White Paper on a Database Design and Implementation of Marriages and Divorces.

Rashmit

16200161

COMP40725 Introduction to Relational Databases and SQL Programming,

School of Computer Science,

University College Dublin, Belfield, Dublin 4, Ireland

rashmit@ucdconnect.ie

INTRODUCTION

In this paper, a detail explanation on the Database Design of Marriage and Divorce and its implementation is provided. The Marriage and Divorce system is complex process and for a marriage or divorce to be successful, it should be legal. These laws are different for different religion and country. Hence, to have an effective storage and retrieval system these laws should be considered based on different Nationalities.

In the field of database design, getting the right design is ambiguous for who can suggest what is exactly right? "Right" means that the database is working fine, productivity is high, performance is optimal, the flexibility is present in the database schema.

"Wrong" can mean everything from operational constraints to the development of partial transactions being applied to the database. While schema design is the foundation of a well performing database taking database performance and backup is also necessary.

Database modeling and design is at the core of a healthy database management and performance foundation. Getting it right the first time will enhance the bottom line in a variety of ways. It will:

- Eliminate haphazard approaches to maintaining data.
- Minimize maintenance and upkeep of the database schema.
- Ensure data accuracy and reliability.

Hence, to get the right design is the utmost challenging. In this document, we provide a step by step analysis on an effective design from its first state of information to the final database design stage. All the major laws and design issues are kept in consideration for designing the Marriage and Divorce database. The case of a bad database design may lead to the rework on the schemas and the application performance caused by flaws in Design Implementation.

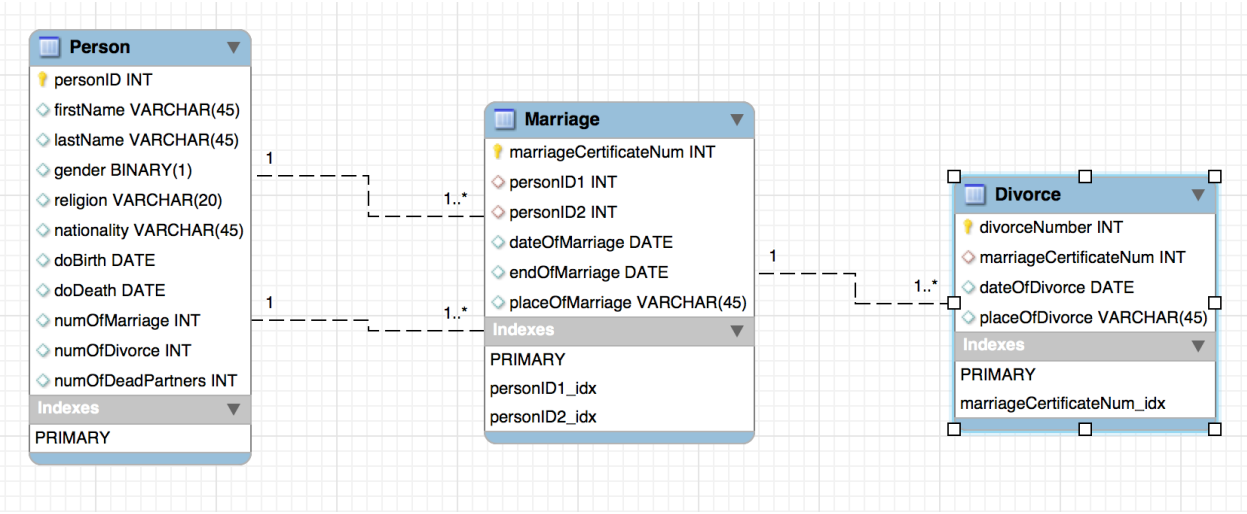
The Challenge - Designing the Right Database

"The Marriage and Divorce Database system" is designed to have an effective storage and retrieval system of the information being queried. This database system stores the person's details, relationship details i.e Marriage and Divorce which considers legal laws like the Marriageable age, polygamy, marriage between same sex.

Example - This database allows:

1. King Henry VIII to marry six times (though never twice at the same time).
2. Richard Burton and Elizabeth Taylor to marry (after each of them were divorced from other partners), then divorced, then remarried to others, then divorced from others, then remarried to each other, and then divorced again from each other.

Indeed these are complex systems and required considerable approach of design. Hence, there will be certain scope of this database. The first level of database schema is shown below:



This database design contains all the details about a person in a person table where a person can have more than one nationality. Therefore this table is not in 1NF i.e. all the attribute of the table being atomic. Similarly in the marriage details of the person is contained in the Marriage table which has many to many relationship with the person table. As two persons are required to marry

and also a person can have multiple marriages. Further, the divorce details of the person is captured in the Divorce Table. As this database has redundancy and is not normalized, this is considered as bad way of storing the data. To make it right and to abide by all the rules is a challenge.

What laws are considered in Database Design?

a. *Homosexual Marriage:* In this database, we allow same sex marriage . It does not matter if a person is male, female or a transgender. Every sex has full rights to marry any other sex. The number of combination can be shown as a cartesian product of {M,F,T} and {M,F,T}.

b. *Age:* According to the marriage laws for any nationality and religion, the person should be equal or above 18 years. This is another constraint to have a marriage in our database system.

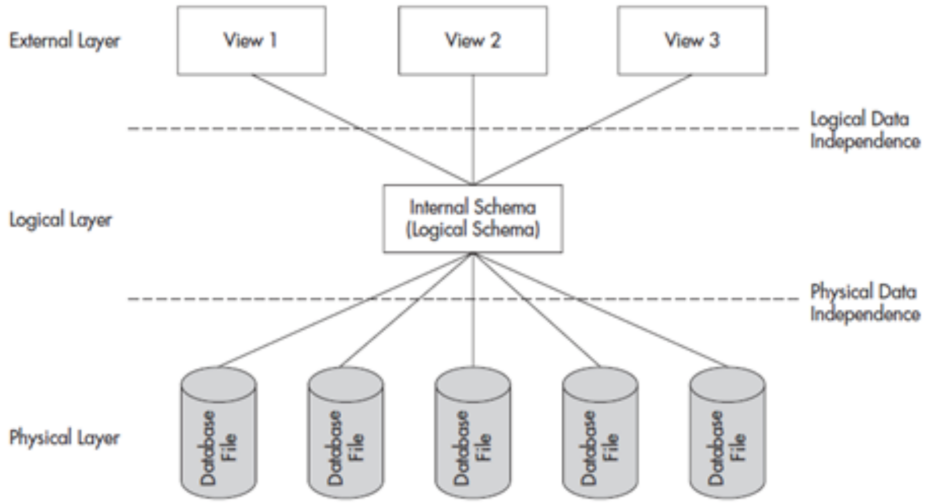
c. *Nationality:* The person can marry only in his own country. Similarly, the Divorce can only be issued in the country the marriage has taken place. In particular case, if the person marries a partner with different Nationality, it is allowed. The constraint in the database would be to always enter the partner's detail first in the database.

d. *Polygamy:* According to almost all religion today multiple marriages are not allowed. Still, Polygamy is allowed in certain religions. Our database does provide this flexibility of multiple marriage if the person's religion allow. Example - In Muslim religion, marriage up-to 4 is allowed.

e. *Death:* After death of a partner, person is eligible to marry another person. Further, The constraints in the database checks a person cannot marry a dead person or himself or to a person who is not eligible to marry.

Different Layers of Database

A database has different layers as shown in the below diagram. The physical layer contains the actual database files or tables. The logical layer contains the individual schema of the tables with the relationship with all other tables. This can be explained with the help of a Entity Relationship. The external layer contains the views which helps the process of abstraction from the end users. Using views users check the contents of the database without making changes to them.



