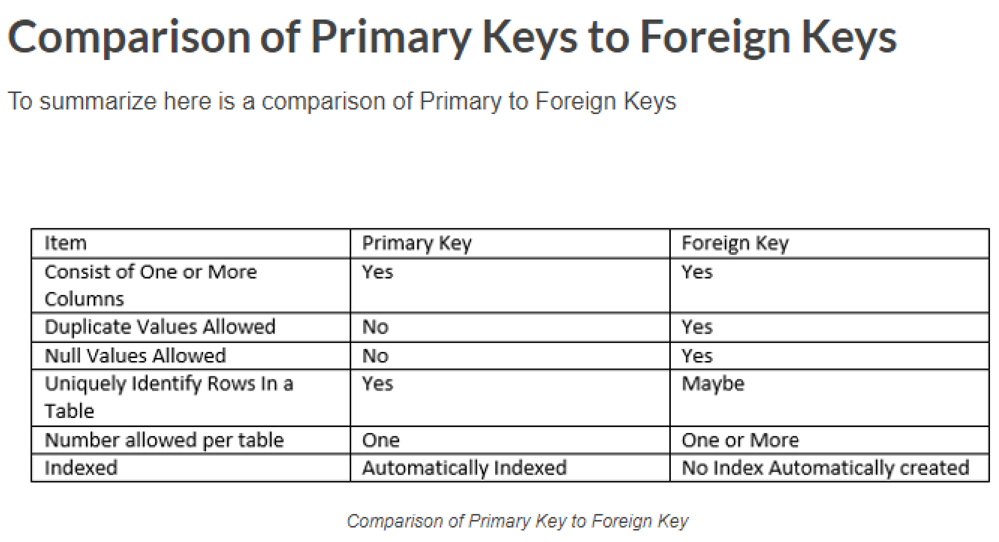
When a primary key is composed of multiple columns, the data from each column is used to determine whether a row is unique.

In order to be a primary key, several conditions must hold true. First, as we mentioned, the columns must be unique. To clarify, we're referring to the data within the rows, not the column names themselves. Also, no value in the columns can be blank or NULL.

When defining a table you specify the primary key. A table has just one primary key, and its definition is mandatory.

The primary key for each table is stored in an index. The index is used to enforce the uniqueness requirement. It also makes it easy for foreign key values to refer back to corresponding primary key values, as we will learn about in the following section.

Figure 30: About primary key



Comparison of Primary Keys to Foreign Keys

To summarize here is a comparison of Primary to Foreign Keys

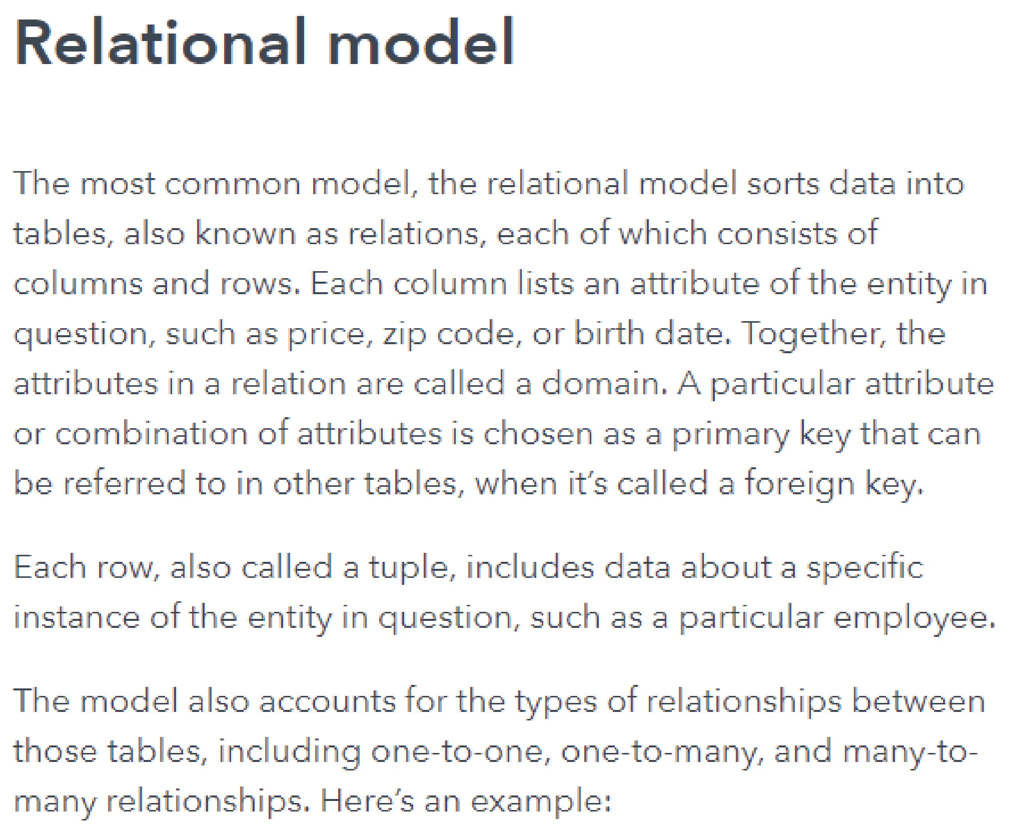
Item	Primary Key	Foreign Key
Consist of One or More Columns	Yes	Yes
Duplicate Values Allowed	No	Yes
Null Values Allowed	No	Yes
Uniquely Identify Rows In a Table	Yes	Maybe
Number allowed per table	One	One or More
Indexed	Automatically Indexed	No Index Automatically created

