

1a. The following system requirements are to be met for this Project:

Node JS V16 or above
Any modern web browser
MySQL Workbench 8.0

The project involves DNS Visualization. From the ER Diagram given below, it becomes clear that we will be requiring IP Addresses of the three servers (root, net and DLB), verify them at each step and ultimately provide the final IP Address of the website entered.

We have involved four tables overall. The queries table acts as the cache server with a known IP Address of 111.11.11.1. If the value stored is not null corresponding to it, then the cached value is returned and this value is reset at regular intervals(say, 10 mins). If it is Null, a call is made to the root table which acts as the root server with an IP Address of 111.11.11.0. The 'link' provided is then broken up into its components based on which calls are made from this root server, which gives an IP Address of the net server, from where we obtain the address of the dlb server, which finally gives us the IP Address required. This is very useful in ensuring security and protecting our online access from any outside unauthorized access.

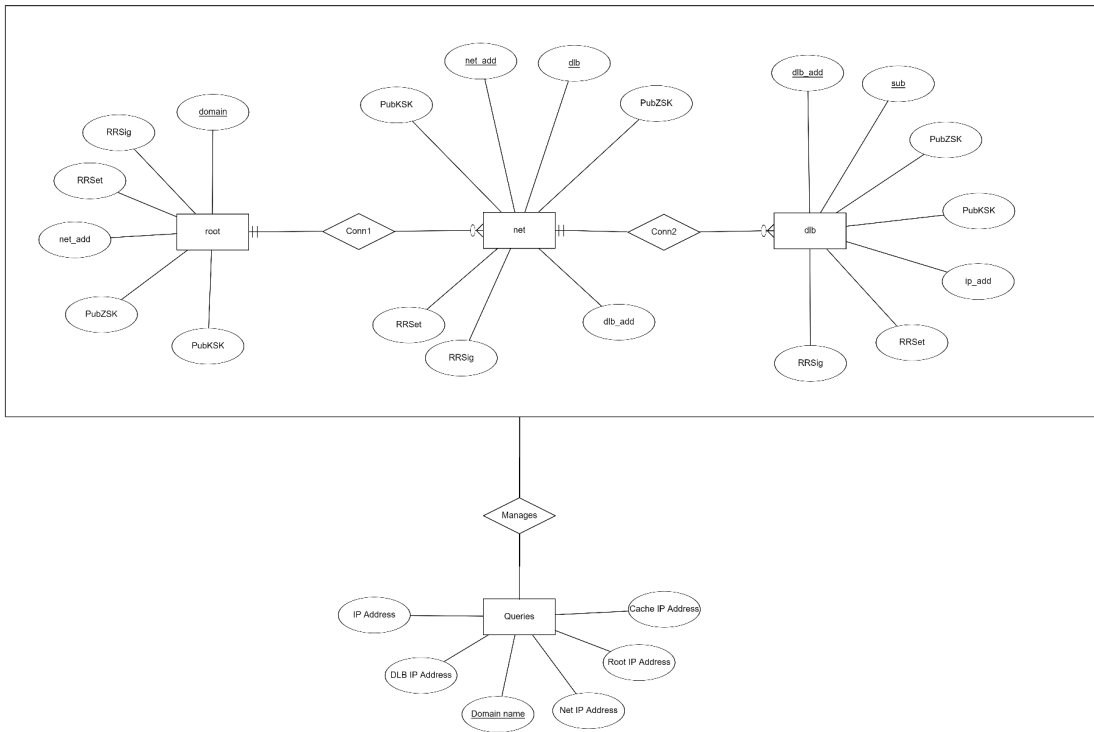
Thus, the Cached server(ISP) makes **recursive** calls to these 3 servers one after the other while each of the servers make an **iterative** call searching for the required IP Address stored. These are what are referred to as Recursive and Iterative Queries in regards to the DNS project, and we have implemented them as such.

Everytime a server returns an IP Address, it also returns a PubKSK and a PubZSK (Public Key Security Key and Public Zone Security Key), along with an RRSet and an RRSig(Set and Signature of Resource Records which contains information about a given record type for a given name, along with a digital signature for authentication.) which are encrypted using the Server's Private keys(PvtKSK and PvtZSK). At each step, we decrypt these values using the Public keys provided to check authenticity at all times. If they do not match, an error is thrown- else the process continues. We have implemented this entire authentication process via the MySQL platform.