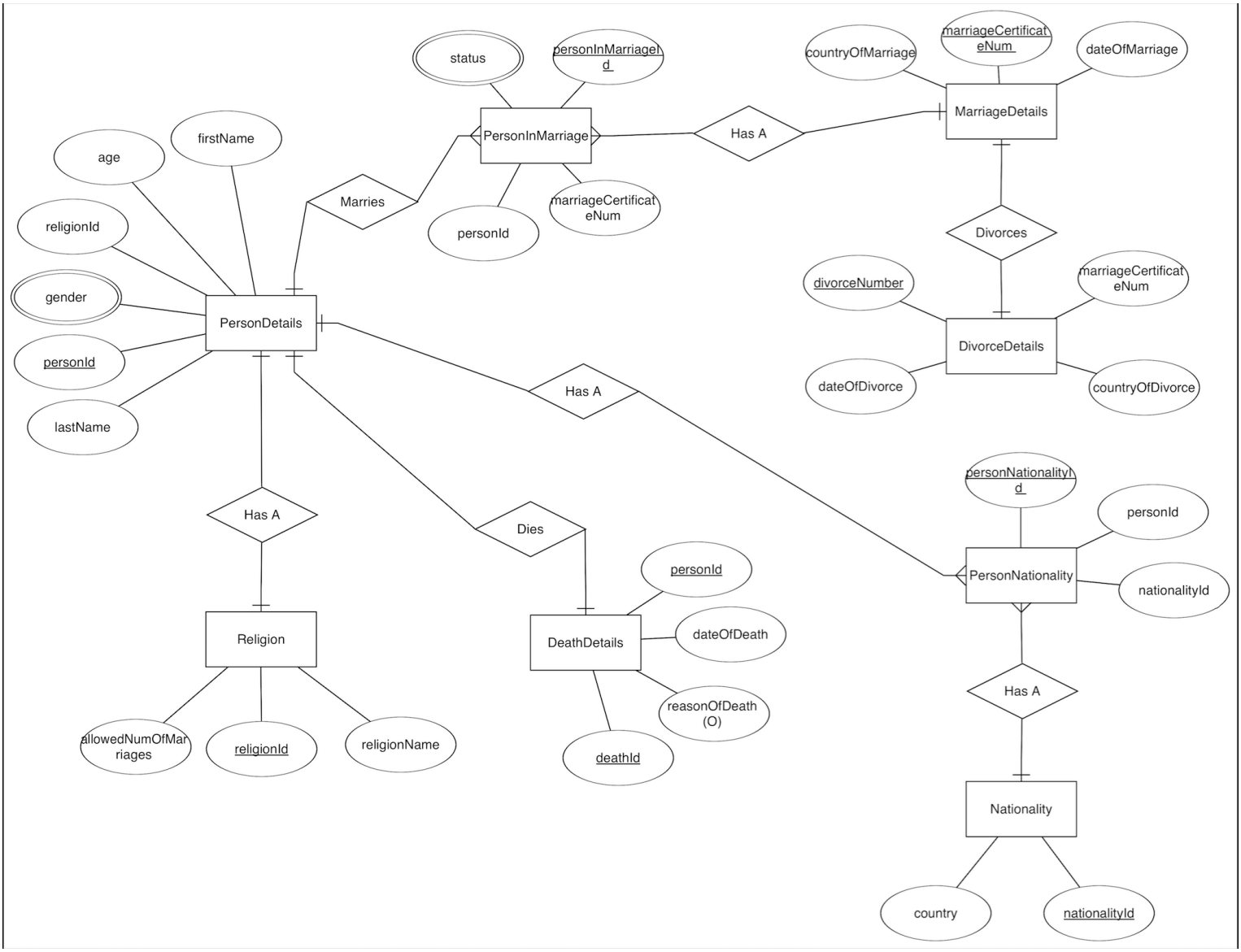
ER Diagram - The Logical Layer

The entity relationship diagram (ERD) shows the relationships of the entity sets stored in a database. ER Diagram illustrates the logical structure of the data. The ER diagram looks like a flowchart and it has a specialized symbols. Each symbol has it's own meaning.

ER diagram is connected via solid lines. These solid lines begins and ends with some special symbol which typically represents the relationship between the entities. These relationships can be One to One Relationship, One to Many Relationship, Many to One Relationship, Many to Many Relationship, Partly Optional Relationship and the Fully Optional Relationship.

The relationship between the tables are established based on how many objects of a table needs to be associated with the another.

In the below diagram, all the relationships between different entities or tables is illustrated. The entities are represented in squares while each column or attribute of the table is represented by the circles with primary key having the double circle.
The relationship between each entity with the other has a logic. Example - PersonDetails Entity has a relation ‘marries’ PersonInMarriage.

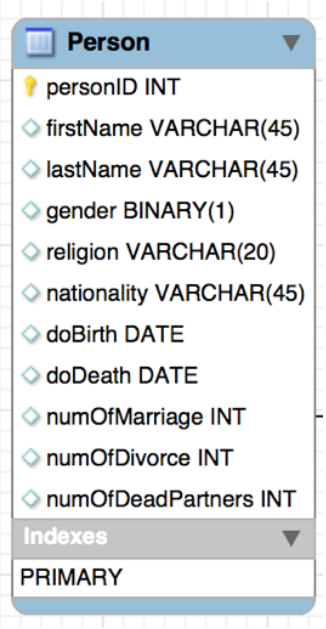


Normalization of tables from our first design:

Database Normalization is a technique of organizing the data in the database in which tables are decomposed to eliminate redundant data. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. Normalization is used for mainly two purpose i.e eliminating redundant data and ensuring data dependencies make sense i.e data is logically stored.

Before having a look on individual schema, let us take our tables from step 1 and normalize them to get an efficient Design for our database. The person table contains all the individual information which is required for his marriage or divorce.

(i) PersonDetails Table



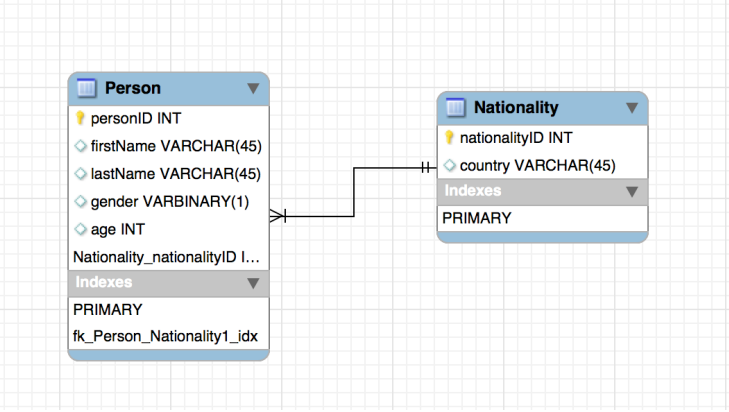
Step -1: ***1NF***

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row.

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

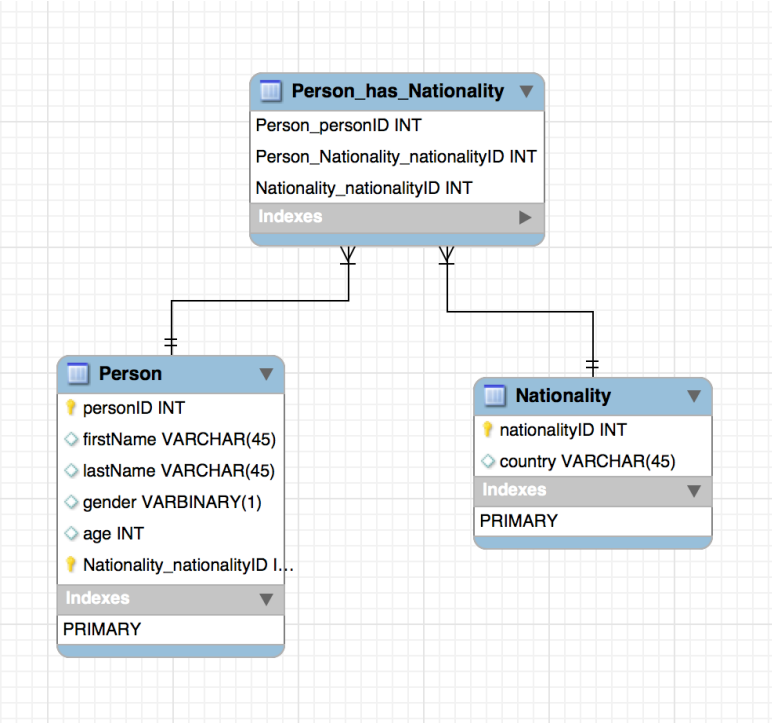
In the Person table, we have non atomic attributes nationality. Hence, creating a separate Nationality table.

Now, we have all the attributes in the table as atomic, but Person can have multiple Nationalities. Hence, this is a Many to Many relationship.



Step 2: **Considering Many to Many Relationships:**

Many-to-many relations require an intermediary table to manage the relation. The simplest implementation of the intermediary table would consist of just two columns for storing the foreign keys pointing to each related pair of records. The pivot table name should be formed by concatenating the two related model names together with an underscore separating the names.

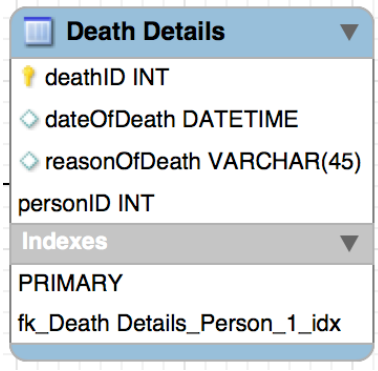


Hence, modifying this table further resulted into one more new table as shown in the diagram.

With the addition of the PersonNationality table, we now have one to many relationship with Person and Nationality. Also, this table has its own primary key and PersonId and nationalityId as the foreign key. This solves the Many to Many relationship problem.

Step 3: **2NF**

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails Second normal form.