Comparison of Primary Key to Foreign Key

Figure 31: difference between the constraint keys

5. What is the database model (Lucid chart)

This website describes different types of database models in the current day this is where relational diagram was described quite well.



Relational model

The most common model, the relational model sorts data into tables, also known as relations, each of which consists of columns and rows. Each column lists an attribute of the entity in question, such as price, zip code, or birth date. Together, the attributes in a relation are called a domain. A particular attribute or combination of attributes is chosen as a primary key that can be referred to in other tables, when it's called a foreign key.

Each row, also called a tuple, includes data about a specific instance of the entity in question, such as a particular employee.

The model also accounts for the types of relationships between those tables, including one-to-one, one-to-many, and many-to-many relationships. Here's an example:

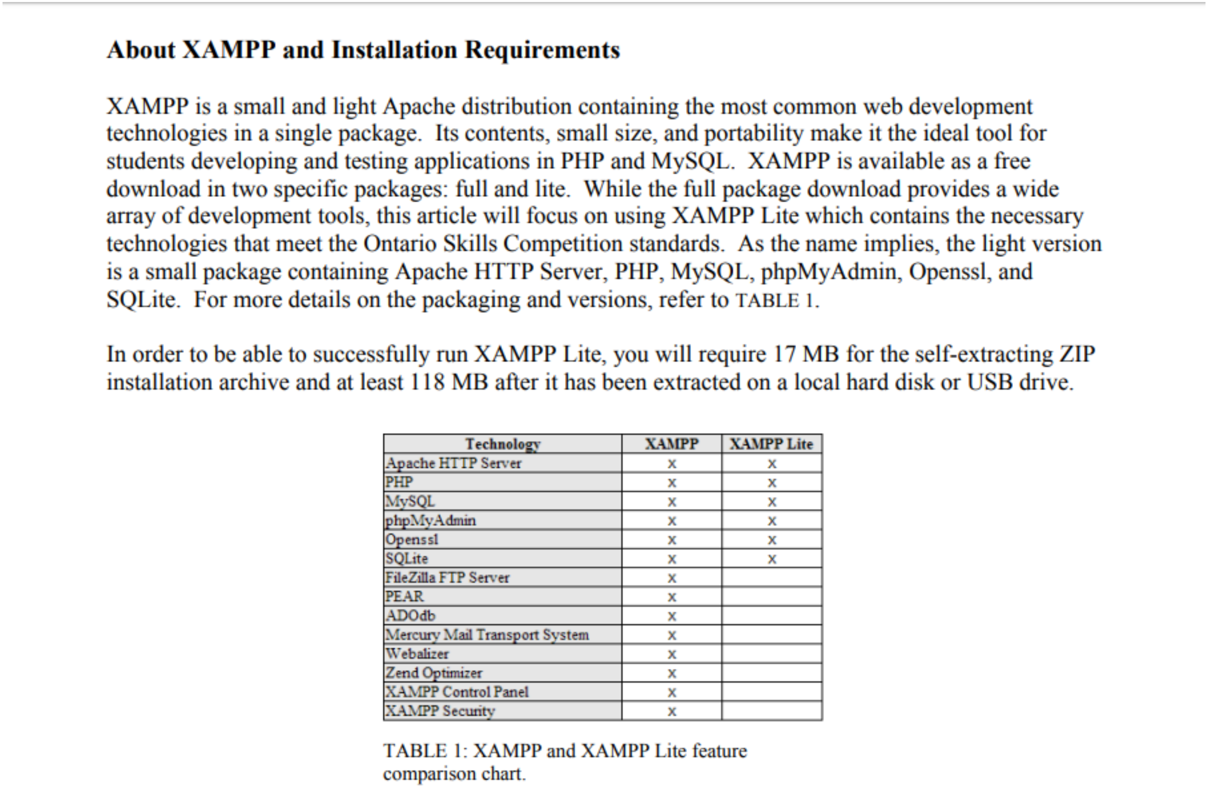
Figure 32: Relational model described by lucid charts (lucid chart, 2018)

