IT214 - DBMS

Final Project Report Submission

Case Study Database: Laborlist and Wages Database

Prepared by

Team No. 17 | Group 6 | Section 10

Group Members

Dhruv Patel - 202001271

Jaykumar Navadiya - 202001465

Under the guidance of

Prof. Chhaya

Teaching Assistant - Vinay Maharaj

Date

November 23, 2022

Index

IT214 - DBMS

Final SRS Submission

Case Study Database: Laborlist and Wages Database

Prepared by

Team No. 17 | Group 6 | Section 10

Group Members

Dhruv Patel - 202001271

Jaykumar Navadiya - 202001465

Under the guidance of

Prof. Chhaya

Teaching Assistant - Vinay Maharaj

Date

September 30, 2022

1. Introduction

1.1. Purpose

The purpose of this document is to present a detailed description of the database of LaborList and Wages. The database will maintain a record of all work done by laborers on a daily basis to store and distribute wages. It will explain the purpose and features of the database such as the schema and relations of the database, how the Labor Supervisor will benefit from using such a database instead of using a file management system, the constraints under which the database must operate (i.e. the constraints on some attributes). The database will be a part and backend of the web application (PERN stack) consisting of features like managing, hiring and scheduling of laborers and providing a dashboard to the firms with the help of the database at the backend.

1.2. Intended Audience and Reading Suggestions

This document is intended for users of the database, the labor supervisor, the company that hired the laborers and labor management teams, the government officials from the ministry of labor and employment, people who need labor services from time to time and want to hire laborers, firms looking for a large workforce.

The document is also intended for the laborers who are looking for jobs and want to make a great profile for themselves (Such laborers can be plumbers, electricians, maids, therapists, home painters or butlers), freelancers who gets paid on the work and the quality of work they do (designers, web developers, photographers) and also developers working on the database that might improve the database, add some functionalities later on to the database or fix some existing bugs in the database.

1.3. Product Scope

The LaborList and Wage database will provide an easy management system for all the businesses who want to manage a workforce (laborers), laborers who want to get hired and who wants to hire someone to get their work done. The Web Application will also have hiring features for normal users who will hire laborers depending on their requirements. The application will support many cities and districts of India and many categories of laborers.

The firms will be provided with an interactive dashboard for managing laborers. The application will provide an easy way for laborers to get hired based on their profiles. Their profiles will have their work experience in the field, ratings and comments of their previous work. The goal is to make an end to end system/application for businesses, laborers and normal users to have an idea of what's going on in their company while making it easy to pay employees/laborers and supervise them, creating job opportunities and making it easier for the users to hire and get their work done faster than any other conventional way.

Labor supervisors' can also be managed by firms with the help of the app and supervisors can see all aspects of the labor's work including start time of the work, end time, the type of work they are doing and availability of the labor. Supervisors will also be able to pay the laborers without calculating their wages because of the automatic wage distribution feature of the database.

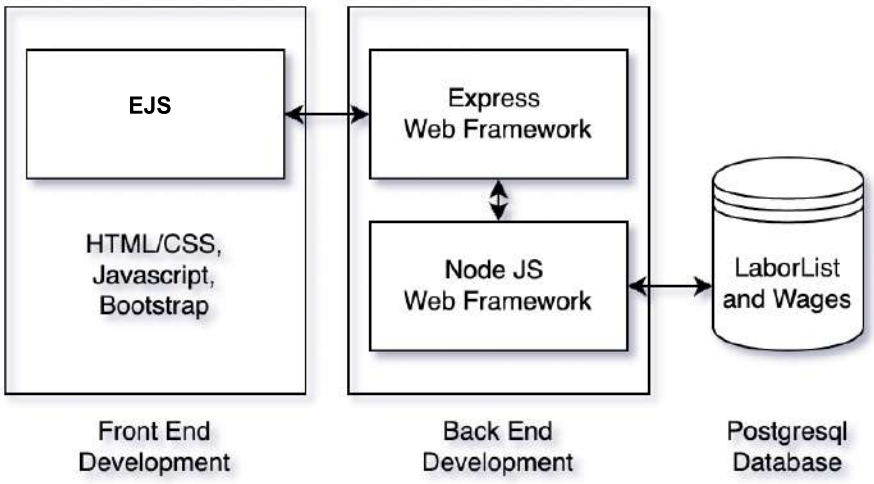
Also, special features will be implemented for the ministry of labor and employment of India such as a monitoring system that indicates any red flags towards anyone using the application and disobeying the rules and regulations.

1.4. Description

1.4.1. Product Perspective:

This database is developed for managing the workforce, getting hiring opportunities for job finders and hiring a workforce for recruiters. The database can be used and integrated with a web application, a mobile or a desktop application with the help of APIs. We will be integrating the database in our web application which uses the PEEN (PGSQL, ExpressJS,EJS, NodeJS) stack.

Below is the diagram explaining the PEEN Stack.



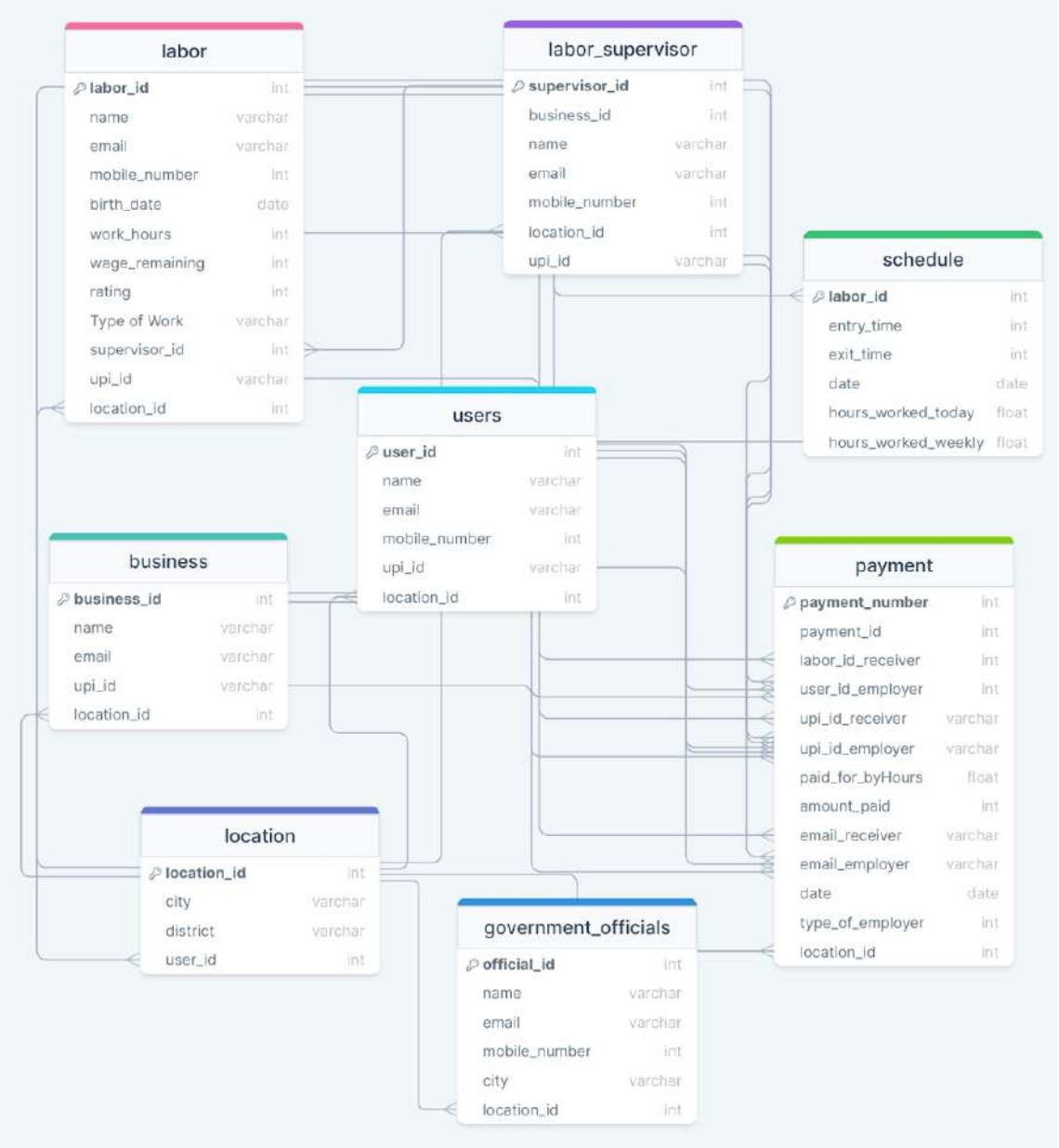
1.4.2. Product Functions:

There are various functions and features the database has. Some of the Application functions are listed below.

1. Add and delete laborers - adds or deletes the labor from the database and also saves personal details of the labors.
2. Add or delete a laborers under a supervisor - adds or deletes the labor from the supervision of a particular supervisor. (Available only for business head)

3. Show working laborers - shows the currently working laborers in the dashboard. (Available only for Supervisor head and Business head)
4. Admin - allows the developer to see the database with dummy entries and debugging features. Also, shows all the data of the database as required.
5. History of laborers - shows the history of the laborers working under a particular supervisor or a firm with the time stamps. (Available only for Supervisor head and Business head)
6. Supervisor List - Shows all the supervisors working in a business. (Available only for Supervisor head and Business head)
7. Schedule Labors (start time, end time) - gives the optimal list of employees for working in the provided time slot based on their productivity and time slot preferences.
8. Pay Laborers (Date) - distribute the wage to the laborers who worked on the given date according to their working hours.
9. Pay Laborers (Week) - only pay the laborers who worked the whole week and opted for a weekly wage option.
10. Leaves (start time, end time) - shows the list of laborers who took a leave in a given interval.
11. Hire (Type of work) - hires a specific type of worker according to an individual's needs. (for individuals)
12. Book (Type of work) - book a laborer with specific skills. (for individuals)
13. Remarks and Ratings - Give ratings to the laborers for fair hiring and wages. (For individuals and firms)
14. Automatic Payments - Automatic payments for laborers once the work is done. The money will automatically be deducted from the employer’s account.
15. Bank Information - Shows the bank details of any class of users for payment purposes.

The initial version of the ER model is given below. That shows the schema of the database.



- Everything is joined by Location ID, it is foreign key in the tables in which it is present.
- Labor ID, Business ID, Supervisor ID, User ID are all primary keys in their table and foreign keys in other tables.
- The arrow ending with 3 edges (<) is foreign key (many to one relation) in that table and the attribute that starts with the arrow is the primary key in that table. Primary key is also indicated with the key symbol (🔑).

1.4.3. User Classes and Characteristics:

3 types of user classes will be using the database: Hiring Managers and Firms (Labor Supervisor), Individual recruiters who will hire on a temporary basis, Laborers who want to get hired by individuals or by firms. Every type of class is equally likely to use the database. While the frequency of use might change. A firm will hire once in a while but a normal user might hire more frequently for smaller tasks like cleaning, plumbing.

1.4.4. Operating Environment:

The Database is made using postgresql and will be compatible with any operating system like Windows, MACOS X, Linux, Android, iOS and iPad OS because it can be integrated with any of the applications by using Node.JS or any other backend services or development environments such as Android Studio and XCode.

2. Document the Requirements Collection/ Fact-Finding Phase

2.1. Background Readings

There are many current workforce management systems like workforce.com, easymetrics.com and many sites for hiring laborers for individual employers such as urbancompany.com. Also, The Gazette of India has published the Code on Wages by Authority under the Ministry of Law and Justice as an act to amend and consolidate the laws relating to wages and bonus and matters connected therewith or incidental thereto.

1. Workforce.com: It schedules laborers according to their convenience and the employer's needs. It provides insights on labor's productivity and shows their work on a dashboard provided to the supervisor. It also

integrates the payroll within the system. So the employees are paid automatically without the company having to manually pay them. Though normal users cannot hire laborers from the website, we have a feature of labor profile, that allows the laborer data to be used by the system and recommend the laborers to normal users.

2. Urbancompany.com: It has very nice features such as hiring any type of laborers individually. But it has no dashboard feature for labor supervisors or firms that have to manage a large number of laborers. It also has no weekly payment or automatic payroll integrations.

3. Easymetrics.com: This website is a fully featured labor management system that has good machine learning algorithms but has no support for hiring capabilities. Nor does it have scheduling capabilities. Our database can be used for hiring, scheduling and for automatic payroll on hourly or weekly basis.

References used in this part of the SRS:

1. Urban Company Website: www.urbancompany.com
2. Workforce Website: https://workforce.com/
3. Easymetrics Website: easymetrics.com

List of requirements from background readings:

Need a database that can automatically pay the laborers and supervisors, have the ability to allocate the laborers to particular jobs faster than conventional methods and also provide job opportunities to the laborers while providing hiring ease for individual employers.

2.2. Interviews

Interview 1 (with supervisor of canteen workers - Role Play):

Interview plan:

System: Laborlist and Wages Database
Project Reference: LLWD/2022/10

Interviewee: Rahil, Designation: Supervisor - Canteen Workers
Interviewer: Jay Navadiya, Designation: Developer - LaborList and Wages

Date: September 27, 2022 Time: 5:00 PM
Duration: 25 Minutes Place: Canteen, DA-IICT

Purpose of the interview: To find problems and requirements of the Labor Supervisor side of the database LaborList and Wages.

Agenda/Questions prepared to ask in the interview:

1. How do you keep track of the working hours of the workers?
2. How do you distribute wages between the workers?
3. If manually, would you want an automated system that takes care of the payment on time to time without any hurdles?
4. How do you keep track of the productivity of the workers?
5. How do you schedule the workers for the job that you give them?
6. If a system is developed with features such as automatic scheduling, would you use and benefit from it?

Interview summary:

Results of the Interview:

- Supervisors cannot always keep track of the work hours of the laborers.

- Wage distribution has to be done manually and it's a burden if the work hours are not tracked properly.
- Productivity of the laborers is impossible to track manually.
- Scheduling the laborers to their preferences is hard without using a scheduling algorithm.
- The supervisors at the canteen do not have any existing labor management systems either. They would love to use such a system.

Interview 2 (with one of the labor of the canteen - Role Play):

Interview plan:

System: Laborlist and Wages Database
Project Reference: LLWD/2022/10

Interviewee: Priyansh, Designation: Canteen worker (chef)
Interviewer: Dhruv Patel, Designation: Developer - LaborList and Wages

Date: September 28, 2022 Time: 3:00 PM
Duration: 40 Minutes Place: Canteen, DA-IICT

Purpose of the interview: To find problems and requirements of the Labor Supervisor side of the database LaborList and Wages.

Agenda/Questions prepared to ask in the interview:

1. Do you get paid by cash or by online banking?
2. And if you get paid by online banking, is it done automatically, regularly?
3. Do you get the fair wage for the hours of work that you do?
4. Is the work hours managed? If yes, is it managed/updated manually by the supervisor or in some structured better way?

5. Do you think that an automated payment system will help you in contrast to the conventional payment system that is implemented right now?
6. Can you shift your job and get another job that has more wages easily with your current job? How hard is it?
7. Will you work for individual employers after regular working hours for getting more wage and helping individuals?
8. Can you keep track of your working hours and productivity of work that you do?

Interview summary:

Results of the Interview:

- Employees/ Laborers get paid in cash and it makes it hard for them to remember if they were paid fully or not for the work they have done.
- It would help the laborers in many ways if the payroll system gets automated as sometimes the supervisor doesn't give the wage on time.
- Having a labor profile and rating system would really help the laborers to get new jobs (part time and full time) and also to get higher wages based on their performance.

Interview 3 (with one of the government official - Role Play):

Interview plan:

System: Laborlist and Wages Database
Project Reference: LLWD/2022/10

Interviewee: Mr. Patel, Designation: Secretary - ministry of labor and employment
Interviewer: Dhruv Patel, Designation: Developer - LaborList and Wages

Date: September 27, 2022 Time: 9:00 PM
Duration: 40 Minutes Place: Canteen, DA-IICT

Purpose of the interview: To find problems and requirements of the government official side of the database LaborList and Wages.

Agenda/Questions prepared to ask in the interview:

1. Do you need a system to manage all the employers/laborers in a particular region?
2. Does it happen that the data represented in the papers are found to be wrong later on?
3. Is live tracking of the employers/laborers helpful to the ministry of labor and employment?

Interview summary:

Results of the Interview:

- There is a need for supervision of the age of the laborers as there are many child laborers currently working in many companies.
- Real time tracking of labor activities could help in regulating the labor laws and also it helps the government officials to not go from the paperwork submitted by the companies as it's faked countless times before. Having a real time tracking feature could prevent all that.

Interview 4 (with a normal citizen - actual):

Interview plan:

System: Laborlist and Wages Database
Project Reference: LLWD/2022/10

Interviewee: Manisha Navadiya, Designation: A teacher
Email of Interviewee: manishanavadiya11@gmail.com
Interviewer: Jay Navadiya, Designation: Developer - LaborList and Wages

Date: September 29, 2022 Time: 10:00 PM
Duration: 1 Hour Place: Canteen, DA-IICT

Purpose of the interview: To find problems and requirements of the normal users who can make use of the database of Laborlist and wages.

Agenda/Questions prepared to ask in the interview:

1. In the past, had you required any kind of labor services?
2. If yes, how hard was it to hire someone?
3. Can you get someone to do your work in under an hour?
4. Would you use an application that does the hiring process for you and find you the best laborer to do the job?
5. Has it happened that the worker you hired did not satisfactorily complete the assigned work to them?

Interview summary:

Results of the Interview:

- Normal users have to hire someone by using local connections who might not be available at times.

- They sometimes get the job done very nicely but at other times have no idea of what to do in some tasks.
- Experts of a particular field cannot always be hired so there's a requirement of such a hiring feature which lets one hire experts in just one click.

Combined observations/Final List of Requirements listed from the Interviews based on the answers given by the interviewees:

- Need an automatic payment system for the laborers instead of paying in cash or paying manually.
- Need a scheduling algorithm that does the scheduling part for the labor supervisors.
- For Labors, a dashboard is also needed for them to keep track of the work they are doing.
- For normal users, a recommendation system and easy to hire feature needs to be implemented that hires the laborers/workers according to their needs.
- For government officials, a dashboard is required for monitoring for regulation purposes in a specific area.
- Government officials also need the database as the data of laborers are many times misrepresented in the offline and more conventional format. Live tracking would be better for regulating the guidelines.

2.3. Questionnaire/s

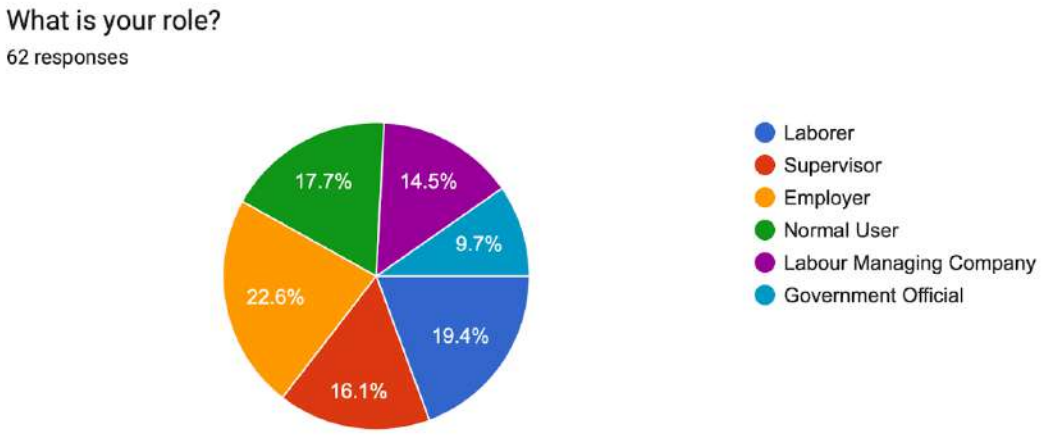
We made a google form and it was shared between laborers, friends (normal users mostly) and labor supervisors (Canteen supervisors of DA-IICT). The form screenshot is given below.

Question 1:

What is your role? *

- ◯ Laborer
- ◯ Supervisor
- ◯ Employer
- ◯ Normal User
- ◯ Labour Managing Company
- ◯ Government Official

Response:



Intent of the question:
To get to know the major user groups of the system.

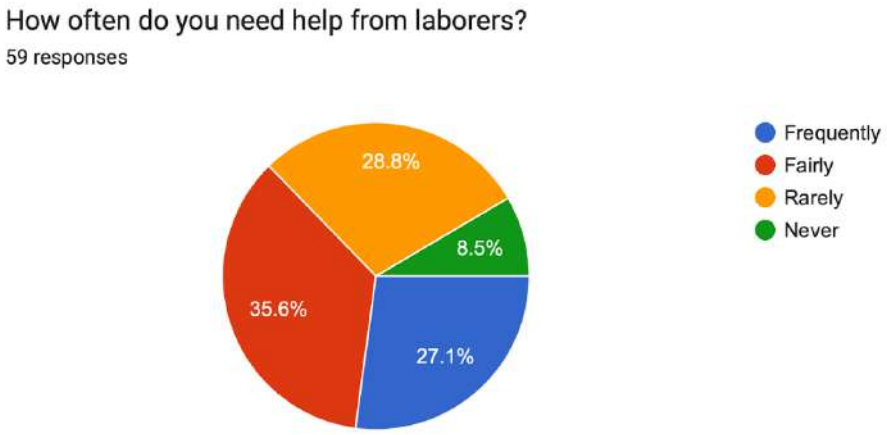
Observation from the response:
Since all the targeted users require this system, the groups are distributed almost equally with the laborers and their employers forming the majority.

Question 2:

How often do you need help from laborers?
(Fill only if you are not a laborer)

- ◯ Frequently
- ◯ Fairly
- ◯ Rarely
- ◯ Never

Response:



Intent of the question:
To gain knowledge of how often people need help with labor.

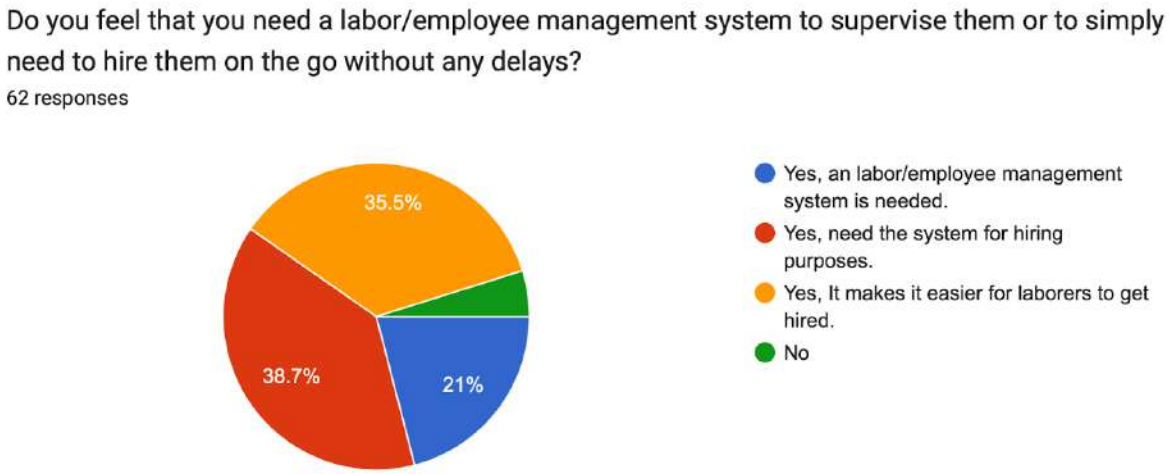
Observation from the response:
Most people require help from labor in their everyday life. There is a very small percentage of people who do not require their help.

Question 3:

Do you feel that you need a labor/employee management system to supervise them or to simply need to hire them on the go without any delays? *

- ◯ Yes, an labor/employee management system is needed.
- ◯ Yes, need the system for hiring purposes.
- ◯ Yes, It makes it easier for laborers to get hired.
- ◯ No

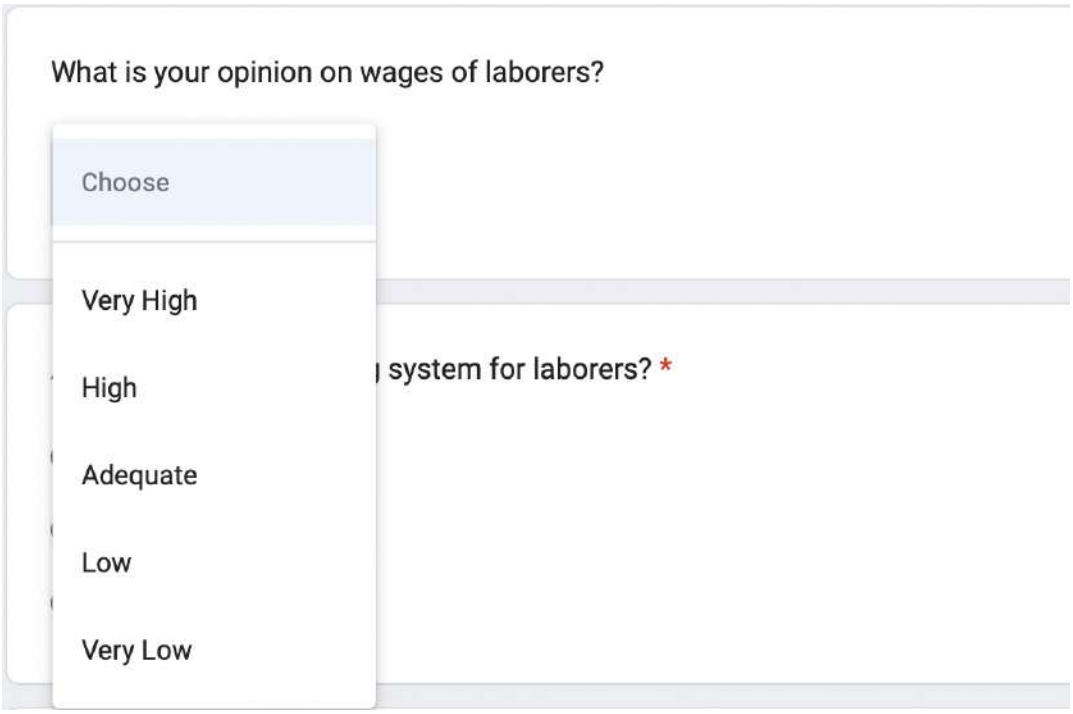
Response:



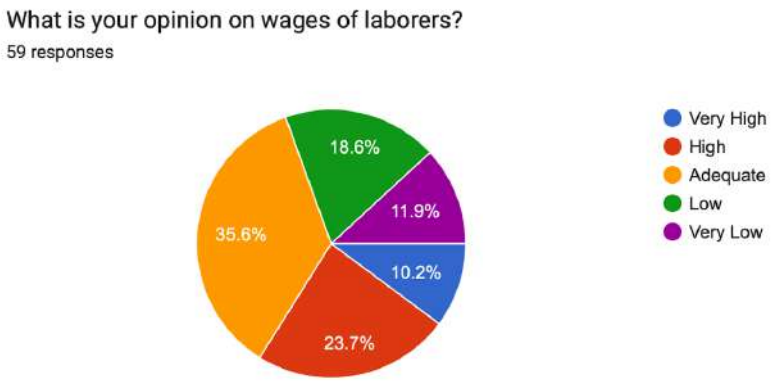
Intent of the question:
The main intent of this question was to understand whether this system should be utilized for hiring purposes or not.

Observation from the response:
There is a unanimous opinion among both employers and laborers that such a system is required for hiring of laborers.

Question 4:



Response:



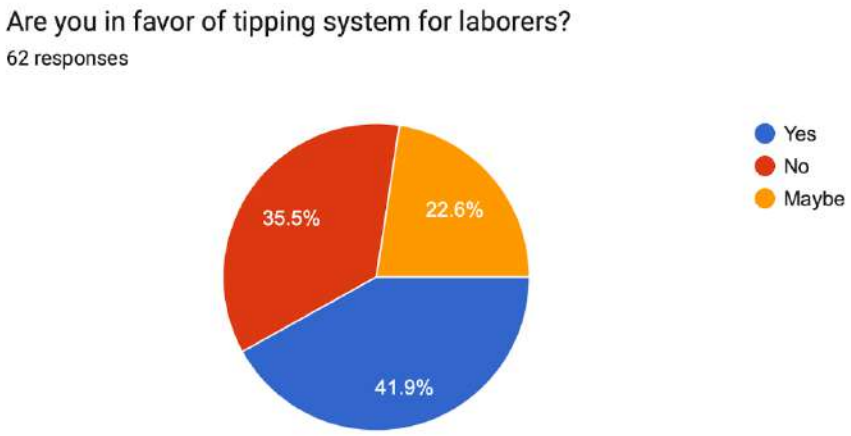
Intent of the question:
To get to know what is the general sentiment about the current wages of laborers among the users in order to develop the payment relation accordingly.

Observation from the response:
Relatively small percentage of people have extreme views on this question while the majority has its opinion somewhere in the middle.
Question 5:

Are you in favor of tipping system for laborers? *

- ◯ Yes
- ◯ No
- ◯ Maybe

Response:



Intent of the question:
To get to know whether people are in favor of a tipping system for laborers and if it is required to be implemented within the system.

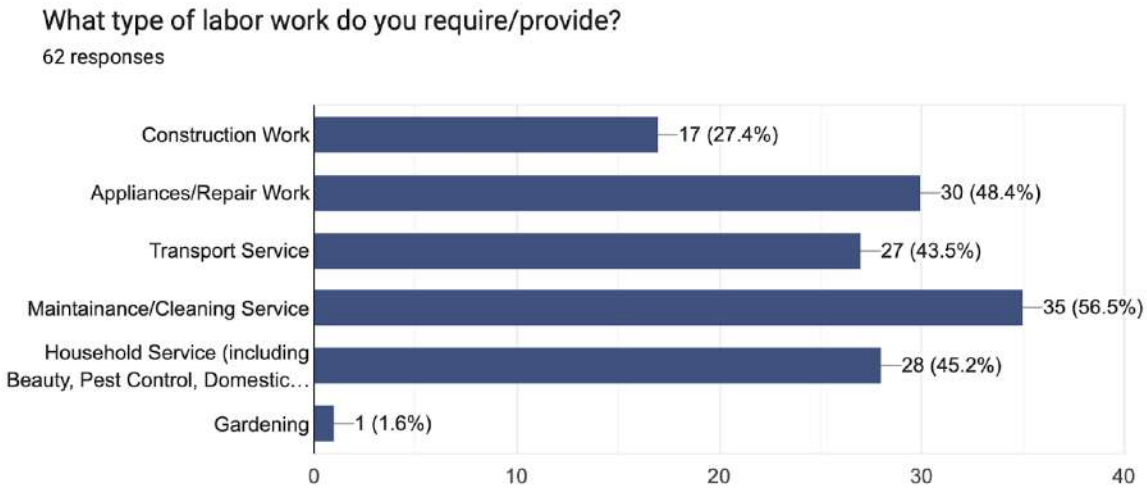
Observation from the response:
Not all users are in favor of the tipping system while many are divided on this topic. So, it is better to leave the tipping decision to the employer if they are impressed with the laborer's work. There is no significance of storing the data of tips received since we are already storing the ratings of hired laborers.

Question 6:

What type of labor work do you require/provide?

- [ ] Construction Work
- [ ] Appliances/Repair Work
- [ ] Transport Service
- [ ] Maintainance/Cleaning Service
- [ ] Household Service (including Beauty, Pest Control, Domestic Work, etc.)
- [ ] Other: _____

Response:



Intent of the question:
To gain knowledge of the type of work for which laborers are hired so one can define the categories for labor accordingly.

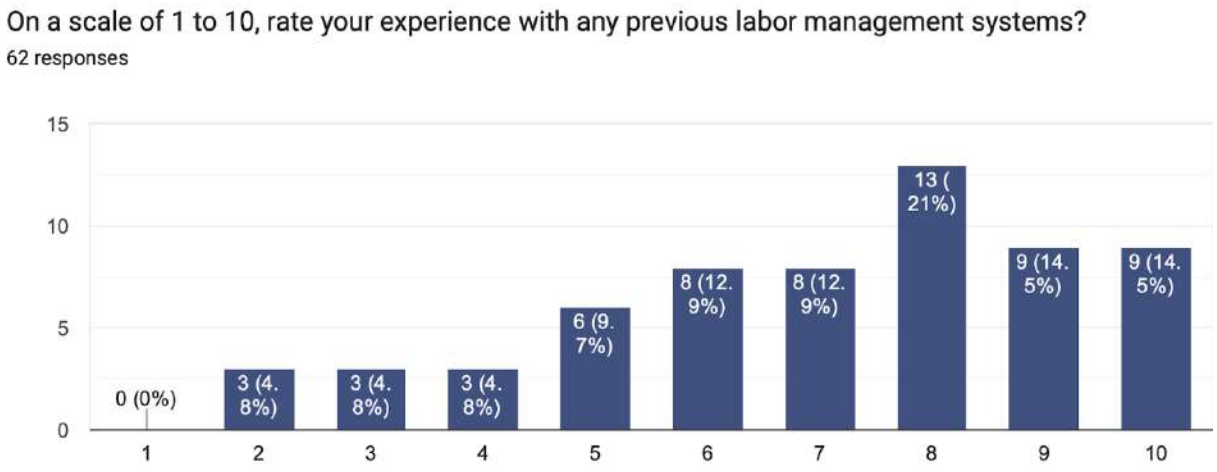
Observation from the response:
All types of labor work are required by the users almost equally. The diversity of work provided by the laborers in these general categories is enough while keeping them under the same category.