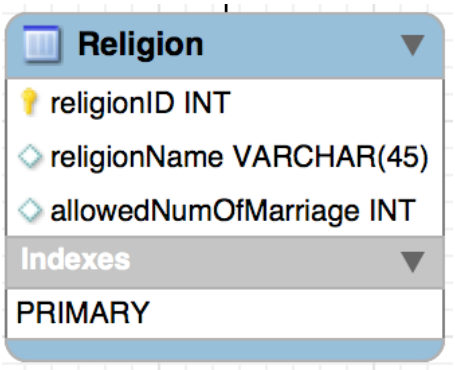
In our table, we have date of death, reason of death which is not fully dependent on the personId. Hence, to remove the partial dependency another table is created with the death Details.

Also, a person can have only one Death, hence it will have Partial one to one relationship with the PersonDetails table. The table has deathId as the primary Key which is also the index of the table and the personId as the foreign Key.

Step 4: **3NF**

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this *transitive functional dependency* should be removed from the table and also the table must be in Second Normal form.

In our table, religion name is not dependent on the personId and has a transitive dependency. Hence, creating a table which contains the religion information.

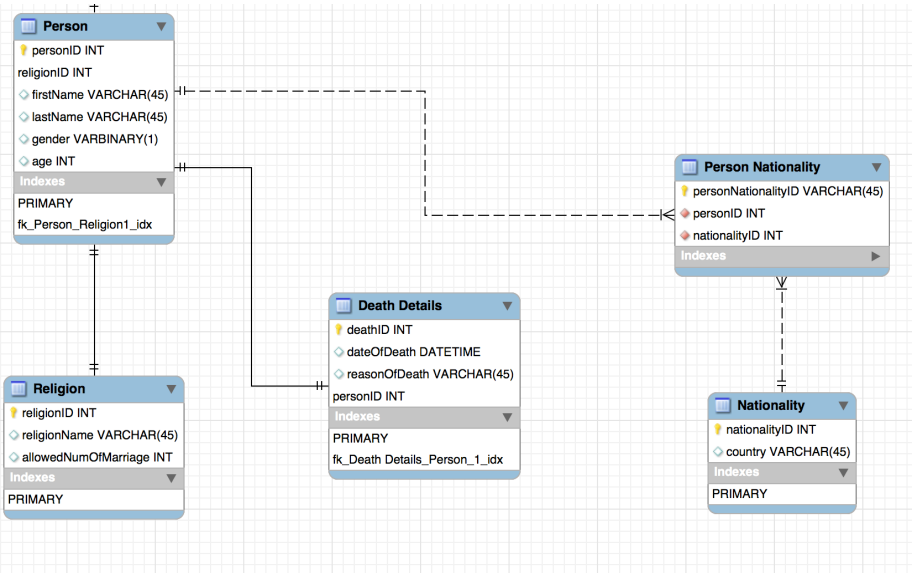


This table contains the religionId as the primary key. The table has one to one relationship with the PersonDetails table. This is because a person can be follower of one religion. This table is very important to know how many marriages does the religion allow for the person at a time.

The religionID is also the foreign Key in the PersonDetails table. Hence, to find the religion of the person the query can be possible by using religionId as index to this table.

Step 5 : **Boyce and Codd Normal Form**

This is a higher version of the Third Normal form. This 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, the conditions must be satisfied i.e. the table should be in 3rd Normal form and for each functional dependency (X -> Y), X should be a super Key.



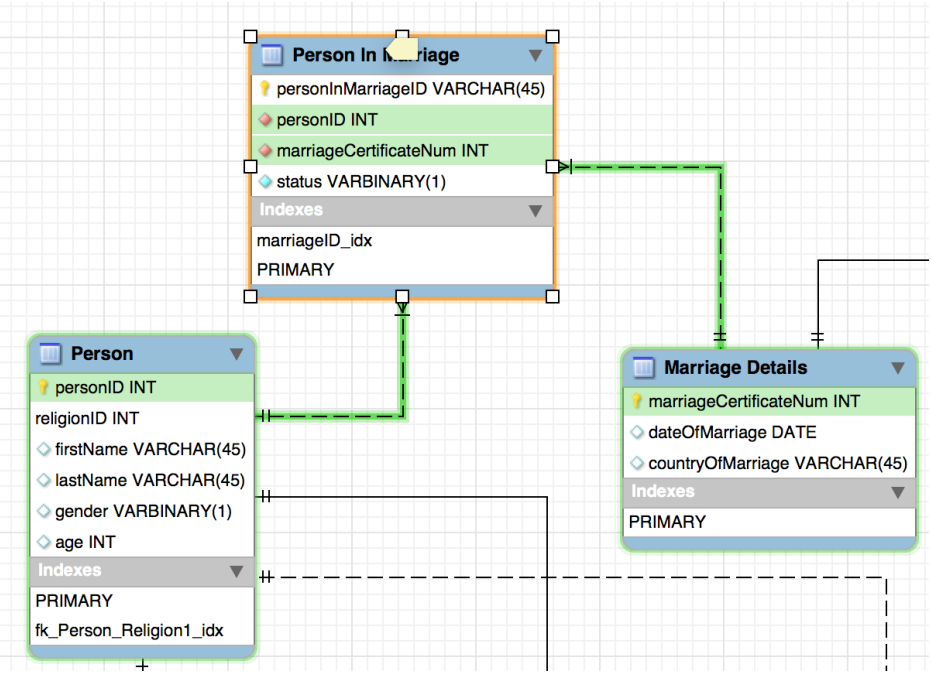
This diagram shows the normalized PersonDetails table from our previous step.

The Person table does not have any other candidate key in which our attributes are dependent. Similarly, in case of all other tables and hence, the tables are in BCNF.

(ii) **MarriageDetials Table:**

As per our database, a person can have multiple Marriages and a marriage is associated with two people. Hence, this leads us to **Many to Many relationship.**

This problem is then solved by using an Intermediary table called PersonInMarriage. This table helps to eliminate the redundant information about two people associated in marriage. Instead of creating different tables to store the information about two people, this table can be used to store the information individually by connecting them using a MarriageCertificate Number.



The MarriageCertificate number is the foreign key to this table and hence can occur twice in this table. Every person who marries will be associated with the unique PersonInMarriageId which is also the primary key and index of this table.

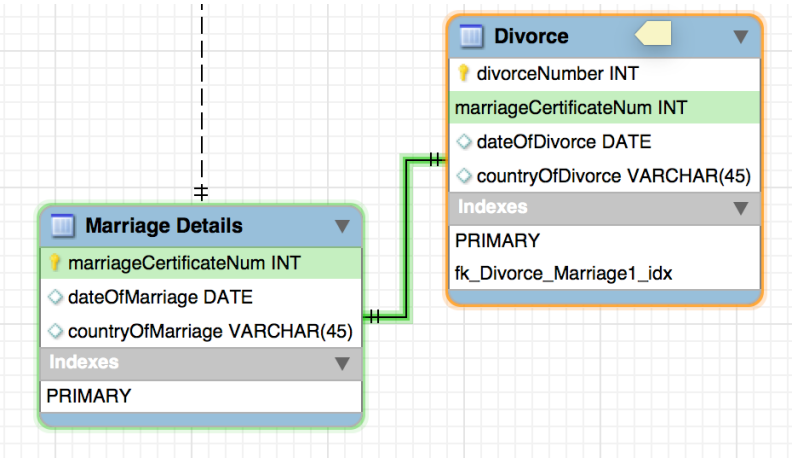
The status variable in the table is very useful in our database which tells whether a marriage is currently active or not. If the status is 'Y' i.e. the marriage is active else not.

1NF : Every attribute in PersonInMarriage and MarriageDetails is atomic in nature.
Also, In both the table there is no partial dependency and transitive dependency and hence they abide 2NF and 3NF forms.