The final queries table includes the addresses of the 3 servers, the cache server itself and also the final IP Address obtained. The queries table can contain null values and so it may have a value not queried or added yet.

This gives rise to the ER Diagram described below with all the data mentioned above stored :

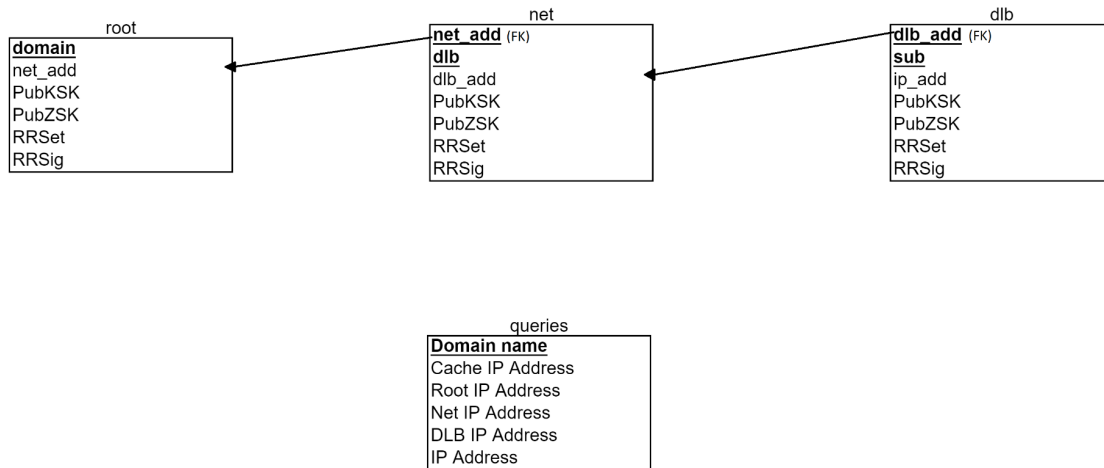
2a.



Since the Conn1 and Conn2 relations are part of a Many-to-one relationship, their primary keys can be included in the respective tables contributing to the 'many' part- thereby eliminating the need to have them as separate relations and removing redundancy.

In the Manages relation, we only need to obtain the final data from the Aggregation and store the same in the queries table. Once again, to remove redundancies, we have eliminated the requirement of the Manages relation and storing the final results directly in the queries table giving us the following Table designs:

2b.



- *root*(**domain**,*net_add*,*PubKSK*,*PubZSK*,*RRSet*,*RRSig*)
- *net*(**net_add**,**dlb**,*dlb_add*,*PubKSK*,*PubZSK*,*RRSet*,*RRSig*)
- *dlb*(**dlb_add**,**sub**,*ip_add*,*PubKSK*,*PubZSK*,*RRSet*,*RRSig*)
- *queries*(**'Domain name'**,*'Cache IP Address'*,*'Root IP Address'*,*'Net IP Address'*,*'DLB IP Address'*,*'IP Address'*)

All the data types here have been stored in Varchar format for ease of access and display. Only the RRSet and RRSig attributes have been stored as blob values(using aes_encrypt and aes_decrypt functions.)

2c. It is possible that the same domain name is contained under different domain name suffixes and also same subdomain names under different domain names, however it is not compulsory that all these possibilities are met. So a BCNF would be much more appropriate.

In the Tables we have created, there are no multivalued attributes (all atomic) so we are in 1NF.

There are no partial dependencies in any of the tables(since no subset of a candidate key determines any other non primary attribute as we have modeled the primary keys as such) and so we are also in 2NF.

Now, for any FD in all tables, on the Left hand there are only superkeys, so the tables are in 3NF and BCNF form also, as this condition suffices both.

