DMDS-HW-1

By:

R. MANI NIHARIKA

(1819748)

G. VAMSI KRISHNA GUNTURI

(1794653)

P.VENKATA ABHINAY

(1819771)

Part – 1 – Database built from scratch

SYRIA DATA BASE:

```
>create database Syria;
>use Syria;
```

In this data base we created schemas for people died in latest incident which occurred in Syria. There are some schemas in this data base which stores information about people's generic information(name, sex, place of birth etc), their death cause, their dependents, In which country they can immigrate and based on these details we calculate their flight timings to respective countries.

Create table:

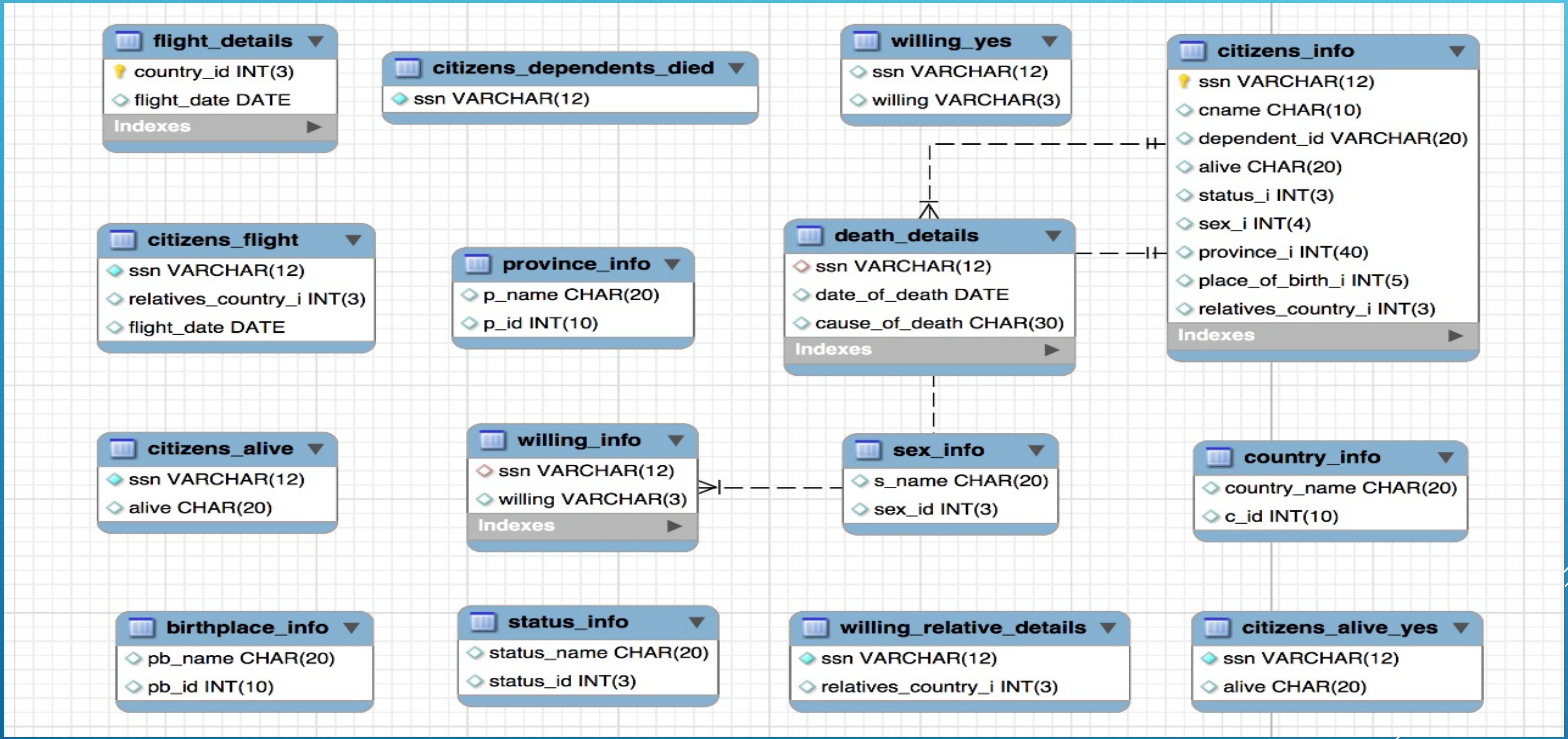
```
>create table citizens_info(ssn varchar(12),cname char(20),status varchar(20), province char(30),place_of_birth char(30),dependent_id int(20), relation_country int(30), alive char(30),primary key(ssn));
```

Like wise we created so many tables using the above syntax.

