H1B 2017 Database

Project 2

CS-3265-01

Spring 2021

Abrar Haroon, Ammar Bin Zulqarnain

Introduction

Data Link:
https://www.kaggle.com/jonamjar/h1b-data-set-2017?select=H-1B_Disclosure_Data_FY17.csv

For this application we are using data of h1b visas in the United States for the year 2017. The application is designed to functionally decompose data into more logical tables so the users can view the specific data categories they desire. It also has a search functionality so that users can look at data or for specific cases they are interested in.

Finally, the application is designed to analyze a few data points such as case percentages by state and by region as well as the percentages or various case statuses. We choose this application as it will show data that is relevant to anyone analyzing h1b results. We chose this dataset as it meets the criteria for the application and was manageable to filter and insert into sql.

Final Implementation

Our use cases include searching for data using the case number. This gives the searcher data on all related data to the matched case number. Case number is a unique identifier so there will be a maximum of 1 result returned. If there is no match no results will be returned. The user will be able to search through the megatable for all the case fields and also through decomposed tables to get the limited case fields they are interested in an easy to read format.

We will also allow users to search by employer name. This allows users to see all h1b cases related to a certain employer. For this use case the minimum results will be none in cases of no match. And there can be multiple results returned for any employer that is matched by name.

Another use case is the ability to edit the data. So we will grant the user permission to insert data directly into the mega table through a form allowing the user to enter values for all case fields. Data will only be entered if the values match the criteria set for the data fields. For instance varchar values have a limit of size and some values must be integers.

Additionally, we have also used a trigger (entry_before_insert) to ensure that primary keys fields for the megatable and all decomposed tables are not null. If they are, then the trigger will throw an error and the command will not be completed. To insert the data from the megatable to the decomposed tables, we have used a stored procedure (auto_insert). This procedure will take all values from the form and insert them into the decomposed tables one by one. This procedure will automatically be called by a trigger entry_after_insert.

Another use case is related to updates. So we have allowed the user to update certain values and fields that are likely to change. These fields include case_status. Decision_date, strart_date, end_date, and employer_address. These changes will be made directly to the megatable by the user using a form. The form will ask for the related case number, the field they want to update and the new value. Again they will need to fulfill the criteria set for the data type. To update data in the corresponding decomposed tables, we have used separate stored procedures for each data field. So we have five update stored procedures. Additionally, since the employer address is part of a primary key in the employer address info decomposed table, we have used a trigger (change_before_update) to ensure that the user cannot update this value to NULL.

Finally we have given the user permission to delete by using the case ID. The use will delete from the megatable and as a result of foreign key constraint “on delete cascade”, data will be deleted from the decomposed tables.

Finally we also data analysis as a use case. The analysis include:
1) h1b case percentages by state
2) h1b cases percentages by case statuses
3) h1b case percentage of agents representing employers.

For each of these analyses we have made views to gather the data, and return results as a percentage in a tabular format.

Final Database Design

Functional Decomposition

1) We started off with the mega table which had the following attributes all dependent on the Case Number and Case ID:
CASE_ID, CASE_NUMBER, CASE_STATUS, CASE_SUBMITTED, DECISION_DATE, VISA_CLASS, EMPLOYMENT_START_DATE, EMPLOYMENT_END_DATE, EMPLOYER_NAME, EMPLOYER_BUSINESS_DBA, EMPLOYER_ADDRESS, EMPLOYER_CITY, EMPLOYER_STATE, EMPLOYER_POSTAL_CODE, EMPLOYER_COUNTRY, EMPLOYER_PROVINCE, EMPLOYER_PHONE, EMPLOYER_PHONE_EXT, AGENT_REPRESENTING_EMPLOYER, AGENT_ATTORNEY_NAME, AGENT_ATTORNEY_CITY,AGENT_ATTORNEY_STATE, JOB_TITLE, SOC_CODE, SOC_NAME, NAICS_CODE, TOTAL_WORKERS, NEW_EMPLOYMENT, CONTINUED_EMPLOYMENT, CHANGE_PREVIOUS_EMPLOYMENT, NEW_CONCURRENT_EMPLOYMENT, CHANGE_EMPLOYER, AMENDED_PETITION, FULL_TIME_POSITION, PREVAILING_WAGE, PW_UNIT_OF_PAY ,PW_WAGE_LEVEL, PW_SOURCE, PW_SOURCE_YEAR, PW_SOURCE_OTHER, WAGE_RATE_OF_PAY_FROM, WAGE_RATE_OF_PAY_TO, WAGE_UNIT_OF_PAY, H1B_DEPENDENT, WILLFUL_VIOLATOR, SUPPORT_H1B, LABOR_CON_AGREE, PUBLIC_DISCLOSURE_LOCATION,WORKSITE_CITY,WORKSITE_COUNTY,WORKSITE_STATE, WORKSITE_POSTAL_CODE,ORIGINAL_CERT_DATE