Journals and Journal articles:

Journals provides vast knowledge and information in the form of traditional papers and articles. Journals and articles are usually legit so they can be trusted.

1. INSTALLING, CONFIGURING, AND DEVELOPING WITH XAMPP

This article written by Dalibor D. Dvorski gives detailed description on the installation, configuration and running XAMPP



About XAMPP and Installation Requirements

XAMPP is a small and light Apache distribution containing the most common web development technologies in a single package. Its contents, small size, and portability make it the ideal tool for students developing and testing applications in PHP and MySQL. XAMPP is available as a free download in two specific packages: full and lite. While the full package download provides a wide array of development tools, this article will focus on using XAMPP Lite which contains the necessary technologies that meet the Ontario Skills Competition standards. As the name implies, the light version is a small package containing Apache HTTP Server, PHP, MySQL, phpMyAdmin, Openssl, and SQLite. For more details on the packaging and versions, refer to TABLE 1.

In order to be able to successfully run XAMPP Lite, you will require 17 MB for the self-extracting ZIP installation archive and at least 118 MB after it has been extracted on a local hard disk or USB drive.

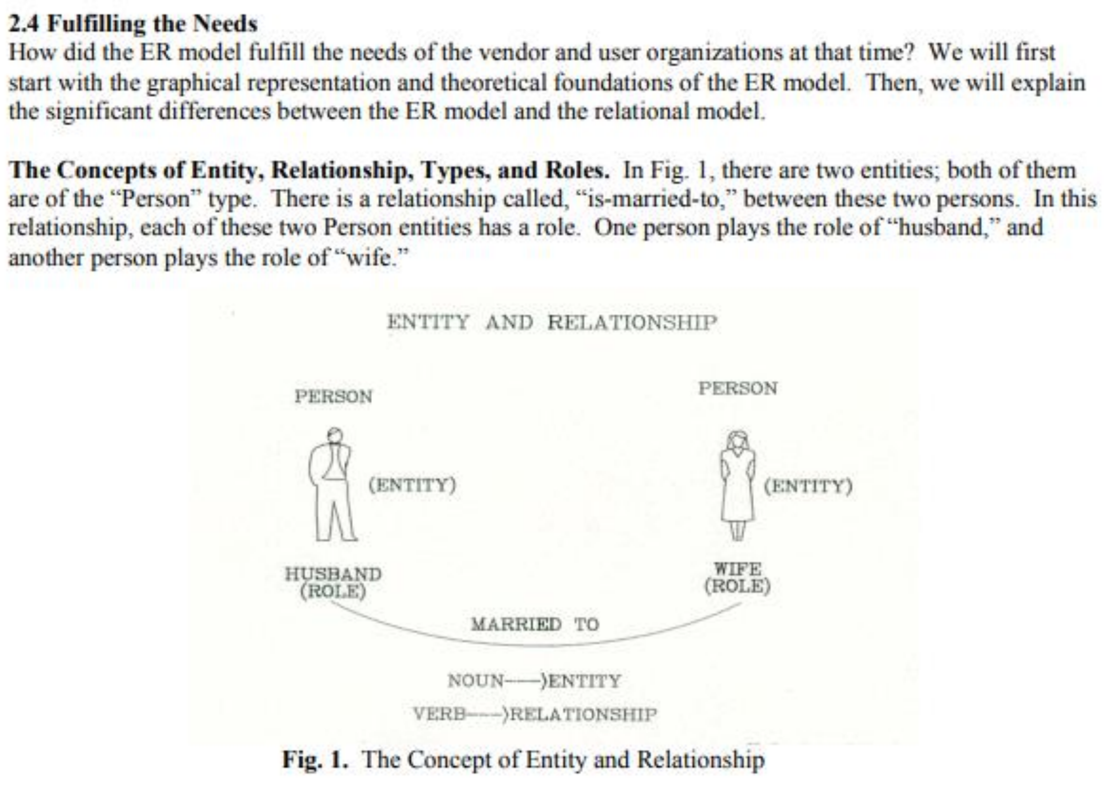
Technology	XAMPP	XAMPP Lite
Apache HTTP Server	x	x
PHP	x	x
MySQL	x	x
phpMyAdmin	x	x
Openssl	x	x
SQLite	x	x
FileZilla FTP Server	x	
PEAR	x	
ADOdb	x	
Mercury Mail Transport System	x	
Webalizer	x	
Zend Optimizer	x	
XAMPP Control Panel	x	
XAMPP Security	x	