(iii) **Divorce Details**

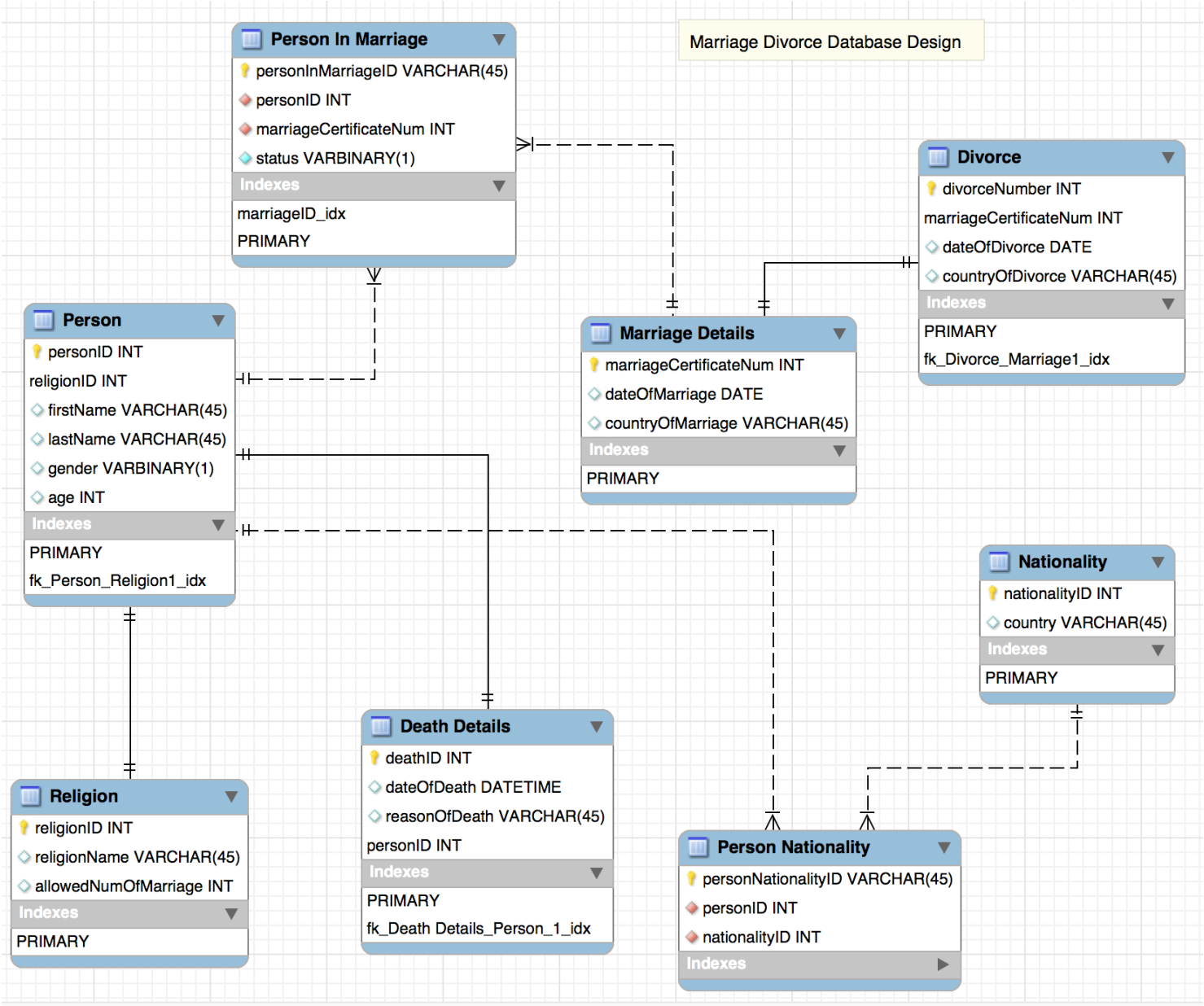
The Divorce table consists of the divorce details of the person. The divorceNumber is the primary key and index of this table.

The marriage certificate number is the foreign key of this table. It has one to one relationship with the MarriageDetails table.



As a marriageCertificate number can have only one Divorce number. All the attributes in the table are atomic, there is no partial dependency and transitive dependency and the attributes are completely dependent on the divorceId.

Below diagram represents the final Enhanced ER diagram of the database.



Definitions of Tables:

As above, we have already defined our scope and designed our relationship diagram, let us see the definitions of our tables one by one. The main focus of this section would be the usage of each table and how it helps to get all our requirements covered in the design of data. We can also determine the expected use cases of the data. This could be identifying candidate keys, the foreign keys and defining the constraints.

Also, we will focus on how our data is normalized and what are our constraints. I have created separate schema for each table to represent that where the following constraints will be referred.

NOT NULL - Ensures field cannot be empty.
Default - A default value is provided.
Primary Key - Uniquely specifies each record in the table.
Foreign Key - Uniquely specifies every record in other table.
Check - The check constraint is fulfilled by triggers. As per documentation of MySQL, check constraints are bypassed and hence as a workaround triggers will replace the checks.

Each table in the given design has a primary key - it is a unique identifier for the record. To maintain the referential integrity of the data, many tables contains foreign key. A foreign key is a field in one table which uniquely identifies a row of another table. A foreign key can be a primary key of both the table.

A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.

Tables can also have candidate keys. A candidate key can be any column or a combination of columns that qualifies as unique key in database. A table can have more than one candidate keys.

Finally, details about my database design which handle complex matters like how it doesn't allow to marry person themselves, or a polygamy marriage or to divorce a person without marrying.

1. PersonDetails -

The Person table records all the personal information of person with unique personId, person's first name, last name, gender, religionID and age. This table also has a foreign key religionID which is primary key of Religion table which can be used to check whether the person is eligible for marriage according to the number of marriages allowed in his religion.

Also, this table is normalized to 1NF - (Every record in the table is atomic and has primary key 'personId')

2NF - The columns have no partial dependency between them.
3NF - Every column is dependent only on the Primary key.
BCNF - This table has super key and hence, it is normalized to BCNF.

Further, separate table was created to log the Religion details, Marriage Details, Nationality Details of the person. A person has more than one marriage and hence, it satisfied 1:N relationship with MarriageInPerson table.

Also, to check whether a person is alive. The death information is collected in a table called death. The person table has 1:1 relation in case a person is dead.

Column Name	Type	Valid Values and Constraints	Primary Key?	Allow Nulls?	Description
personId	varchar(8)	-	Yes	No	Unique identifier for a person.
firstName	varchar(50)	-	No	No	First Name of the person
lastName	varchar(50)		No	No	Last Name of the person
gender	varbinary(1)	F or M or T	No	No	Gender of the person
religionID	varbinary(15)	-	No	No	Religion of the person
age	int	>18	No	No	age of the candidate to be married.

2. **DeathDetails**

This table consists of data of all the dead people. It only has two columns: person Id, date of death of that person and the reason of death. Once a person is dead, a marriage or divorce is not possible anymore and all the data related to that person in marriage or divorce table might be freeze.

This is a very simple table with a primary key - deathID. The table is in 1NF. We have other columns personID, reason of death and date of death just to capture details in our database. All the attributes are fully dependent on the primary key death id. There are no transitive dependency in this table making it in 3NF. This table has super key and hence, it is normalized to BCNF as well.

The personId is the foreign key in the table, to back track the partner of the died person. The partner of the died person would be eligible for another marriage.

Column Name	Type	Valid Values	Primary Key?	Allow Nulls?	Description
deathId	int	-	Yes	No	This field is an alphanumeric identifier for a divorce.
dateofdeath	date	-	No	No	Date of person's death
reasonOfDeath	varchar(45)	-	No	No	reason of death to capture the details of death.
personID	int	-	No	No	the personID of the person who is dead

(iii) Nationality

The Nationality table consists of all the required information of the person's nationality. As a person can have more than one Nationality and hence it is many to many relationship which is further sorted by PersonNationality Table.

This table doesn't have any repeating groups and has a primary key. These condition makes this table in 1NF. In this table no column is a candidate key. So there is no question of dependencies, hence this table satisfies the requirement of 2NF. Also, there is no dependency between columns with each other, hence there are no transitivity dependency. This clearly shows that it is in 3NF form as well. This table has super key and hence, it is normalized to BCNF as well.

Column Name	Type	Valid Values	Primary Key?	Allow Nulls?	Description
NationalityId	int	-	yes	No	Primary Key
Country	varchar(20)	-	No	No	Country with nationality ID

(iv) **PersonNationality**

As described above the PersonNationality table solves the issue of Many to Many relationship consists of all the foreign keys to both PersonDetails and Nationality table. This table also has its unique key and index as PersonNationalityId.

This table is atomic and has a primary key and hence these condition makes this table in 1NF. In this table no column is a candidate key. So there is no question of dependencies, hence this table satisfies the requirement of 2NF. Also, there is no dependency between columns with each other, hence there are no transitivity dependency. This clearly shows that it is in 3NF form as well. This table has super key and hence, it is normalized to BCNF as well.

Column Name	Type	Valid Values	Primary Key?	Allow Nulls?	Description
PersonNationalityId	int	-	yes	No	Primary Key
NationalityId	int	-	No	No	Foreign Key to Nationality table
PersonID	int	-	No	No	Foreign Key to Personality table

(v) **Religion**

This table consist of the very important information about the number of marriages allowed with religion name and the unique ID i.e. religion ID. The religionId attribute is the primary key of the table and also serves as index to this table and is also the foreign key to the personDetails table.

This table is atomic and has a primary key and hence these condition makes this table in 1NF. In this table no column is a candidate key. So there is no question of dependencies, hence this table satisfies the requirement of 2NF. Also, there is no dependency between columns with each other, hence there are no transitivity dependency. This clearly shows that it is in 3NF form as well. This table has super key and hence, it is normalized to BCNF as well.

Column Name	Type	Valid Values	Primary Key?	Allow Nulls?	Description
religionId	int	-	yes	No	Primary Key
religionName	varchar(45)	-	No	No	Religion name
allowedNumOfMarriages	int	-	No	No	Religion details

(vi) **MarriageDetails**

The marriage table consists of all the information about marriage. It contains marriageCertificate number and the details associated with this certificate number. The marriage certificate number is the primary key or the index of this table. This also serves as the foreign key to the personInMarriage table. Two people will be associated with every certificate number. This marriage certificate number is used to compute the marriage parter's details.