2) Using these attributes, we found all functional dependencies between the attributes using the code:

```
SELECT A
FROM mega_table
GROUP BY A
HAVING count (distinct B) > 1;
```

If no results were returned, then there was a functional dependency (FD) A -> B

3) Below are the non-trivial FD’s we were able to find :

- CASE_SUBMITTED -> CASE_STATUS, DECISION DATE
- EMPLOYER_ADDRESS, EMPLOYER_POSTAL_CODE -> EMPLOYER_CITY,EMPLOYER_STATE,EMPLOYER_COUNTRY
- AGENT_ATTORNEY_NAME -> AGENT_REPRESENTING_EMPLOYER, AGENT_ATTORNEY_CITY,AGENT_ATTORNEY_STATE
- JOB_TITLE,SOC_CODE -> SOC_NAME

• WORKSITE_POSTAL_CODE -> WORKSITE_CITY, WORKSITE_COUNTY, WORKSITE_STATE,

The following had no other FD other than CASE_ID, CASE_NUMBER:

• CASE_ID, CASE_NUMBER -> EMPLOYMENT_START_DATE, EMPLOYMENT_END_DATE, NAICS_CODE

- CASE_ID,CASE_NUMBER -> EMPLOYER_NAME,EMPLOYER_BUSINESS_DBA, EMPLOYER_PROVINCE,EMPLOYER_PHONE,EMPLOYER_PHONE_EXT,TOTAL_WORKERS
- CASE_ID,CASE_NUMBER -> FULL_TIME_POSITION, PREVAILING_WAGE, PW_UNT_OF_PAY, PW_WAGE_LEVEL, PW_SOURCE, PW_SOURCE_YEAR, PW_SOURCE_OTHER, WAGE_RATE_OF_PAY_FROM, WAGE_RATE_OF_PAY_TO, WAGE_UNIT_OF_PAY
- CASE_ID,CASE_NUMBER -> WILLFUL_VIOLATOR, SUPPORT_H1B, LABOR_CON_AGREE, PUBLIC_DISCLOSURE_LOCATION, ORIGINAL_CERT_DATE, VISA_CLASS
- CASE_ID,CASE_NUMBER -> NEW_EMPLOYMENT,CONTINUED_EMPLOYMENT, CHANGE_PREVIOUS_EMPLOYMENT, NEW_CONCURRENT_EMPLOYMENT, CHANGE_EMPLOYER,AMENDED_PETITION

4) Each of these FD's (in separate bullets) was evaluated to ensure 3NF using the formula in part 2. It was ensure that each non- primary attribute was dependant on the whole key and nothing but the key

5) Each of these FD's were turned into tables with appropriate names based on attributes e.g. CASE_SUBMITTED -> CASE_STATUS, DECISION DATE was named case_info.
In forming these tables, CASE_ID,CASE_NUMBER was included in each table (if it was not present already) as a necessary foreign key, from a preceding table as shown in the UML, to not only maintain referential integrity but also to assist in inserting values in each table. The primary key for each table was a combination of A in each (A -> B) FD and CASE_ID,CASE_NUMBER to ensure compliance with 3NF. This also ensured lossless joins as each table had the same number of rows and if joined using CASE_ID,CASE_NUMBER. See the appendix as for example of how the foreign key constraints were used in decomposed tables.

6) Dependencies were preserved by separating and attributes and making the independent terms the primary keys for decomposed tables. This tactic also helped ensure 3NF.