TABLE 1: XAMPP and XAMPP Lite feature comparison chart.

Figure 33: Dalibor D. Dvorski's guide for xampp

2. Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned

This article clears out the concepts about the relations of entities and how they can be represented in graphical form.



2.4 Fulfilling the Needs
How did the ER model fulfill the needs of the vendor and user organizations at that time? We will first start with the graphical representation and theoretical foundations of the ER model. Then, we will explain the significant differences between the ER model and the relational model.

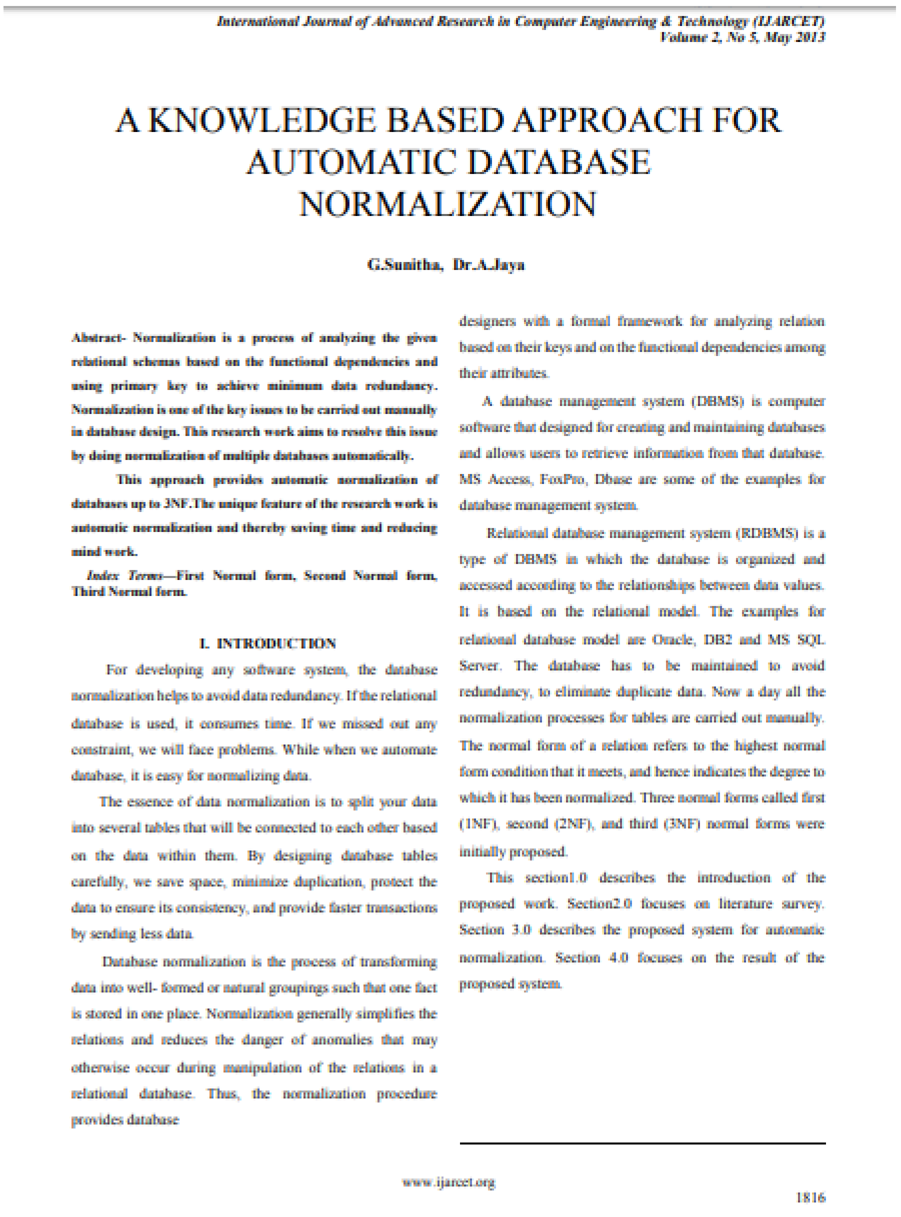
The Concepts of Entity, Relationship, Types, and Roles. In Fig. 1, there are two entities; both of them are of the "Person" type. There is a relationship called, "is-married-to," between these two persons. In this relationship, each of these two Person entities has a role. One person plays the role of "husband," and another person plays the role of "wife."