Question 7:

On a scale of 1 to 10, rate your experience with any previous labor management systems?

	1	2	3	4	5	6	7	8	9	10	
Unsatisfactory	○	○	○	○	○	○	○	○	○	○	Satisfactory

Response:



Intent of the question:
To get to know if people were satisfied with their experience with any labor management systems they used previously.

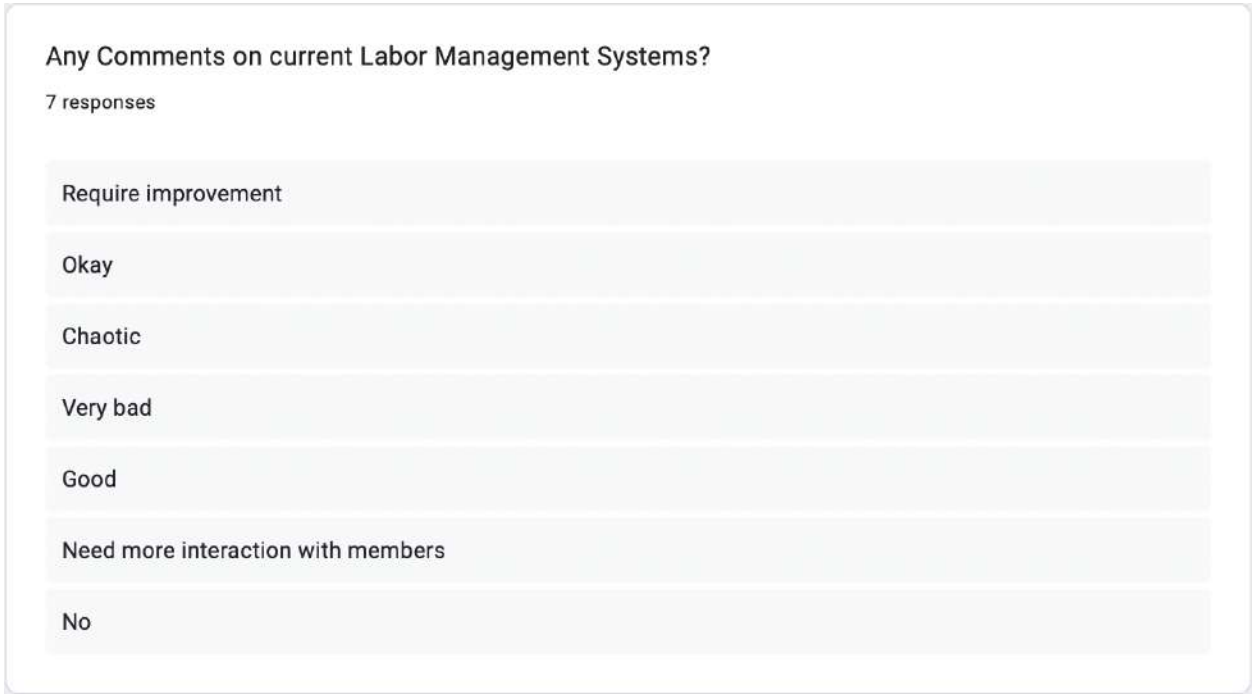
Observation from the response:
People felt that the systems were on par with their needs but still required many improvements based on the responses.

Question 8:

Any Comments on current Labor Management Systems?

Your answer

Response:



Intent of the question:
To get to know the user group's specific opinion on the current system of labor management.

Observation from the response:
The respondents feel the current system does not have all the required features and is quite challenging to manage.

Combined observations/Final list of requirements based on the answers of the questionnaires:

- As we can see in the responses, the users are very different from each other.
- The users are willing to use software like the one we are developing.
- No tipping system needs to be implemented for the laborers as shown in the response.
- The current labor management systems are not that responsive nor do they consist of all the features needed by the intended audience.

2.4. Observations

System: LaborList and Wages
Project Reference: LLWD/2022/10

Observation Subjects: 2 Laborers, 1 Labor Hiring Company head
Observations by: Dhruv Patel, Jay Navadiya

Date: 1/10/2022 Time: 15:00
Duration: 2 Hours Place: DAIICT Cafeteria

Observations:

- There is no proper system to collect information about availability of laborers. It needs to be filled manually by the supervisor in a register.
- There is no formal way to inform laborers about their duties. They need to be informed in-person individually.
- Even if a government official needs to be informed about the status of laborers, it is not tabulated anywhere. They need to rely on word-of-mouth.
- There is no proper method of payment to laborers. It is done on a daily basis which is not assured by the employer.
- There is no system of ratings/feedback which is used to let laborers and their employers know about their work quality.

3. Fact Finding Chart

Objective	Technique	Subject(s)	Time Commitment
To get background on current Labor Management companies and types of labor	Background Reading (Websites/ Blogs)	Company Websites/Reports/ blogs	1 day
To gain understanding of the current system from the subjects (laborers)	Interviews	1 Laborer	40 minutes
To understand how current Labor management works, how the payroll is managed and To establish what labor work records are kept	Interviews	1 Labor Supervisor/ Labor hiring company head	25 Minutes
To gain understanding of the Labor wages system and its business implications	Observation	2 Laborers, 1 Labor Hiring Company head	2 Hours
To find out new additional requirements to make the system streamlined and To find out how normal users hire laborers when required or if at all they require hiring laborers	Interview	1 Laborer hiring Citizen	1 Hours
To find out the needs of the government officials of the ministry of labor and employment	Interview	1 Government Official	40 Minutes

department			
To understand the user's perspective	Questionnaire	User group Respondents	3 Hours

4. List Requirements

- An automatic payment system for the laborers.
- A scheduling algorithm that does the scheduling part for the labor supervisors.
- A dashboard to keep track of the work laborers are doing.
- A recommendation system and easy to hire feature needs to be implemented to fulfill requirements.
- An algorithm needs to be developed which tracks the productivity of the laborers which lowers the load of supervisors.
- A dashboard is required for monitoring for regulation purposes in a specific area for government officials.
- Live tracking of databases for government officials to monitor the data and issue guidelines accordingly.
- Proper hardware and software for the system users to get the intended usage.

5. User Categories and Privileges

I. Laborer
Access Privileges: Relations Labor, Schedule

The Laborer will have access to all the information required by them under the two relations. They can gain knowledge of their work, payments, rating and schedules through these tables.

II. Supervisor

Access Privileges: Relations Labor, Payment, Supervisor, Location, Schedule

The Supervisor will have access to all the information required by them under these relations. They can gain knowledge of the work, payments, rating and locations of the laborers under them using these relations. Also, they would be able to schedule the laborer's work for the laborers when they are available.

III. Employer
Access Privileges: Relations Labor, Payment, Business, Supervisor, Location, Schedule

An Employer will have access to all the information required by them under these relations. They can gain knowledge of the supervisors under them and also the work, payments, rating and locations of the laborers under these supervisors using these relations. Also, they would be able to maintain the record of payments using these relations.

IV. Normal User (Citizen)
Access Privileges: Relations Labor, Normal People, Location, Schedule

A Normal Labor Hiring Person will have access to all the information required by them under these relations. They can use these to hire laborers, verify their payments, locate their laborers and also schedule the work of the laborers. Hence, they can update this information directly with the supervisor/employer.

V. Labor Managing Company
Access Privileges: Relations Labor, Payment, Business, Supervisor, Location, Schedule

A Labor Managing Company will have access to every bit of information about laborers and supervisors under the company. They can gain knowledge of the supervisors, their ids and also all the information on laborers working under these supervisors using these relations. Also, they would be able to maintain the record of payments using these relations while monitoring various businesses.

VI. Government Official
Access Privileges: Relations Labor, Business, Supervisor, Location, Government Officials

A Government Official will have access to all the businesses, their supervisors and the laborers working under them. They can use this knowledge along with their identification and location to generate a report on the laborers, their work and wage conditions. Also, they would be able to supervise all of these while being in contact with other government officials.

6. Assumptions

The following are the main assumptions made for this system:

- The user of the system has the required hardware which also fulfills the software specifications to use the system.
- The available slot of the workers is timely updated so that the supervisor and the employer can hire them accordingly.
- All the ratings and contacts provided are genuine and not used to manipulate the data in the relations.

7. Business Constraints

- All the Laborers have a registered bank account which is linked to UPI. This ID should be used to receive the payment via UPI so that the payments can be monitored.
- All the Businesses / Labor Managing Companies have a current account registered for their business which is used to carry out their business practices.
- There will be certain charges for the subscription to use the system since it needs to be hosted on a server.

8. References used for making of the SRS

1. Urban Company Website: www.urbancompany.com
2. Naukri Website: https://www.naukri.com/
3. Workforce Website: https://workforce.com/
4. Easymetrics Website: easymetrics.com
5. IEEE Template for System Requirement Specification Documents
6. Ministry of Labor and Employment, India: https://labour.gov.in/
7. Airline Database SRS
8. The Code on Wages, The Gazette of India: https://egazette.nic.in/WriteReadData/2019/210356.pdf

IT214 - DBMS

(E-R Diagram Documentation)

Case Study Database: Laborlist and Wages Database

Prepared by
Team No. 17 | Group 6 | Section 10

Group Members
Dhruv Patel - 202001271
Jaykumar Navadiya - 202001465

Under the guidance of
Prof. Chhaya
Teaching Assistant - Vinay Maharaaj

Date
October 14, 2022

9. Final Problem Description

9.1. Purpose of the Project:

The database, Laborlist and Wages will maintain a record of all work done by laborers on a daily basis to store and distribute wages. It will explain the purpose and features of the database such as the schema and relations of

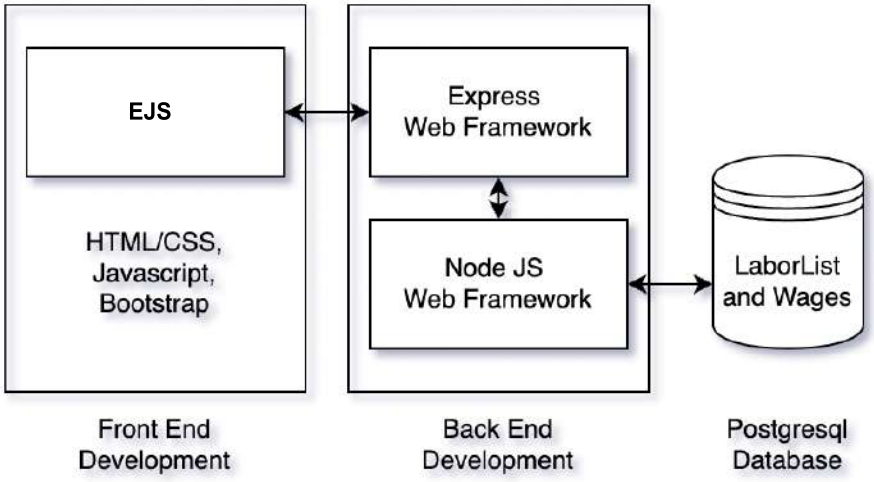
the database, how the Labor Supervisor will benefit from using such a database instead of using a file management system, the constraints under which the database must operate (i.e. the constraints on some attributes). The database will be a part and backend of the web application (PEEN stack) consisting of features like managing, hiring and scheduling of laborers and providing a dashboard to the firms with the help of the database at the backend.

The project is required for a more efficient, regulated and secure system of managing laborers and their wages. The current system which utilizes manual records and files is chaotic and not secure since data can be manipulated easily by anyone.This project makes the process completely digital and accessible to all the target user groups in order to maintain accountability.

9.2. Product Perspective:

This database is developed for managing the workforce, getting hiring opportunities for job finders and hiring a workforce for recruiters. The database can be used and integrated with a web application, a mobile or a desktop application with the help of APIs. We will be integrating the database in our web application which uses the PEEN (PGSQL, ExpressJS, EJS, NodeJS) stack.

Below is the diagram explaining the PEEN Stack.



9.3. Product Functions and General overview of the system:

Overview of the Laborlist and wages database:

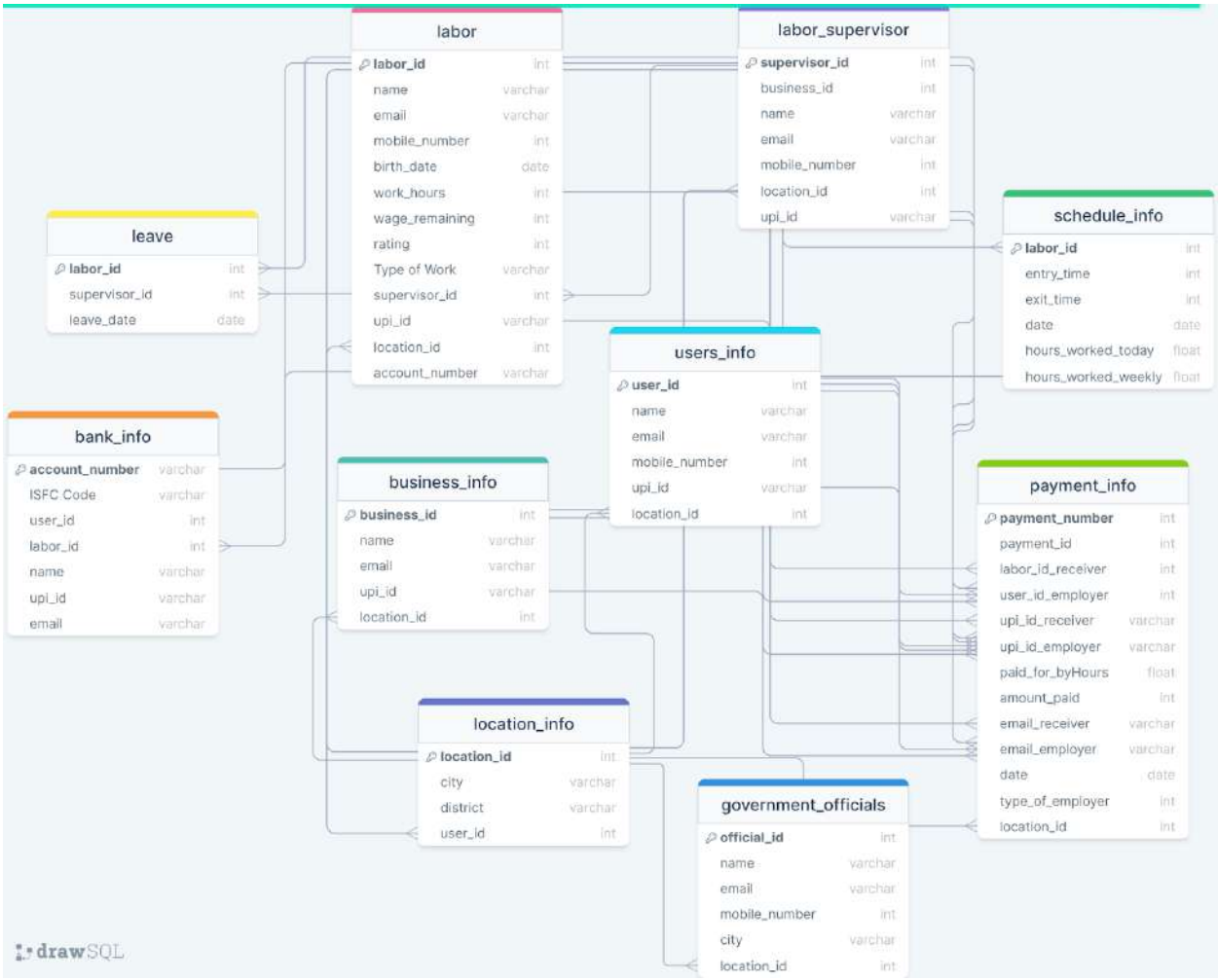
The database will consist of all the information of labor, labor supervisor, businesses who have both laborers and labor supervisors, government officials who can supervise the whole town or district with their designation in the ministry of labor and employment of India, normal users who are potential employers of the laborers who can make a profile of their own with the help of our database and software solution. Personal details may involve mobile number, email address, age and birthdate. The database will also provide solutions for scheduling of laborers and maintaining the work hours by using start and end time of working of the laborers. Also, the payroll system will be automated by using bank_details of everyone that consists of every banking related details like account number, UPI ID and IFSC Code. Every payment that happens through the platform will be stored via the payment table of the database with proper timestamps.

There are various functions and features the database has. Some of the Application functions are listed below.

1. Add and delete laborers - adds or deletes the labor from the database and also saves personal details of the labors.
2. Add or delete a laborers under a supervisor - adds or deletes the labor from the supervision of a particular supervisor. (Available only for business head)
3. Show working laborers - shows the currently working laborers in the dashboard. (Available only for Supervisor head and Business head)
4. Admin - allows the developer to see the database with dummy entries and debugging features. Also, shows all the data of the database as required.
5. History of laborers - shows the history of the laborers working under a particular supervisor or a firm with the time stamps. (Available only for Supervisor head and Business head)
6. Supervisor List - Shows all the supervisors working in a business. (Available only for Supervisor head and Business head)
7. Schedule Labors (start time, end time) - gives the optimal list of employees for working in the provided time slot based on their productivity and time slot preferences.
8. Pay Laborers (Date) - distribute the wage to the laborers who worked on the given date according to their working hours.
9. Pay Laborers (Week) - only pay the laborers who worked the whole week and opted for a weekly wage option.
10. Leaves (start time, end time) - shows the list of laborers who took a leave in a given interval.
11. Hire (Type of work) - hires a specific type of worker according to an individual's needs. (for individuals)
12. Book (Type of work) - book a laborer with specific skills. (for individuals)
13. Remarks and Ratings - Give ratings to the laborers for fair hiring and wages. (For individuals and firms)
14. Automatic Payments - Automatic payments for laborers once the work is done. The money will automatically be deducted from the employer’s account.

15. Bank Information - Shows the bank details of any class of users for payment purposes.

The initial version of the tables with its attributes is given below. That shows the initial schema of the database.



- Everything is joined by Location ID, it is foreign key in the tables in which it is present.
- Labor ID, Business ID, Supervisor ID, User ID are all primary keys in their table and foreign keys in other tables.
- The arrow ending with 3 edges (<) is foreign key (many to one relation) in that table and the attribute that starts with the arrow is

the primary key in that table. Primary key is also indicated with the key symbol ( ).

9.4. User Classes and Characteristics:

3 types of user classes will be using the database: Hiring Managers and Firms (Labor Supervisor), Individual recruiters who will hire on a temporary basis, Laborers who want to get hired by individuals or by firms. Every type of class is equally likely to use the database. While the frequency of use might change. A firm will hire once in a while but a normal user might hire more frequently for smaller tasks like cleaning, plumbing.

User Categories and Privileges:

I. Laborer
Access Privileges: Relations Labor, Schedule

The Laborer will have access to all the information required by them under the two relations. They can gain knowledge of their work, payments, rating and schedules through these tables.

II. Supervisor
Access Privileges: Relations Labor, Payment, Supervisor, Location, Schedule

The Supervisor will have access to all the information required by them under these relations. They can gain knowledge of the work, payments, rating and locations of the laborers under them using these relations. Also, they would be able to schedule the laborer's work for the laborers when they are available.

III. Employer

Access Privileges: Relations Labor, Payment, Business, Supervisor, Location, Schedule

An Employer will have access to all the information required by them under these relations. They can gain knowledge of the supervisors under them and also the work, payments, rating and locations of the laborers under these supervisors using these relations. Also, they would be able to maintain the record of payments using these relations.

IV. Normal User (Citizen)
Access Privileges: Relations Labor, Normal People, Location, Schedule

A Normal Labor Hiring Person will have access to all the information required by them under these relations. They can use these to hire laborers, verify their payments, locate their laborers and also schedule the work of the laborers. Hence, they can update this information directly with the supervisor/employer.

V. Labor Managing Company
Access Privileges: Relations Labor, Payment, Business, Supervisor, Location, Schedule

A Labor Managing Company will have access to every bit of information about laborers and supervisors under the company. They can gain knowledge of the supervisors, their ids and also all the information on laborers working under these supervisors using these relations. Also, they would be able to maintain the record of payments using these relations while monitoring various businesses.

VI. Government Official

Access Privileges: Relations Labor, Business, Supervisor, Location, Government Officials

A Government Official will have access to all the businesses, their supervisors and the laborers working under them in certain locations. They can use this knowledge along with their identification and location to generate a report on the laborers, their work and wage conditions. Also, they would be able to supervise all of these while being in contact with other government officials.

9.5. Operating Environment:

The Database is made using postgresql and will be compatible with any operating system like Windows, MACOS X, Linux, Android, iOS and iPad OS because it can be integrated with any of the applications by using Node.JS or any other backend services or development environments such as Android Studio and XCode.

10. Noun & Verb Analysis

10.1. Extracted Nouns and Verbs from Problem Description using Noun Analysis Method

Serial no.	Nouns	Verbs
1	database	maintain
2	Laborlist	store
3	Wages	distribute

Serial no.	Nouns	Verbs
4	record	explain
5	work	benefit
6	laborers	operate
7	wages	hiring
8	schema	managing
9	relations	scheduling
10	Labor supervisor	provide
11	File	integrating
12	System	help
13	constraints	add
14	attributes	delete
15	web-application	saving
16	features	supervising
17	dashboard	showing
18	firms	allow
19	database	working
20	backend	schedule
21	product	give
22	features	pay
23	workforce	leave
24	labor	opting

Serial no.	Nouns	Verbs
25	opportunities	booking
26	desktop-application	hire
27	APIs	update
28	PEEN	consists of
29	PGSQL	access
30	ExpressJS	locate
31	EJS	verify
32	NodeJS	identify
33	stack	modify
34	diagram	insert
35	functions	sorting
36	application	completing
37	details	register
38	supervisor	hold
39	supervision	manipulate
40	business	
41	head	
42	Admin	
43	developer	
44	entries	
45	data	

Serial no.	Nouns	Verbs
46	history	
47	timestamps	
48	schedule	
49	time	
50	start-time	
51	end-time	
52	list	
53	employees	
54	time-slot	
55	preferences	
56	date	
57	hours	
58	week	
59	option	
60	Leaves	
61	Leave_date	
62	interval	
63	hire	
64	Type of work	
65	skills	
66	individuals	

Serial no.	Nouns	Verbs
67	remarks	
68	ratings	
69	Automatic payments	
70	Bank Information	
71	bank details	
72	users	
73	payment	
74	tables	
75	labor_id	
76	name	
77	email	
78	mobile_number	
79	birth_date	
80	work_hours	
81	wage_remaining	
82	rating	
83	type_of_work	
84	supervisor_id	
85	upi_id	
86	location_id	
87	labor_supervisor	

Serial no.	Nouns	Verbs
88	supervisor_id	
89	business_id	
90	schedule_laborers	
91	entry_time	
92	exit_time	
93	date	
94	hours_worked_today	
95	hours_worked_weekly	
96	payment_info	
97	payment_numer	
98	payment_id	
99	labor_id_receiver	
100	user_id_employer	
101	upi_id_receiver	
102	upi_id_employer	
103	paid_by_hours	
104	amount_paid	
105	email_receiver	
106	email_employer	
107	date	
108	type_of_employer	

Serial no.	Nouns	Verbs
109	government_officials	
110	official_id	
111	location	
112	city	
113	district	
114	location_id	
115	users_info	
116	user_id	
117	business_info	
118	business_id	
119	Individual recruiters	
120	Hiring managers	
121	firms	
122	foreign key	
123	primary key	
124	symbol	
125	MACOS X	
126	Windows	
127	Android	
128	iOS	
129	Android Studio	

Serial no.	Nouns	Verbs
130	XCode	
131	Normal User	
132	account_number	
133	ISFC Code	
134	upi_id	

10.3. Accepted Nouns List

Candidate Entity Set	Candidate Attribute Set	Candidate Relationship Set
labor	Labor_ID, Work Hours, Wage_remaining, Rating, Supervisor_ID, account_number, Type of Work, Location ID, bank_number	has
labor_supervisor	Supervisor_ID, Business_ID, account_number, Email, Mobile No., Location_ID	Identifies as, hires, pays, supervises
schedule_info	Labor ID, Entry Time, Exit Time, time_slot, date, hours_worked_weekly, hours_worked_today	consists of, registers
users_info	user_id, account_id, location_id	Identify by, hires, pays, employes
payment_info	Payment_number, payment_id, labor_id_receiver, user_id_employer,	Pay using

	account_number_employer, account_number_receiver, paid_for_byHours, amount_paid, email_receiver, email_employer, date, type_of_employer, location_id	
bank_info	Account_number, ISFC_code, upi_id	consists of
location_info	Location_id, city, district, user_id	Has, consists of
business_info	Business_id, account_number, location_id	Pays, consists of, employes
government_officials	Official_id, location_id	consists of, regulates and supervises
leaves	Labor_id, supervisor_id, leave_date	Registers, modifies
personal_information	User_id, name, email, mobile_number, type_of_user	
hourly_labors	Labor_id, hourly_wage, hours_worked	
weekly_labors	Labor_id, weekly_wage, hours_worked	

10.4. Rejected Nouns list with the reason of rejection

Noun	Reject Reason
database	General

Noun	Reject Reason
Laborlist	General
Wages	General
record	General
work	Irrelevant
laborers	Duplicate
wages	Duplicate
schema	General
relations	General
File	General
System	General
constraints	General
attributes	General
web-application	General
features	General
dashboard	General
firms	Duplicate
database	General
backend	General
product	General
features	General
workforce	Vague

Noun	Reject Reason
labor	Vague
opportunities	Irrelevant
desktop-application	General
APIs	General
PEEN	General
PGSQL	General
ExpressJS	General
EJS	General
NodeJS	General
stack	Irrelevant
diagram	Irrelevant
functions	General
application	General
details	Vague
supervisor	Duplicate
supervision	Association
head	Association
Admin	Association
developer	Irrelevant
entries	General
data	General

Noun	Reject Reason
history	General
timestamps	Vague
time	Vague
start-time	Attribute
end-time	Attribute
list	General
employees	Association
time-slot	Duplicate
preferences	Vague
date	Irrelevant
hours	Irrelevant
week	Irrelevant
option	General
interval	Irrelevant
hire	General
Type of work	Duplicate
skills	Irrelevant
individuals	Vague
remarks	Duplicate
Automatic payments	Irrelevant
Bank Information	Duplicate

Noun	Reject Reason
tables	General
labor_id	Attribute
name	Attribute
email	Attribute
mobile_number	Attribute
birth_date	Attribute
work_hours	Attribute
wage_remaining	Attribute
rating	Attribute
type_of_work	Attribute
supervisor_id	Attribute
upi_id	Attribute
location_id	Attribute
labor_supervisor	Attribute
supervisor_id	Attribute
business_id	Attribute
schedule_laborers	Attribute
entry_time	Attribute
exit_time	Attribute
date	Attribute
hours_worked_today	Attribute

Noun	Reject Reason
hours_worked_weekly	Attribute
payment_info	Attribute
payment_numer	Attribute
payment_id	Attribute
labor_id_receiver	Attribute
user_id_employer	Attribute
upi_id_receiver	Attribute
upi_id_employer	Attribute
paid_by_hours	Attribute
amount_paid	Attribute
email_receiver	Attribute
email_employer	Attribute
date	Attribute
type_of_employer	Attribute
government_officials	Attribute
official_id	Attribute
location	Attribute
city	Attribute
district	Attribute
location_id	Attribute
users_info	Attribute

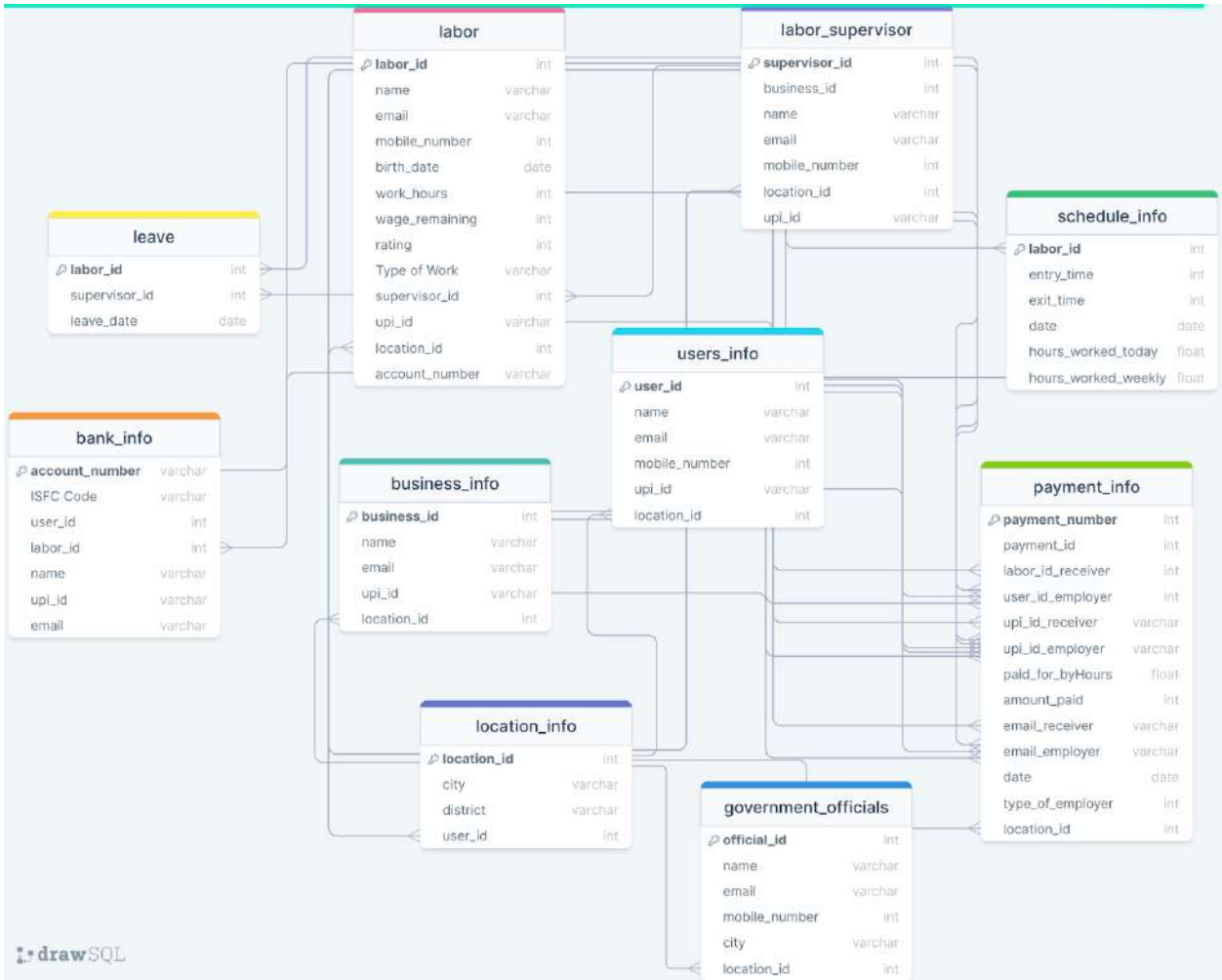
Noun	Reject Reason
user_id	Attribute
business_info	Attribute
business_id	Attribute
Individual recruiters	Association
Hiring managers	Association
firms	Duplicate
foreign key	General
primary key	General
symbol	Irrelevant
MACOS X	General
Windows	General
Android	General
iOS	General
Android Studio	General
XCode	General
Normal User	Duplicate
Leave date	Attribute
Account number	Attribute
IFSC Code	Attribute
UPI ID	Duplicate

11. ER Diagrams

11.1. Version 1:

→ This version contains entities and attributes in the form of a basic diagram only.