This table has constraints with the nationality of the person i.e. the person can only marry in the country he has nationality or the partner's nationality.

The table is normalized to 1NF as all the records are atomic and has MarriageNum as Primary key. Also, any column does not have any partial dependency between them. Third, they all are fully dependent on the primary key to achieve the 3NF form. This table has super key and hence, it is normalized to BCNF as well.

A Marriage may end either due to death of spouse or due to divorce and hence the divorce details was captured in another table with MarriageCertificateNum as foreign Key.
The Marriage table will have 1:1 with Divorce table in case a divorce has happened.

Column Name	Type	Valid Values/ Constraints	Primary Key?	Allow Nulls?	Description
marriageCertificateNum	int	-	Yes	No	Unique identifier for a marriage.
dateOfMarriage	date	-	No	No	Date of marriage
countryOfMarriage	varchar(45)	-	No	No	place where marriage took place

(vii) **PersonInMarriage**

As discussed above, the PersonInMarriage table solves the issue of many to many relationship. It contains marriageCertificate number which is the foreign key to the marriage details table and the details associated with this certificate number. The marriage certificate number is the primary key personID which is the foreign key to the person details table . It also has the unique primary key PersonInMarriageId.

The table is normalized to 1NF as all the records are atomic and has MarriageNum as Primary key. Also, any column does not have any partial dependency between them. Third, they all are fully dependent on the primary key to achieve the 3NF form. This table has super key and hence, it is normalized to BCNF as well.

Column Name	Type	Valid Values/ Constraints	Primary Key?	Allow Nulls?	Description
PersonInMarriageID	int	-	Yes	No	Unique identifier for a marriage.
marriageCertificateNumber	int	-	No	No	foreign key to the marriage details
personId	int	-	No	No	foreign key to the person details

(viii) **Divorce Details**

The divorce table consists of all the information about the marriageCertificateNumber which has been divorced. This table has one to one relationship with the Marriage details table as one marriage certificate can be used to divorce only once.

This table consists of divorceId as primary key and marriageCertificate number as the foreign key. It also consists of the dateOfDivorce. The constraint is date of divorce should always be greater than the date of marriage.

Divorce Id is a primary key of this table, hence all the data in this table is unique. This tells that this table is in the 1NF. In this table, the non-prime key is date of divorce.

This date of divorce depends upon the primary key divorce id but not on the marriage id which is the candidate key. Hence this table is in the 2NF. Also, there is no transitive dependency in this table. This makes this table in 3NF.

This table has super key and hence, it is normalized to BCNF as well.

Column Name	Type	Valid Values/ Constraints	Primary Key?	Allow Nulls?	Description
divorceId	int	-	Yes	No	Unique identifier for a divorce.
marriageCertificateNum	int	-	No	No	foreign key from marriage table
dod	date	-	No	No	Date of divorce of P1 and P2
countryOfDivorce	varchar(45)	Should match country of marriage	No	No	This field should match with the place of marriage.

Implementation - The Physical Layer

This physical schema or the database tables are implemented using Mysql. MySQL is the widely used open source database. MySQL is the backend database of most of the websites. The advantages of MySQL are:

1. MySQL is an open source database system. Hence it can be downloaded and used by the developer for free.
2. MySQL is robust and it provides excellent performance due to usage of MyISAM.
3. MySQL occupies very less disk space.
4. MySQL can be easily installed in all major operating systems like Microsoft Windows, Linux, UNIX.
5. MySQL can be easily learnt using the tutorials that are available on internet.
6. Though MySQL is open source, it offers most of the features provided by Oracle and other leading databases.
7. MySQL is best suited for small and medium applications.

Since MySQL is acquired by Sun, Java is soon expected to include enhanced MySQL connectivity.

But MySQL has drawbacks as well. Out of few the one which have a direct impact in the database implementation is:

MySQL does not comply with the full SQL standards for some of the implemented functionality, including foreign key references and check constraints. [References: InnoDB and FOREIGN KEY Constraints and Bug #3464: Constraints: support CHECK]

Here it is clearly mentioned to use triggers as work around for MySQL check constraints. We're addressing several constraints for this complex database design.

So as a workaround we have used triggers instead of check constraints. A **trigger** can be set to activate either before or after the **trigger** event.

For example, we can have a **trigger** activate before each row that is inserted into a table or after each row that is updated. **MySQL triggers** activate only for changes made to tables by SQL statements. There are several advantages of using MySQL triggers:

1. SQL triggers provide an alternative way to check the integrity of data. MySQL doesn't recognize the check constraint and bypass it while parsing. Hence to check any condition, it's good to use triggers.
2. They can also catch errors in business logic in the database layer.
3. SQL triggers provide an alternative way to run scheduled tasks.
4. SQL triggers are very useful to audit the changes of data in tables.

We've various constraints which we can check before entering or after entering the values into the tables. Few of them are:

1. Marriage of a person allowed only for age > 18.
2. Marital status needs to be updated in the personInMarriageTable once the divorce has happened or in case of death of a person.
3. The place of marriage and divorce must be the same.

4. The date of divorce should be higher than the date of marriage.
5. The place of marriage must be same to the nationality of either partners.
6. Restricting polygamy marriages if person's religion doesn't allow.
7. Restricting person to marry oneself or with dead person.

The MySQL's SIGNAL SQLSTATE statement's is used to return an error/warning condition to the caller from a stored program.

They are written with the customized messages with each SIGNAL SQLSTATE statement and making it to point to the specific error that where the error has occurred and why. Notice that 45000 is a generic SQLSTATE value that illustrates an unhandled user-defined exception. Triggers also used delimiters.

Also used the delimiter as $$, it could be any sign. It changes the statement delimiter from ; to $ $. By doing this, we can write ; in the trigger definition without the MySQL client misinterpreting that as meaning we're done with it. In the end of the trigger we set delimiter back to ;.

Creation of Tables: DATA DEFINITIONS

Step 1: Check if the database with the same name exit, then drop it
Step 2: Create a new database and use that database.

```
mysql> # Drop database if already exists
mysql> drop database if exists marriage_divorce_DB;
Query OK, 0 rows affected (0.00 sec)

mysql> # Create database if not exists
mysql> create database if not exists marriage_divorce_DB;
Query OK, 1 row affected (0.00 sec)

mysql> # Use database we created just now
mysql> use marriage_divorce_DB;
Database changed
```

Step 3: Now, we are ready to create tables into this database. The following tables will be created:

1. NationalityDetails
2. ReligionDetails
3. PersonNationality
4. PersonDetails
5. DeathDetails
6. MarriageDetails
7. PersonInMarriage
8. DivorceDetails

1. **NationalityDetails** - This table has the primary key natioanlityId and NULL constraints.

```
mysql> # Table: Nationality
mysql> # Primary Key: nationalityId
mysql> create table Nationality(
    -> nationalityId int NOT NULL,
    -> country varchar(45) NOT NULL,
    -> primary key(nationalityId)
    -> );
```

2. **ReligionDetails** - This table has primary key - religionId and NULL constraints.

```
mysql> # Table: Religion
mysql> # Primary Key: religionId
mysql> create table Religion(
    -> religionId int NOT NULL,
    -> religionName varchar(45) NOT NULL,
    -> allowedNumOfMarriages int NOT NULL,
    -> primary key(religionId)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

3. **PersonNationality** - This table has personalityNationalityId as primary key, nationalityId as foreign key and Null constraints.

```
# Table: PersonNationality
# Primary Key: personNationalityId
# Foreign Key: personId
# Foreign Key: nationalityId
create table PersonNationality(
        personNationalityId int NOT NULL,
        personId int NOT NULL,
        nationalityId int NOT NULL,
        primary key(personNationalityId),
        foreign key(personId) REFERENCES PersonDetails(personId),
        foreign key(nationalityId) REFERENCES Nationality(nationalityId)
);
```

4. **PersonDetails** - This table has personId as primary key, religionId as foreign key Null constraints.

```
mysql> # Table: PersonDetails
mysql> # Primary Key: personId
mysql> # Foreign Key: religionId
mysql> create table PersonDetails(
    -> personId int NOT NULL,
    -> religionId int NOT NULL,
    -> firstName varchar(45) NOT NULL,
    -> lastName varchar(45) NOT NULL,
    -> gender varbinary(1) NOT NULL,
    -> age int NOT NULL,
    -> primary key(personId),
    -> foreign key(religionId) REFERENCES Religion(religionId)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

5. **DeathDetails** - This table has deathId as primary key, personId as foreign key and has NULL constraints.

```
mysql> # Table: DeathDetails
mysql> # Primary Key: deathId
mysql> # Foreign Key: personId
mysql> create table DeathDetails(
    -> deathId int NOT NULL,
    -> personId int NOT NULL,
    -> dateOfDeath date NOT NULL,
    -> reasonOfDeath varchar(100),
    -> primary key(deathId),
    -> foreign key(personId) REFERENCES PersonDetails(personId)
    -> );
```

6. **MarriageDetails** - This table contains the marriagecertificateNumber as primary key and null constraints.

```
mysql> # Table: MarriageDetails
mysql> # Primary Key: marriageCertificateNum
mysql> create table MarriageDetails(
    -> marriageCertificateNum int NOT NULL,
    -> dateOfMarriage date NOT NULL,
    -> countryOfMarriage varchar(45) NOT NULL,
    -> primary key(marriageCertificateNum)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

7. **PersonInMarriage** - This table contains the marriagecertificateNumber, personId as foreign key , personMarriageId as primary key and null constraints.