Fig. 1. The Concept of Entity and Relationship

Figure 34: Entities and relations

3. A knowledge based approach for automatic database normalization

This journal deals with data normalization and advanced techniques of data normalization but also includes vital info about the data normalization techniques and the methods to reduce data redundancy and data inconsistency.



International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, No 5, May 2013

A KNOWLEDGE BASED APPROACH FOR AUTOMATIC DATABASE NORMALIZATION

G.Sunitha, Dr.A.Jaya

Abstract- Normalization is a process of analyzing the given relational schemas based on the functional dependencies and using primary key to achieve minimum data redundancy. Normalization is one of the key issues to be carried out manually in database design. This research work aims to resolve this issue by doing normalization of multiple databases automatically.

This approach provides automatic normalization of databases up to 3NF.The unique feature of the research work is automatic normalization and thereby saving time and reducing mind work.

***Index Terms*—First Normal form, Second Normal form, Third Normal form.**

I. INTRODUCTION

For developing any software system, the database normalization helps to avoid data redundancy. If the relational database is used, it consumes time. If we missed out any constraint, we will face problems. While when we automate database, it is easy for normalizing data.

The essence of data normalization is to split your data into several tables that will be connected to each other based on the data within them. By designing database tables carefully, we save space, minimize duplication, protect the data to ensure its consistency, and provide faster transactions by sending less data.

Database normalization is the process of transforming data into well- formed or natural groupings such that one fact is stored in one place. Normalization generally simplifies the relations and reduces the danger of anomalies that may otherwise occur during manipulation of the relations in a relational database. Thus, the normalization procedure provides database designers with a formal framework for analyzing relation based on their keys and on the functional dependencies among their attributes.

A database management system (DBMS) is computer software that designed for creating and maintaining databases and allows users to retrieve information from that database. MS Access, FoxPro, Dbase are some of the examples for database management system.

Relational database management system (RDBMS) is a type of DBMS in which the database is organized and accessed according to the relationships between data values. It is based on the relational model. The examples for relational database model are Oracle, DB2 and MS SQL Server. The database has to be maintained to avoid redundancy, to eliminate duplicate data. Now a day all the normalization processes for tables are carried out manually. The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized. Three normal forms called first (1NF), second (2NF), and third (3NF) normal forms were initially proposed.