<u>Final Database Design</u>

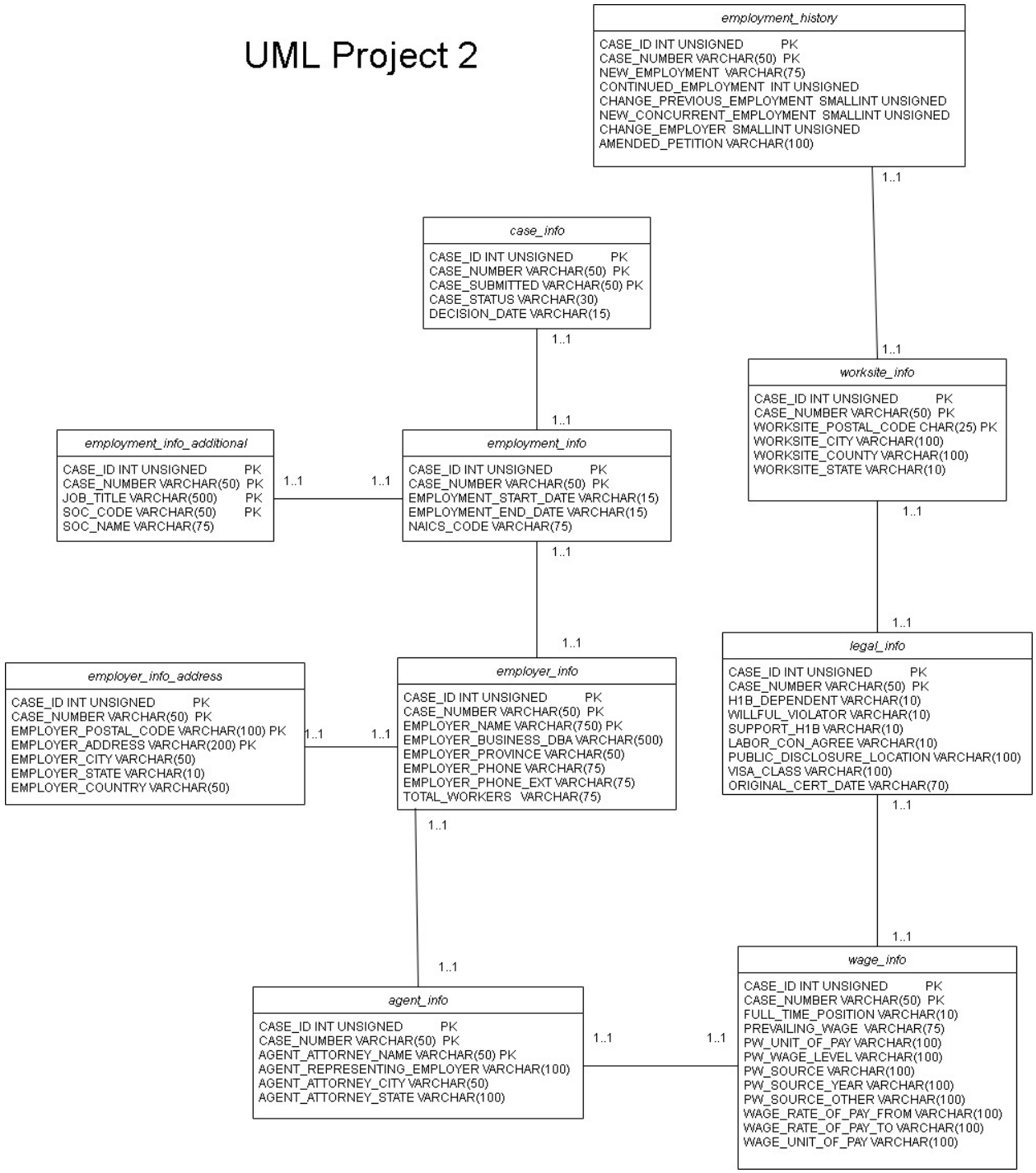
We have 11 tables including the megatable. The megatable contains all data fields as listed in the functional dependencies. The remaining tables and relevant data fields are as follows:

1) Case Info: CASE_ID, CASE_NUMBER,CASE_STATUS,CASE_SUBMITTED,DECISION_DATE
2) Employment info: CASE_ID, CASE_NUMBER, EMPLOYMENT_START_DATE,EMPLOYMENT_END_DATE, NAICS_CODE
3) Employment info additional: CASE_ID, CASE_NUMBER,JOB_TITLE, SOC_CODE,SOC_NAME