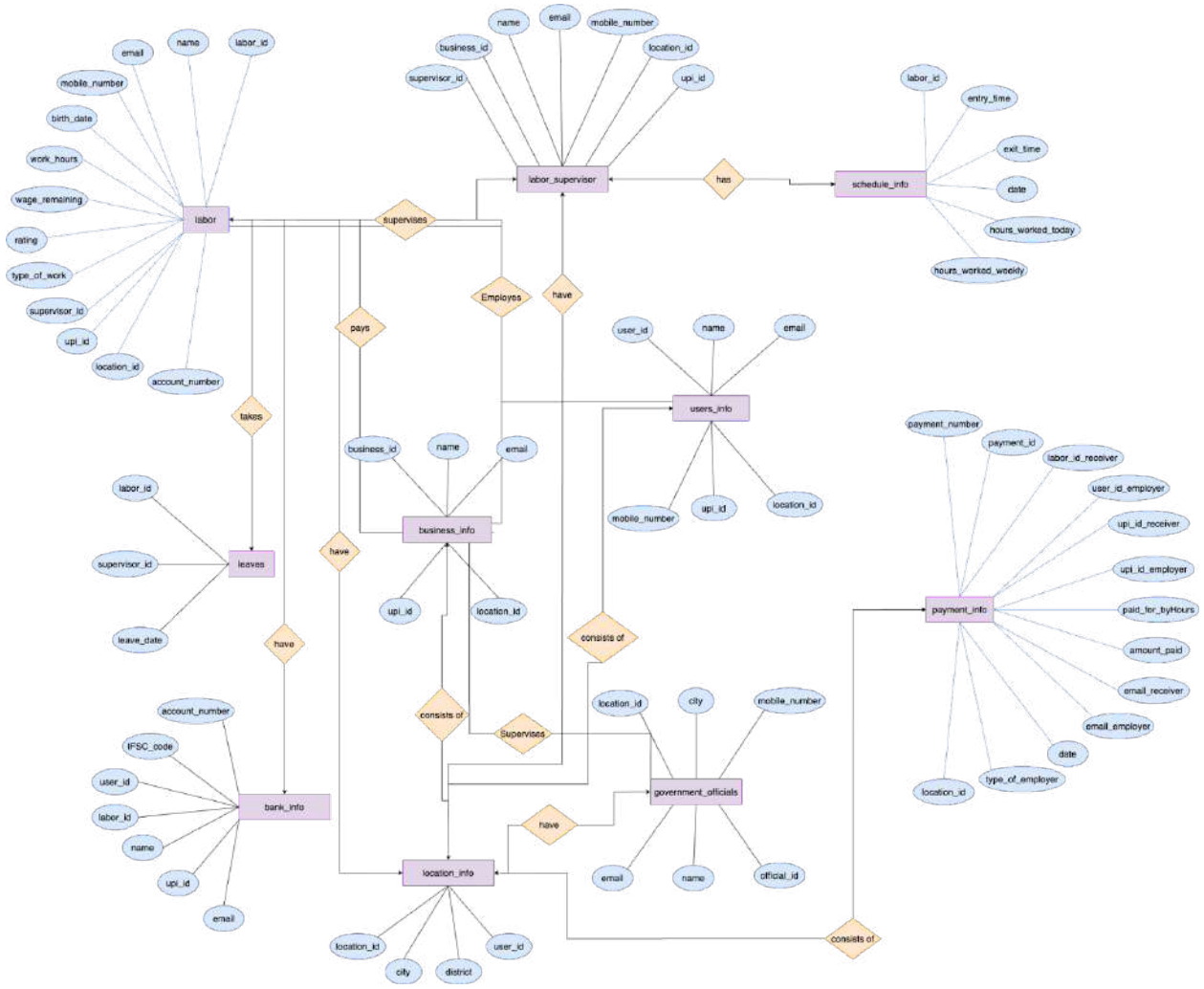
→ This diagram is used just to visualize the database system for future versions of the E-R diagram.The E-R diagram will be built upon this illustration using the draw.io tool.



11.2. Version 2:

→ This version consists of Entities, Attributes and relations between entities.

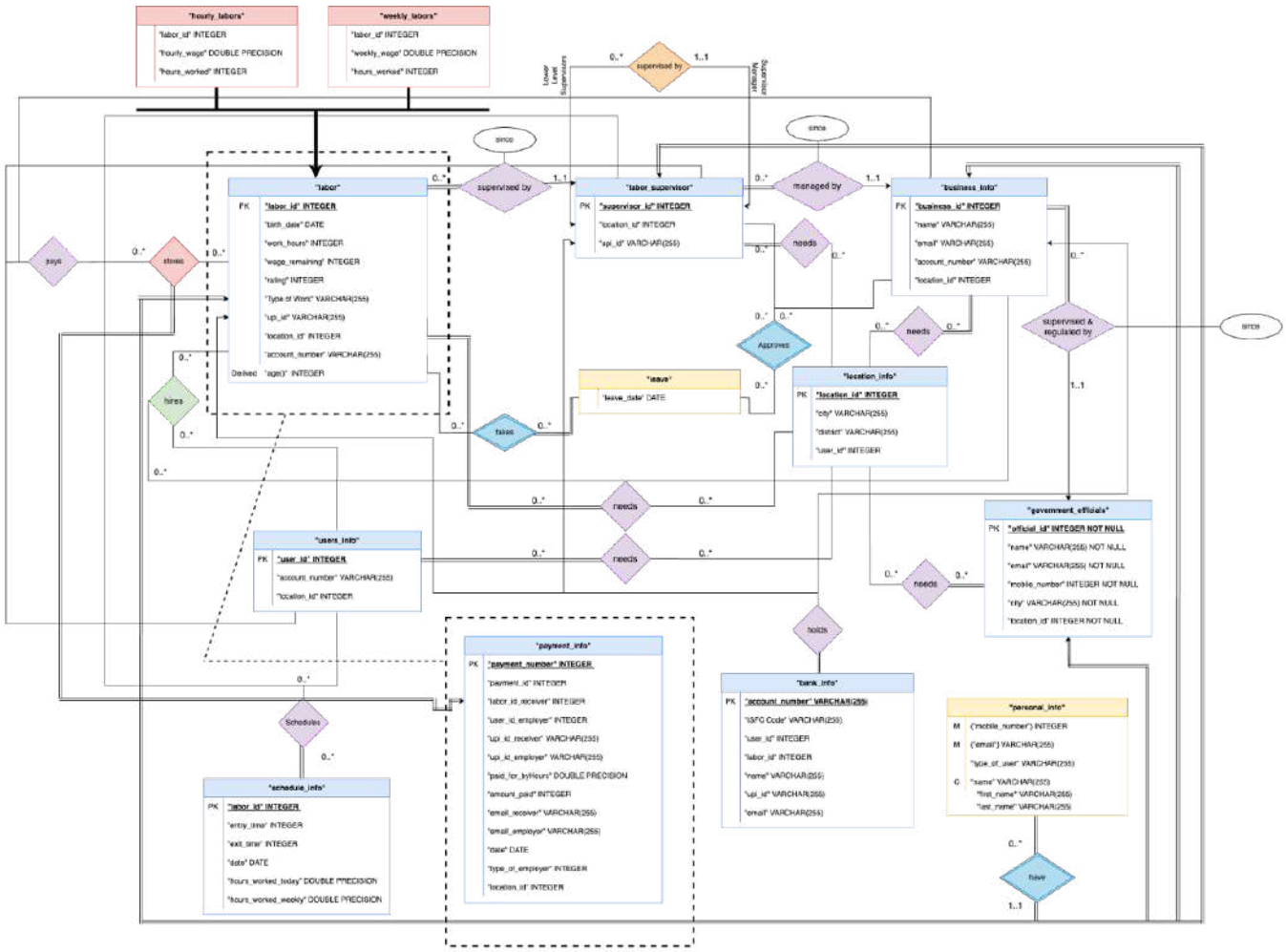
→ The E-R Diagram model used for this version is the Chen Model for database notation. It provides an abstract view of the E-R diagram with using different shapes which provide a good representation of Entities, Attributes and Relations for initial versions and brainstorming.



11.3. Final Version:

→ The final version of the E-R diagram has all of the features implemented in the diagram and are listed below. The previous version did not have types of entities and relationships defined in the diagram.

→ The following version consists of an E-R diagram that contains all the information regarding the entities and the relations.



Notation

non binary relationship

recursive relationship

binary relationship

Aggregation

Identifying Relationship

Entity

Weak Entity

Strong Entity

Entities part of Aggregation

Specialised Entity

many-to-many relationship

one-to-one relationship

many-to-one relationship

cardinality limits

Attribute type is written on the left pane of the entity table and also follows standard notation

PK - Primary Key
C - Composite Attribute
D() - Derived Attribute
M {} - Multivalued Attribute

Descriptive Attribute

Total Participation

Partial Participation

The ER Diagram is also available at draw.io platform. The editable PNG file is attached in the zip file.

Entity Types:

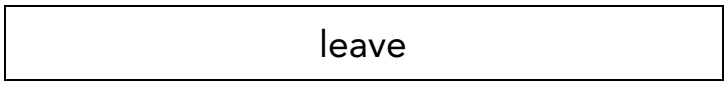
The type of entities used in the E-R diagram are:

1] Strong Entity: The system consists of many strong entities which have a primary key defined and are not dependent on the attributes.

Strong Entities
labor
labor_supervisor
schedule_info
business_info
government_officials
payment_info
user_info
location_info
bank_info

2] Weak Entity: There are two weak entities in the system which do not have a primary key i.e., a unique key to define and are dependent on other attributes.

Weak Entities
personal_info



Extended Features of E-R Diagram:

The type of relationships used in the E-R diagram are:

1] Hierarchy (Generalization & Specialization): There are certain ISA hierarchies used in the LaborList and Wages System which are:

1) Hourly Laborers and Weekly Laborers → Labor

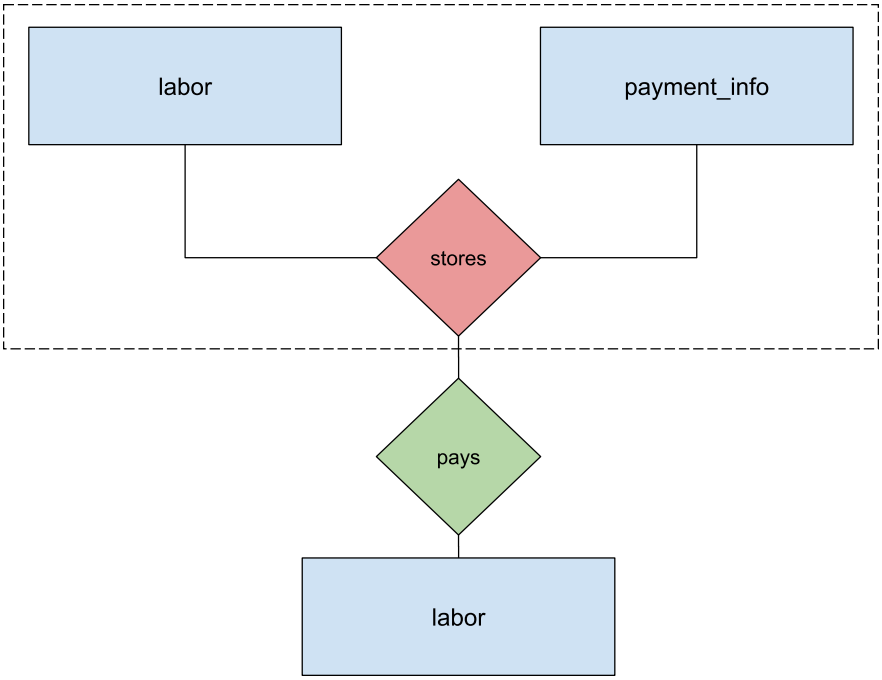
Used for the working of the Labor Management and to keep a track or record of its working. The users can utilize it to gain only the required data and thus function accordingly.

2) Normal Users & Labor Supervisors & Business → Employers

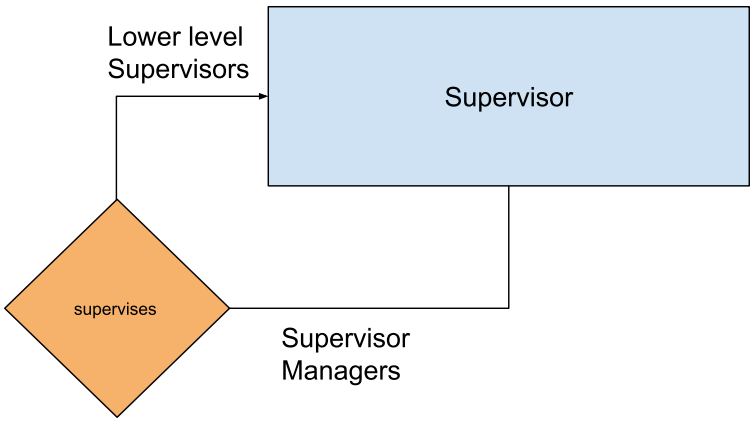
Used for generalizing the employers who are to hire a laborer. The employer can see which laborers are available for hiring and the laborers will have access to the information provided by their employer.

These hierarchies can be used to provide access to certain information to the user who should have the privilege to do so.

2] Aggregation: There is a simple usage of aggregation in the database between the entities labor and payment_info. The entities aggregate together to form a more meaningful entity which is used for storing the payment information of laborers used to make their wage payments.

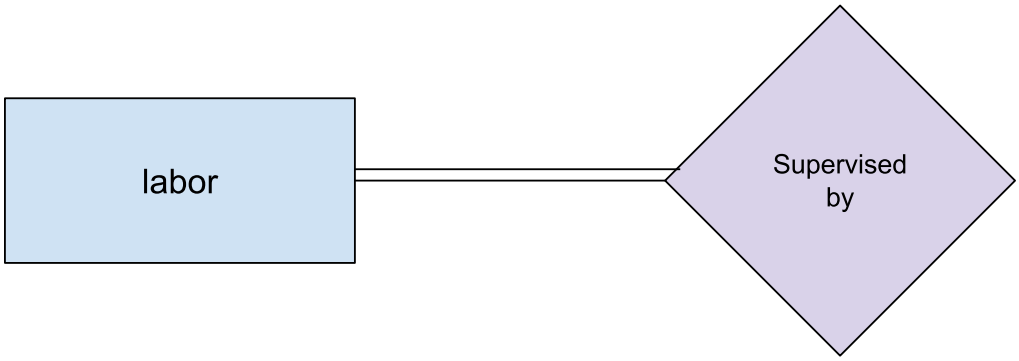


3] Recursive Relationship: There is one recursive relationship in the system which is to supervise. The supervisors on a lower level can be under a higher level supervisor whose task is to keep records of all the lower level supervisors under them. Thus, the supervisor relation becomes recursive in case there are multiple levels of supervisors in a labor management company.



Participation Constraints:

Total Participation: There is one relation in the system which has total participation. All the laborers are required to have a supervisor who would be assigned to them. Hence, the relationship of laborers being supervised on the entity labor requires total participation.



Partial participation: It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set.

Example of partial participation in the system: All government officials might not be supervising or regulating the laborer system. So it's called partial participation of government officials.

Types of attributes used in the ER Diagram:

1] Single-valued Attribute: Attributes that have only one value per entity instance. There are multiple instances of single-valued attributes in the system since they are very common.

2] Multi-valued Attribute: Whenever an attribute has several values for a given entity instance, it is considered to be a multi-valued attribute. Email and mobile_number are examples of such attributes in the system since they can take up multiple values for a single user.
3] Derived Attribute: A derived attribute is an attribute that can be derived from another attribute. One such instance of derived attribute

used in the system is the age of a user which can be derived from their birth_date.

4] Composite Attribute: The term composite attribute refers to an attribute whose components can be split. In the system, name is one such composite attribute in the personal_info entity. This attribute can be further split into first_name and last_name.

5] Descriptive Attribute: The attributes which are used to describe a relationship in an E-R model are called descriptive attributes. For example, since is an attribute which is used multiple times to describe the relationships and their time.

IT214 - DBMS
(E-R Diagram to Relational Model, DDL Script)

Case Study Database: Laborlist and Wages Database

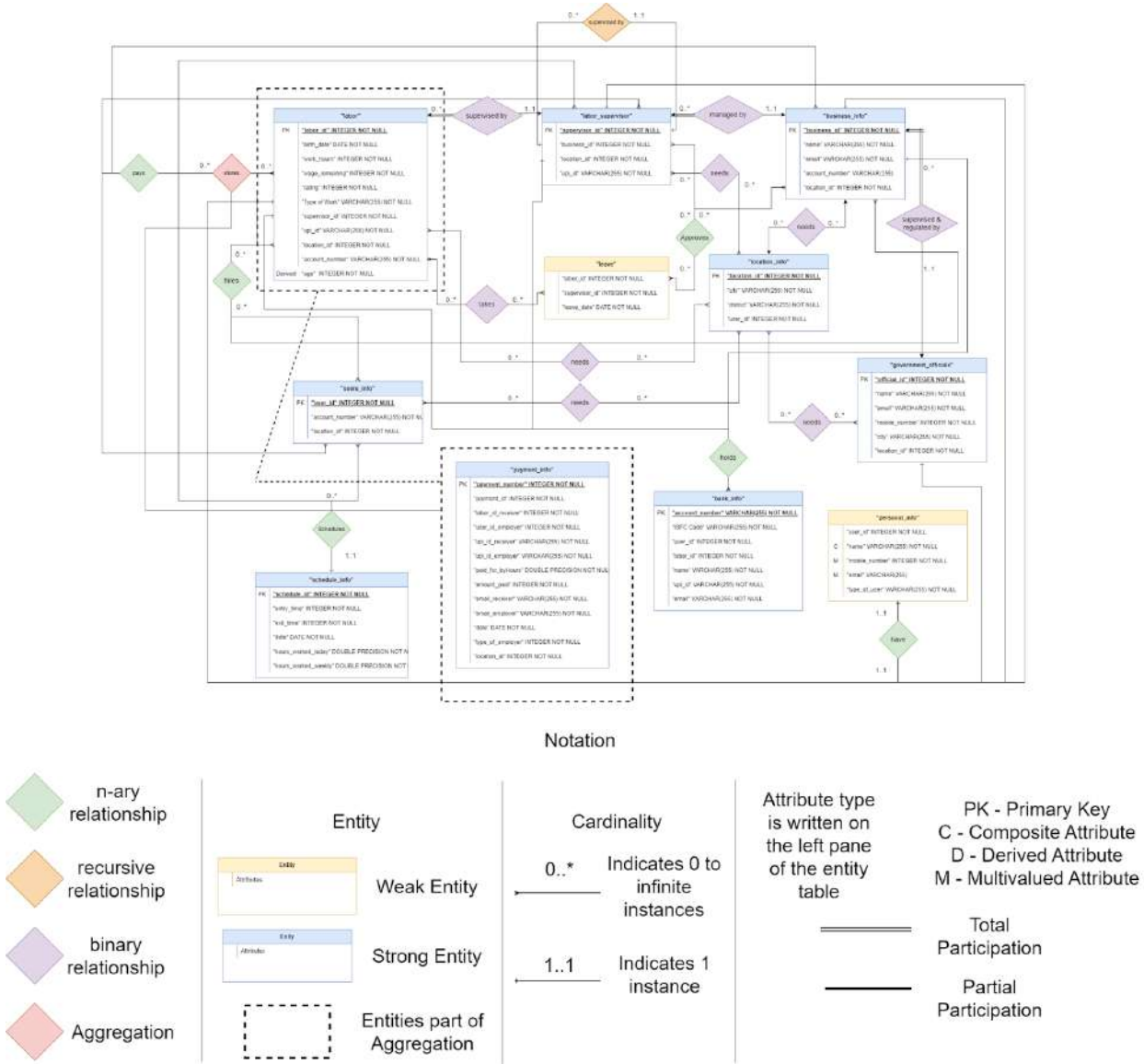
Prepared by
Team No. 17 | Group 6 | Section 10

Group Members
Dhruv Patel - 202001271
Jaykumar Navadiya - 202001465

Under the guidance of
Prof. Chhaya
Teaching Assistant - Vinay Maharaaj

Date
November 4, 2022

12. ER Diagram Final Version



Notation

13. E-R Model to Relational Model Mapping

1] labor(labor_id, birth_date, work_hours, wage_remaining, rating, type_of_work, supervisor_id ,supervised_since, location_id, account_number, password)

2] labor_supervisor(supervisor_id, managed_since, business_id, location_id, account_number, password)

3] location_info(location_id, city, district, user_id)

4] government_officials(official_id, location_id)

5] business_info(business_id, name, email, location_id, official_id, regulated_since, password)

6] users_info(user_id, account_number, location_id, password)

7] payment_info(payment_number, payment_id, labor_id_receiver, user_id_employer, account_number_receiver, account_number_employer, paid_for_byHours, amount_paid, email_receiver, email_employer, date, type_of_employer, location_id)

8] schedule_info(schedule_id, entry_time, exit_time, date, hours_worked_today, hours_worked_weekly)

9] bank_info(account_number, IFSC Code, user_id, labor_id, name, upi_id, email)

10] hourly_labors(labor_id, hourly_wage, hours_worked)

11] weekly_labors(labor_id, weekly_wage, hours_worked)

12] leave(leave_id, leave_date)

13] personal_info(user_id, labor_id, supervisor_id, business_id, official_id, mobile_number, type_of_user, name, email)

14] supervisor_manager(lowerlevel_supervisor_id,manager_id, supervised_since)

15] needs_location(user_id, labor_id, supervisor_id, business_id, official_id, location_id)

16] schedules(user_id, supervisor_id,labor_id)

17] hires(user_id, business_id, labor_id)

18] takes_leave(user_id,leave_id, business_id, supervisor_id, labor_id)

19] approves(user_id,leave_id, business_id, supervisor_id, labor_id)

20] holds(user_id, labor_id, supervisor_id, business_id)

21] stores(labor_id, payment_number)

22] pays(business_id, supervisor_id, labor_id, payment_number)

14. DDL Scripts

```
CREATE TABLE IF NOT EXISTS "labor"(
        "labor_id" INTEGER NOT NULL,
        "birth_date" DATE ,
        "work_hours" INTEGER ,
        "wage_remaining" INTEGER ,
        "rating" INTEGER ,
        "Type of Work" VARCHAR(255) ,
        "supervisor_id" INTEGER ,
         "age" INTEGER ,
        "supervised_since" INTEGER ,
```

```
        "location_id" INTEGER ,
        "account_number" VARCHAR(255) ,
        PRIMARY KEY("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id")
);
```

```
CREATE TABLE IF NOT EXISTS "labor_supervisor"(
        "supervisor_id" INTEGER NOT NULL,
        "managed_since" INTEGER ,
        "business_id" INTEGER ,
        "location_id" INTEGER ,
        "account_number" VARCHAR(255) ,
        PRIMARY KEY("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id")
);
```

```
CREATE TABLE IF NOT EXISTS "location_info"(
        "location_id" INTEGER NOT NULL,
        "city" VARCHAR(255) ,
        "district" VARCHAR(255) ,
        "user_id" INTEGER ,
        PRIMARY KEY("location_id")
);
```

```
CREATE TABLE IF NOT EXISTS "government_officials"(
        "official_id" INTEGER NOT NULL,
        "city" VARCHAR(255) ,
        "location_id" INTEGER ,
        PRIMARY KEY("official_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id")
);
```

```
CREATE TABLE IF NOT EXISTS "business_info"(
        "business_id" INTEGER NOT NULL ,
        "name" VARCHAR(255) ,
```

```
        "email" VARCHAR(255) ,
        "account_number" VARCHAR(255) ,
        "location_id" INTEGER ,
        "official_id" INTEGER ,
        "regulated_since" DATE ,
        PRIMARY KEY("business_id"),
        FOREIGN KEY("official_id") REFERENCES "government_officials"("official_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id")
);
```

```
CREATE TABLE IF NOT EXISTS "users_info"(
        "user_id" INTEGER NOT NULL,
        "account_number" VARCHAR(255) ,
        "location_id" INTEGER ,
        PRIMARY KEY("user_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id")
);
```

```
CREATE TABLE IF NOT EXISTS "payment_info"(
        "payment_number" INTEGER NOT NULL,
        "payment_id" VARCHAR(255) ,
        "labor_id_receiver" INTEGER ,
        "user_id_employer" INTEGER ,
        "account_number_receiver" VARCHAR(255) ,
        "account_number_employer" VARCHAR(255) ,
        "paid_for_byHours" DOUBLE PRECISION ,
        "amount_paid" INTEGER ,
        "email_receiver" VARCHAR(255) ,
        "email_employer" VARCHAR(255) ,
        "date" DATE ,
        "type_of_employer" VARCHAR(255) ,
        "location_id" INTEGER ,
        PRIMARY KEY("payment_number")
);
```

```
CREATE TABLE IF NOT EXISTS "schedule_info"(
        "schedule_id" INTEGER NOT NULL,
        "entry_time" TIME(0) WITHOUT TIME ZONE ,
        "exit_time" TIME(0) WITHOUT TIME ZONE ,
```

```
        "date" DATE ,
        "hours_worked_today" DOUBLE PRECISION ,
        "hours_worked_weekly" DOUBLE PRECISION ,
        PRIMARY KEY("schedule_id")
);
```

```
CREATE TABLE IF NOT EXISTS "bank_info"(
        "account_number" VARCHAR(255) NOT NULL,
        "ISFC Code" VARCHAR(255) NOT NULL,
        "user_id" INTEGER NOT NULL,
        "labor_id" INTEGER NOT NULL,
        "name" VARCHAR(255) ,
        "upi_id" VARCHAR(255) NOT NULL,
        "email" VARCHAR(255) ,
        PRIMARY KEY("account_number")
);
```

```
CREATE TABLE IF NOT EXISTS "hourly_labors"(
        "labor_id" INTEGER NOT NULL,
        "hourly_wage" INTEGER ,
        "hours_worked" INTEGER ,
        PRIMARY KEY("labor_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id")
);
```

```
CREATE TABLE IF NOT EXISTS "weekly_labors"(
        "labor_id" INTEGER NOT NULL,
        "weekly_wage" INTEGER ,
        "hours_worked" INTEGER ,
        PRIMARY KEY("labor_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id")
);
```

```
CREATE TABLE IF NOT EXISTS "leave"(
        "leave_id" INTEGER NOT NULL,
        "leave_date" DATE,
        PRIMARY KEY("leave_id"),
        ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS "personal_info"(
        "user_id" INTEGER  ,
        "labor_id" INTEGER ,
        "supervisor_id" INTEGER  ,
        "business_id" INTEGER ,
        "official_id" INTEGER ,
        "mobile_number" INTEGER ,
        "type_of_user" VARCHAR(255) NOT NULL ,
        "name" VARCHAR(255) ,
        "email" VARCHAR(255) ,
        UNIQUE("user_id","labor_id","supervisor_id","business_id","official_id","mobi
        le_number","email"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
        FOREIGN KEY("official_id") REFERENCES "government_officials"("official_id"),
        ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS "supervisor_manager"(
        "lowerlevel_supervisor_id" INTEGER NOT NULL,
        "manager_id" INTEGER NOT NULL,
        "supervised_since" DATE ,
        PRIMARY KEY("lowerlevel_supervisor_id","manager_id")
);
```

```
CREATE TABLE IF NOT EXISTS "needs_location"(
        "user_id" INTEGER ,
        "labor_id" INTEGER ,
        "supervisor_id" INTEGER ,
        "business_id" INTEGER ,
        "official_id" INTEGER ,
        "location_id" INTEGER ,
        UNIQUE("user_id","labor_id","supervisor_id","business_id","official_id","locat
        ion_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
```

```
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
        FOREIGN KEY("official_id") REFERENCES "government_officials"("official_id")
);
```

```
CREATE TABLE IF NOT EXISTS "schedules"(
        "user_id" INTEGER ,
        "supervisor_id" INTEGER ,
        "labor_id" INTEGER NOT NULL,
        UNIQUE("user_id","supervisor_id","labor_id"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("schedule_id") REFERENCES "schedule_info"("schedule_id")
);
```

```
CREATE TABLE IF NOT EXISTS "hires"(
        "user_id" INTEGER ,
        "business_id" INTEGER ,
        "labor_id" INTEGER NOT NULL ,
        UNIQUE("user_id","business_id","labor_id"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id")
);
```

```
CREATE TABLE IF NOT EXISTS "takes_leave"(
        "user_id" INTEGER ,
        "leave_id" INTEGER ,
        "business_id" INTEGER ,
        "supervisor_id" INTEGER,
        "labor_id" INTEGER NOT NULL,
        UNIQUE("user_id","business_id","labor_id","leave_id","supervisor_id"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
```

```
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
        FOREIGN KEY("leave_id") REFERENCES "leave"("leave_id")
);
```

```
CREATE TABLE IF NOT EXISTS "approves"(
        "user_id" INTEGER,
        "leave_id" INTEGER ,
        "business_id" INTEGER ,
        "supervisor_id" INTEGER,
        "labor_id" INTEGER NOT NULL,
        UNIQUE("user_id","business_id","labor_id","leave_id","supervisor_id"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
        FOREIGN KEY("leave_id") REFERENCES "leave"("leave_id")
);
```

```
CREATE TABLE IF NOT EXISTS "holds"(
        "user_id" INTEGER ,
        "labor_id" INTEGER ,
        "supervisor_id" INTEGER ,
        "business_id" INTEGER ,
        UNIQUE("user_id","labor_id","supervisor_id","business_id"),
        FOREIGN KEY("location_id") REFERENCES "location_info"("location_id"),
        FOREIGN KEY("user_id") REFERENCES "users_info"("user_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
        FOREIGN KEY("official_id") REFERENCES "government_officials"("official_id")
);
```

```
CREATE TABLE IF NOT EXISTS "stores"(
        "labor_id" INTEGER NOT NULL,
        "payment_number" INTEGER NOT NULL,
        PRIMARY KEY("labor_id","payment_number"),
```

```
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("payment_number") REFERENCES
        "payment_info"("payment_number")
);
```

```
CREATE TABLE IF NOT EXISTS "pays"(
        "business_id" INTEGER NOT NULL,
        "supervisor_id" INTEGER NOT NULL,
        "labor_id" INTEGER NOT NULL,
        "payment_number" INTEGER NOT NULL,
        PRIMARY KEY("labor_id","business_id","supervisor_id"),
        FOREIGN KEY("labor_id") REFERENCES "labor"("labor_id"),
        FOREIGN KEY("payment_number") REFERENCES
        "payment_info"("payment_number")
        FOREIGN KEY("supervisor_id") REFERENCES
        "labor_supervisor"("supervisor_id"),
        FOREIGN KEY("business_id") REFERENCES "business_info"("business_id"),
);
```

IT214 - DBMS
(Normalization and Schema Refinement)

Case Study Database: Laborlist and Wages Database

Prepared by
Team No. 17 | Group 6 | Section 10

Group Members
Dhruv Patel - 202001271
Jaykumar Navadiya - 202001465

Under the guidance of
Prof. Chhaya
Teaching Assistant - Vinay Maharaaj

Date
November 11, 2022

15. Normalization & Schema Refinement

15.1 Relations and Schemas (Initial)

1] labor(labor_id, birth_date, work_hours, wage_remaining, rating, Type of work, age, supervisor_id ,supervised_since, location_id, account_number)

2] labor_supervisor(supervisor_id, managed_since, business_id, location_id, account_number)

3] location_info(location_id, city, district, user_id)

4] government_officials(official_id, city, location_id)

5] business_info(business_id, name, email, account_number, location_id, official_id, regulated_since)

6] users_info(user_id, account_number, location_id)

7] payment_info(payment_number, payment_id, labor_id_receiver, user_id_employer, account_number_receiver, account_number_employer, paid_for_byHours, amount_paid, email_receiver, email_employer, date, type_of_employer, location_id)

8] schedule_info(schedule_id, entry_time, exit_time, date, hours_worked_today, hours_worked_weekly)

9] bank_info(account_number, IFSC Code, user_id, labor_id, name, upi_id, email)

10] hourly_labors(labor_id, hourly_wage, hours_worked)

11] weekly_labors(labor_id, weekly_wage, hours_worked)

12] leave(leave_id, leave_date)

13] personal_info(user_id, labor_id, supervisor_id, business_id, official_id, mobile_number, type_of_user, name, email)

14] supervisor_manager(lowerlevel_supervisor_id,manager_id, supervised_since)

15] needs_location(user_id, labor_id, supervisor_id, business_id, official_id, location_id)

16] schedules(user_id, supervisor_id,labor_id)

17] hires(user_id, business_id, labor_id)

18] takes_leave(user_id,leave_id, business_id, supervisor_id, labor_id)

19] approves(user_id,leave_id, business_id, supervisor_id, labor_id)

20] holds(user_id, labor_id, supervisor_id, business_id)

21] stores(labor_id, payment_number)

22] pays(business_id, supervisor_id, labor_id, payment_number)

15.2 Types of Dependencies

labor

PK	labor_id
FK	supervisor_id, location_id
Functional Dependencies	labor_id → birth_date, labor_id → work_hours, labor_id → wage_remaining, labor_id → rating, labor_id → Type of work, labor_id → age, labor_id → supervisor_id , labor_id → supervised_since, labor_id → location_id, labor_id → account_number, birth_date → age

labor_supervisor	
PK	supervisor_id
FK	business_id, location_id
Functional Dependencies	supervisor_id → managed_since, supervisor_id → business_id, supervisor_id → location_id,supervisor_id → account_number

location_info	
PK	location_id
FK	
Functional Dependencies	location_id → city, location_id → district, city → district, location_id → user_id

government_officials	
PK	official_id
FK	location_id
Functional Dependencies	official_id → city, official_id → location_id

business_info	
PK	business_id
FK	official_id, location_id
Functional Dependencies	business_id → name, business_id → email, business_info → account_number, email → account_number, business_id → location_id, business_id → official_id, business_id → regulated_since, email → account_number

users_info	
PK	user_id
FK	location_id
Functional Dependencies	user_id → account_number, user_id → location_id

payment_info	
PK	payment_number
FK	
Functional Dependencies	payment_number → payment_id, payment_number → labor_id_receiver, payment_number → user_id_employer, payment_number → account_number_receiver, payment_number → account_number_employer, payment_number → paid_for_byHours, payment_number → amount_paid, payment_number → email_receiver, payment_number → email_employer, payment_number → date, payment_number → type_of_employer, payment_number → location_id

schedule_info	
PK	schedule_id
FK	
Functional Dependencies	schedule_id → entry_time, schedule_id → exit_time, schedule_id → date, schedule_id → hours_worked_today, schedule_id → hours_worked_weekly

bank_info	
PK	account_number, user_id, upi_id
FK	
Functional Dependencies	(account_number, user_id, upi_id) → (IFSC Code, user_id, labor_id, name, upi_id, email)

hourly_labors	
PK	labor_id
FK	labor_id
Functional Dependencies	labor_id → (hourly_wage, hours_worked)

weekly_labors	
PK	labor_id
FK	labor_id
Functional Dependencies	labor_id → (weekly_wage, hours_worked)

leave

PK	leave_id
FK	
Functional Dependencies	leave_id → leave_date

personal_info	
PK	{ user_id, labor_id, supervisor_id, business_id, official_id, mobile_number, email }
FK	user_id, labor_id, supervisor_id, business_id, official_id
Functional Dependencies	(user_id, labor_id, supervisor_id, business_id, official_id, mobile_number, email) → (mobile_number, type_of_user, name, email)

supervisor_manager	
PK	{lowerlevel_supervisor_id, manager_id }
FK	
Functional Dependencies	(lowerlevel_supervisor_id, manager_id)→ supervised_since

schedules	
PK	{ user_id, supervisor_id, labor_id }
FK	user_id, labor_id, supervisor_id, schedule_id
Functional Dependencies	(user_id, supervisor_id,labor_id) → (user_id, supervisor_id,labor_id)

hires	
PK	{ user_id, business_id, labor_id }
FK	user_id, business_id, labor_id
Functional Dependencies	(user_id, business_id, labor_id) → (user_id, business_id, labor_id), user_id → labor_id, business_id → labor_id

takes_leave	
PK	{ user_id, business_id, labor_id, leave_id, supervisor_id }

FK	user_id, business_id, labor_id, leave_id, supervisor_id
Functional Dependencies	(user_id, business_id, labor_id, leave_id, supervisor_id) → (user_id, business_id, labor_id, leave_id, supervisor_id)

approves	
PK	{ user_id, business_id, labor_id, leave_id, supervisor_id }
FK	user_id, business_id, labor_id, leave_id, supervisor_id
Functional Dependencies	(user_id, business_id, labor_id, leave_id, supervisor_id) → (user_id, business_id, labor_id, leave_id, supervisor_id)

holds	
PK	{ user_id, business_id, labor_id, supervisor_id }
FK	user_id, business_id, labor_id, location_id, supervisor_id, official_id

Functional Dependencies	(user_id, business_id, labor_id, supervisor_id) → (user_id, business_id, labor_id, supervisor_id)

stores	
PK	{ labor_id, payment_number }
FK	labor_id, payment_number
Functional Dependencies	(labor_id, payment_number) → (labor_id, payment_number)

pays	
PK	{ labor_id, business_id, supervisor_id }
FK	labor_id, payment_number, supervisor_id, business_id
Functional Dependencies	(labor_id, business_id, supervisor_id) → payment_number

15.3 Schema Investigation

15.3.1 Redundancy and General Analysis

1] labor(labor_id, birth_date, work_hours, wage_remaining, rating, Type of work, age, supervisor_id ,supervised_since, location_id, account_number)

- There is one transitive dependency i.e., birth_date → age (because labor_id → birth_date)
- This can be removed by making a new table with columns age(birth_date, age).
- Redundancy: for example, a type of work can be plumbing, now there can be many laborers who are plumbers.