In order to see all the tables in this Syria data base, use the following syntax:

```
>show tables;
```

- The schema of all the tables look like:



The following are the queries we done on the Syria data base:

1) Find the citizens who are alive and are willing to immigrate to another countries?

```
> Select citizens_info.alive, willing_info.willing from citizens_info left join
willing_info on citizens_info.ssn=willing_info.ssn;
```

2) Find the citizens who are alive,willing to go and their flight dates?
For this we've divided the query into three parts.

- Create table with citizens that are alive.
- Create table with alive citizens that are willing to go.
- Create table for the willing citizens with respect to their relations country

```
>create table citizens_flight as (select
c.ssn,c.relative_country_i,f.flight_date from willing_relative_details c,
flight_deatils f where c.relatives_country_i=f.country_id);
```

3) Find all the citizens whose dependents died?

```
>create table citizens_dependents_died as (select c.ssn from
citizens_info c, death_details d where c.dependent_id=d.ssn);
>select *from citizens_dependents_died;
```

4)Who are the head of the family?

```
>select ssn from citizens_info where dependent_id=‘no’;
```

5)Find all the citizens whose head of the family died?

```
>select ssn.cname from citizens_info where dependent_id=‘no’ and
alive=‘no’;
```

6) Find all the citizens whose name starts with ‘A’?

```
>select *from citizens_info where cname LIKE ‘A%’;
```

7)Select the top 10 citizens from the citizens_info?

```
>select *from citizens_info LIMIT 10;
```

8) Find how many citizens are there in citizens_info table?

```
>select count(ssn) from citizens_info;
```

9) Use the wildcards in the your quries?

```
>select *from death_details where cause_of_death LIKE ‘s%%g’;
```

10) Create a view for deaths which are caused by shooting?

```
>create view shooting_death as select cause_of_death,ssn from
death_details where cause_of_death=‘shooting’;
```

11)Create a table about civilians death and see if there is any non_civilian the query should return error?

```
>create table civilian_death(ssn varchar(20),cname char(20),status
char check (status=‘civilian’));
```

12)Return all the citizens whose ssn is not between 2 and 20?

```
>select *from citizens_info where ssn not between ‘2’ and ‘20’;
```

13)Find the citizen name=‘Ahin’ is dead or alive and if he is dead find the death_date?

```
>select date_of_death from death_details where exists(select ssn from
citizens_info where citizens_info.cname=‘Ahin’ and
citizens_info.ssn=death_details.ssn);
```

Part – 2 – Using existing data base already built

YELP DATA BASE:
YELP is a company which publishes crowd sourced reviews of businesses like restraunts, hotels etc.., so that users can be well informed on what business to choose based on their preferences, location and host of other filters

The database is available for download in the below link,
https://www.yelp.com/dataset/documentation/sql

The yelp dataset is provided by yelp.com that sizes up to **1.5 GB of data** comprising of **61,000+** businesses, **1.5 million** reviews and **481,000+** attributes for all businesses.

The dataset is available as SQL and JSON formats

We used following command to import the entire database content (that was downloaded as .sql file) –

```
mysql -u username -p database_name < file.sql
```

This command will import the data base content in the file.sql file and create a new database with the name supplied

Part – 2 – Using existing data base already built

The imported yelp dataset has following tables –

Attribute – Contains **1310575** records. Comprises of information such as attribute of a particular business for example does a restraunt is suitable for kids, is alcohol served etc..,

Business – Contains **174567** records. Comprises of information about individual businesses like where it is located and its reviews by users

Category – Contains **667527** records. Comprises of list of all business mapped to their corresponding categories.