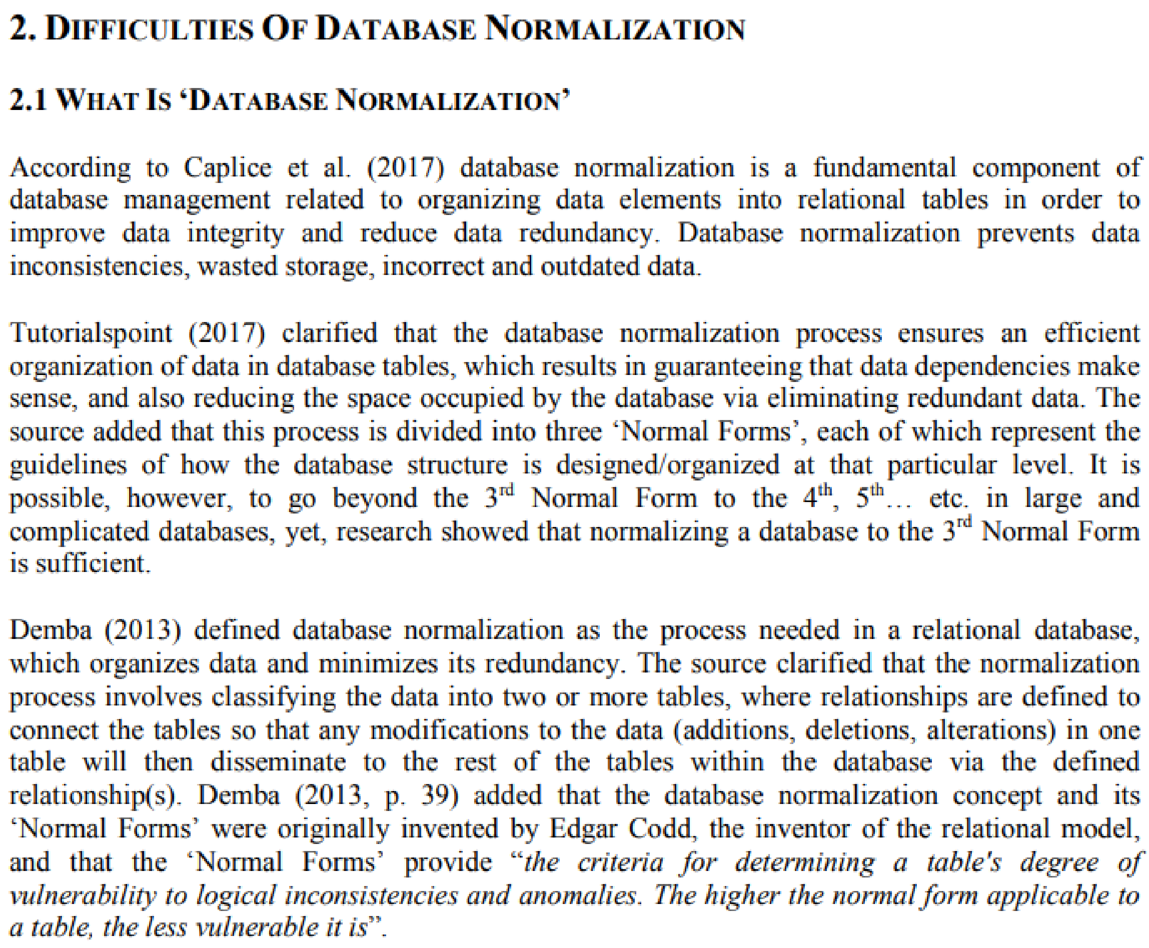
This section1.0 describes the introduction of the proposed work. Section2.0 focuses on literature survey. Section 3.0 describes the proposed system for automatic normalization. Section 4.0 focuses on the result of the proposed system.

www.ijarcet.org 1816

Figure 35: A knowledge based approach for automatic database normalization journal (Rajan, et al., 2012)

4. Simplifying database normalization within a visual interactive simulation model

This database journal deals on data normalization and discusses what data normalization is.



2. DIFFICULTIES OF DATABASE NORMALIZATION

2.1 WHAT IS 'DATABASE NORMALIZATION'

According to Caplice et al. (2017) database normalization is a fundamental component of database management related to organizing data elements into relational tables in order to improve data integrity and reduce data redundancy. Database normalization prevents data inconsistencies, wasted storage, incorrect and outdated data.