2] labor_supervisor(supervisor_id, managed_since, business_id, location_id, account_number)

- There are no transitive dependencies.
- Redundancy: In 1 business there can be many supervisors hence making the business_id redundant.

3] location_info(location_id, city, district, user_id)

- There is one transitive dependency i.e., city → district (because location_id → city)
- This can be removed by making a new tables with columns (city, district) and location_info(location_id, city, user_id).

4] government_officials(official_id, city, location_id)

- There are no transitive dependencies.
- Redundancy: In 1 city there can be many officials hence making the city redundant.

5] business_info(business_id, name, email, account_number, location_id, official_id, regulated_since)

- There are transitive dependencies, such as email can give the account_number and vice versa. Hence this table is in 2NF and needs to be converted into 3NF.
- Hence we can make tables like business_account(email, account_number) and business_info(business_id, name, email, location_id, official_id, regulated_since).

6] users_info(user_id, account_number, location_id)

- There are no transitive dependencies.

7] payment_info(payment_number, payment_id, labor_id_receiver, user_id_employer, account_number_receiver, account_number_employer, paid_for_byHours, amount_paid, email_receiver, email_employer, date, type_of_employer, location_id)

- There are no transitive dependencies.

8] schedule_info(schedule_id, entry_time, exit_time, date, hours_worked_today, hours_worked_weekly)

- There are no transitive dependencies.

9] bank_info(account_number, IFSC Code, user_id, labor_id, name, upi_id, email)

- There can be many upi_id linked to 1 account. There are no non-prime to non-prime dependencies. But there is one dependency (account_number → upi_id) that can be removed.
- Making this into BCNF form we can decompose into 2 tables bank_info(account_number, IFSC Code, user_id, labor_id, name, email) and upi_info(account_number, upi_id)

10] hourly_labors(labor_id, hourly_wage, hours_worked)

- There are no transitive dependencies.

11] weekly_labors(labor_id, weekly_wage, hours_worked)

- There are no transitive dependencies.

12] leave(leave_id, leave_date)

- There are no transitive dependencies.

13] personal_info(user_id, labor_id, supervisor_id, business_id, official_id, mobile_number, type_of_user, name, email)

- There are no transitive dependencies.
- Redundancy: For any type of users, there will be many users of that type.

14] supervisor_manager(lowerlevel_supervisor_id,manager_id, supervised_since)

- There are no transitive dependencies.

15] needs_location(user_id, labor_id, supervisor_id, business_id, official_id, location_id)

- There are no transitive dependencies.

16] schedules(user_id, supervisor_id,labor_id)

- There are no transitive dependencies.

17] hires(user_id, business_id, labor_id)

- There are no transitive dependencies.

18] takes_leave(user_id,leave_id, business_id, supervisor_id, labor_id)

- There are no transitive dependencies.

19] approves(user_id,leave_id, business_id, supervisor_id, labor_id)

- There are no transitive dependencies nor partial dependencies.

20] holds(user_id, labor_id, supervisor_id, business_id)

- There are no transitive dependencies.

21] stores(labor_id, payment_number)

- There are no transitive dependencies.

22] pays(business_id, supervisor_id, labor_id, payment_number)

- There are no transitive dependencies.

15.3.2 List of update, delete and insert anomalies

1. Hires table (user_id → labor_id, business_id → labor_id)
Here combined (user_id, business_id → labor_id) can give us labor_id. Now, when there is an individual user hiring a laborer, business_id will be null. So, It creates an insertion anomaly as we are inserting NULL values in the business_id column and vice versa.

2. The personal_info table has personal information of all types of users. But while inserting information about an individual user, other types of users such as labor or supervisor will remain NULL, hence creating an insertion anamoly.

15.4 1NF Normalization

Each table is already in 1NF form. The table personal_info was already separated and mobile_number was a part of the primary key. Hence while making the final portal/website, users will have to login using their ID and mobile number. As mobile number is a part of the primary key, for 1 user multiple mobile numbers can be stored and 1NF is achieved.

15.5 2NF Normalization

All of the tables are in 2NF form as there are no Partial Functional Dependency in any table even though there might be

15.6 3NF/BCNF Normalization

The table/relation which has only 1 Candidate Key and if it is already in 3NF, it will be in BCNF (since they are equivalent).

For 3NF Normalization, we need to remove all the transitive dependencies from the 2NF form. For instance, in the relation business_info:

1] business_info(business_id, name, email, account_number, location_id, official_id, regulated_since).

There are transitive dependencies, such as email which can give the account_number and vice versa from the business_id. Hence this table is in 2NF and needs to be converted into 3NF.

To achieve this, we decompose business_info to get a new table which is business_account(email,account_number) and business_info(business_id, name, email, location_id, official_id, regulated_since).

2] bank_info(account_number, IFSC Code, user_id, labor_id, name, upi_id, email)

- There can be many upi_id linked to 1 account. There are no non-prime to non-prime dependencies. But there is one dependency (account_number → upi_id) that can be removed.
- Making this into BCNF form we can decompose into 2 tables bank_info(account_number, IFSC Code, user_id, labor_id, name, email) and upi_info(account_number, upi_id)

15.7 Final Relations

1] labor(labor_id, birth_date, work_hours, wage_remaining, rating, type_of_work, supervisor_id ,supervised_since, location_id, account_number, password)

2] labor_supervisor(supervisor_id, managed_since, business_id, location_id, account_number, password)

3] location_info(location_id, city,district)

4] government_officials(official_id, location_id)

5] business_info(business_id, location_id, official_id, name, email, regulated_since, password)

6] users_info(user_id, account_number, location_id, password)

7] payment_info(payment_number, payment_id, paid_for_byHours, amount_paid, date)

8] schedule_info(schedule_id, entry_time, exit_time, date)

9] bank_info(account_number, isfc_code, name, email, type_of_user, user_id)

10] hourly_labors(labor_id, hourly_wage, hours_worked)

11] weekly_labors(labor_id, weekly_wage, hours_worked)

12] leave(leave_id, leave_date)

13] personal_info(*user_id, labor_id, supervisor_id, official_id,* mobile_number, type_of_user, name, email) //Here unique key is in italics

14] supervisor_manager(lowerlevel_supervisor_id,manager_id, supervised_since)

15] schedules(user_id, supervisor_id,labor_id, schedule id)

16] hires(*user_id, business_id, labor_id*)

17] takes_leave(user_id, leave_id, supervisor_id, labor_id)

18] approves(*user_id, leave_id, supervisor_id, labor_id*) //Here unique key is in italics

19] pays(user_id, supervisor_id, labor_id, payment_number)

20] business_account(email, account_number)

21] upi_info(account_number, upi_id)

→ District, holds, stores, needs_location tables were eventually dropped as they were not required. We allow only 1 location per user and they can update it afterwards. In one to one relationship we do not require a new table. These were removed since they were not required for any specific purpose and created redundancy in the database. All the relations associated with their attributes already had access to them.

→ Also, password attributes were added in relations labor, labor_supervisor, business_info and users_info. These are eventually required for authentication purposes for the web pages created for the system to recognize the users by the passwords.

→ user_id was removed from the primary key of the relation bank_info since account_number was sufficient for the primary key.

16. Re-write DDL Scripts

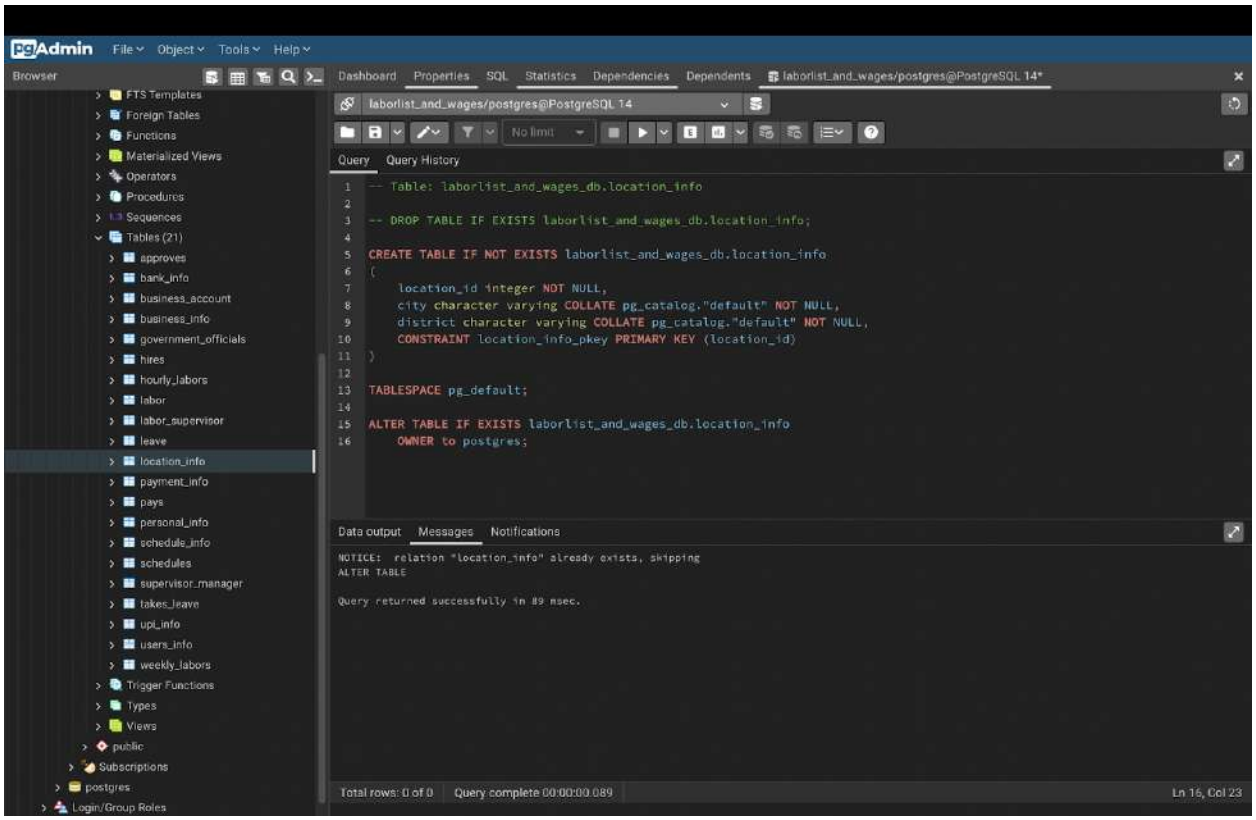
16.1 & 16.2 DDL Scripts for New Design (3NF/BCNF Form) and DDL Snapshots

1. Location_info Table

DDL Script:

```
-- Table: laborlist_and_wages_db.location_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.location_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.location_info
(
    location_id integer NOT NULL,
    city character varying COLLATE pg_catalog."default" NOT NULL,
    district character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT location_info_pkey PRIMARY KEY (location_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.location_info
    OWNER to postgres;
```

DDL Snapshot:

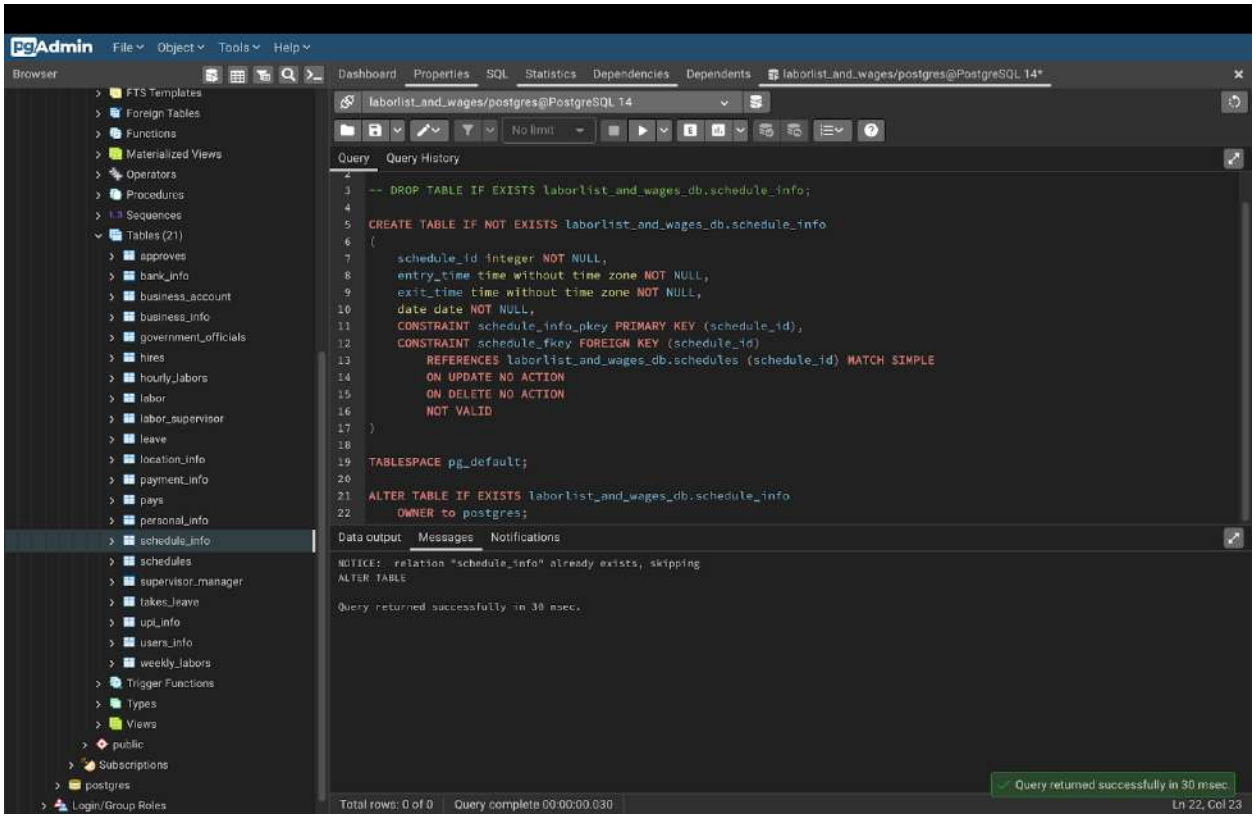


2. Schedule_info Table

DDL Script:

```
-- Table: laborlist_and_wages_db.schedule_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.schedule_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.schedule_info
(
    schedule_id integer NOT NULL,
    entry_time time without time zone NOT NULL,
    exit_time time without time zone NOT NULL,
    date date NOT NULL,
    CONSTRAINT schedule_info_pkey PRIMARY KEY (schedule_id),
    CONSTRAINT schedule_fkey FOREIGN KEY (schedule_id)
        REFERENCES laborlist_and_wages_db.schedules (schedule_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.schedule_info
    OWNER to postgres;
```

DDL Snapshot:



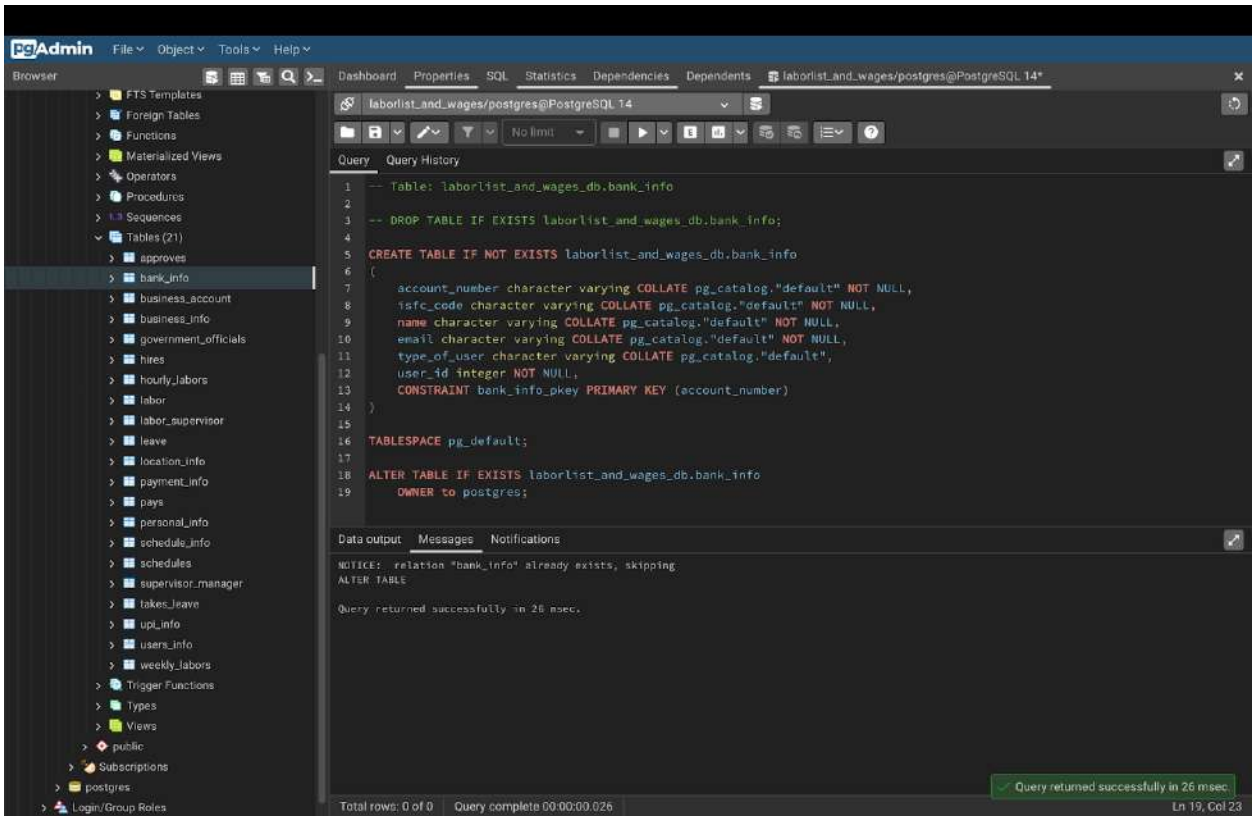
3. Bank_info table

DDL Script:

```
-- Table: laborlist_and_wages_db.bank_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.bank_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.bank_info
(
    account_number character varying COLLATE pg_catalog."default" NOT NULL,
    isfc_code character varying COLLATE pg_catalog."default" NOT NULL,
    name character varying COLLATE pg_catalog."default" NOT NULL,
    email character varying COLLATE pg_catalog."default" NOT NULL,
    type_of_user character varying COLLATE pg_catalog."default",
    user_id integer NOT NULL,
    CONSTRAINT bank_info_pkey PRIMARY KEY (account_number)
)

TABLESPACE pg_default;
```

```
ALTER TABLE IF EXISTS laborlist_and_wages_db.bank_info
    OWNER to postgres;
```

DDL Snapshot:



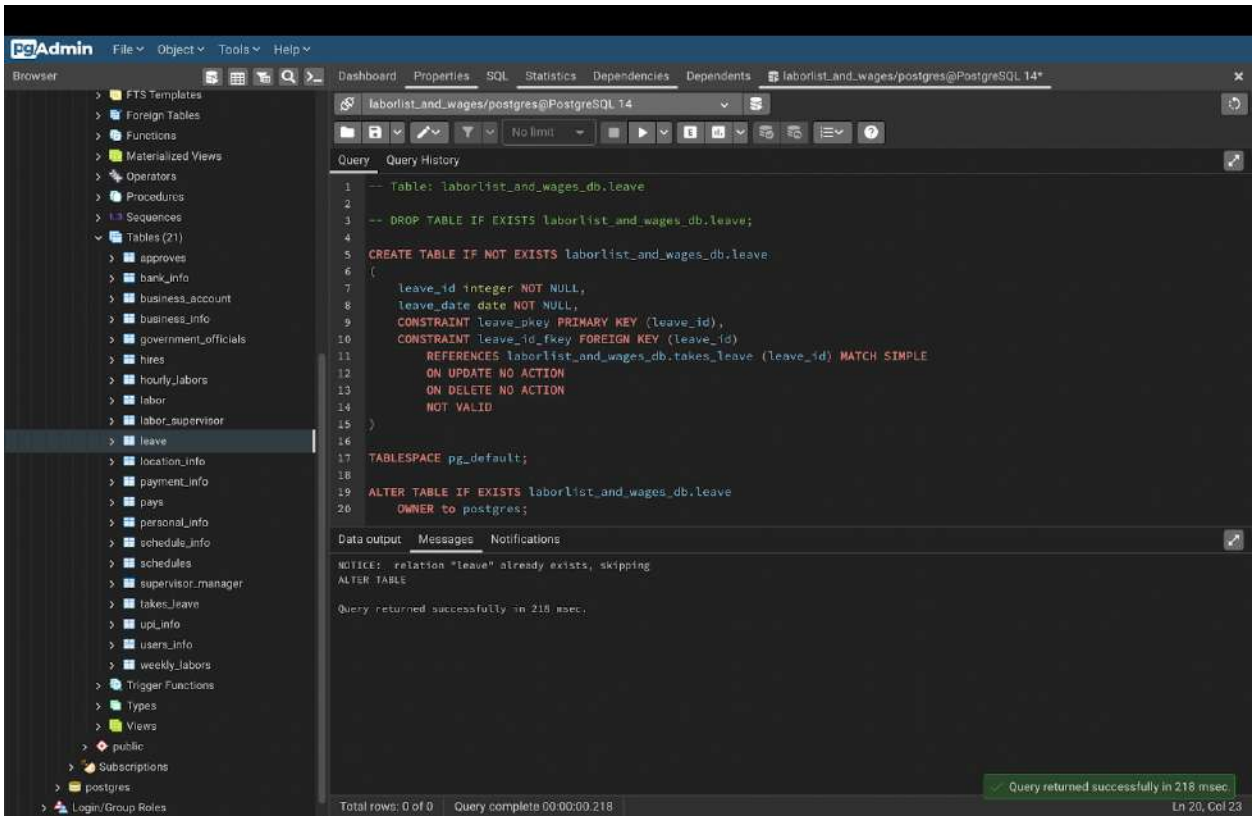
4. Leave table

DDL Script:

```
-- Table: laborlist_and_wages_db.leave

-- DROP TABLE IF EXISTS laborlist_and_wages_db.leave;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.leave
(
    leave_id integer NOT NULL,
    leave_date date NOT NULL,
    CONSTRAINT leave_pkey PRIMARY KEY (leave_id),
```

```
    CONSTRAINT leave_id_fkey FOREIGN KEY (leave_id)
        REFERENCES laborlist_and_wages_db.takes_leave (leave_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.leave
    OWNER to postgres;
```

DDL Snapshot:



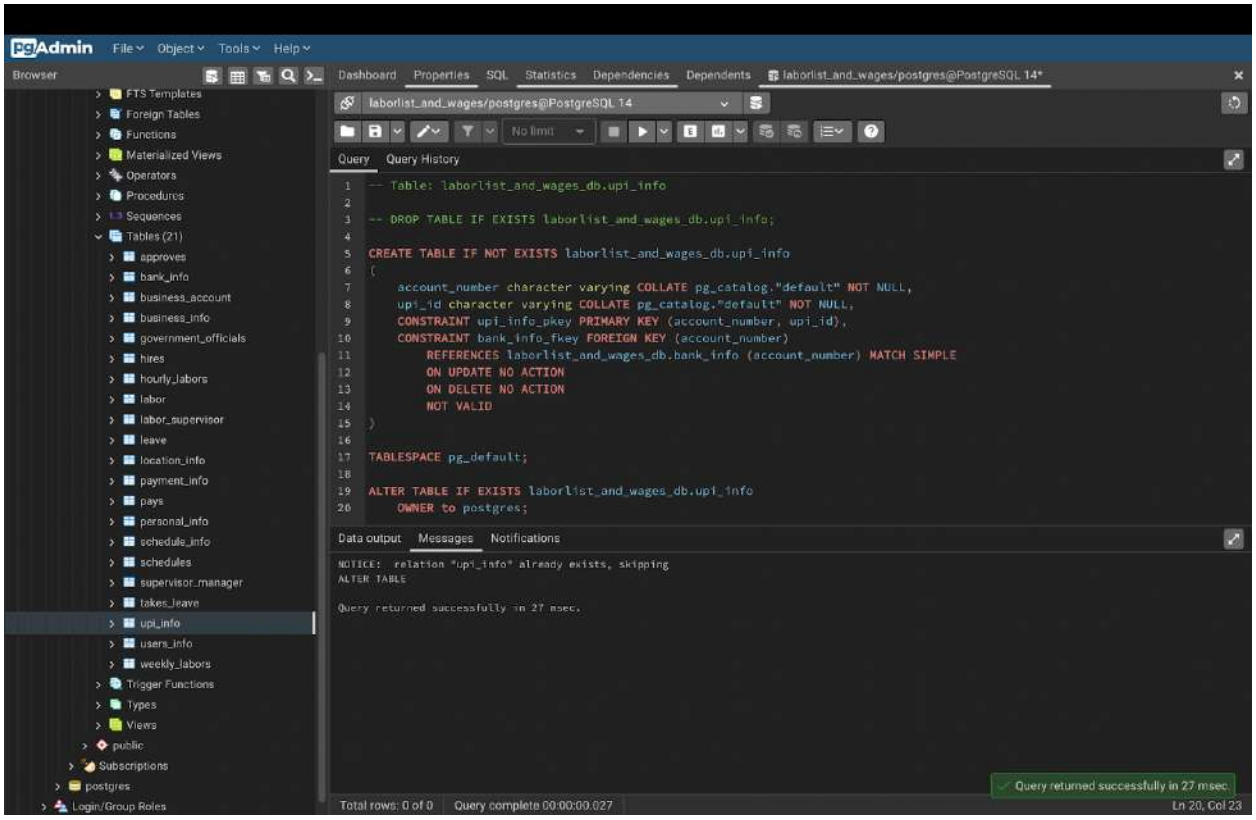
5. Upi_info table

DDL Script:

```
-- Table: laborlist_and_wages_db.upi_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.upi_info;
```

```
CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.upi_info
(
    account_number character varying COLLATE pg_catalog."default" NOT NULL,
    upi_id character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT upi_info_pkey PRIMARY KEY (account_number, upi_id),
    CONSTRAINT bank_info_fkey FOREIGN KEY (account_number)
        REFERENCES laborlist_and_wages_db.bank_info (account_number) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.upi_info
    OWNER to postgres;
```

DDL Snapshot:

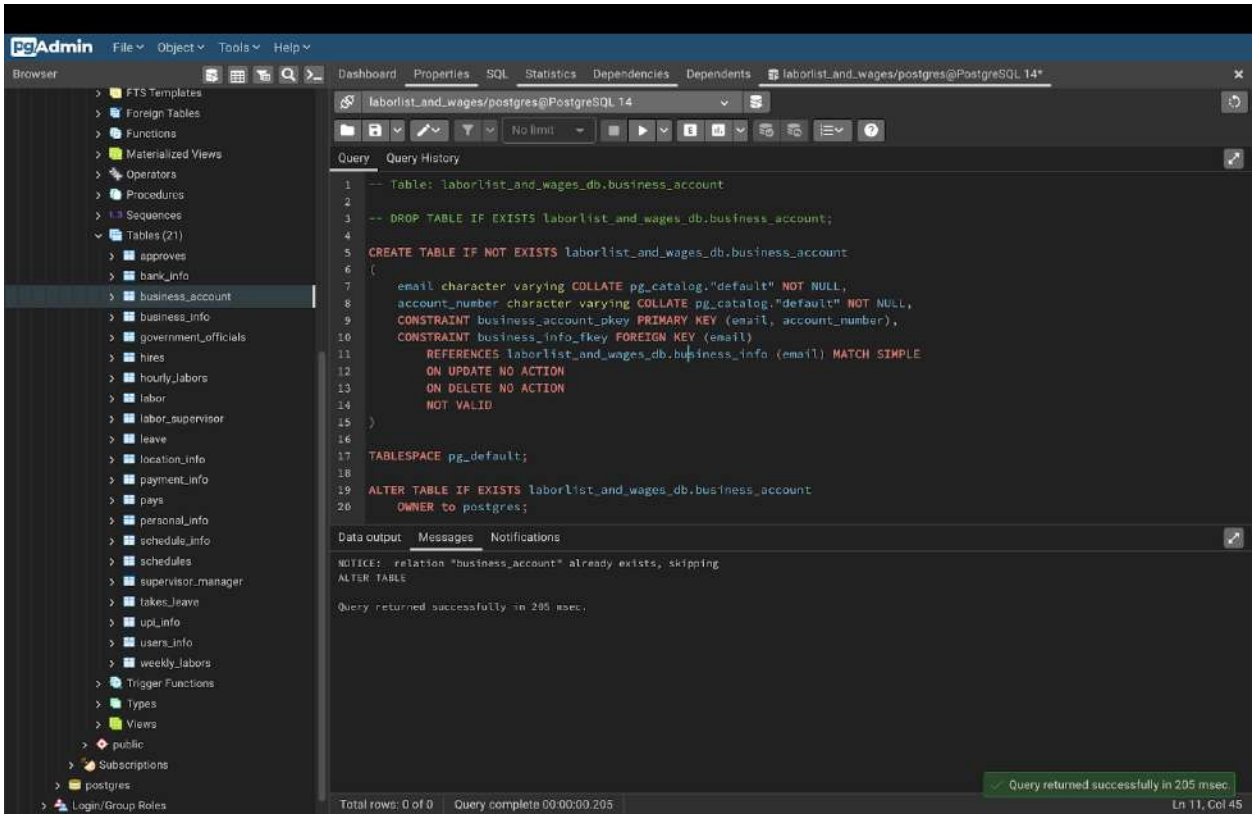


6. Business_account table

DDL Script:

```
-- Table: laborlist_and_wages_db.business_account

-- DROP TABLE IF EXISTS laborlist_and_wages_db.business_account;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.business_account
(
    email character varying COLLATE pg_catalog."default" NOT NULL,
    account_number character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT business_account_pkey PRIMARY KEY (email, account_number),
    CONSTRAINT business_info_fkey FOREIGN KEY (email)
        REFERENCES laborlist_and_wages_db.business_info (email) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.business_account
    OWNER to postgres;
```

DDL Snapshot:



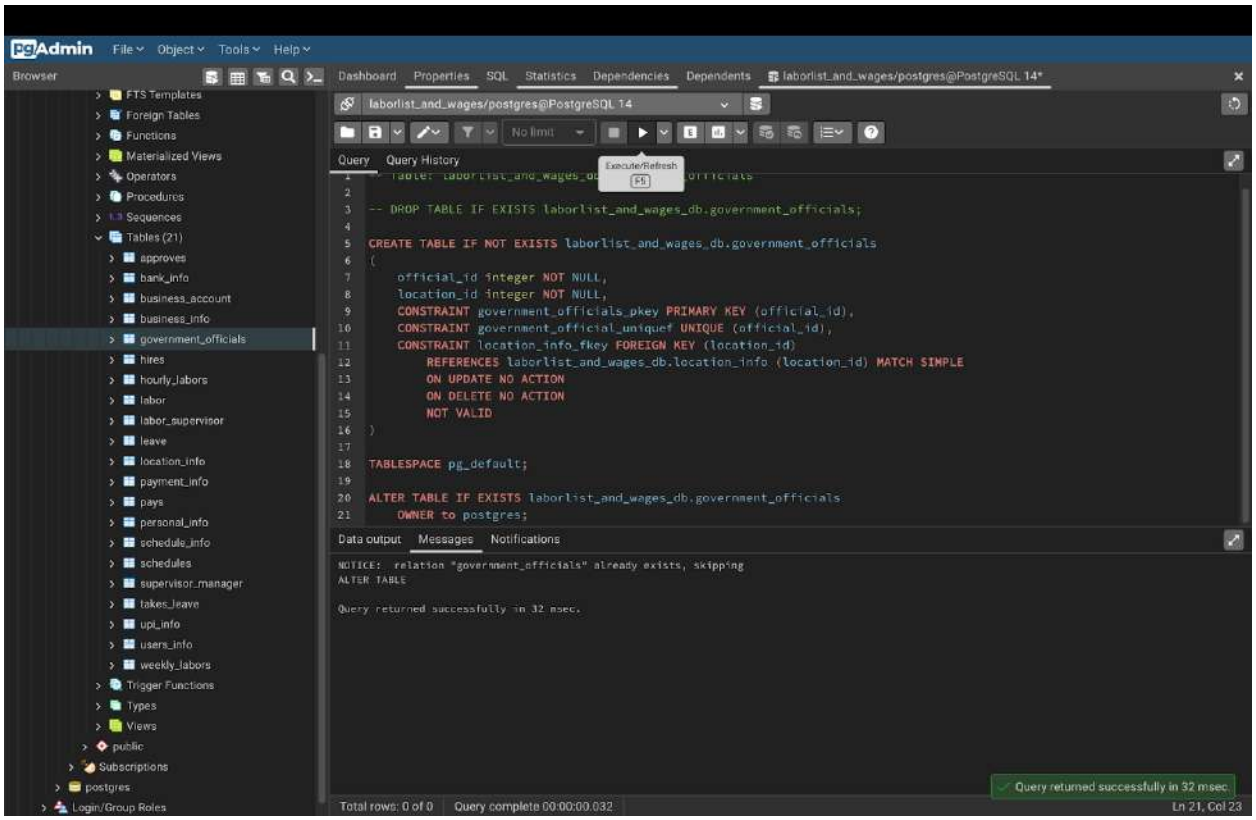
7. Government_officials table

DDL Script:

```
-- Table: laborlist_and_wages_db.government_officials

-- DROP TABLE IF EXISTS laborlist_and_wages_db.government_officials;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.government_officials
(
    official_id integer NOT NULL,
    location_id integer NOT NULL,
    CONSTRAINT government_officials_pkey PRIMARY KEY (official_id),
    CONSTRAINT government_official_uniquef UNIQUE (official_id),
    CONSTRAINT location_info_fkey FOREIGN KEY (location_id)
        REFERENCES laborlist_and_wages_db.location_info (location_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
```

```
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.government_officials
    OWNER to postgres;
```

DDL Snapshot:



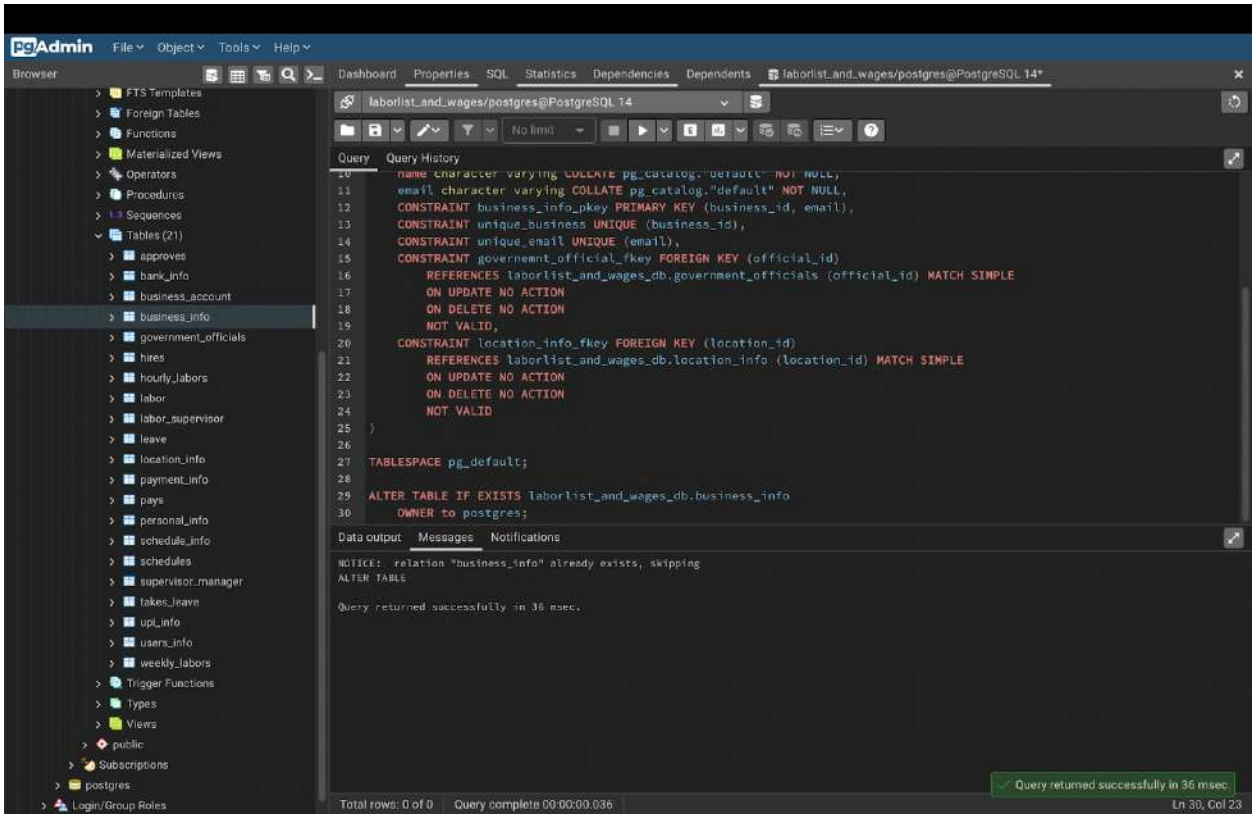
8. Business_info table

DDL Script:

```
-- Table: laborlist_and_wages_db.business_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.business_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.business_info
(
```

```
    business_id integer NOT NULL,
    location_id integer NOT NULL,
    official_id integer NOT NULL,
    name character varying COLLATE pg_catalog."default" NOT NULL,
    email character varying COLLATE pg_catalog."default" NOT NULL,
    regulated_since date,
    password character varying DEFAULT 'busy'::character varying NOT NULL
    CONSTRAINT business_info_pkey PRIMARY KEY (business_id, email),
    CONSTRAINT unique_business UNIQUE (business_id),
    CONSTRAINT unique_email UNIQUE (email),
    CONSTRAINT governemnt_official_fkey FOREIGN KEY (official_id)
        REFERENCES laborlist_and_wages_db.government_officials (official_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT location_info_fkey FOREIGN KEY (location_id)
        REFERENCES laborlist_and_wages_db.location_info (location_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.business_info
    OWNER to postgres;
```

DDL Snapshot:



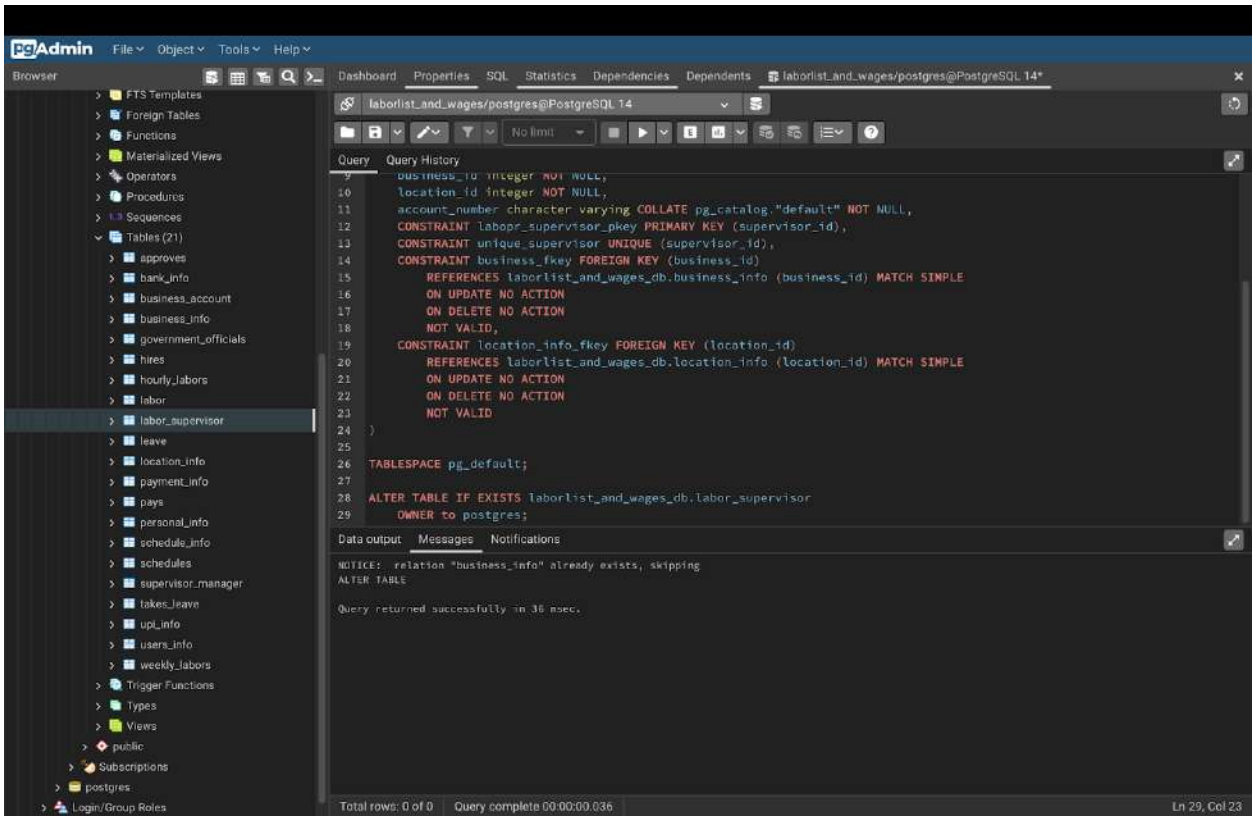
9. Labor_supervisor table

DDL Script:

```
-- Table: laborlist_and_wages_db.labor_supervisor

-- DROP TABLE IF EXISTS laborlist_and_wages_db.labor_supervisor;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.labor_supervisor
(
    supervisor_id integer NOT NULL,
    managed_since date NOT NULL,
    business_id integer NOT NULL,
    location_id integer NOT NULL,
    account_number character varying COLLATE pg_catalog."default" NOT NULL,
    password character varying DEFAULT 'super'::character varying NOT NULL
    CONSTRAINT labopr_supervisor_pkey PRIMARY KEY (supervisor_id),
    CONSTRAINT unique_supervisor UNIQUE (supervisor_id),
    CONSTRAINT business_fkey FOREIGN KEY (business_id)
```

```
        REFERENCES laborlist_and_wages_db.business_info (business_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT location_info_fkey FOREIGN KEY (location_id)
        REFERENCES laborlist_and_wages_db.location_info (location_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.labor_supervisor
    OWNER to postgres;
```

DDL Snapshot:

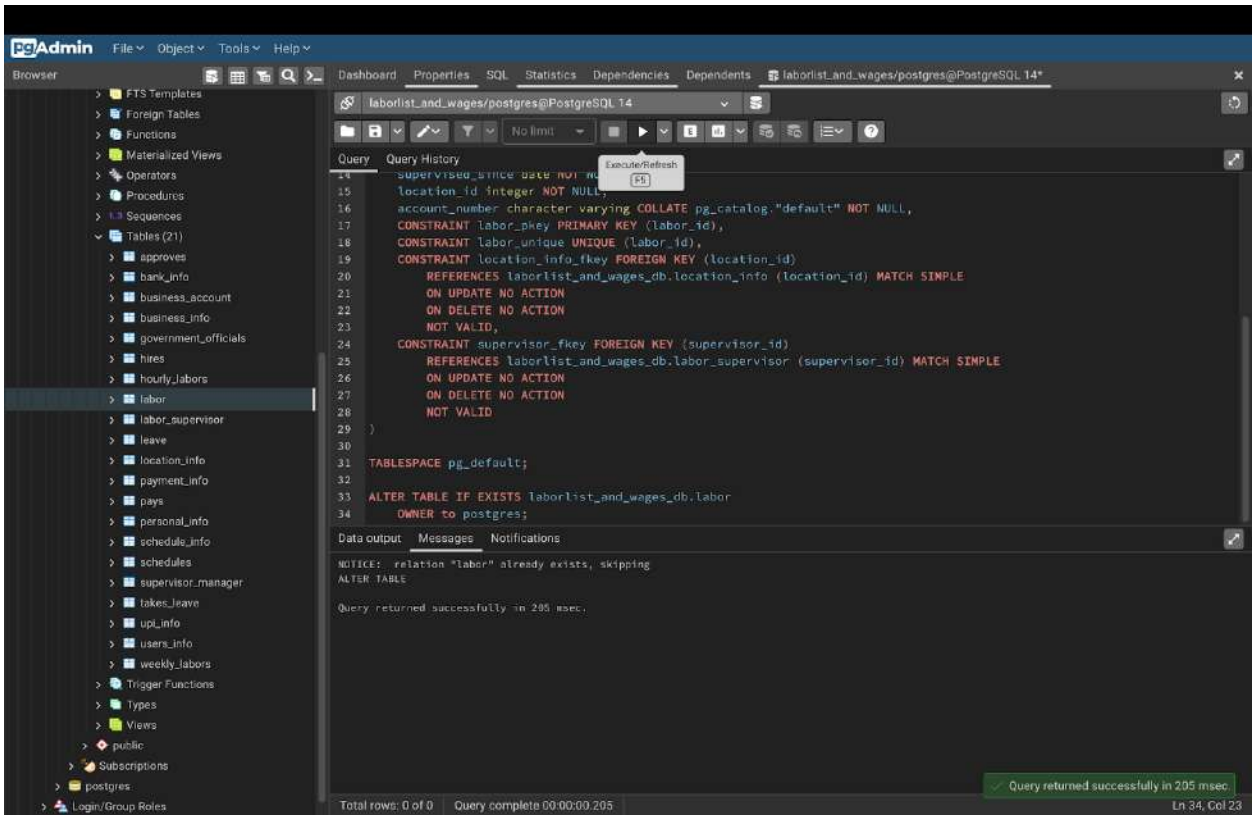


10. Labor table

DDL Script:

```
-- Table: laborlist_and_wages_db.labor

-- DROP TABLE IF EXISTS laborlist_and_wages_db.labor;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.labor
(
    labor_id integer NOT NULL,
    birth_date date NOT NULL,
    work_hours integer NOT NULL,
    wage_remaining integer NOT NULL,
    rating integer NOT NULL,
    type_of_work character varying COLLATE pg_catalog."default" NOT NULL,
    supervisor_id integer NOT NULL,
    supervised_since date NOT NULL,
    location_id integer NOT NULL,
    account_number character varying COLLATE pg_catalog."default" NOT NULL,
    password character varying DEFAULT 'labor'::character varying NOT NULL
    CONSTRAINT labor_pkey PRIMARY KEY (labor_id),
    CONSTRAINT labor_unique UNIQUE (labor_id),
    CONSTRAINT location_info_fkey FOREIGN KEY (location_id)
        REFERENCES laborlist_and_wages_db.location_info (location_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT supervisor_fkey FOREIGN KEY (supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.labor
    OWNER to postgres;
```

DDL Snapshot:



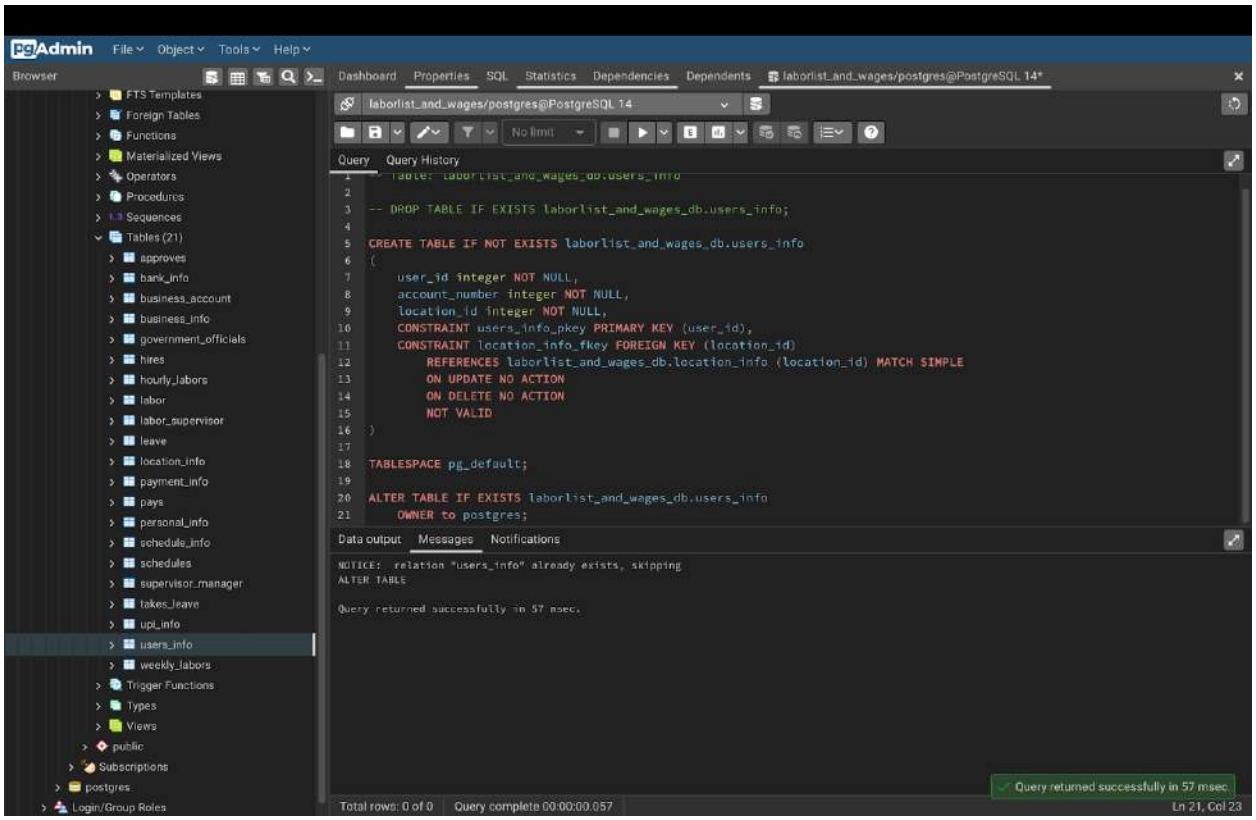
11. Users_info table

DDL Script:

```
-- Table: laborlist_and_wages_db.users_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.users_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.users_info
(
    user_id integer NOT NULL,
    account_number integer NOT NULL,
    location_id integer NOT NULL,
    password character varying DEFAULT 'majoor'::character varying NOT NULL
    CONSTRAINT users_info_pkey PRIMARY KEY (user_id),
    CONSTRAINT location_info_fkey FOREIGN KEY (location_id)
        REFERENCES laborlist_and_wages_db.location_info (location_id) MATCH
SIMPLE
```

```
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.users_info
    OWNER to postgres;
```

DDL Snapshot:



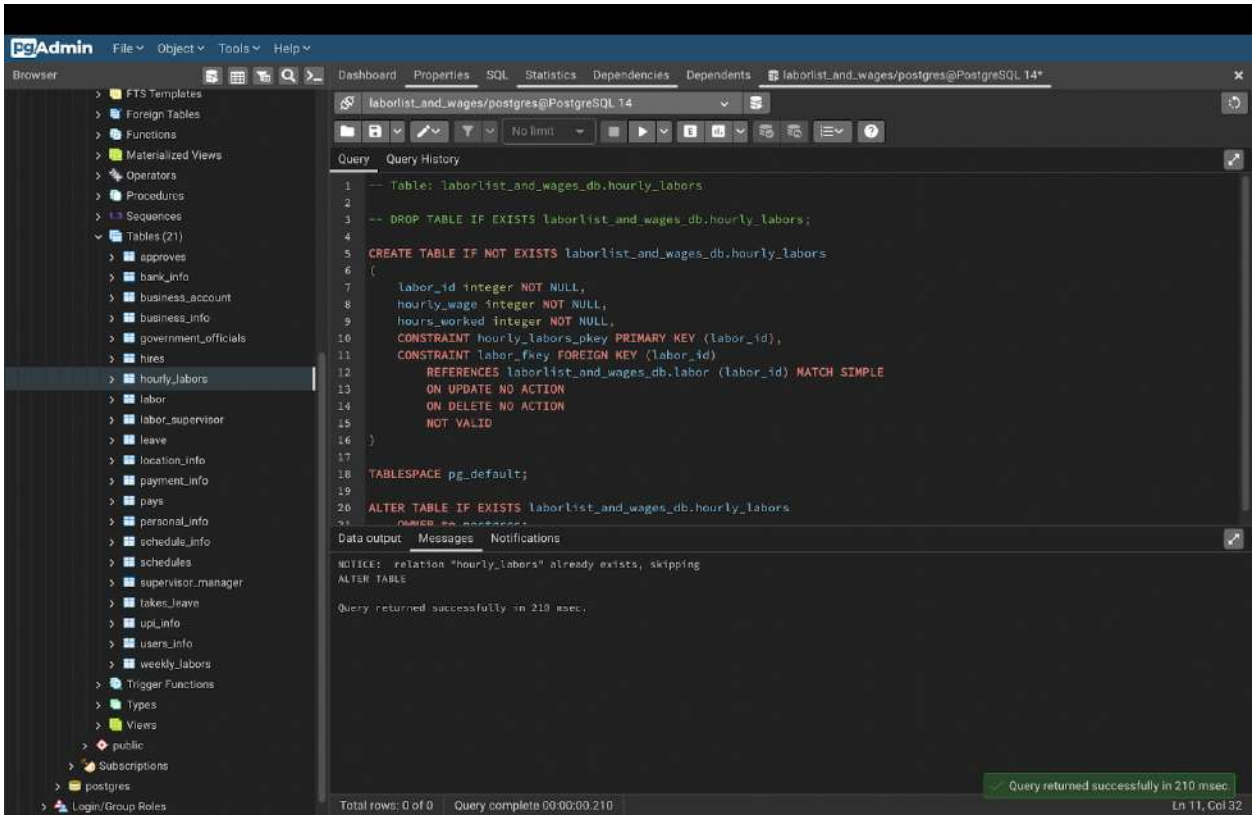
12. Hourly_labors table

DDL Script:

```
-- Table: laborlist_and_wages_db.hourly_labors

-- DROP TABLE IF EXISTS laborlist_and_wages_db.hourly_labors;
```

```
CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.hourly_labors
(
    labor_id integer NOT NULL,
    hourly_wage integer NOT NULL,
    hours_worked integer NOT NULL,
    CONSTRAINT hourly_labors_pkey PRIMARY KEY (labor_id),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.hourly_labors
    OWNER to postgres;
```

DDL Snapshot:

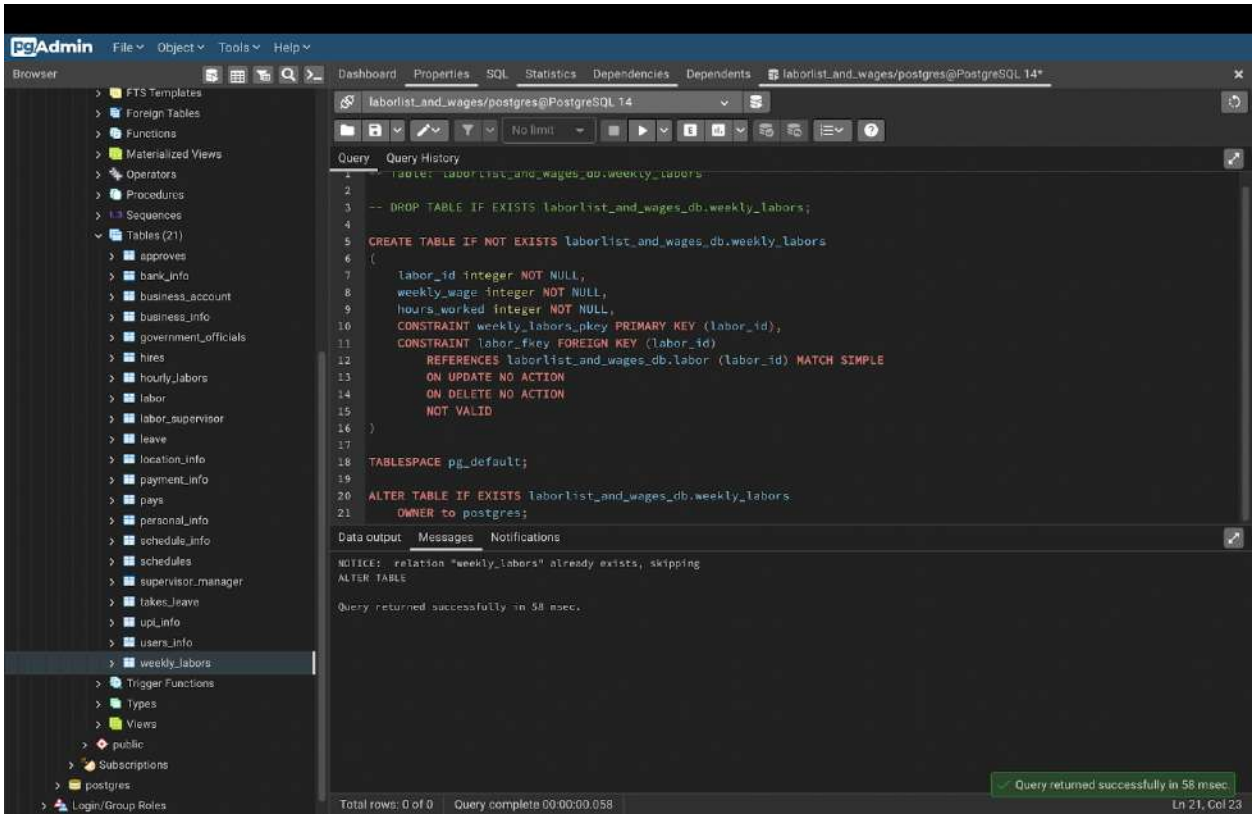


13. Weekly_labors table

DDL Script:

```
-- Table: laborlist_and_wages_db.weekly_labors

-- DROP TABLE IF EXISTS laborlist_and_wages_db.weekly_labors;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.weekly_labors
(
    labor_id integer NOT NULL,
    weekly_wage integer NOT NULL,
    hours_worked integer NOT NULL,
    CONSTRAINT weekly_labors_pkey PRIMARY KEY (labor_id),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.weekly_labors
    OWNER to postgres;
```

DDL Snapshot:



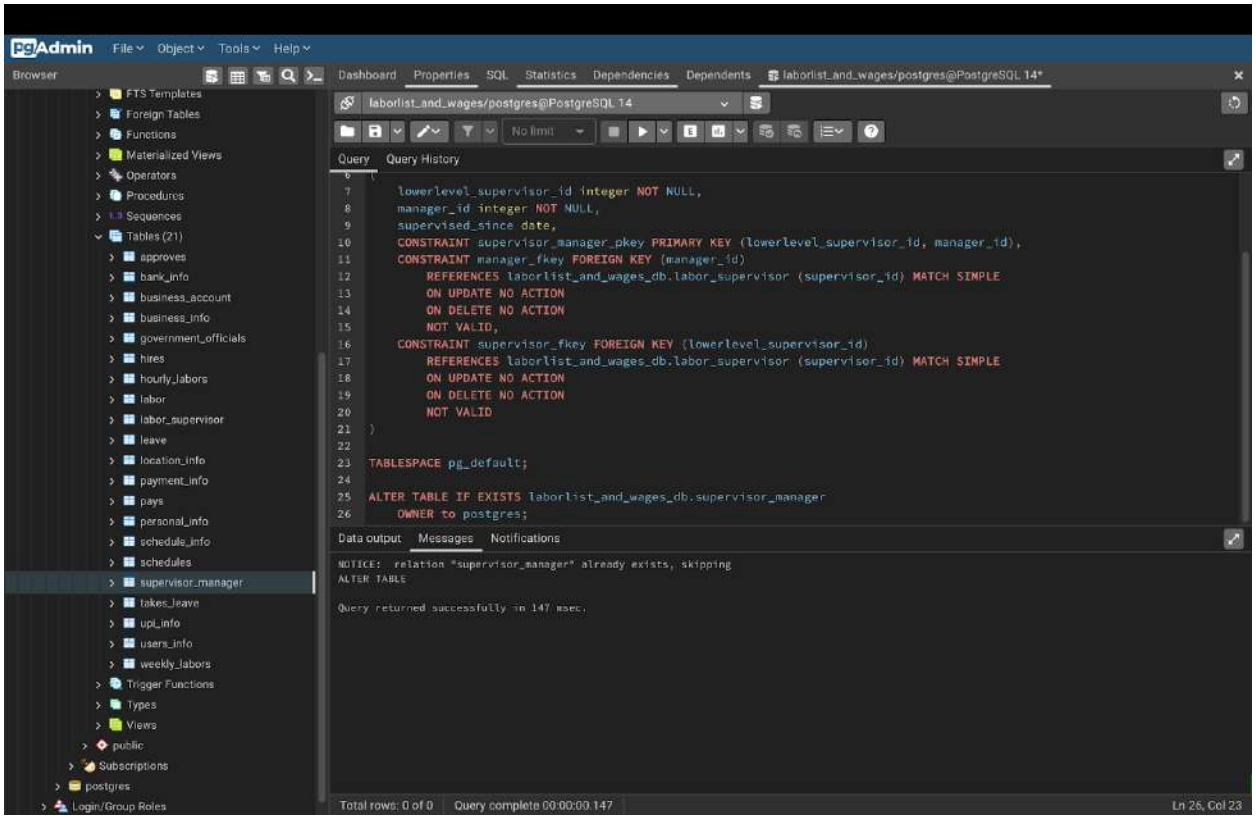
14. Supervisor_manager table

DDL Script:

```
-- Table: laborlist_and_wages_db.supervisor_manager

-- DROP TABLE IF EXISTS laborlist_and_wages_db.supervisor_manager;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.supervisor_manager
(
    lowerlevel_supervisor_id integer NOT NULL,
    manager_id integer NOT NULL,
    supervised_since date,
    CONSTRAINT supervisor_manager_pkey PRIMARY KEY (lowerlevel_supervisor_id,
manager_id),
    CONSTRAINT manager_fkey FOREIGN KEY (manager_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
```

```
        NOT VALID,
    CONSTRAINT supervisor_fkey FOREIGN KEY (lowerlevel_supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.supervisor_manager
    OWNER to postgres;
```

DDL Snapshot:



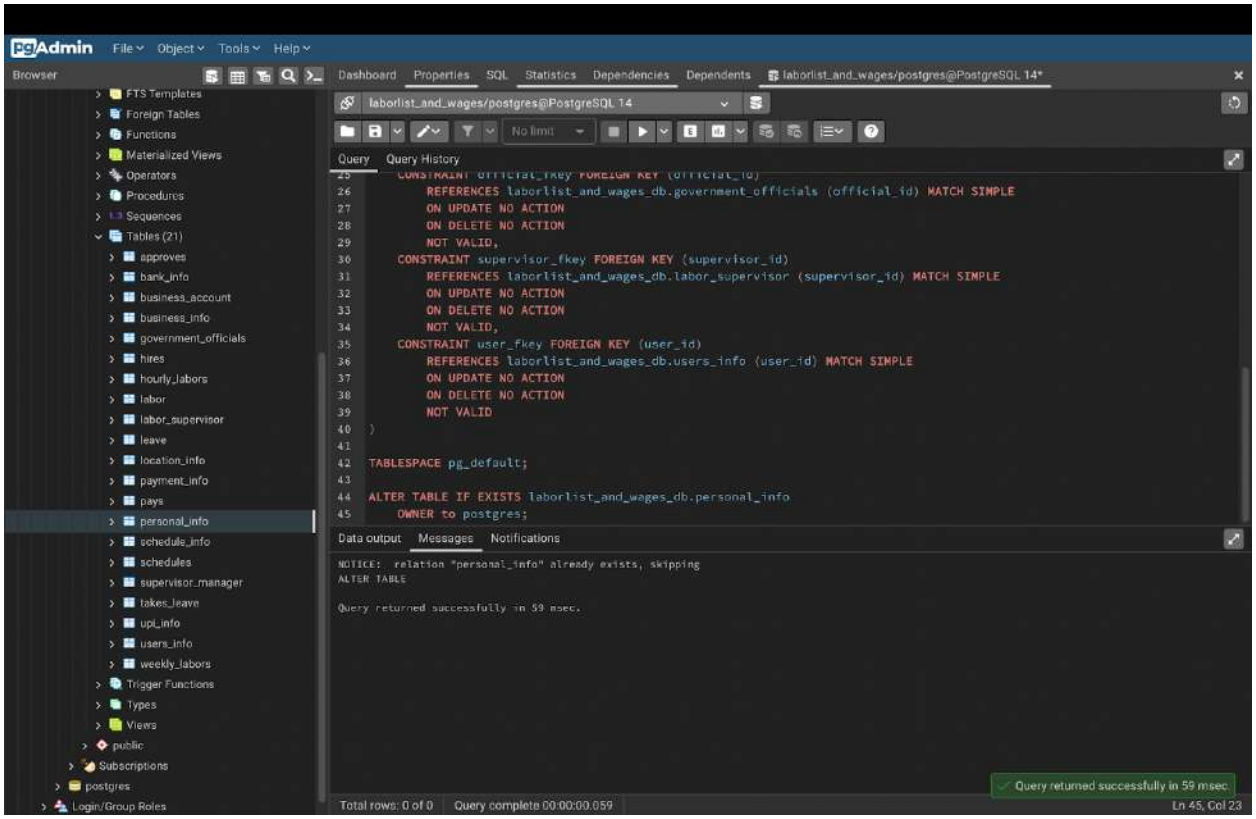
15. Personal_info table

DDL Script:

```
-- Table: laborlist_and_wages_db.personal_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.personal_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.personal_info
(
    user_id integer,
    labor_id integer,
    supervisor_id integer,
    official_id integer,
    mobile_number integer,
    type_of_user character varying COLLATE pg_catalog."default",
    name character varying COLLATE pg_catalog."default",
    email character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT personal_info_pkey PRIMARY KEY (email),
    CONSTRAINT unique_labor UNIQUE (labor_id),
    CONSTRAINT unique_official UNIQUE (official_id),
    CONSTRAINT unique_supervisor_pinfo UNIQUE (supervisor_id),
    CONSTRAINT unique_user UNIQUE (user_id),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT official_fkey FOREIGN KEY (official_id)
        REFERENCES laborlist_and_wages_db.government_officials (official_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT supervisor_fkey FOREIGN KEY (supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT user_fkey FOREIGN KEY (user_id)
        REFERENCES laborlist_and_wages_db.users_info (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)
```

```
TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.personal_info
    OWNER to postgres;
```