```
mysql> # Table: PersonInMarriage
mysql> # Primary Key: personInMarriageId
mysql> # Foreign Key: personId
mysql> # Foreign Key: marriageCertificateNum
mysql> create table PersonInMarriage(
    -> personInMarriageId int NOT NULL,
    -> personId int NOT NULL,
    -> marriageCertificateNum int NOT NULL,
    -> status varbinary(1) NOT NULL,
    -> primary key(personInMarriageId),
    -> foreign key(personId) REFERENCES PersonDetails(personId),
    -> foreign key(marriageCertificateNum) REFERENCES MarriageDetails(marriageCertificateNum)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

8. DivorceDetails - This table contains the marriagecertificateNumber as foreign key,

```
mysql> # Table: DivorceDetails
mysql> # Primary Key: divorceNumber
mysql> # Foreign Key: marriageCertificateNum
mysql> create table DivorceDetails(
    -> divorceNumber int NOT NULL,
    -> marriageCertificateNum int NOT NULL,
    -> dateOfDivorce date NOT NULL,
    -> countryOfDivorce varchar(45) NOT NULL,
    -> primary key(divorceNumber),
    -> foreign key(marriageCertificateNum) REFERENCES MarriageDetails(marriageCertificateNum)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

divorceNumber as primary key and null constraints.

Step -4 After the successful creation of tables, triggers were created and executed. Following are the triggers:

1. Trigger to check if the person age is > 18

```
mysql> # Trigger: To check if age of a person is greater than or equal to 18
mysql> delimiter $$
mysql> create trigger ageCheck before insert on PersonDetails
    -> for each row
    -> begin
    -> # Disallow people in PersonDetails who are younger than 18 years of age
    ->   if NEW.age < 18 then
    ->     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "age should be > 18.";
    ->   end if;
    -> end; $$
Query OK, 0 rows affected (0.03 sec)
```

2. Trigger to update the personInMarriage table the partner is dead

```
mysql> # Trigger: Update status of partners in PersonInMarriage to "N" after death.
mysql> delimiter $$
mysql> create trigger afterDeath after insert on DeathDetails
    -> for each row
    -> begin
    -> declare mID int;
    -> # On insertion of persionId into DeathDetails, fetch that personID to get marriageCertificateNum
    -> # Convert status of people associated with this marriageCertificateNum to "N"
    -> select marriageCertificateNum into mID from PersonInMarriage where PersonInMarriage.personId = NE
W.personId;
    -> update PersonInMarriage set PersonInMarriage.status = 'N' where PersonInMarriage.marriageCertific
ateNum = mID;
    -> end; $$
Query OK, 0 rows affected (0.04 sec)
```

3. Trigger to update the personInMarriage table once the divorce has happened for a particular marriage certificate number or if

```
mysql> # Trigger: Update status of partners in PersonInMarriage to "N" after divorce.
mysql> delimiter $$
mysql> create trigger afterDivorce before insert on DivorceDetails
    -> for each row
    -> begin
    -> # Convert status of people associated with this marriageCertificateNum to "N"
    -> update PersonInMarriage set PersonInMarriage.status = 'N' where PersonInMarriage.marriageCertific
ateNum = NEW.marriageCertificateNum;
    -> end; $$
Query OK, 0 rows affected (0.04 sec)
```

4. Trigger to check if the place of divorce is same as the place of marriage and if divorce date is after the marriage date.

```
mysql> create trigger beforeDivorce before insert on DivorceDetails
    -> for each row
    -> begin
    -> declare com varchar(45);
    -> declare dom date;
    ->
    -> # Fetch the place of marriage of a person based on marriageCertificateNum.
    -> # Restrict divorce if place of divorce is different than place of marriage.
    -> select countryOfMarriage into com from MarriageDetails where NEW.marriageCertificateNum = Marriag
eDetails.marriageCertificateNum;
    -> select dateOfMarriage into dom from MarriageDetails where NEW.marriageCertificateNum = MarriageDe
tails.marriageCertificateNum;
    ->
    -> if NEW.countryOfDivorce != com then
    -> SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Divorce place must be same as Marriage Place.";
    -> end if;
    ->
    -> if NEW.dateOfDivorce < dom then
    -> SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Divorce cannot happen before marriage";
    -> end if;
    ->
    -> end; $$
Query OK, 0 rows affected (0.02 sec)
```

5. Trigger to check if a person is eligible for marriage i.e. polygamy check, dead partner.. etc.

```
mysql> create trigger marriageCount before insert on PersonInMarriage
    -> for each row
    -> begin
    -> declare rn int;
    -> declare am int;
    -> declare cms int;
    -> declare cn1 varchar(45);
    -> declare d int;
    -> # get the count of marriages allowed in a religion of person
    -> select religionId into rn from PersonDetails where PersonDetails.personId = NEW.personId;
    -> select allowedNumOfMarriages into am from Religion where Religion.religionId = rn;
    -> select count(status) into cms from PersonInMarriage where PersonInMarriage.personId = NEW.personI
d and PersonInMarriage.status = "Y";
    ->
    -> # get the country of marriage of person
    -> select countryOfMarriage into cn1 from MarriageDetails where marriageCertificateNum IN (select ma
rriageCertificateNum from PersonInMarriage where personId = NEW.personId);
    ->
    -> # get the person_id from DeathDetails
    -> select deathId into d from DeathDetails where DeathDetails.personId =  NEW.personId;
    ->
    -> # check if number of marriages allowed in religion is equals to current number of marriage relati
onships
    -> # it it is equal, restrict person to marry again.
    -> if cms = am then
    -> SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Allowed limit exceeded.";
    -> end if;
    -> # check if nationality of the person is equal to the marriage place
    -> # if not, then restrict marriage based on nationality issues.
```

Step -5 Data Manipulation Language- Insert Data into tables

Religion Table:

```
mysql> #Insert Religion_Details
mysql> insert into Religion values(1,'Bhuddism',1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into Religion values(2,'Sikhism',1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into Religion values(3,'Jainism',1);
Query OK, 1 row affected (0.00 sec)
```

Nationality Table

```
mysql> #Insert Nationality_Details
mysql> insert into Nationality values(1,'Ireland');
Query OK, 1 row affected (0.00 sec)

mysql> insert into Nationality values(2,'India');
Query OK, 1 row affected (0.00 sec)
```

PersonDetails Table

```
mysql> #Insert PersonDetails
mysql> insert into PersonDetails values(1,2, 'Andreas','Themis','M',37);
Query OK, 1 row affected (0.00 sec)

mysql> insert into PersonDetails values(2,3, 'Harbajan','Singh','M',28);
Query OK, 1 row affected (0.00 sec)
```

Nationality Table

```
mysql> # Insert Personality Nationality
mysql> insert into PersonNationality values(1,1,2);
Query OK, 1 row affected (0.00 sec)

mysql> insert into PersonNationality values(2,19,3);
Query OK, 1 row affected (0.00 sec)
```

PersonInMarriage Table

```
mysql> insert into PersonInMarriage values(1, 1, 111, 'Y');
Query OK, 1 row affected (0.00 sec)

mysql> insert into PersonInMarriage values(2, 5, 111, 'Y');
Query OK, 1 row affected (0.00 sec)
```

Step - 6 Deleting the marriages in which only two people are not associated.

```
mysql> # Delete people from PersonInMarriage when only 1 marriageCertificateNum is present
mysql> delete from PersonInMarriage
    -> where marriageCertificateNum in (
    -> select marriageCertificateNum from (select * from PersonInMarriage) as mcn
    -> group by marriageCertificateNum
    -> having count(*) != 2
    -> );
Query OK, 1 row affected (0.01 sec)
```

Step -7 **Testing** as all the triggers are implemented and successfully executed. We can check with different criteria and display how these triggers are working. Let's take few examples:

1. *Polygamy marriages if religion permits*: Trigger to check if religion allows polygamy or not is based on the religion. Example - In Muslim religion, they are allowed to marry unto 4 people at a time. We can observe from the below test that for muslims database allows polygamy but for others it restricts.