2d. List of tables required are:

1. root

Field	Type	Null	Key
domain	varchar(50)	NO	PRI
net_add	varchar(50)	NO	UNI
PubKSK	varchar(50)	NO	
PubZSK	varchar(50)	NO	
RRSet	varbinary(160)	YES	
RRSig	varbinary(160)	YES	

2. net

Field	Type	Null	Key
net_add	varchar(50)	NO	MUL
dlb	varchar(50)	NO	
dlb_add	varchar(50)	NO	PRI
PubKSK	varchar(50)	NO	
PubZSK	varchar(50)	NO	
RRSet	varbinary(160)	YES	
RRSig	varbinary(160)	YES	

3. dlb

Field	Type	Null	Key
dlb_add	varchar(50)	NO	MUL
sub	varchar(50)	NO	
ip_add	varchar(50)	NO	PRI
PubKSK	varchar(50)	NO	
PubZSK	varchar(50)	NO	
RRSet	varbinary(160)	YES	
RRSig	varbinary(160)	YES	

4. queries

Field	Type	Null	Key
Domain name	varchar(50)	NO	PRI
Cache IP Address	varchar(50)	YES	
Root IP Address	varchar(50)	YES	
Net IP Address	varchar(50)	YES	
DLB IP Address	varchar(50)	YES	
IP Address	varchar(50)	YES	UNI

The details of all these tables have been elaborated and also shown in the ER Diagram above.

2e. We have used three procedures:

1. splitname(IN name varchar(50),out dlb varchar(50),out net varchar(50),out root varchar(50)) : This splits the entered website into its domain suffix, domain and sub domain. Access the OUTs directly.
2. set_ip(IN link varchar(50), IN ntad varchar(50), IN dlbad varchar(50), IN ipad VARCHAR(50)) : This does multiple checks and finally adds the link into the database

while ensuring all the Integrity Constraints. Access the existing tables to see the updated changes then.

3. `get_ip(IN link varchar(50),OUT rad varchar(50),OUT ntad varchar(50),OUT dad varchar(50))` : This fetches the required IP Address either from the cache(until timeout) or via querying through the database in the fashion explained above, while handling all the edge. The results can be obtained directly from the OUTs cases or visible via the view `view` or queries table.

Dataset:

Since our project required us to have an initially static database for a start, we have stored the suffixes '.com', '.net', '.org' and '.edu', the domains 'facebook', 'google' and 'youtube' and the subdomains 'www.', 'www.test.' and 'www.blog.' giving a total of 36 entries to begin with. We have also kept an option of adding as many valid database entries as required so that the database is dynamic in nature and can also cover a wide array of data while maintaining consistency and concurrency checks at the same time.

Consistency:

Since in our project, Consistency involves that no two IP Addresses should clash and that each server has a foreign key to the prior, it can be maintained using the unique, not null and foreign key constraints on the attributes in our tables. Moreover, since every procedure implemented at each step also checks for these conditions and also authenticates the values returned via encryption and decryption methods with the help of the Keys provided, we have ensured that the database remains and operates consistently at all times.

If we used Triggers, we would then have to individually check for any repeated IP Addresses in other columns anyway thereby requiring us to do double-work. So instead of that, Consistency checks have been performed altogether by enforcing the relevant constraints and with the help of procedures as well.

Concurrency:


Whenever there is a write operation involved, We have set the transaction isolation level to Repeatable read mode at the global level. This solves the problem of concurrency but has only the problem of phantom reads. However, since our queries require exactly what we seek instead of some range, and since we do not involve any delete operations, we need not concern ourselves with this issue either.

Error Handling:

1. The basic errors when we try to add a duplicate value or any illegal values are handled by SQL itself and also the frontend taking inputs.
2. If at any stage the PubKSK or PubZSK cannot decrypt and verify authenticity, it throws an error saying 'Error in Authentication'.
3. If we try to access a web address not in the database, it gives the error 'Sorry but Web Address does not exist in our database. Try adding a new entry.'
4. When entering a new IP Address, if any of the Addresses conflicts with an existing address, it shows the error 'Error in details entered(d).' with (d) signaling duplicates'.
5. When entering a new IP Address, if the net address supplied conflicts with the existing net address, it shows the error 'Error in details entered(1).' with (1) signaling error in net address.
6. When entering a new IP Address, if the dlb address supplied conflicts with the existing dlb address, it shows the error 'Error in details entered(2).' with (2) signaling error in dlb address.

Thus we have thoroughly covered various cases and provided an overall error handling mechanism.

References:

1.  DNSSec Explained
2. <https://dnsviz.net/>

P.S. All the assets used in this project have been edited by us. The Frontend is hosted on Netlify Free-Tier.

Also please note that a few extra constraints on the size and qualities of the IP Address and the Web servers have been imposed via the frontend as it became unnecessarily complicated and didn't make sense to be done on MySQL, however , everything else has been done entirely on MySQL.