4) Employer info:
   CASE_ID,CASE_NUMBER,EMPLOYER_NAME,EMPLOYER_BUSINESS_DBA,
   EMPLOYER_PROVINCE,EMPLOYER_PHONE,EMPLOYER_PHONE_EXT,TOTAL_WORKERS
5) Employer info address: CASE_ID,CASE_NUMBER,EMPLOYER_ADDRESS,
   EMPLOYER_CITY,EMPLOYER_STATE,EMPLOYER_POSTAL_CODE,EMPLOYER_COUNTRY
6) Agent info: CASE_ID, CASE_NUMBER, AGENT_REPRESENTING_EMPLOYER,
   AGENT_ATTORNEY_NAME, AGENT_ATTORNEY_CITY,AGENT_ATTORNEY_STATE
7) Wage info:
   CASE_ID,CASE_NUMBER,FULL_TIME_POSITION,PREVAILING_WAGE,PW_UNIT_OF_PAY,PW_WAGE_LEVEL, PW_SOURCE,
   PW_SOURCE_YEAR,PW_SOURCE_OTHER, WAGE_RATE_OF_PAY_FROM,
   WAGE_RATE_OF_PAY_TO, WAGE_UNIT_OF_PAY
8) Legal info: CASE_ID,CASE_NUMBER,WILLFUL_VIOLATOR,
   SUPPORT_H1B,LABOR_CON_AGREE, PUBLIC_DISCLOSURE_LOCATION,
   ORIGINAL_CERT_DATE, VISA_CLASS
9) Worksite info:
   CASE_ID,CASE_NUMBER,WORKSITE_CITY,WORKSITE_COUNTY,WORKSITE_STATE, WORKSITE_POSTAL_CODE
10) Employment history:
    CASE_ID,CASE_NUMBER,NEW_EMPLOYMENT,CONTINUED_EMPLOYMENT,
    CHANGE_PREVIOUS_EMPLOYMENT, NEW_CONCURRENT_EMPLOYMENT,
    CHANGE_EMPLOYER, AMENDED_PETITION

    Note that each of these decomposed tables contains a foreign key CASE ID, CASE_NUMBER to the preceding table e.g 10 refers to 9. Case info takes the same foreign key from the megatable. Exceptions to this are employer info that takes the foregin key from employment info and agent info that takes the foreign key from employer info

UML



Above is the UML diagram for our table with related cardinality. It accurately displays the decomposed tables, their component attributes and the dependencies between them.

Testing Data

We first tested the edge case including null values for primary keys and foreign key constraints. We created this data on our own using testing guidelines learned in class. Then we downloaded

data for h1b 2018 and inserted that into the table, updated a few fields manually and also deleted the data. This helped us test our inserts, updates and deletes as well as all related views, triggers and stored procedures. Total rows used were over 5000. They were acquired from the same link.

Summary of implemented use cases

All use cases explained above were implemented on the front end. To summarize, these are:

1) Search and view case data
2) Insert data
3) Update data
4) Delete data
5) Analyze data

Illustration of Functionality

Search and view case data: You can search by case number only. To test input: I-200-16064-557834, this should return a result. Then test AA-200-16064-557834, this should not give you any results.

Insert data: To demonstrate this you will need to input 52 fields for the megatable that were listed above. To test insert (fields delimited by commas) (“ ” refers to empty string) use data :

I-200-26064-557834,CERTIFIED-WITHDRAWN,2016-03-05,2016-10-10,H-1B,2016-09-16,2019-09-16,DFS SERVICES, “ ” ,2501 LAKE COOK ROAD,RIVERWOOD,CA,99000,UNITED STATES OF AMERICA, “ ” ,2244050905, “ ”,Y, ELLSWORTH, CHAD,NEW YORK,NY,JUNIOR ASSOCIATE,15-2031,OPERATIONS RESEARCH ANALYSTS,522210,1,1,0,0,0,0,0,Y,49800.0,Year, “ ” ,Other,2015.0,TOWERS WATSON DATA SERVICES 2015 CSR PROFESSIONAL (ADMINISTRATIVE AND SALES) C,53000.0,57200.0,Year,N,N, “ ” ,Y, “ ” ,RIVERWOODS,LAKE,IL,60015,2016-03-08

This should go through. You can then search by case number: I-200-26064-557834 and see the inserted data.

To demonstrate Delete: Input the following case number: I-200-13266-966905. Then search using the case number, no results should be returned.

To demonstrate Updates for CASE_STATUS or DECISION_DATE, OR EMPLOYMENT_START_DATE, OR EMPLOYMENT_END_DATE, OR EMPLOYER_ADDRESS,
Use Case Number: I-200-13294-544059 and new value: 06-12-2020, then search using case number and you should see inserted values.

To demonstrate analytics, click on the analytics tab on the front end and see results for various analyses.

Summary Discussion

Status

The project is mostly complete but additional functionality could be added in the future. This includes greater ease of searchability by any data type. Additionally it would be helpful to have

users and accounts and only allow users to see selected data. Finally we can allow certain users to update additional fields than the ones shown. Note that all project requirements were met in the implementation.

Challenges

The initial challenge was loading data into the megatable as certain fields had commas and were also delimited by commas. We used the enclosed by clause to resolve this. Additionally there was some faulty data that was not allowing the load into the megatable; we used the IGNORE clause to resolve this issue. On the front end, none of us had knowledge of php so Ammar had to learn this.