```
mysql> # 1. POLYGAMY MARRIAGE EXAMPLE
mysql> # Inserting people to PersonDetails table whose religion allows multiple marriage
mysql> INSERT INTO PersonDetails VALUES(13, 5, 'X', 'T', 'M', 30);
ERROR 1062 (23000): Duplicate entry '13' for key 'PRIMARY'
mysql> INSERT INTO PersonDetails VALUES(14, 5, 'S', 'H', 'F', 25);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonDetails VALUES(15, 5, 'T', 'B', 'F', 30);
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> # Inserting Nationality of people into PersonNationality
mysql> INSERT INTO PersonNationality VALUES(14, 13, 10);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonNationality VALUES(15, 14, 10);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonNationality VALUES(16, 15, 10);
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> # Marrying person 13-14 and 13-15 and entering for certificate in MarriageDetails table
mysql> INSERT INTO MarriageDetails VALUES(107, '2006-10-12', 'Scotland');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MarriageDetails VALUES(108, '2016-01-24', 'Scotland');
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> # Associating these marriage with persons involved in PersonInMarriage table
mysql> INSERT INTO PersonInMarriage VALUES(1013, 13, 107, "Y");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1014, 14, 107, "Y");
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> # For second marriage between 13-15
mysql> INSERT INTO PersonInMarriage VALUES(1015, 13, 108, "Y");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1016, 15, 108, "Y");
Query OK, 1 row affected (0.01 sec)
```

2. *Same sex marriage: I*n our database, same sex marriage is allowed. Example- In the below code two marries each other and the system allows.

```
mysql> # 2. SAME SEX MARRIAGE EXAMPLE
mysql> # Inserting people to PersonDetails table with same sex
mysql> INSERT INTO PersonDetails VALUES(16, 3, 'Marry', 'Brown', 'F', 30);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonDetails VALUES(17, 3, 'Leona', 'Hurley', 'F', 28);
Query OK, 1 row affected (0.00 sec)

mysql> # Inserting Nationality
mysql> INSERT INTO PersonNationality VALUES(17, 16, 1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonNationality VALUES(18, 17, 1);
Query OK, 1 row affected (0.00 sec)

mysql> # Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(109, '2016-01-24', 'Ireland');
Query OK, 1 row affected (0.00 sec)

mysql> # Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1017, 16, 109, "Y");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1018, 17, 109, "Y");
Query OK, 1 row affected (0.00 sec)
```

3. *Remarriage after divorce either with the same person or the other:* For example, there are complicated marital situations of the rich and famous: King Henry VIII was married six times (though never twice at the same time), while Richard Burton and Elizabeth Taylor were married (after each of them were divorced from other partners), then divorced, then remarried to others, then divorced from others, then remarried to each other, and then divorced again from each other. The key is about marrying one at at time (if religion doesn't permit more). I've handled this situation as well. In below example, a person gets divorce and re-marries. In the above code

person_ids 18 and 19 marries with each other, then gets divorce and then re-marries again.

```
mysql> # 3. DIVORCED PERSON MARRYING EXAMPLE
mysql> # Inserting people to PersonDetails table with same sex
mysql> INSERT INTO PersonDetails VALUES(18, 1, 'Mark', 'Bush', 'M', 30);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonDetails VALUES(29, 1, 'Michille', 'Anderson', 'F', 28);
Query OK, 1 row affected (0.00 sec)

mysql> # Inserting Nationality
mysql> INSERT INTO PersonNationality VALUES(19, 18, 2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonNationality VALUES(20, 29, 2);
Query OK, 1 row affected (0.00 sec)

mysql> # Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(110, '2016-01-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> # Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1019, 18, 110, "Y");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1020, 29, 110, "Y");
Query OK, 1 row affected (0.00 sec)

mysql> # Divorcing between them
mysql> # | Divorce_Certificate_Number | MarriageCertificateNumber | Divorce_Date | Divorce_Country |
mysql> INSERT INTO DivorceDetails VALUES(2001, 110, "2016-06-06", "India");
Query OK, 1 row affected (0.00 sec)

mysql> # Marrying them again with new certificate of marriage
mysql> # Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(211, '2016-12-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> # Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1021, 18, 211, "Y");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1022, 29, 211, "Y");
Query OK, 1 row affected (0.00 sec)
```

Below is the output after successful execution:

4. *A person can have multiple nationalities:* A person can have multiple nationalities and below person has output for the code where we can see that the person which have 2 different nationalities 1 and 2 are successfully inserted into the database.

```
mysql> # 5. A PERSON CAN HAVE MULTIPLE NATIONALITIES
mysql> INSERT INTO PersonDetails VALUES(22, 1, 'Jameson', 'Guiness', 'F', 25);
Query OK, 1 row affected (0.00 sec)

mysql> # Inserting multiple nationalities
mysql> INSERT INTO PersonNationality VALUES(23, 22, 1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonNationality VALUES(24, 22, 2);
Query OK, 1 row affected (0.00 sec)
```

5. *Person can marry again if partner dies*: Below test shows where it allows to marry again. The output of the above code is:

```
mysql> # 4. WIDOW PERSON MARRIAGE EXAMPLE
mysql> # Inserting people to PersonDetails table with same sex
mysql> INSERT INTO PersonDetails VALUES(30, 1, 'Mark', 'Bush', 'M', 30);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonDetails VALUES(31, 1, 'Michille', 'Anderson', 'F', 28);
Query OK, 1 row affected (0.00 sec)

mysql> # Inserting Nationality
mysql> INSERT INTO PersonNationality VALUES(21, 30, 2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonNationality VALUES(22, 31, 2);
Query OK, 1 row affected (0.00 sec)

mysql> # Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(112, '2016-01-24', 'India');
ERROR 1062 (23000): Duplicate entry '112' for key 'PRIMARY'
mysql> # Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1023, 30, 112, "Y");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1024, 31, 112, "Y");
Query OK, 1 row affected (0.00 sec)

mysql> # Entering one partner to dead list
mysql> INSERT INTO DeathDetails VALUES(3000, 31, "2016-05-23", "Accident");
Query OK, 1 row affected (0.00 sec)

mysql> #Marrying partner with new certificate
mysql> # Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(156, '2016-12-24', 'India');
Query OK, 1 row affected (0.00 sec)
```

6. *Disallow polygamy if person's religion doesn't allow it*: There are religions who allow polygamy like in Muslims. For example, in below case the system will not allow multiple marriage.

```
mysql> # 6. POLYGAMY IS NOT POSSIBLE FOR A PERSON IF RELIGION DOESN'T ALLOW
mysql> # Inserting people to PersonDetails table with same sex
mysql> INSERT INTO PersonDetails VALUES(50, 1, 'Mark', 'Black', 'M', 30);
Query OK, 1 row affected (0.00 sec)

mysql> # Inserting Nationality
mysql> INSERT INTO PersonNationality VALUES(26, 50, 2);
Query OK, 1 row affected (0.00 sec)

mysql> # Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(237, '2016-01-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonInMarriage VALUES(2224, 50, 237, "Y");
Query OK, 1 row affected (0.01 sec)

mysql> # Marrying again
mysql> INSERT INTO MarriageDetails VALUES(1105, '2016-01-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> # Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1025, 50, 1105, "Y");
ERROR 1644 (45000): Allowed limit exceeded.
```

7. *Cannot marry outside person's nationality country*: The marriage can only take place where either one of the person belongs to. For example, if both person are from India, they can only marry in India and not anywhere else.

```
mysql> INSERT INTO MarriageDetails VALUES(116, '2016-01-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1027, 2, 116, "Y");
ERROR 1644 (45000): Nationality Issue
mysql> -- ERROR 1644 (45000): Cannot Marry Outside Person Nationality Country.
```

8. *Child marriage prohibited*: In most of the countries child marriage is prohibited and the minimum age of marriage is 18 years old for both males and females. Hence, testing the same.

```
mysql> # 8. CHILD MARRIAGE IS NOT POSSIBLE
mysql> INSERT INTO PersonDetails VALUES(23, 1, 'TOMMY', 'HANKS', 'M', 12);
ERROR 1644 (45000): age should be > 18.
```

9. *Self marriage is not possible:* A person cannot marry himself. In the below code a person tries to marry himself and system throws and exception. Here person id 24 tries to marry himself and the system throws following exception:

```
mysql> # 9. SELF MARRIAGE IS NOT POSSIBLE
mysql> -- Inserting people to PersonDetails table with same sex
mysql> INSERT INTO PersonDetails VALUES(24, 1, 'Mark', 'Bush', 'M', 30);
Query OK, 1 row affected (0.00 sec)

mysql> -- Inserting Nationality
mysql> INSERT INTO PersonNationality VALUES(25, 24, 2);
Query OK, 1 row affected (0.00 sec)

mysql> -- Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(117, '2016-01-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> -- Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1028, 24, 117, "Y");
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO PersonInMarriage VALUES(1029, 24, 117, "Y");
ERROR 1644 (45000): Allowed limit exceeded.
```

10. *Cannot marry a dead person*: In this system a dead person marriage is not allowed. In the following code a person tries to marry a dead person and get the exception.