DDL Snapshots:



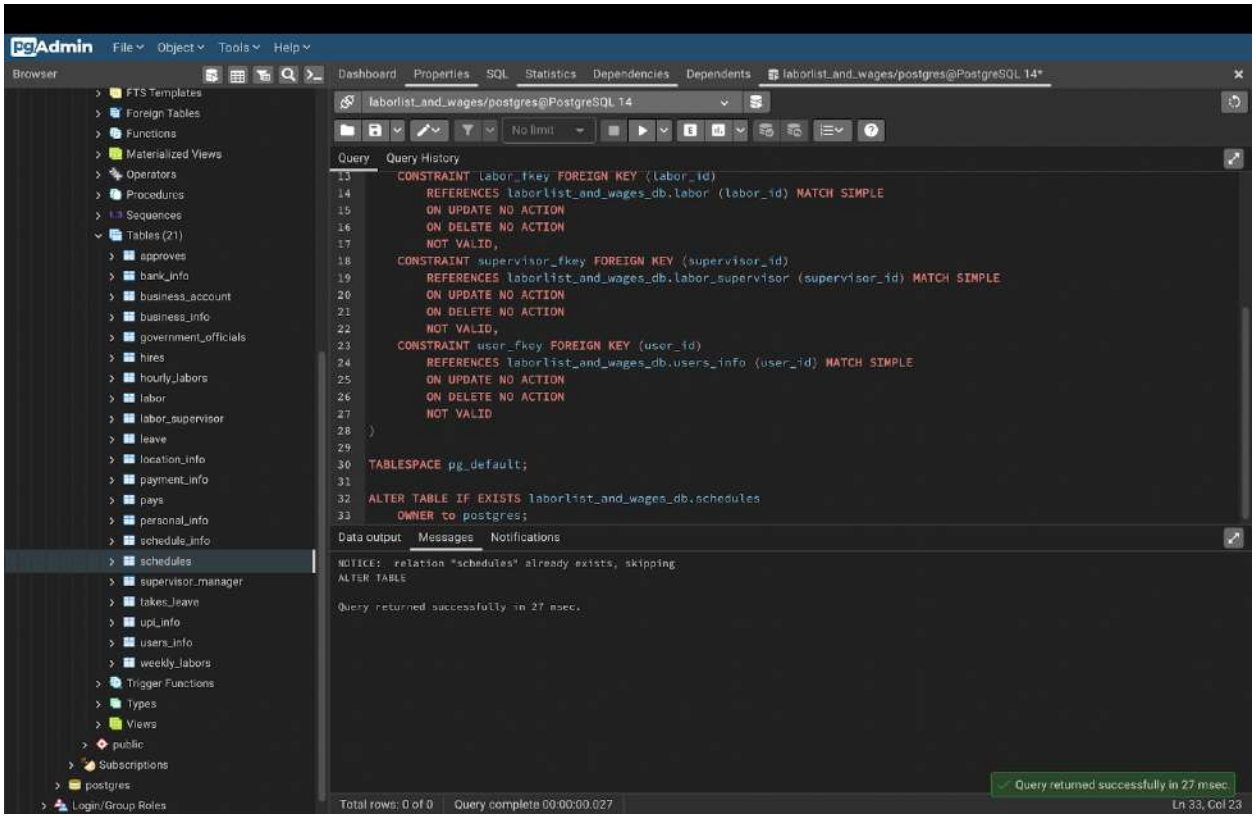
16. Schedules table

DDL Script:

```
-- Table: laborlist_and_wages_db.schedules

-- DROP TABLE IF EXISTS laborlist_and_wages_db.schedules;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.schedules
(
    user_id integer,
```

```
    supervisor_id integer,
    labor_id integer NOT NULL,
    schedule_id integer NOT NULL,
    CONSTRAINT schedules_pkey PRIMARY KEY (schedule_id),
    CONSTRAINT schedule_unique UNIQUE (schedule_id),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT supervisor_fkey FOREIGN KEY (supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT user_fkey FOREIGN KEY (user_id)
        REFERENCES laborlist_and_wages_db.users_info (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.schedules
    OWNER to postgres;
```

DDL Snapshots:



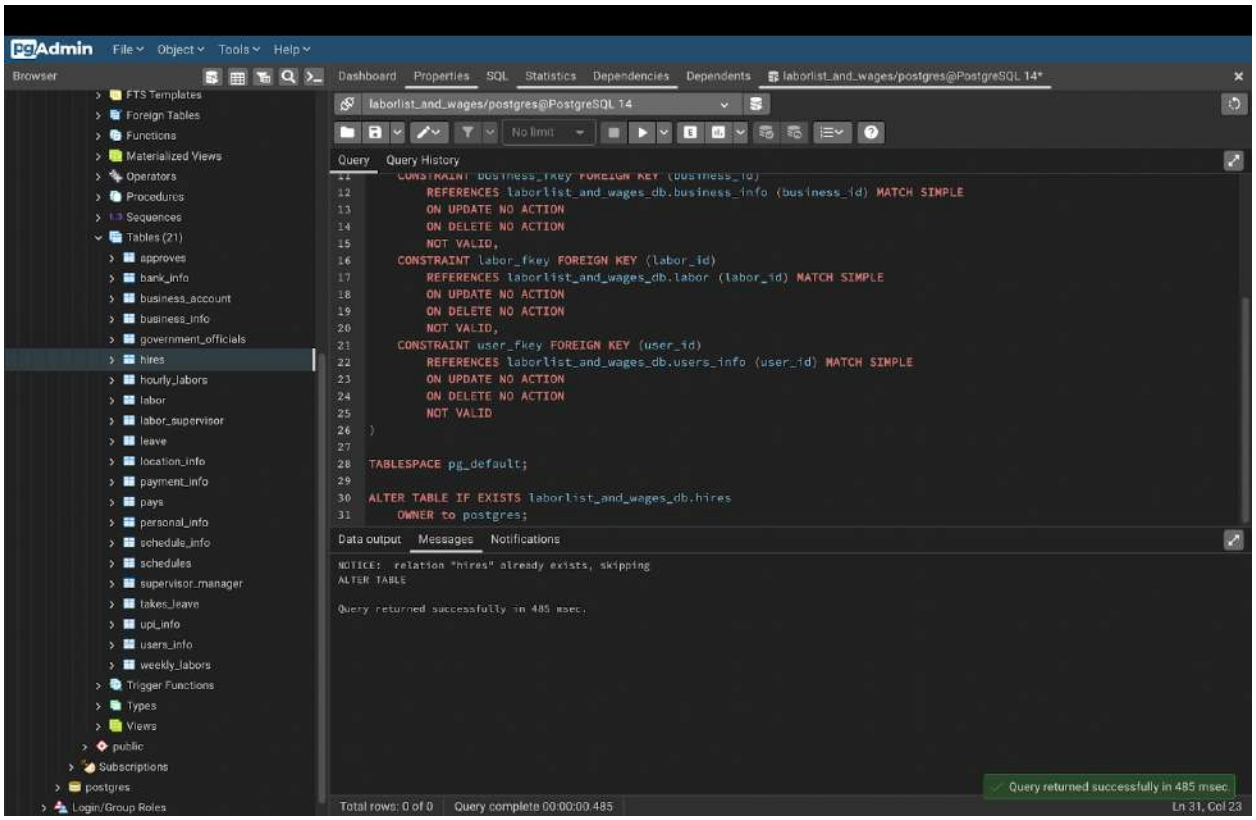
17. Hires table

DDL Script:

```
-- Table: laborlist_and_wages_db.hires

-- DROP TABLE IF EXISTS laborlist_and_wages_db.hires;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.hires
(
    user_id integer,
    business_id integer,
    labor_id integer NOT NULL,
    CONSTRAINT unique_hire UNIQUE (user_id, business_id, labor_id),
    CONSTRAINT business_fkey FOREIGN KEY (business_id)
        REFERENCES laborlist_and_wages_db.business_info (business_id) MATCH
SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
```

```
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT user_fkey FOREIGN KEY (user_id)
        REFERENCES laborlist_and_wages_db.users_info (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.hires
    OWNER to postgres;
```

DDL Snapshots:

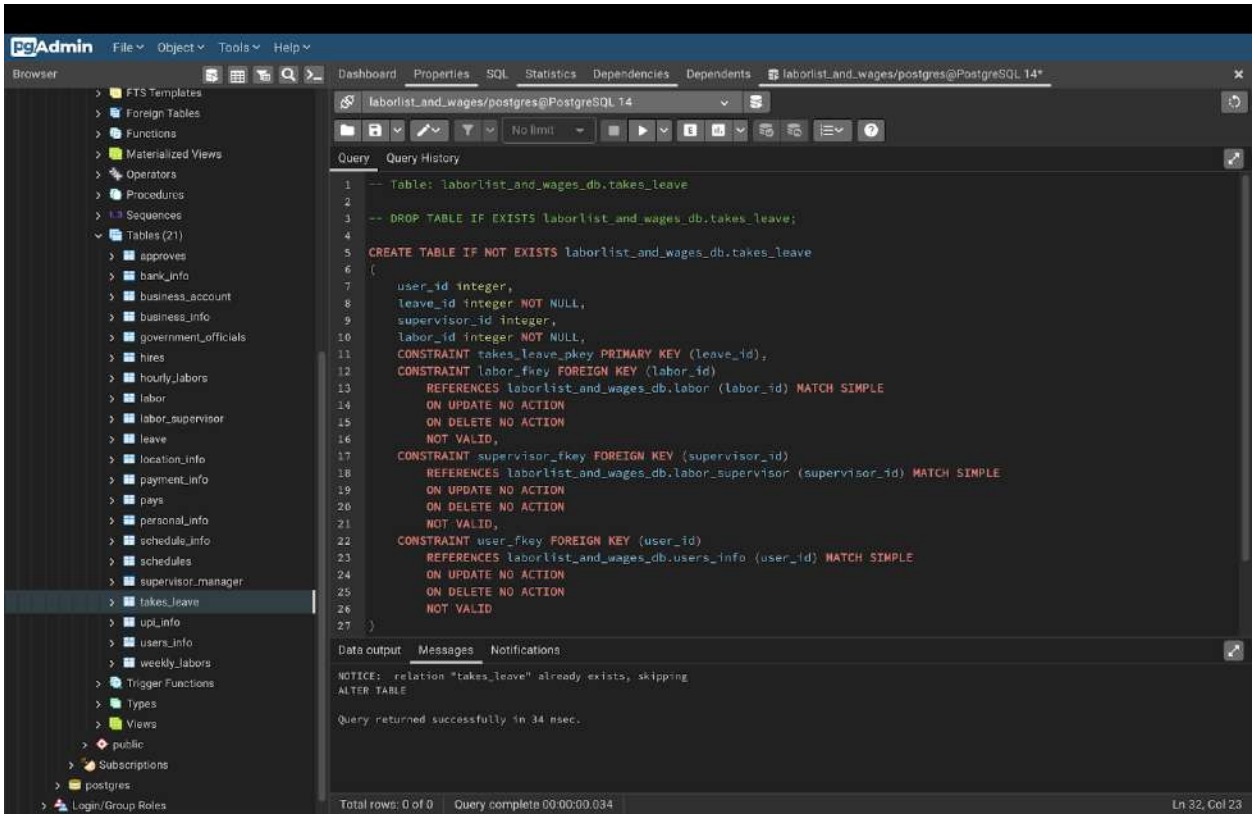


18. Takes_leave table

DDL Script:

```
-- Table: laborlist_and_wages_db.takes_leave

-- DROP TABLE IF EXISTS laborlist_and_wages_db.takes_leave;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.takes_leave
(
    user_id integer,
    leave_id integer NOT NULL,
    supervisor_id integer,
    labor_id integer NOT NULL,
    CONSTRAINT takes_leave_pkey PRIMARY KEY (leave_id),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT supervisor_fkey FOREIGN KEY (supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT user_fkey FOREIGN KEY (user_id)
        REFERENCES laborlist_and_wages_db.users_info (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.takes_leave
    OWNER to postgres;
```

DDL Snapshots:

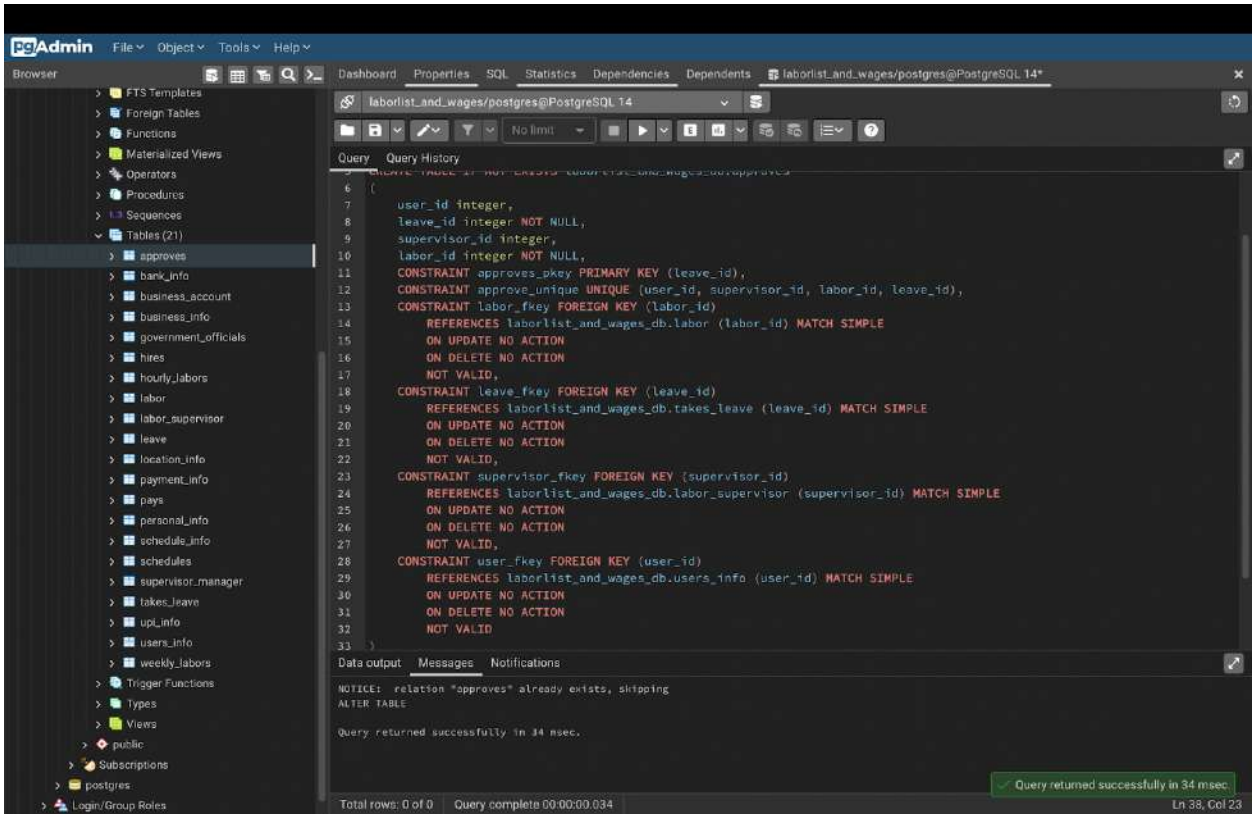


19. Approves table

DDL Script:

```
-- Table: laborlist_and_wages_db.approves

-- DROP TABLE IF EXISTS laborlist_and_wages_db.approves;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.approves
(
    user_id integer,
    leave_id integer NOT NULL,
    supervisor_id integer,
    labor_id integer NOT NULL,
    CONSTRAINT approves_pkey PRIMARY KEY (leave_id),
    CONSTRAINT approve_unique UNIQUE (user_id, supervisor_id, labor_id,
leave_id),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
```

```
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT leave_fkey FOREIGN KEY (leave_id)
        REFERENCES laborlist_and_wages_db.takes_leave (leave_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT supervisor_fkey FOREIGN KEY (supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT user_fkey FOREIGN KEY (user_id)
        REFERENCES laborlist_and_wages_db.users_info (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.approves
    OWNER to postgres;


DDL Snapshots:
```



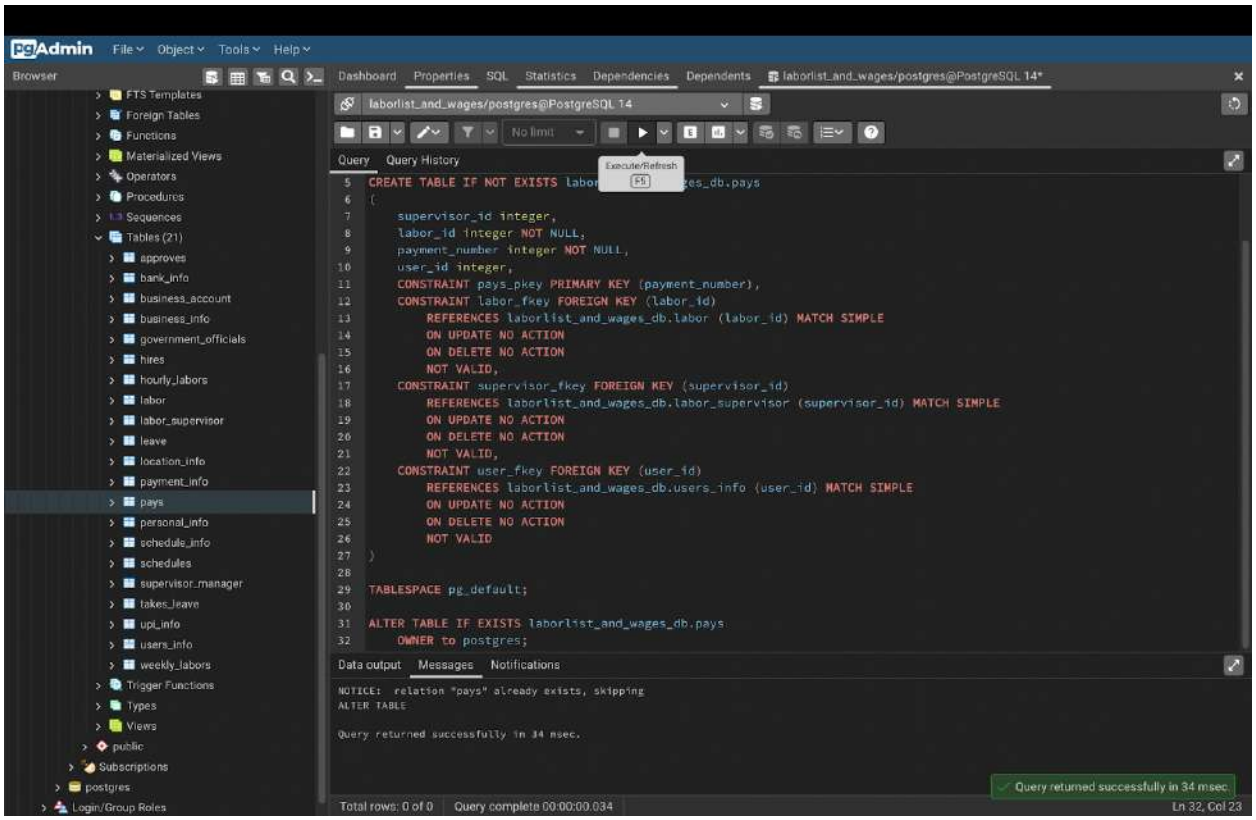
20. Pays table

DDL Script:

```
-- Table: laborlist_and_wages_db.pays

-- DROP TABLE IF EXISTS laborlist_and_wages_db.pays;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.pays
(
    supervisor_id integer,
    labor_id integer NOT NULL,
    payment_number integer NOT NULL,
    user_id integer,
    CONSTRAINT pays_pkey PRIMARY KEY (payment_number),
    CONSTRAINT labor_fkey FOREIGN KEY (labor_id)
        REFERENCES laborlist_and_wages_db.labor (labor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
```

```
    CONSTRAINT supervisor_fkey FOREIGN KEY (supervisor_id)
        REFERENCES laborlist_and_wages_db.labor_supervisor (supervisor_id)
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID,
    CONSTRAINT user_fkey FOREIGN KEY (user_id)
        REFERENCES laborlist_and_wages_db.users_info (user_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.pays
    OWNER to postgres;
```

DDL Snapshots:

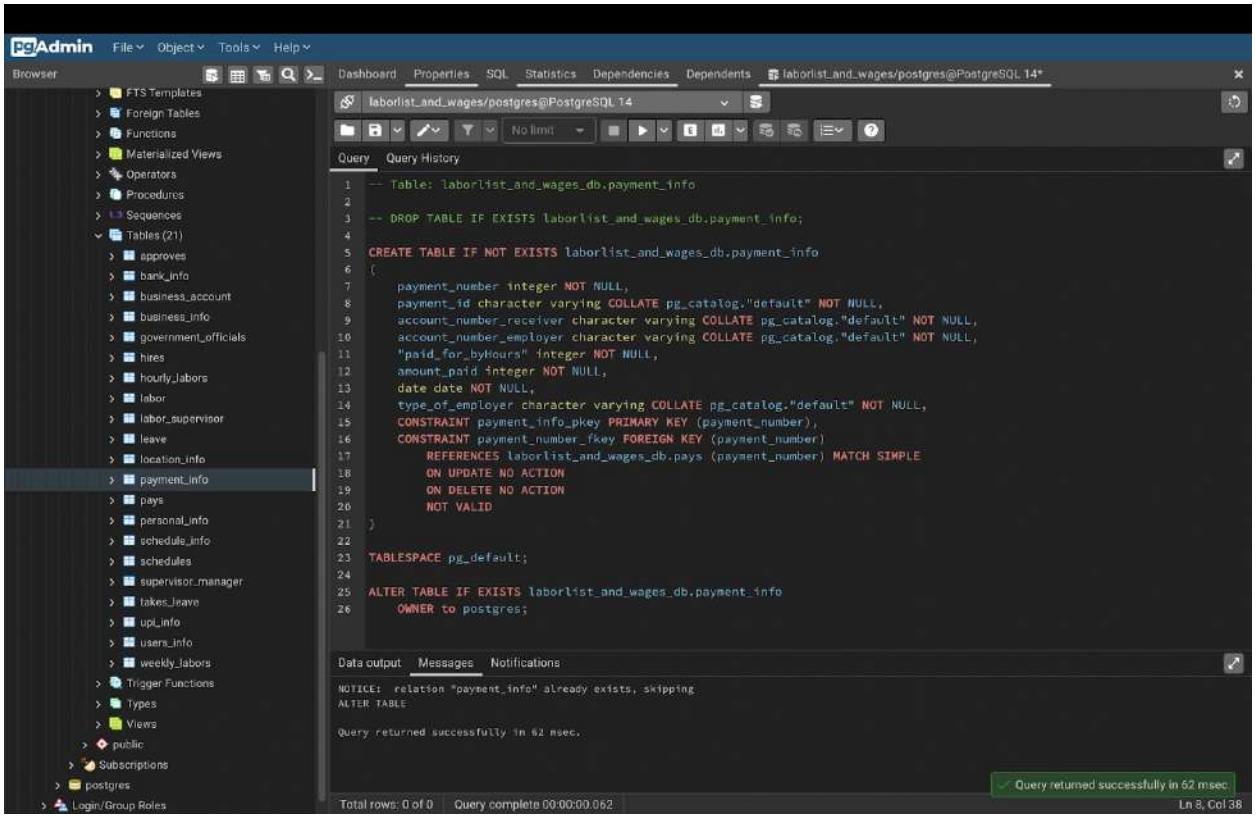


21. Payment_info table

DDL Script:

```
-- Table: laborlist_and_wages_db.payment_info

-- DROP TABLE IF EXISTS laborlist_and_wages_db.payment_info;

CREATE TABLE IF NOT EXISTS laborlist_and_wages_db.payment_info
(
    payment_number integer NOT NULL,
    payment_id character varying COLLATE pg_catalog."default" NOT NULL,
    paid_for_byhours integer NOT NULL,
    amount_paid integer NOT NULL,
    date date NOT NULL,
    CONSTRAINT payment_info_pkey PRIMARY KEY (payment_number),
    CONSTRAINT payment_number_fkey FOREIGN KEY (payment_number)
        REFERENCES laborlist_and_wages_db.pays (payment_number) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS laborlist_and_wages_db.payment_info
    OWNER to postgres;
```

DDL Snapshots:



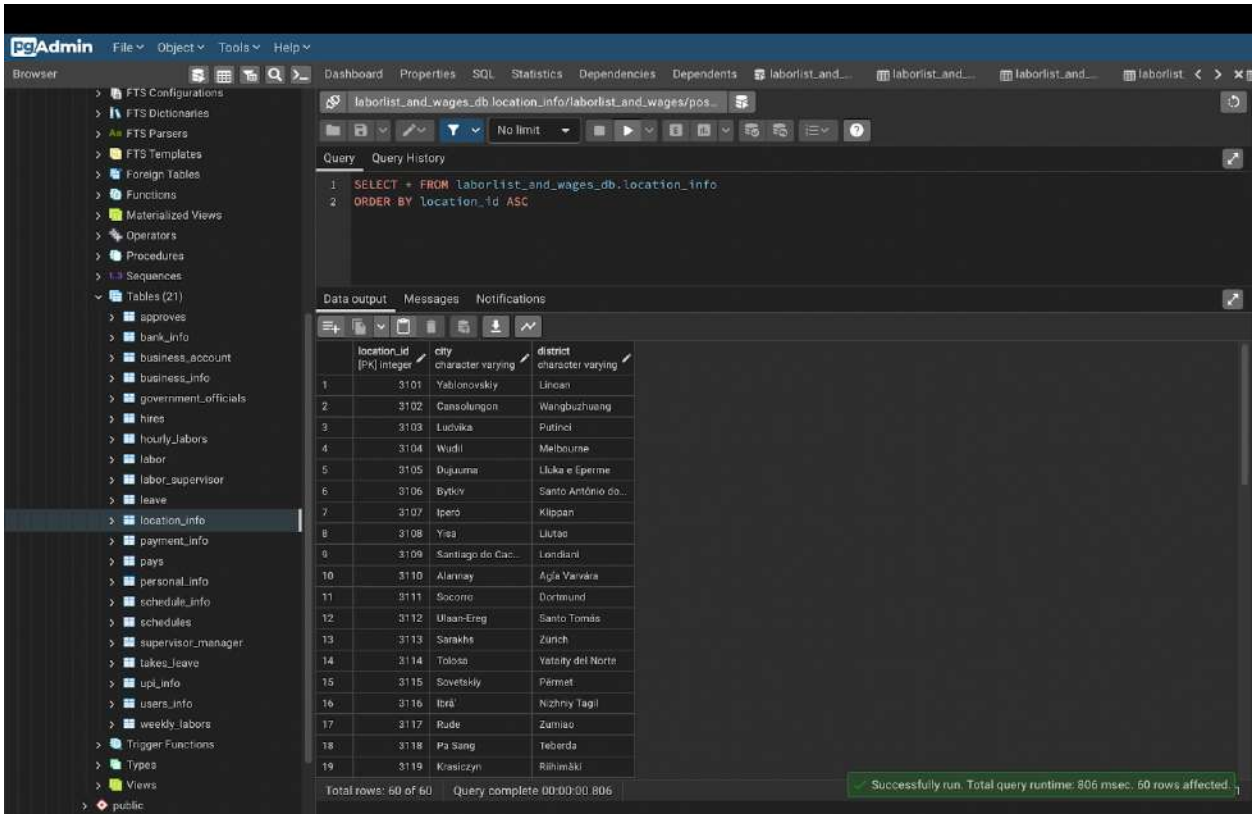
16.3 Data Snapshots

SQL Query for Showing Data:

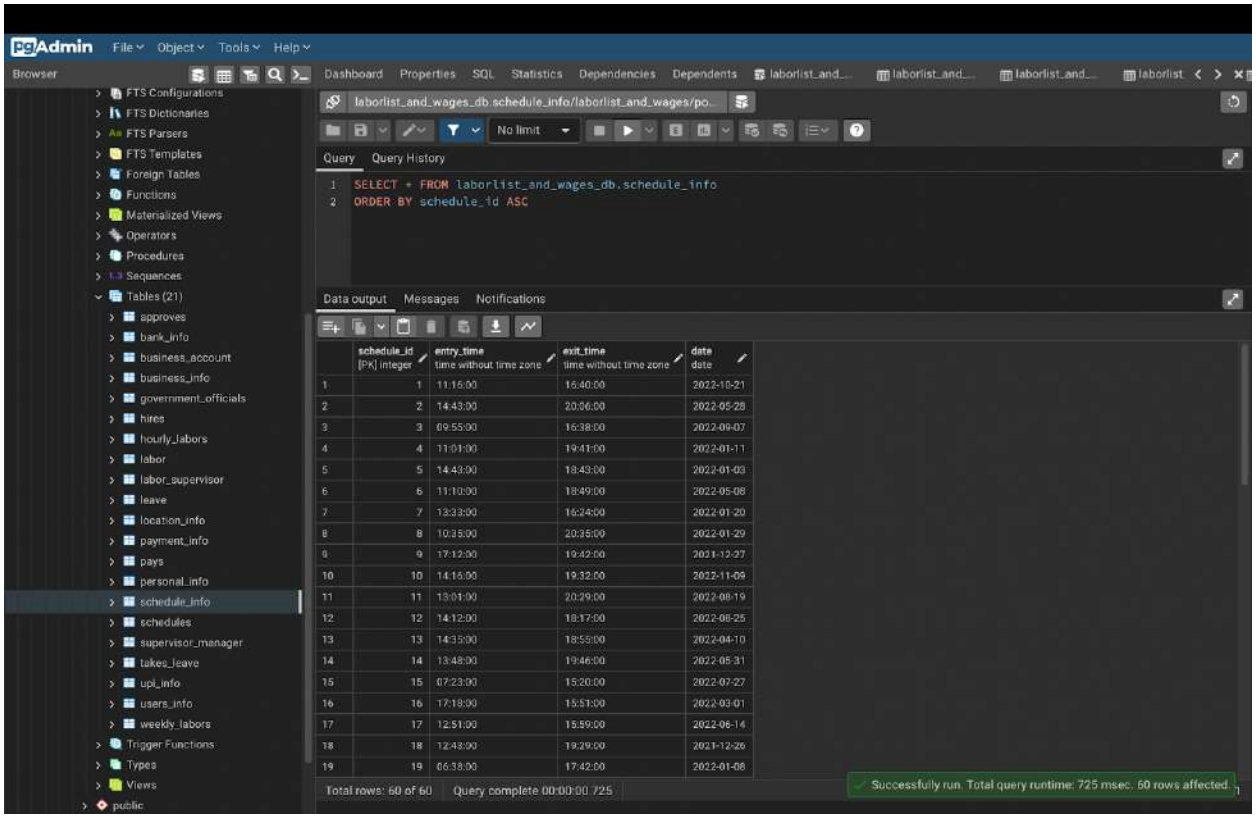
```
SELECT * FROM laborlist_and_wages_db.location_info

SELECT * FROM laborlist_and_wages_db.schedule_info

SELECT * FROM laborlist_and_wages_db.bank_info

SELECT * FROM laborlist_and_wages_db.leave

SELECT * FROM laborlist_and_wages_db.upi_info

SELECT * FROM laborlist_and_wages_db.business_account

SELECT * FROM laborlist_and_wages_db.government_officials

SELECT * FROM laborlist_and_wages_db.business_info
```

```
SELECT * FROM laborlist_and_wages_db.labor_supervisor

SELECT * FROM laborlist_and_wages_db.labor

SELECT * FROM laborlist_and_wages_db.users_info

SELECT * FROM laborlist_and_wages_db.hourly_labors

SELECT * FROM laborlist_and_wages_db.weekly_labors

SELECT * FROM laborlist_and_wages_db.supervisor_manager

SELECT * FROM laborlist_and_wages_db.personal_info

SELECT * FROM laborlist_and_wages_db.schedules

SELECT * FROM laborlist_and_wages_db.hires

SELECT * FROM laborlist_and_wages_db.takes_leave

SELECT * FROM laborlist_and_wages_db.approves

SELECT * FROM laborlist_and_wages_db.pays

SELECT * FROM laborlist_and_wages_db.payment_info
```