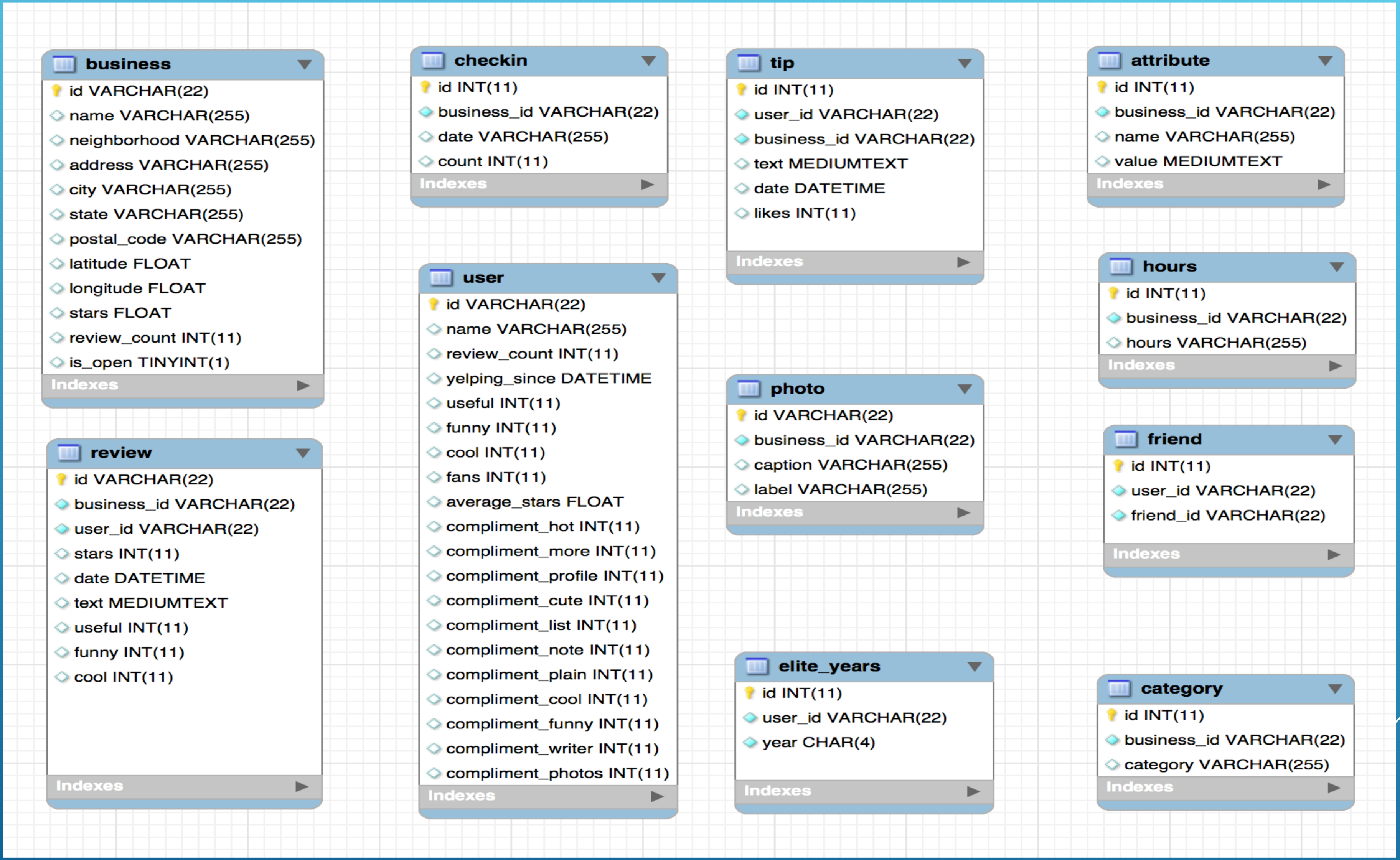
Check-in – Contains **3911218** records. As the name indicates this table contains all the user checkins based on the business id.

Elite_years – Contains **187125** records. Contains the list of years a particular user is active

Friend – Contains **49626957** records. Information about the friends of all the users mapped as user_id and his friend_id

Review – Contains **5261669** records. This table houses all the user reviews

Following is the schema of Yelp data set:



business
- id VARCHAR(22)
- name VARCHAR(255)
- neighborhood VARCHAR(255)
- address VARCHAR(255)
- city VARCHAR(255)
- state VARCHAR(255)
- postal_code VARCHAR(255)
- latitude FLOAT
- longitude FLOAT
- stars FLOAT
- review_count INT(11)
- is_open TINYINT(1)

Indexes

review
- id VARCHAR(22)
- business_id VARCHAR(22)
- user_id VARCHAR(22)
- stars INT(11)
- date DATETIME
- text MEDIUMTEXT
- useful INT(11)
- funny INT(11)
- cool INT(11)