Tutorialspoint (2017) clarified that the database normalization process ensures an efficient organization of data in database tables, which results in guaranteeing that data dependencies make sense, and also reducing the space occupied by the database via eliminating redundant data. The source added that this process is divided into three 'Normal Forms', each of which represent the guidelines of how the database structure is designed/organized at that particular level. It is possible, however, to go beyond the 3rd Normal Form to the 4th, 5th… etc. in large and complicated databases, yet, research showed that normalizing a database to the 3rd Normal Form is sufficient.

Demba (2013) defined database normalization as the process needed in a relational database, which organizes data and minimizes its redundancy. The source clarified that the normalization process involves classifying the data into two or more tables, where relationships are defined to connect the tables so that any modifications to the data (additions, deletions, alterations) in one table will then disseminate to the rest of the tables within the database via the defined relationship(s). Demba (2013, p. 39) added that the database normalization concept and its 'Normal Forms' were originally invented by Edgar Codd, the inventor of the relational model, and that the 'Normal Forms' provide "*the criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is*".

Figure 36: Data Normalization (Attallah, 2017)

5. A Comparative Analysis of Entity-Relationship Diagrams

This article deals on the entity relationship diagrams.

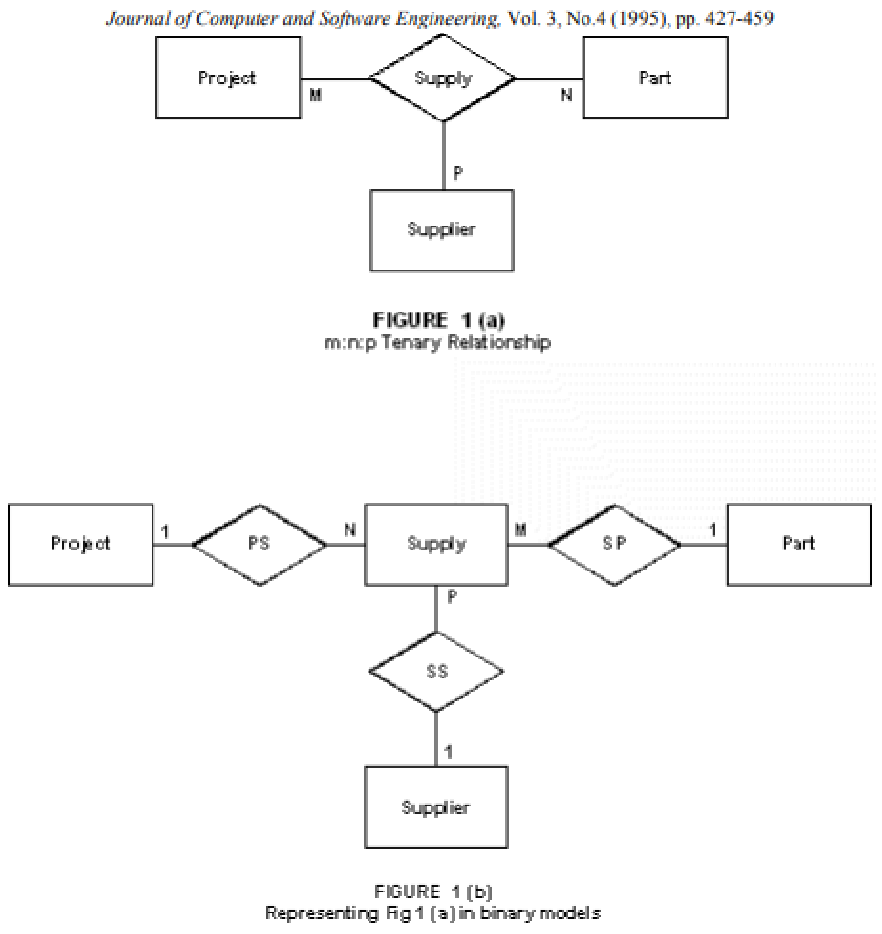

Journal of Computer and Software Engineering, Vol. 3, No.4 (1995), pp. 427-459
Project
Supply
Part
M
N
P
Supplier
FIGURE 1 (a)
m:n:p Tenary Relationship
Project
1
PS
N
Supply
M
SP
1
Part
P
SS
1
Supplier
FIGURE 1 (b)
Representing Fig 1 (a) in binary models