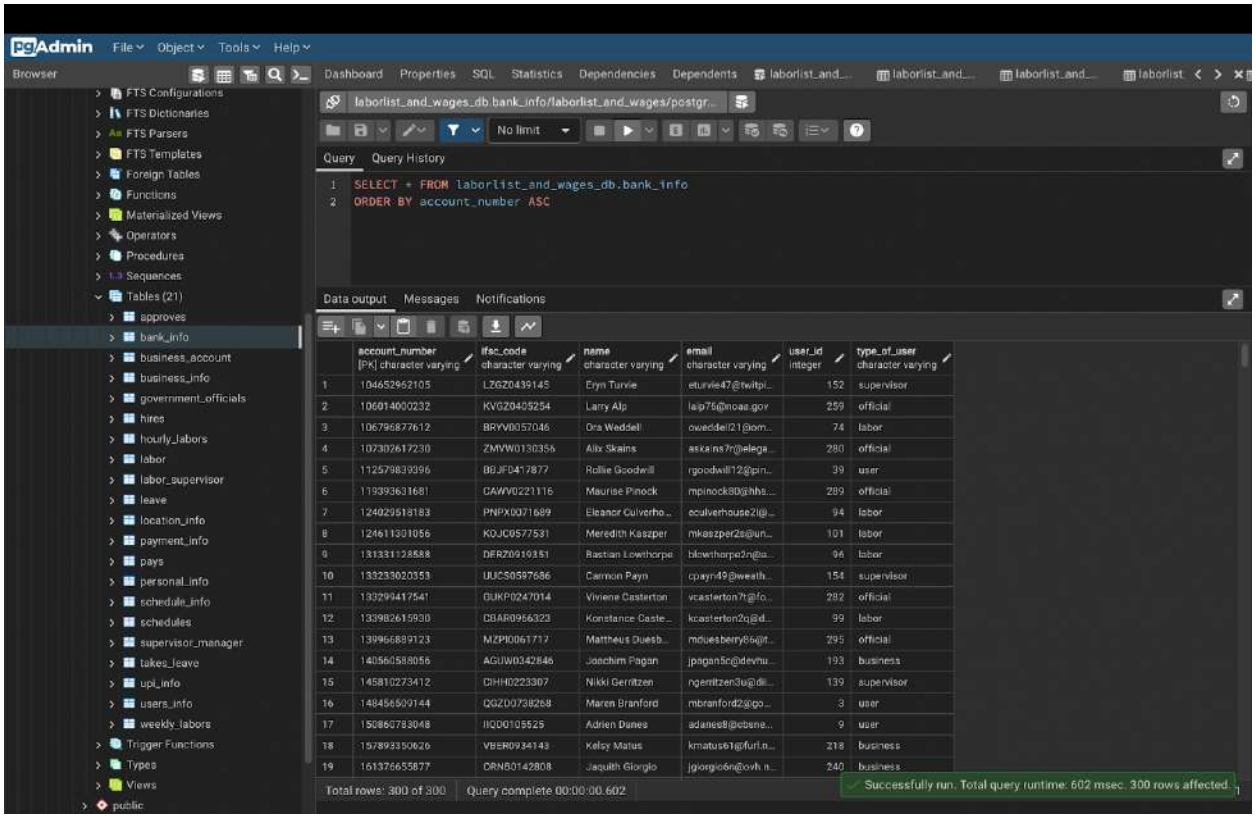
1. Location_info Table



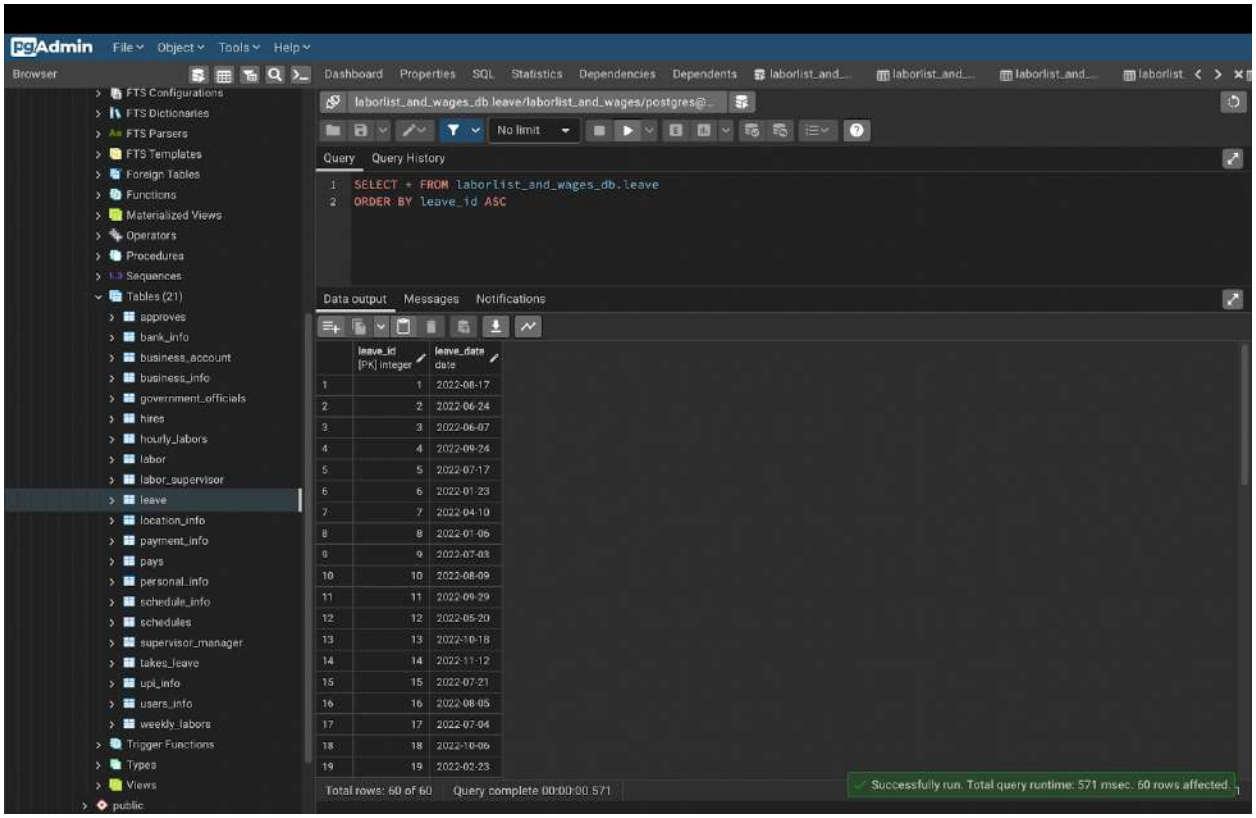
2. Schedule_info Table



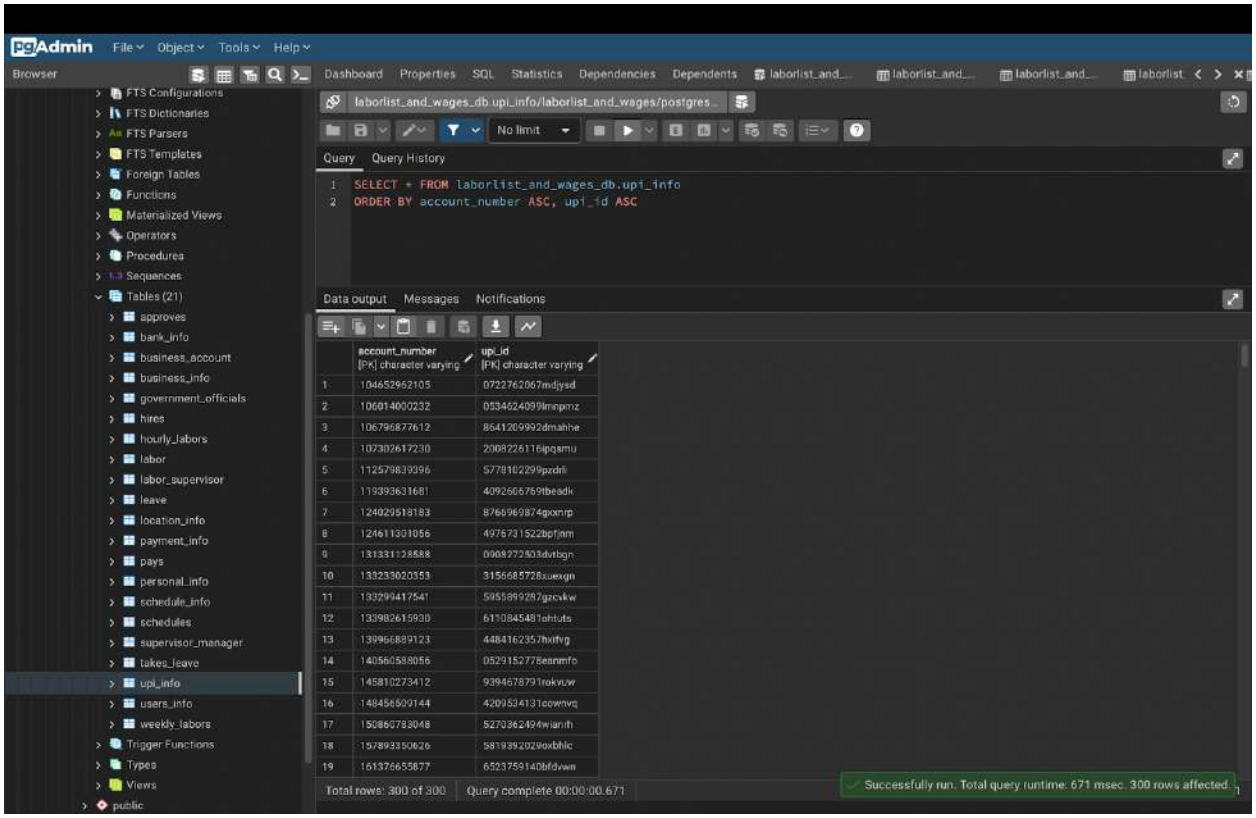
3. Bank_info table



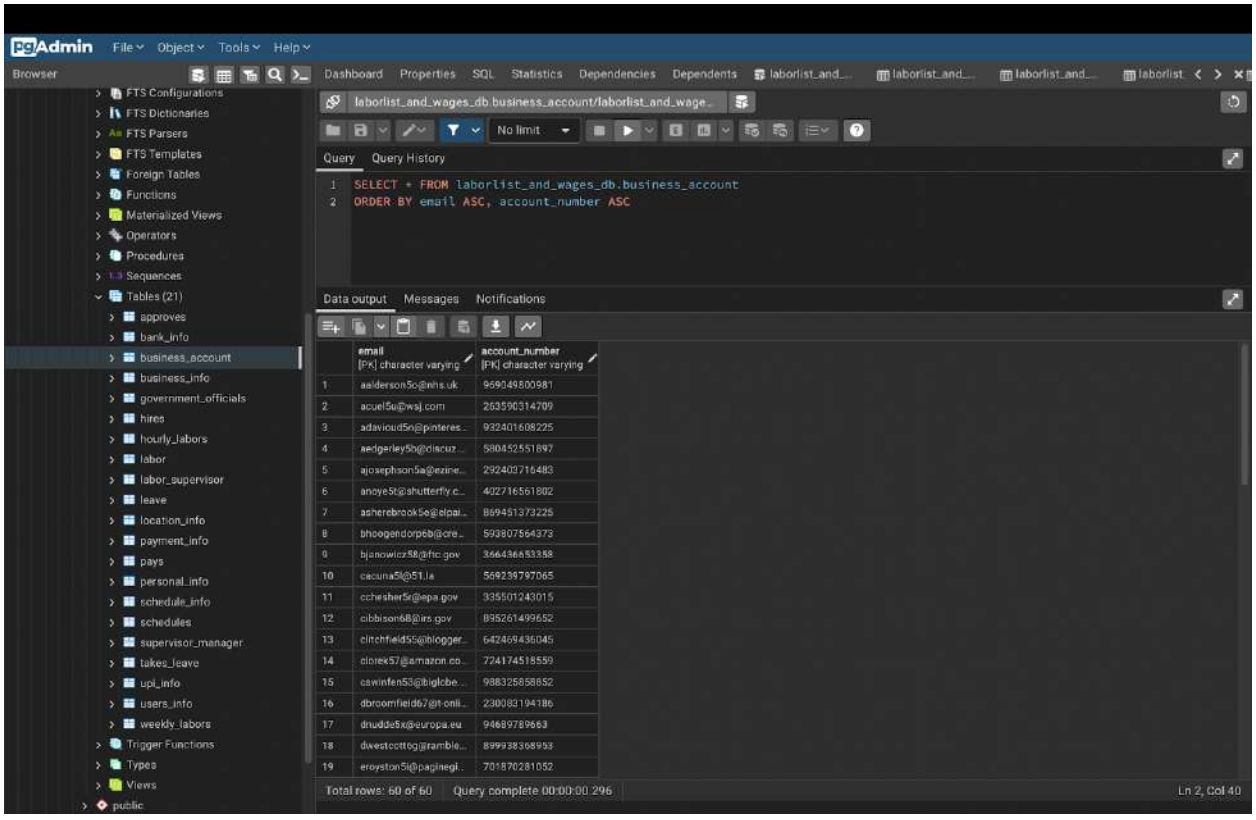
4. Leave table



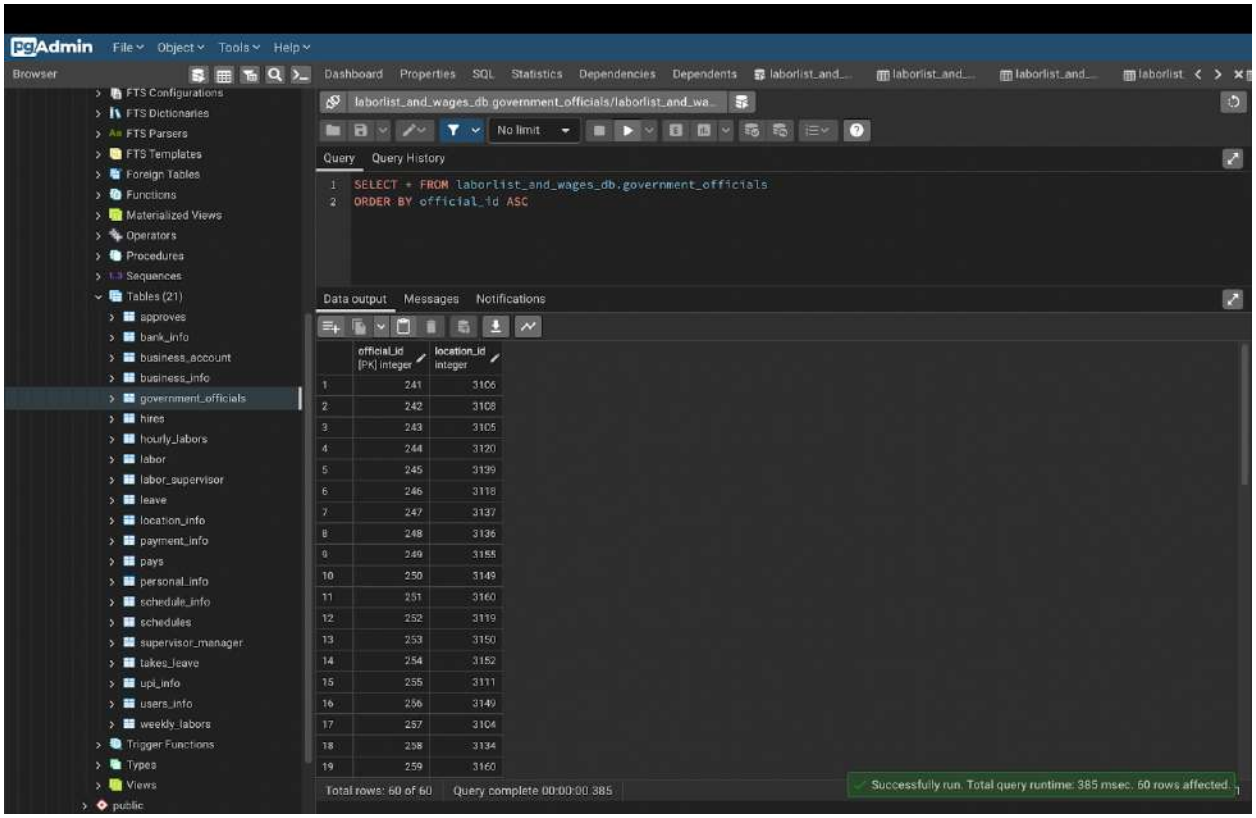
5. Upi_info table



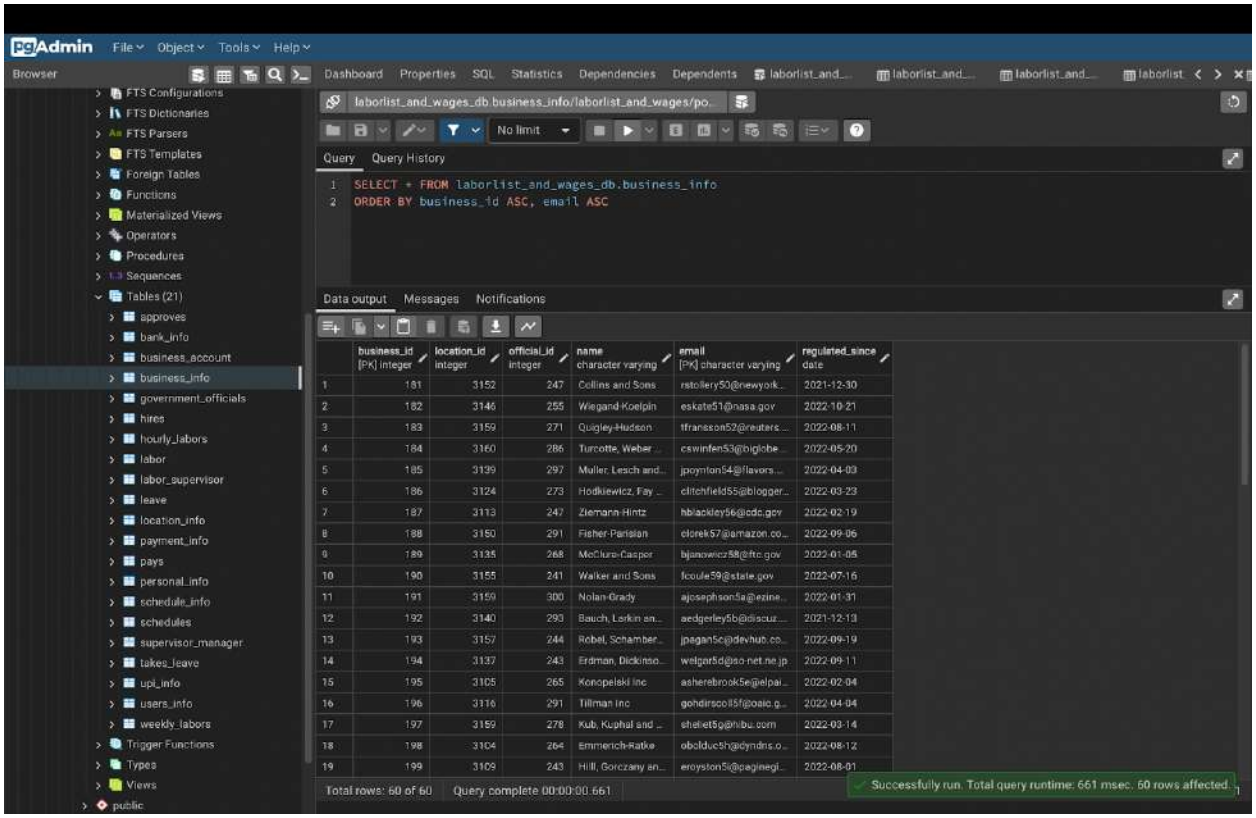
6. Business_account table



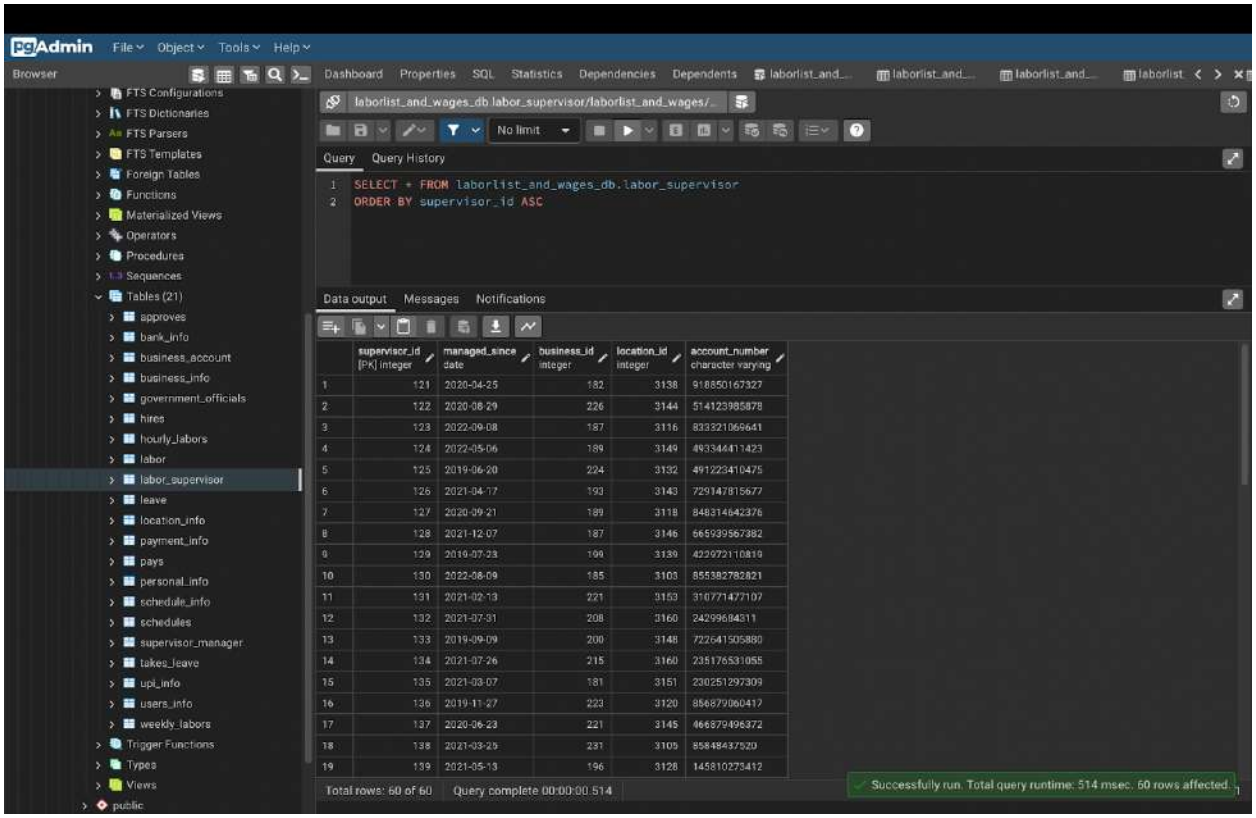
7. Government_officials table



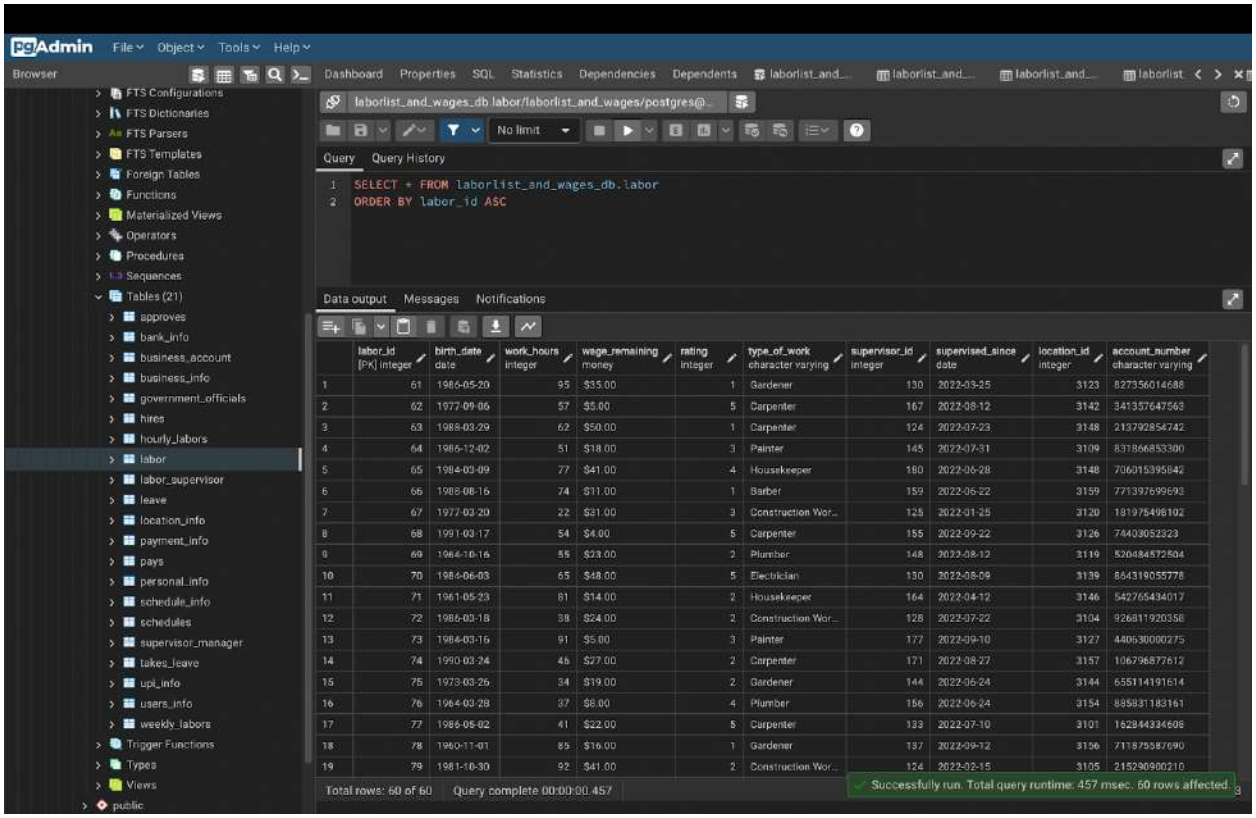
8. Business_info table



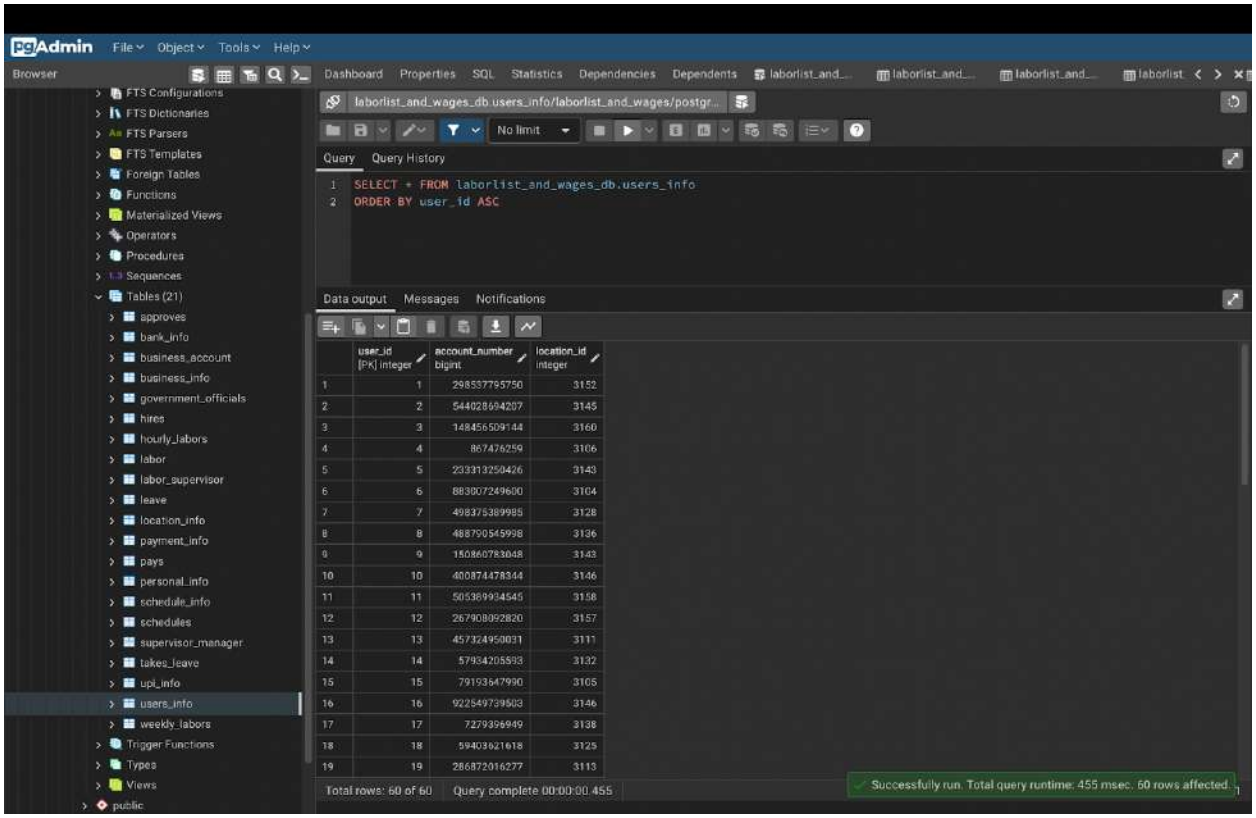
9. Labor_supervisor table



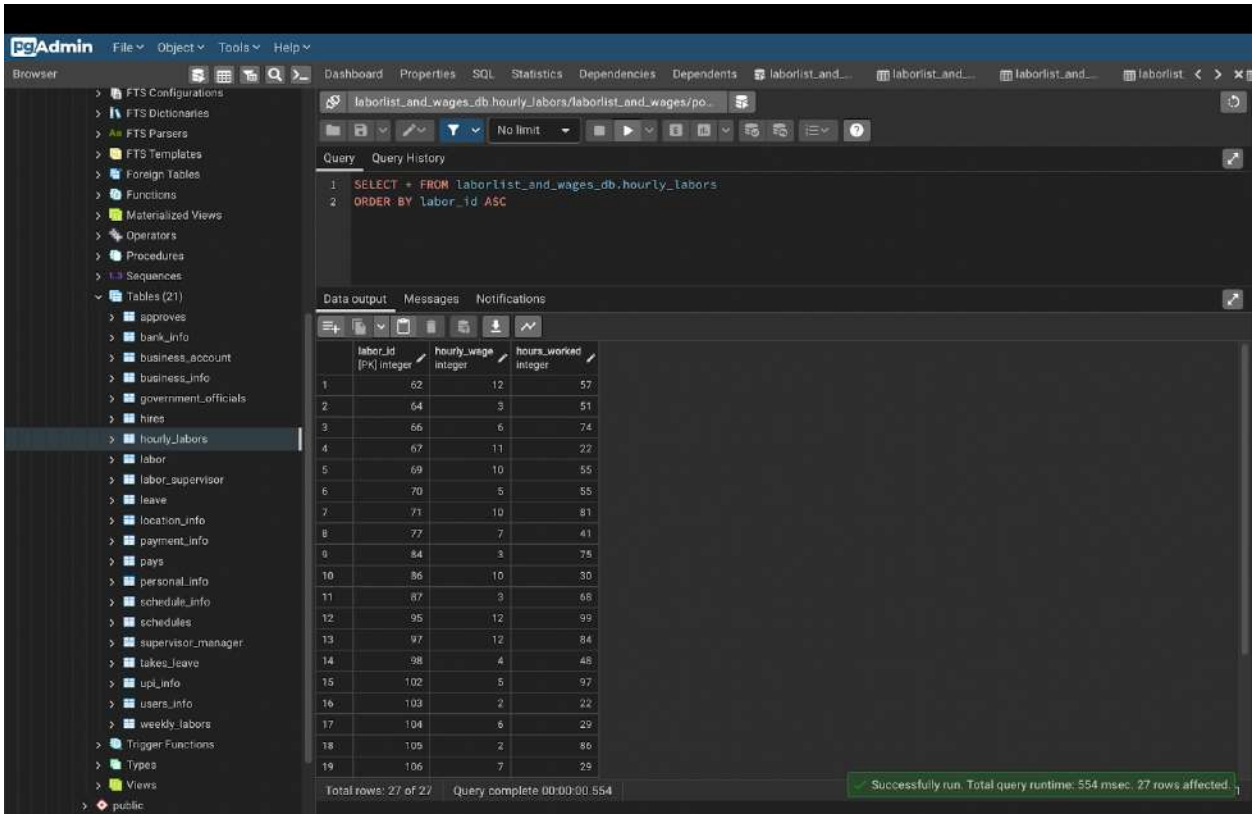
10. Labor table



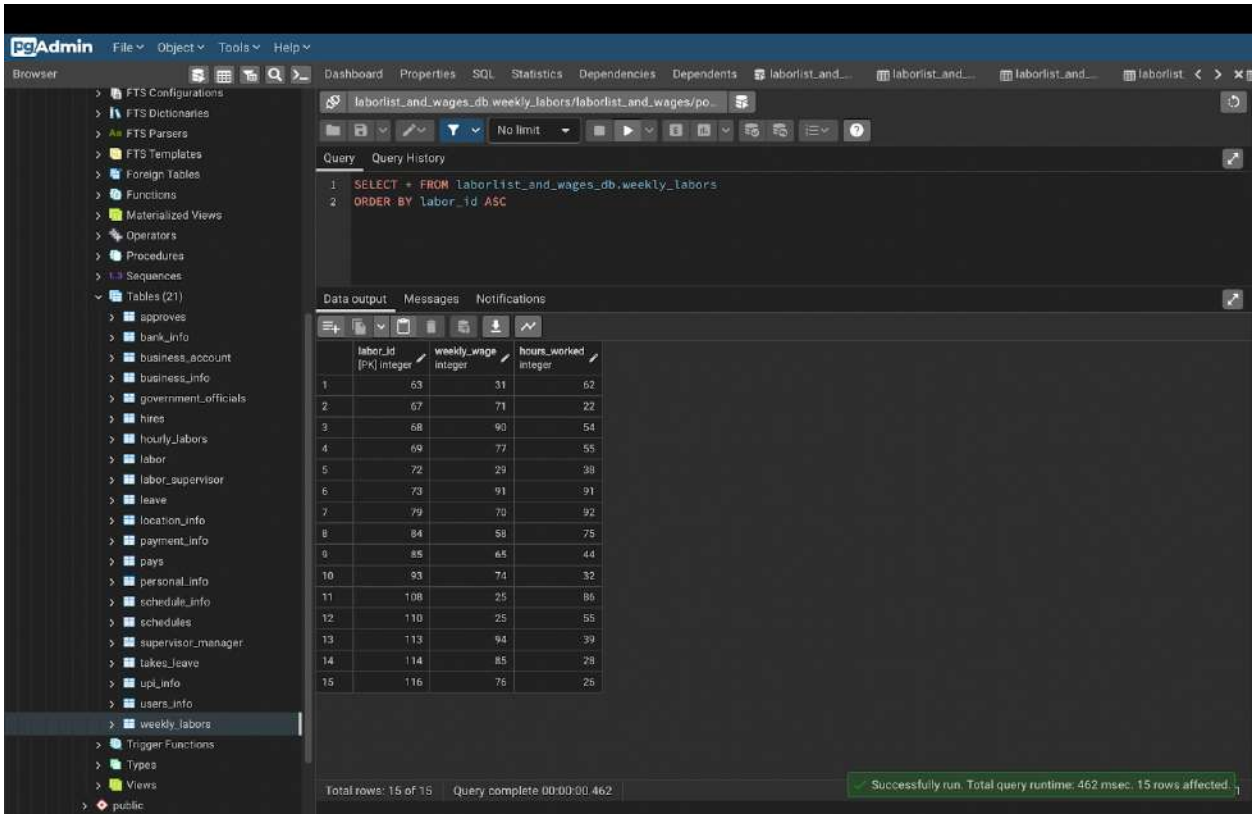
11. Users_info table



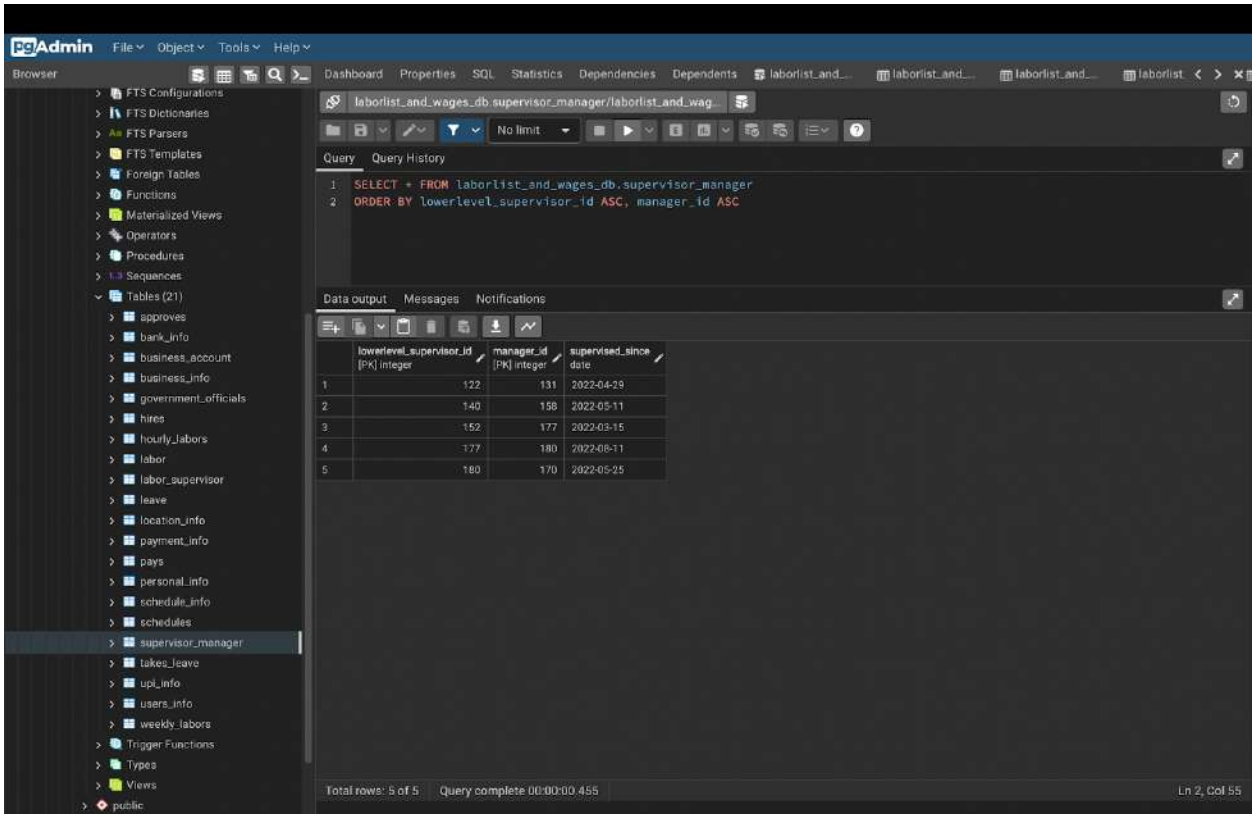
12. Hourly_labors table



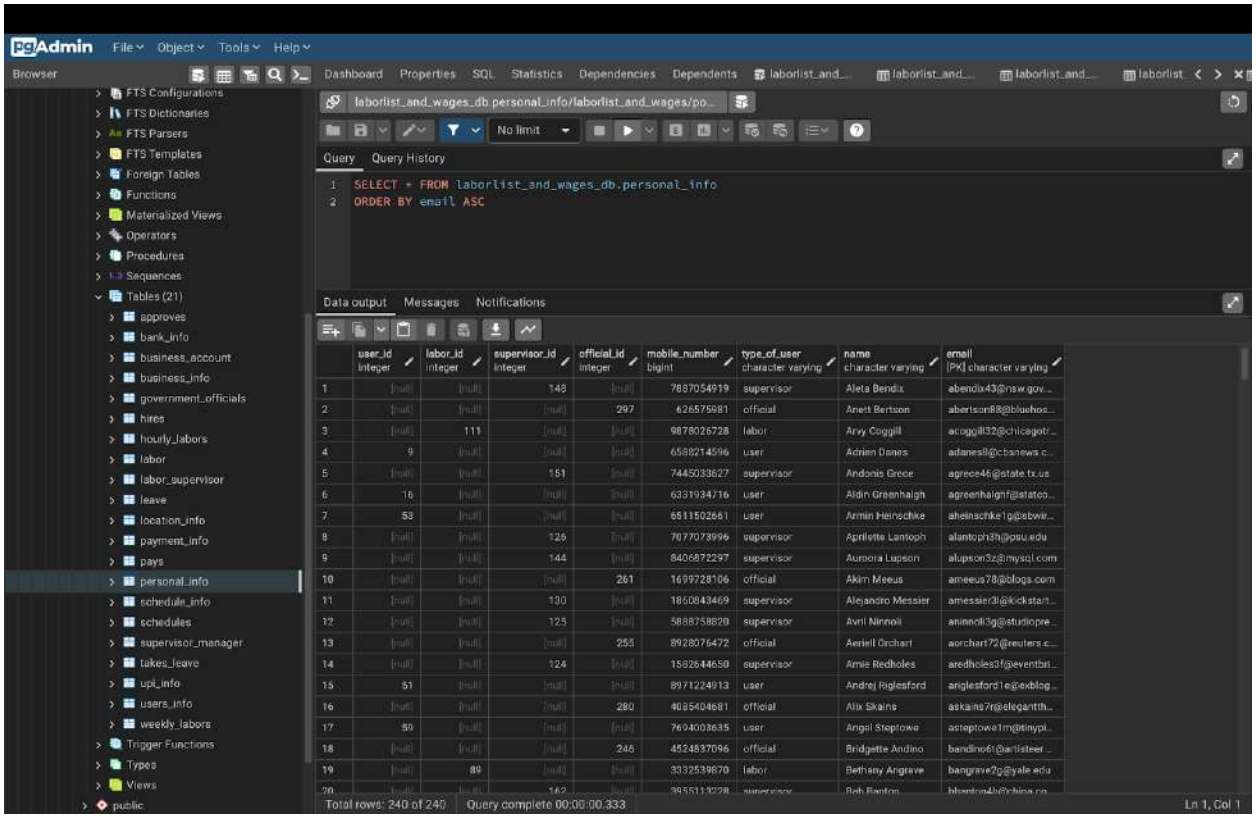
13. Weekly_labors table



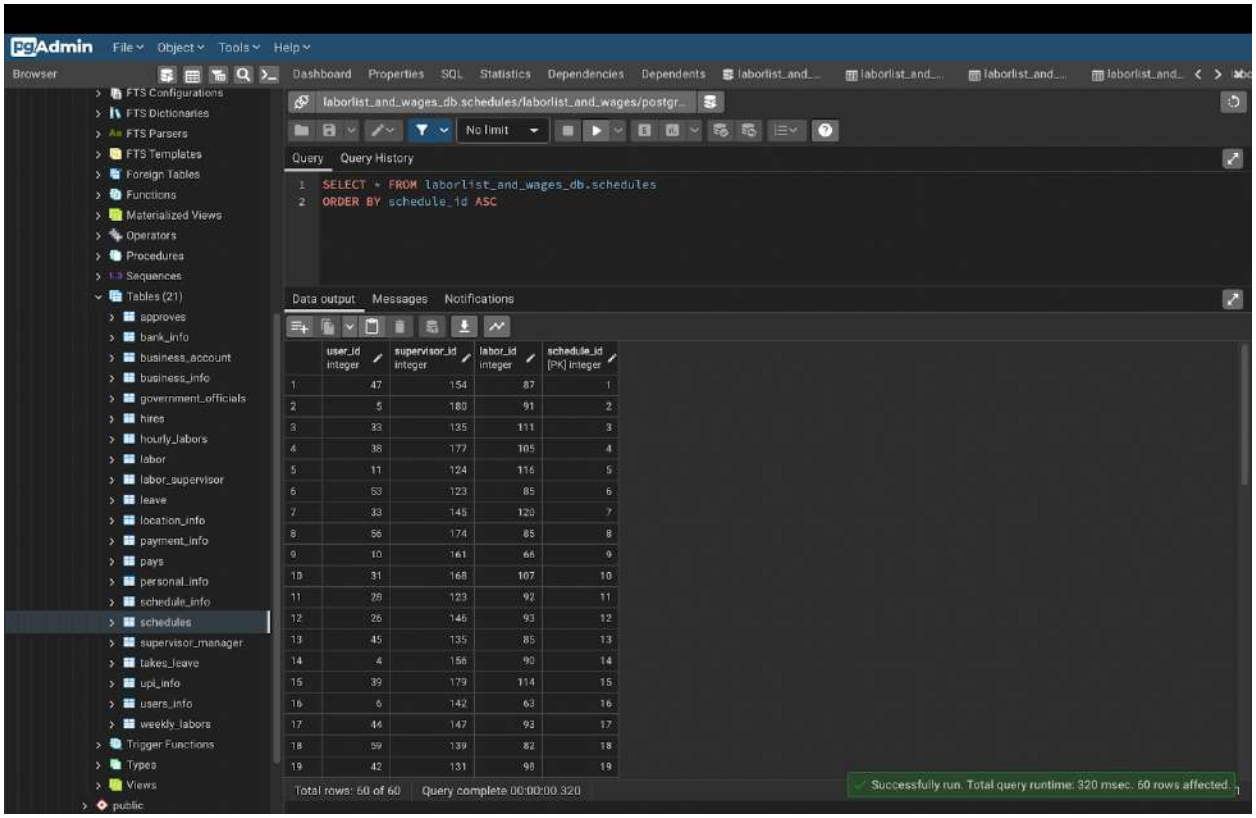