Indexes

checkin
- id INT(11)
- business_id VARCHAR(22)
- date VARCHAR(255)
- count INT(11)

Indexes

user
- id VARCHAR(22)
- name VARCHAR(255)
- review_count INT(11)
- yelping_since DATETIME
- useful INT(11)
- funny INT(11)
- cool INT(11)
- fans INT(11)
- average_stars FLOAT
- compliment_hot INT(11)
- compliment_more INT(11)
- compliment_profile INT(11)
- compliment_cute INT(11)
- compliment_list INT(11)
- compliment_note INT(11)
- compliment_plain INT(11)
- compliment_cool INT(11)
- compliment_funny INT(11)
- compliment_writer INT(11)
- compliment_photos INT(11)

Indexes

tip
- id INT(11)
- user_id VARCHAR(22)
- business_id VARCHAR(22)
- text MEDIUMTEXT
- date DATETIME
- likes INT(11)

Indexes

photo
- id VARCHAR(22)
- business_id VARCHAR(22)
- caption VARCHAR(255)
- label VARCHAR(255)

Indexes

elite_years
- id INT(11)
- user_id VARCHAR(22)
- year CHAR(4)

Indexes

attribute
- id INT(11)
- business_id VARCHAR(22)
- name VARCHAR(255)
- value MEDIUMTEXT

Indexes

hours
- id INT(11)
- business_id VARCHAR(22)
- hours VARCHAR(255)

Indexes

friend
- id INT(11)
- user_id VARCHAR(22)
- friend_id VARCHAR(22)

Indexes

category
- id INT(11)
- business_id VARCHAR(22)
- category VARCHAR(255)

Indexes

What insights can we draw from the dataset -

We have wealth of information from YELP to perform some useful data analysis and draw some conclusions from the patterns observed in the data. Some of the insights that we can do are as follows –

- **Which is the average rating that users give to my business?**
- **Is the number of hours that a business open affecting their ranking mark?**
- **Is the location of my business affecting the ranking mark?**
- **Is my business getting more users depending of the season of the year?**
- **Is the number of services that my business offers affecting their ranking mark?**
- **Predict when a business will be more busy.**

Queries that we prepared -

Following are some queries that we prepared from the data set –

Goal : Top 10 users based on tip count -

Query:

```
SELECT u.name, COUNT(u.name)
FROM  yelp_db.user u JOIN yelp_db.tip t
WHERE  u.id = t.user_id
GROUP BY u.name
ORDER BY 2 DESC
LIMIT 10;
```

Goal : List of all categories to which all businesses belong

Query:

```
SELECT DISTINCT(category) FROM yelp_db.category ORDER BY category ASC; - sorted in alphabetical order
```

Goal : Top 10 users based on useful reviews -

Query:

```
SELECT u.name, TU.likes as useful_reviews
FROM
(SELECT user_id, SUM(useful) as likes
FROM yelp_db.review
GROUP BY user_id
ORDER BY 2 DESC
LIMIT 10) TU JOIN yelp_db.user u
WHERE TU.user_id = u.id;
```

Goal : Top 10 users based on useful reviews -

Query:

Creating a VIEW to improve performance

```
CREATE VIEW top_users_on_useful_reviews
AS
   SELECT user_id, SUM(useful) as likes
   FROM yelp_db.review
   GROUP BY user_id
   ORDER BY 2 DESC
   LIMIT 10

SELECT u.name, TU.likes as useful_reviews
FROM
yelp_db.top_users_on_useful_reviews TU JOIN yelp_db.user u
WHERE TU.user_id = u.id;
```

Improved performance:
With out Views: **244** seconds for displaying results
With Views: **105** seconds for displaying results

Technical reason for improved performance:
After a unique clustered index is created on the view, the view's result set is materialized immediately and persisted in physical storage in the database, saving the overhead of performing this costly operation at execution time.

Goal : Most popular users based on friends count

Query:

```
CREATE VIEW popular_users
AS
   SELECT user_id, COUNT(user_id) as friend_count
   FROM yelp_db.friend
   GROUP BY user_id
   ORDER BY 2 DESC
   LIMIT 10;

CREATE VIEW user_id_to_name_map
AS
   SELECT id, name
   FROM  yelp_db.user;


SELECT UM.name, PU.friend_count
FROM
yelp_db.popular_users PU JOIN yelp_db.user_id_to_name_map UM
WHERE PU.user_id = UM.id;
```

Execution time: **420** seconds