Figure 37: ERD from A Comparative Analysis of Entity-Relationship Diagrams

7. Conclusion:

For developing any type database, complex or simple, it is required to have deeper understanding on how a database functions and can be used to our advantage. A simple model must be created to simplify the process as it can greatly increase productivity by visually guiding us. This will help develop the database faster and makes the procedure simpler. Queries must be created for viewing and for management purposes. The development of the database should be well documented as it can help people to understand the mechanisms of the project. The database is made to be more simple and easy to understand but also contains all the required elements, skills and methods for a proper database. Even though the database is made as an example for a remittance company it is greatly meant for representing database deigns and practicality and should not be implemented in real life scenario. this database is for prototyping and research only and might not be quite useful in day to day operation however it can be tuned to be used in such a way.

To develop the database proper tools should be used for better efficiency. Sublime text is a great text editor for editing simple texts and codes. XAMPP is a simple light weight server made by apache which provides decent interface and console to control and view the database. VISIO, Excel and Word was also used to create various elements of the documentation process.

The database was tested extensively with queries to view data in different formats and imperfections and corrected accordingly. The database was modified various time each time increasing its practicality and simplicity. This resulted in creation of better database which was simple, sort but also consistent of all the skills and techniques that can be implemented in a database. Though this database is simple and minimalistic it has all the required techniques and follows all the specified objectives to accomplish the goals. The database could be made more complex and many tables could have been related to the database but it would make it quite unpractical

References

Anon., 1995. A Comparative Analysis of Entity-Relationship Diagrams. *Journal of Computer and Software Engineering,* 3(4), pp. 427-459.

Attallah, B., 2017. Simplifying database normalization within a visual interactive simulation model. *International Journal of Database Management Systems (IJDMS),* 9(3), p. 2.

Bagui, S. & Earp, R., 2011. *Database Design Using Entity-Relationship Diagrams, Second Edition.* 2nd ed. London: CRC Press.

Chen, P. P., 2007. *Historical Events, Future Trends, and Lessons Learned.* Baton Rouge: Computer Science Department.

Dvorski, D. D., 2007. *Dalibor D. Dvorski.* Ontario: Skills Canada.

Jacobson, I., Booch, G. & Rumbaugh, J., 2000. *The Unified Modeling Language User Guide.* 1st ed. Boston: Addison Wesley Longman.

Kedar, S., 2009. *Database Management Systems.* 1st ed. Delhi: Technical Publications.

Kreines, D., 2003. *Oracle Data Dictionary Pocket Reference.* 1st ed. Chicago: O'Reilly Media, Inc..

lucid chart, 2018. *What is the data base model.* [Online]
Available at: https://www.lucidchart.com/pages/database-diagram/database-models
[Accessed 21 03 2018].

Pratt, P. J. & Last, M. Z., 2014. *Concepts of Database Management.* 8th ed. Boston: Cengage Learning.

Rajan, S. N., Sinha, A. K. & Singh, J. B., 2012. A knowledge based approach for automatic database normalization. *International Journal of Information and Communication Technology Research,* 2(5), pp. 1-3.

Sheldon, R. & Moes, G., 2005. *Beginning MySQL.* 1st ed. Boston: John Wiley & Sons.

Studytonight, 2018. *1NF, 2NF, 3NF and BCNF in Database Normalization | Studytonight.* [Online]

Available at: https://www.studytonight.com/dbms/database-normalization.php
[Accessed 17 March 2018].

tutorialspoint, 2018. *SQL - Constraints.* [Online]
Available at: https://www.tutorialspoint.com/sql/sql-constraints.htm
[Accessed 20 3 2018].

w3Schools, 2018. *SQL FOREIGN KEY Constraint.* [Online]
Available at: https://www.w3schools.com/sql/sql_foreignkey.asp
[Accessed 22 3 2018].

Wenzel, K., 2018. *What is the Difference between a Primary Key and a Foreign Key?.* [Online]
Available at: https://www.essentialsql.com/what-is-the-difference-between-a-primary-key-and-a-foreign-key/
[Accessed 20 3 2018].