```
mysql> -- Inserting people to PersonDetails table with same sex
mysql> INSERT INTO PersonDetails VALUES(25, 1, 'Mark', 'Bush', 'M', 30);
Query OK, 1 row affected (0.00 sec)

mysql> -- Inserting Nationality
mysql> INSERT INTO PersonNationality VALUES(26, 24, 2);
Query OK, 1 row affected (0.00 sec)

mysql> -- Marriage Certificate generation
mysql> INSERT INTO MarriageDetails VALUES(118, '2016-01-24', 'India');
Query OK, 1 row affected (0.00 sec)

mysql> -- Person's details in PersonInMarriage
mysql> INSERT INTO PersonInMarriage VALUES(1028, 24, 117, "Y");
ERROR 1644 (45000): Allowed limit exceeded
mysql> INSERT INTO PersonInMarriage VALUES(1029, 21, 117, "Y");
ERROR 1644 (45000): Allowed limit exceeded
mysql> -- ERROR 1644 (45000): Maximum allowed marriages for this religion exceed.
```

Step: 8 Views - External Layer:

A database view is a virtual table or logical table which is defined as a SQL select query with joins. Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems, including MySQL, allow you to update data in the underlying tables through the database view with some prerequisites.

A database view is dynamic because it is not related to the physical schema. The database system stores database views sql select statement with joins. When the data of the tables changes, the view reflects that changes as well.

Further to this, views were created that has relationships with other tables. Lets see the examples of views in our database one by one.

1. *CurrentMarriedPeopleView*: This view shows all the married couple in a relationship table where both the partners can be seen side by side. Inner join is used to create this view. After executing the following code:

```
mysql> # Married People with their relationship status
mysql> create view currentMarriedPeople as
    -> select a.marriageCertificateNum, a.status, a.personID as PersonID1,
    -> b.personID as PersonID2 from PersonInMarriage a inner join PersonInMarriage b on
    -> (a.marriageCertificateNum = b.marriageCertificateNum and a.personID > b.personID)
    -> where a.status = 'Y';
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> select * from CurrentMarriedPeople;
+------------------------+--------+-----------+-----------+
| marriageCertificateNum | status | PersonID1 | PersonID2 |
+------------------------+--------+-----------+-----------+
|                    107 | Y      |        14 |        13 |
|                    108 | Y      |        15 |        13 |
|                    109 | Y      |        17 |        16 |
|                    211 | Y      |        29 |        18 |
|                    122 | Y      |        26 |        25 |
+------------------------+--------+-----------+-----------+
5 rows in set (0.00 sec)
```

2. *MarriedPeopleView*: This view will show all people who married once. It shows historical data as well. In the status column, one can see if they are currently married or not. Following is the code. When we execute the code we will get this output:

```
mysql> # Married People with their relationship status
mysql> create view marriedPeople as
    -> select a.marriageCertificateNum, a.status, a.personID as PersonID1,
    -> b.personID as PersonID2 from PersonInMarriage a inner join PersonInMarriage b on
    -> (a.marriageCertificateNum = b.marriageCertificateNum and a.personID > b.personID);
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> select * from marriedPeople;
+------------------------+--------+-----------+-----------+
| marriageCertificateNum | status | PersonID1 | PersonID2 |
+------------------------+--------+-----------+-----------+
|                    111 | N      |         5 |         1 |
|                    112 | N      |        11 |         2 |
|                    113 | N      |        19 |        12 |
|                    107 | Y      |        14 |        13 |
|                    108 | Y      |        15 |        13 |
|                    109 | Y      |        17 |        16 |
|                    110 | N      |        29 |        18 |
|                    211 | Y      |        29 |        18 |
|                    112 | N      |        30 |         2 |
|                    112 | N      |        30 |        11 |
|                    112 | N      |        31 |         2 |
|                    112 | N      |        31 |        11 |
|                    112 | N      |        31 |        30 |
|                    122 | Y      |        26 |        25 |
+------------------------+--------+-----------+-----------+
14 rows in set (0.00 sec)
```

3. divorcedPeopleView: This will show historic data of people who divorced. The code executed with output:

```
mysql> # Divorced Marriages with people
mysql> create view divorcedPeople as
    -> select a.marriageCertificateNum, a.personID as Divorcee1,
    -> b.personID as Divorcee2 from PersonInMarriage a inner join PersonInMarriage b on
    -> (a.marriageCertificateNum = b.marriageCertificateNum and a.personID > b.personID)
    -> where a.marriageCertificateNum IN (select marriageCertificateNum from DivorceDetails);
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> select * from divorcedPeople;
+------------------------+-----------+-----------+
| marriageCertificateNum | Divorcee1 | Divorcee2 |
+------------------------+-----------+-----------+
|                    110 |        29 |        18 |
|                    112 |        11 |         2 |
|                    112 |        30 |         2 |
|                    112 |        30 |        11 |
|                    112 |        31 |         2 |
|                    112 |        31 |        11 |
|                    112 |        31 |        30 |
+------------------------+-----------+-----------+
7 rows in set (0.01 sec)
```

4. widowPeopleEligibleForMarriageView: This view will show people who are eligible to marry after their partner's death.

```
mysql> # People who are widow and are eligible for marriage
mysql> create view widowPeopleEligibleForMarriage as
    -> select a.marriageCertificateNum, b.personID as widow
    -> from PersonInMarriage a inner join PersonInMarriage b on
    -> (a.marriageCertificateNum = b.marriageCertificateNum and a.personID != b.personID)
    -> where a.personId IN (select personId from DeathDetails) and
    -> b.status = 'N' and
    -> b.personId NOT IN (select personId from DeathDetails) and
    -> (select count(status) from PersonInMarriage where PersonInMarriage.personId = b.personId and PersonInMarriage.status = "Y")
    -> <
    -> (select allowedNumOfMarriages from Religion where religionId =
    -> (select religionId from PersonDetails where personId = b.personId));
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> select * from widowPeopleEligibleForMarriage;
+------------------------+-------+
| marriageCertificateNum | widow |
+------------------------+-------+
|                    111 |     5 |
|                    112 |    11 |
|                    112 |    11 |
|                    113 |    12 |
|                    112 |    30 |
|                    112 |    30 |
+------------------------+-------+
6 rows in set (0.01 sec)
```

5. PolygamyAllowedForPeopleView: This view shows people whose religion accepts polygamy marriages. Following are the people whose religion allow polygamy marriages:

```
mysql> # People allowed for polygamy according to their religion
mysql>
mysql> create view polygamyAllowed as
    -> select personID , firstName, lastName from PersonDetails p where
    -> (select allowedNumOfMarriages from Religion where religionId = p.religionId)>1;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> select * from polygamyAllowed;
+----------+-----------+----------+
| personID | firstName | lastName |
+----------+-----------+----------+
|       12 | Faiza     | Begum    |
|       13 | Mohammad  | Khan     |
|       14 | S         | H        |
|       15 | T         | B        |
+----------+-----------+----------+
4 rows in set (0.00 sec)
```

6. MarriedMoreThanOnceView: This view shows about the people who married more than once. Following are the people who married more than once in their lifetime.

```
mysql> # People who married more than once due to any reason
mysql>
mysql> create view MarriedMoreThanOnce as
    -> select personID from PersonInMarriage a where
    -> (select count(marriageCertificateNum)
    -> from PersonInMarriage where personId = a.personId)>1;
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> select distinct * from MarriedMoreThanOnce;
+----------+
| personID |
+----------+
|       13 |
|       18 |
|       29 |
+----------+
3 rows in set (0.00 sec)
```

LIMITATIONS:

The database design for marriage and divorce is very much complex and addressing all the conditions for worldwide marriages and divorces are practically impossible.

That's because of so many rules which are based on different religions, regions etc. There are every unimaginable possibilities for marriages which are poorly documented as well. So this database system too have some limitations:

1. In case a person is marrying outside its country then the partner must have the nationality of that country . In our database, partner details must be associated first and then the person's id.

2. There are complex relationships as well like civil partnerships. This system doesn't handle those.

3. Couple of major religion allows males to marry multiple females but only handful of religion and regions allow female to marry many males. In this system there is no restriction as such. Male and female are free to marry equally if their religion allows.

CONCLUSION:

In this project the concepts of Relational Database Management Systems were used to make this database design as robust as possible. The complexity and challenges of Marriage and Divorce Database were taken into account and then the motivation was well established. The design of schemas was created step by step process using the normalization process. The logical schema was established and the logical schema of each table is provided them using Entity-Relationship diagram, which is a graphical representation of entities and their relationships to each other.

In this Marriage_divorce_DB, tables were created and most of them in 3NF normal form. The Enhanced entity–relationship model or EER Model which is a high-level or conceptual data model which is an extension to the ER diagram. It is used to reflect more precisely the properties and constraints that are found in more complex databases like Marriage and Divorce Database.

Then the implementation of the database design is being explained. The code is present in the script file and the Data Definition Language (DDL) to create the structure of database objects in database and Data Manipulation Language (DML) to insert, update, delete etc. items in the database. To check the constraints, I used Triggers extensively in this system.

Then each trigger is being validated and the database is throughly tested if the mentioned conditions are working properly. Large amount of data is inserted and created indexes in each database to make the process fast. Further , took timely back up of the implemented code.

The marriage and divorce database system is indeed a complex system and in this design and implementation, covered many constraints as possible with certain mentioned limitations.