Division of Labor

Work was reasonably split. Abrar did the back-end implementation including tables, advanced features and analytics. He also did the UML and back end related parts of the project report. Ammar worked on the front end functionality which he needed to learn and spent time on before implementation. He was also responsible for most of the testing of the backend features on the front end.

Appendix (Screenshots) Order: 1) Foreign key in Decomposed table 2) Example of an update stored procedure 3) Example of an before update trigger 4) All views related to analytics

```
/* Employment history contains the employmet history info of h1b applicants*/
DROP TABLE IF EXISTS employment_history;
CREATE TABLE IF NOT EXISTS employment_history(
CASE_ID INT UNSIGNED, /* Covers max values */
CASE_NUMBER VARCHAR(50), /*Varchar to save space */
NEW_EMPLOYMENT  VARCHAR(75), /* Covers max values */
CONTINUED_EMPLOYMENT  INT UNSIGNED, /* Covers max values */
CHANGE_PREVIOUS_EMPLOYMENT  SMALLINT UNSIGNED, /* Covers max values */
NEW_CONCURRENT_EMPLOYMENT  SMALLINT UNSIGNED, /* Covers max values */
CHANGE_EMPLOYER  SMALLINT UNSIGNED, /* Covers max values */
AMENDED_PETITION VARCHAR(100), /*Varchar to save space */
  PRIMARY KEY(CASE_ID, CASE_NUMBER),
        CONSTRAINT fk_8
        FOREIGN KEY (CASE_ID, CASE_NUMBER)
            REFERENCES worksite_info(CASE_ID, CASE_NUMBER)
            ON UPDATE CASCADE
            ON DELETE CASCADE
)ENGINE = INNODB;


/*Update case status procedure updates data in case info that was updated by user in megatable*/
DROP PROCEDURE IF EXISTS update_case_status;

DELIMITER //
CREATE PROCEDURE update_case_status(
CASE_ID INT UNSIGNED,
NEW_CASE_STATUS VARCHAR(30)
)
BEGIN

    UPDATE case_info
    SET  case_info.CASE_STATUS = NEW_CASE_STATUS
    WHERE  case_info.CASE_ID = CASE_ID;


END//

DELIMITER ;
```

```
/*change_before_update trigger ensures employer address cannot be updated by user to null
This is the only data field that is chosen for not null as it is the only primary key that we allow
the user to update*/
DROP TRIGGER IF EXISTS change_before_update;
DELIMITER //

CREATE TRIGGER change_before_update
BEFORE UPDATE
ON mega_table
FOR EACH ROW
BEGIN

   IF NEW.EMPLOYER_ADDRESS IS NULL
   THEN SIGNAL SQLSTATE '22003'
   SET MESSAGE_TEXT = 'EMPLOYER_ADDRESS cannot be empty',
   MYSQL_ERRNO = 1264;
   END IF;


END //

DELIMITER ;
```

```
/*case_counts_per_state view returns percentages of h1b cases per state in a tabular form
Note that null values and irregular data was ommitted from results*/
DROP VIEW IF EXISTS case_counts_per_state;
CREATE VIEW case_counts_per_state AS
SELECT EMPLOYER_STATE, ( count(CASE_NUMBER)/MAX(CASE_ID) * 100)  AS Percentage
FROM mega_table
WHERE EMPLOYER_STATE !="" AND EMPLOYER_STATE != "32501"
GROUP BY EMPLOYER_STATE
ORDER BY Percentage DESC;

/*case_counts_per_case_status view returns percentages of h1b cases per each cateorgy of case status*/
DROP VIEW IF EXISTS case_counts_per_case_status;
CREATE VIEW case_counts_per_case_status AS
SELECT CASE_STATUS, ( (count(CASE_NUMBER))/MAX(CASE_ID) * 100)  AS Percentage
FROM mega_table
GROUP BY CASE_STATUS
ORDER BY Percentage DESC;

/*case_counts_per_agent_representing_employer view returns percentages of h1b cases that had agents representing employers
and those who didnt. Note that null values were ommitted from results*/
DROP VIEW IF EXISTS case_counts_per_agent_representing_employer;
CREATE VIEW case_counts_per_agent_representing_employer AS
SELECT AGENT_REPRESENTING_EMPLOYER, ( (count(CASE_NUMBER))/MAX(CASE_ID) * 100)  AS Percentage
FROM mega_table
WHERE AGENT_REPRESENTING_EMPLOYER ="Y" OR AGENT_REPRESENTING_EMPLOYER = "N"
GROUP BY AGENT_REPRESENTING_EMPLOYER
ORDER BY Percentage DESC;
```