14. Supervisor_manager table



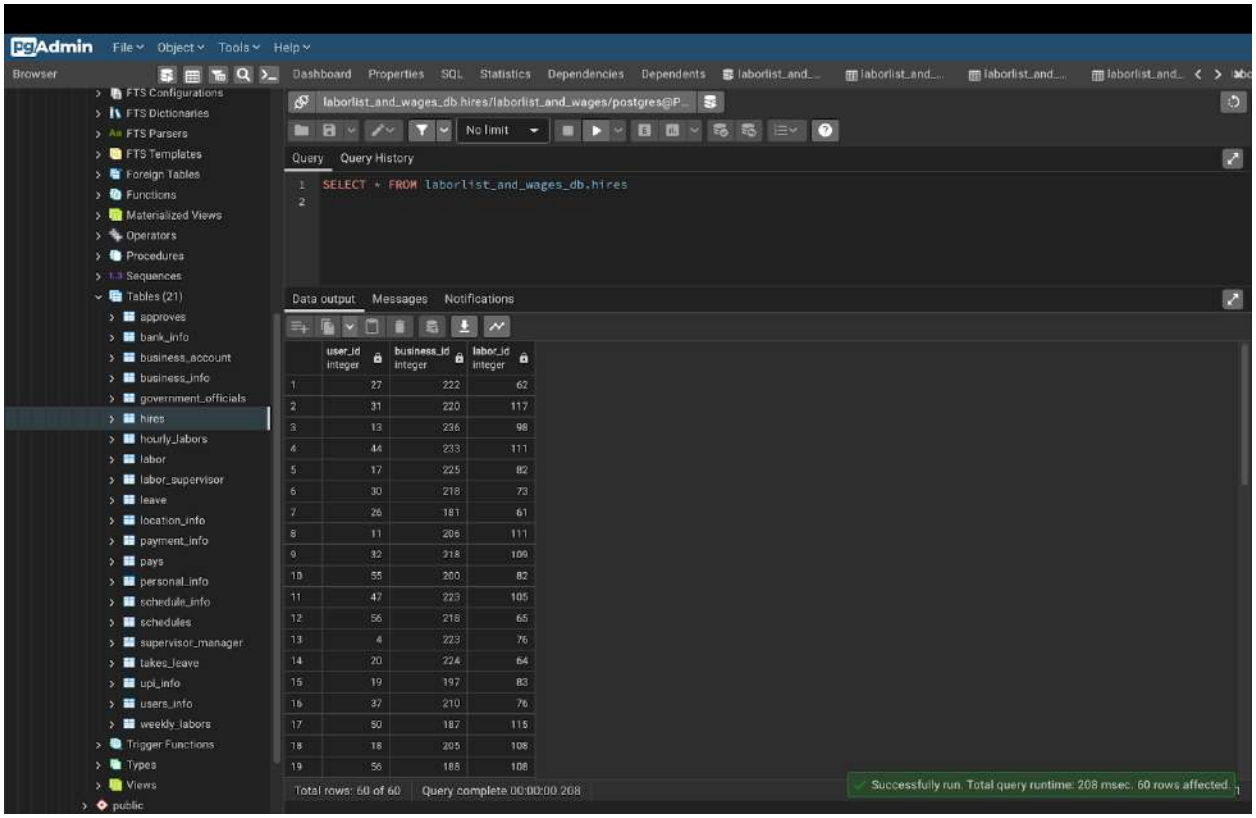
15. Personal_info table



16. Schedules table

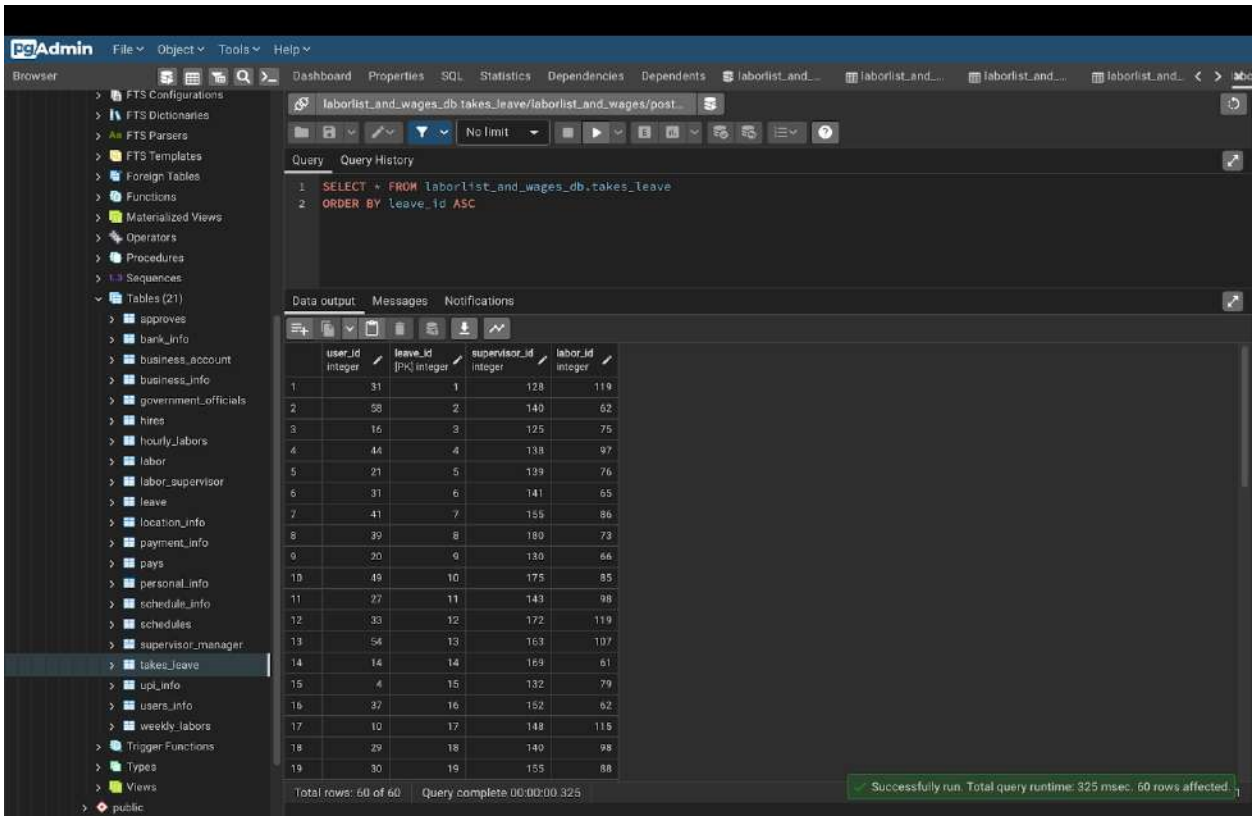


17. Hires table

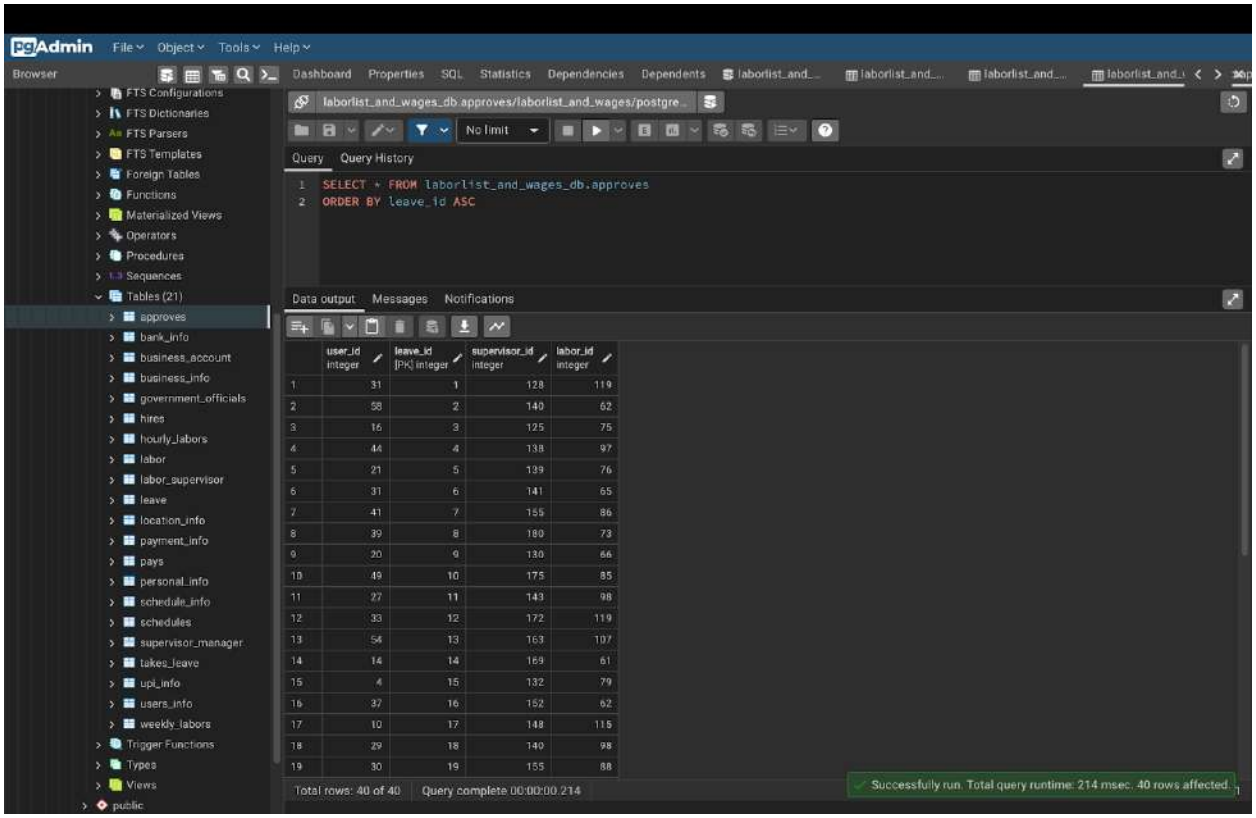


	user_id integer	business_id integer	labor_id integer
1	27	222	62
2	31	220	117
3	13	236	98
4	44	233	111
5	17	225	82
6	30	218	73
7	26	181	61
8	11	206	111
9	32	218	109
10	55	200	82
11	47	223	105
12	56	218	65
13	4	223	76
14	20	224	64
15	19	197	83
16	37	210	76
17	50	187	115
18	18	205	108
19	56	188	108

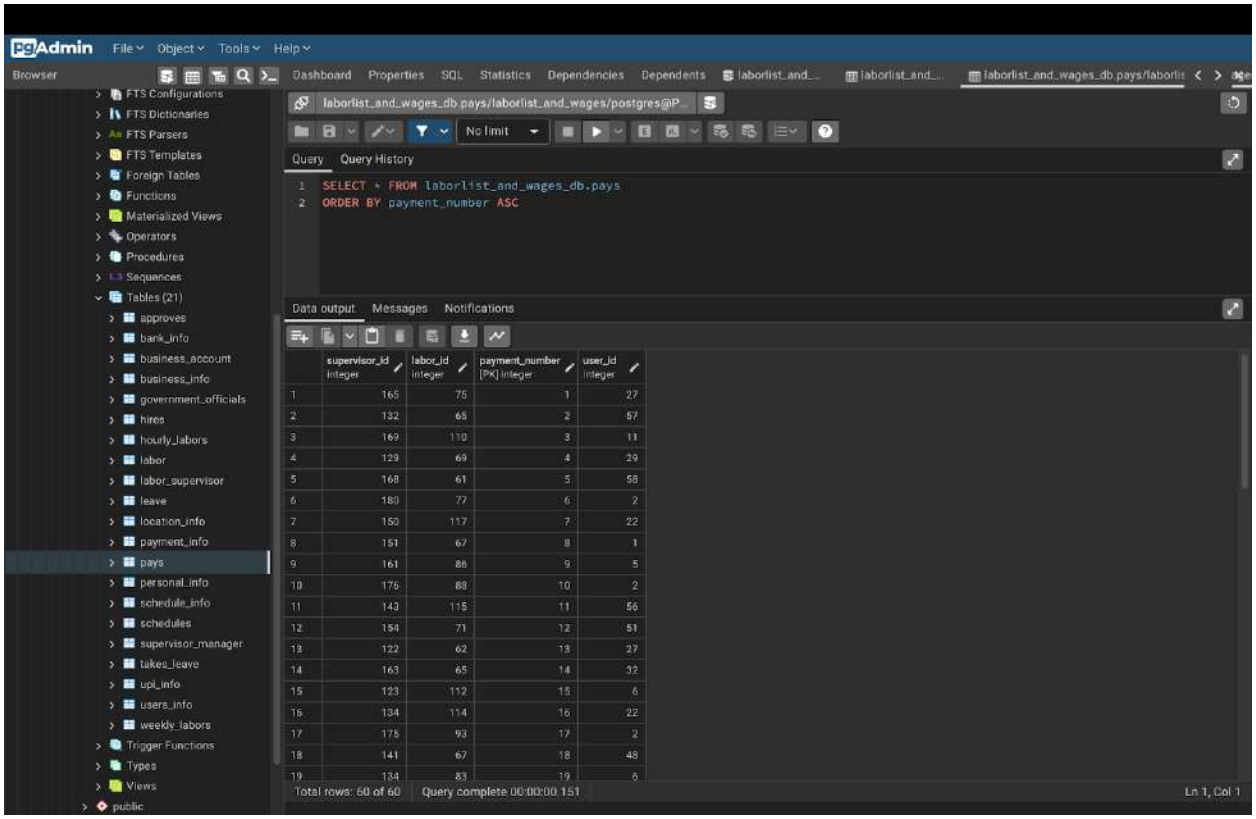
18. Takes_leave table



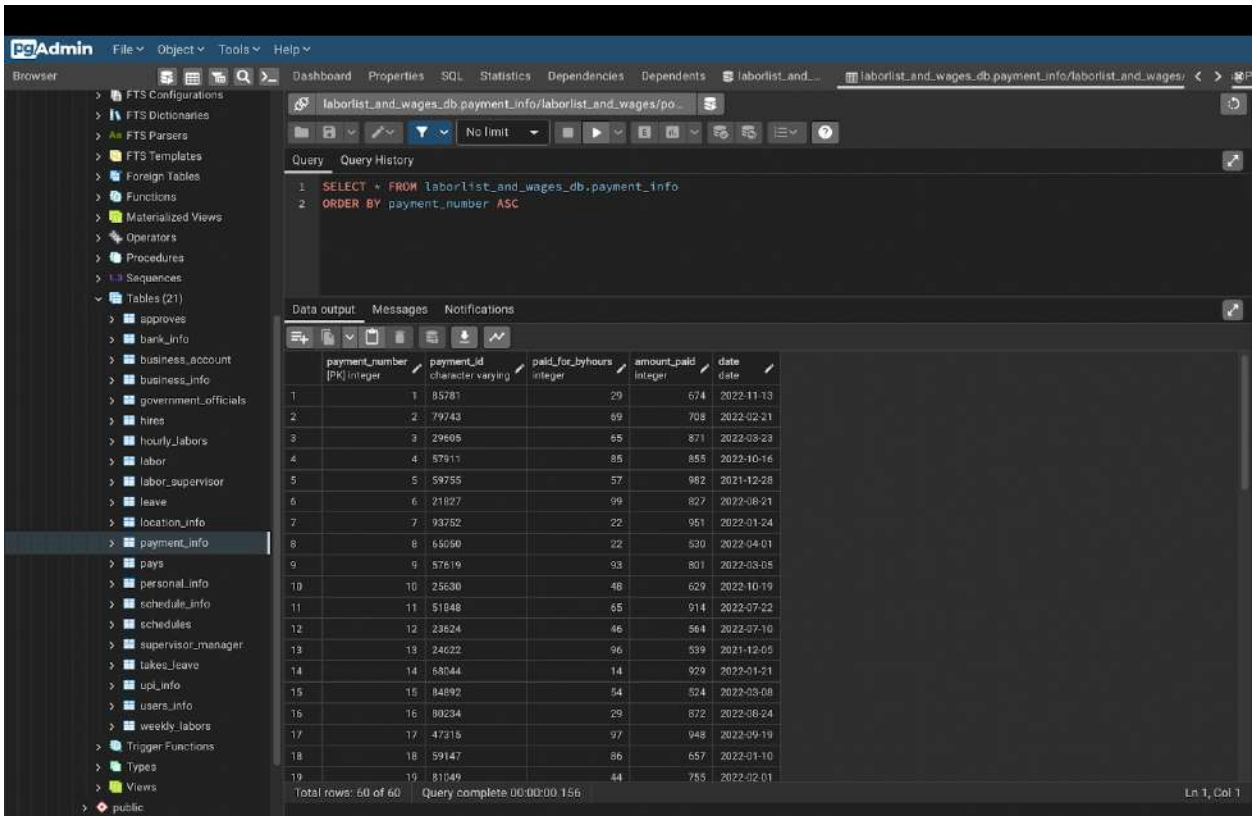
19. Approves table



20. Pays table



21. Payment_info table

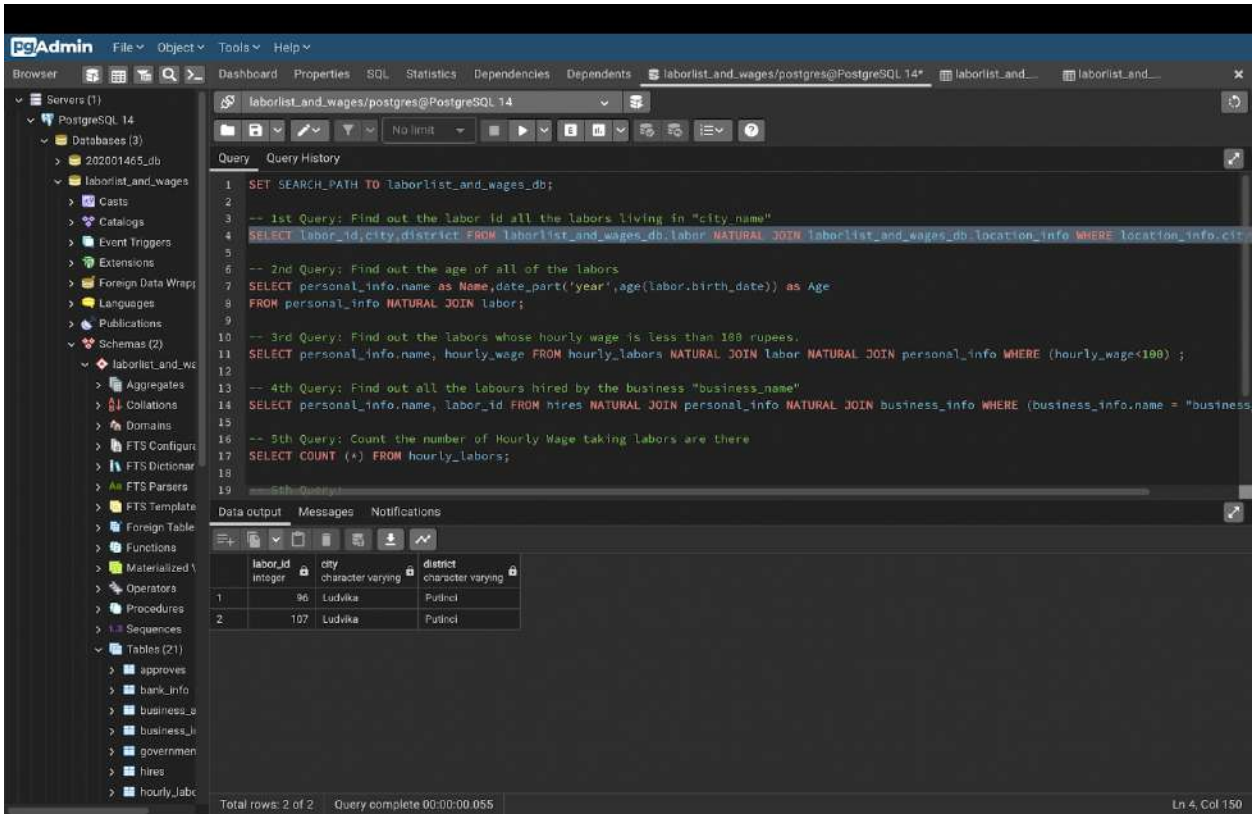


17. SQL Queries

~20 SQL queries and their outputs for the Case Study. (Consists of mostly complex queries and some simple queries because of the database's high complexity)

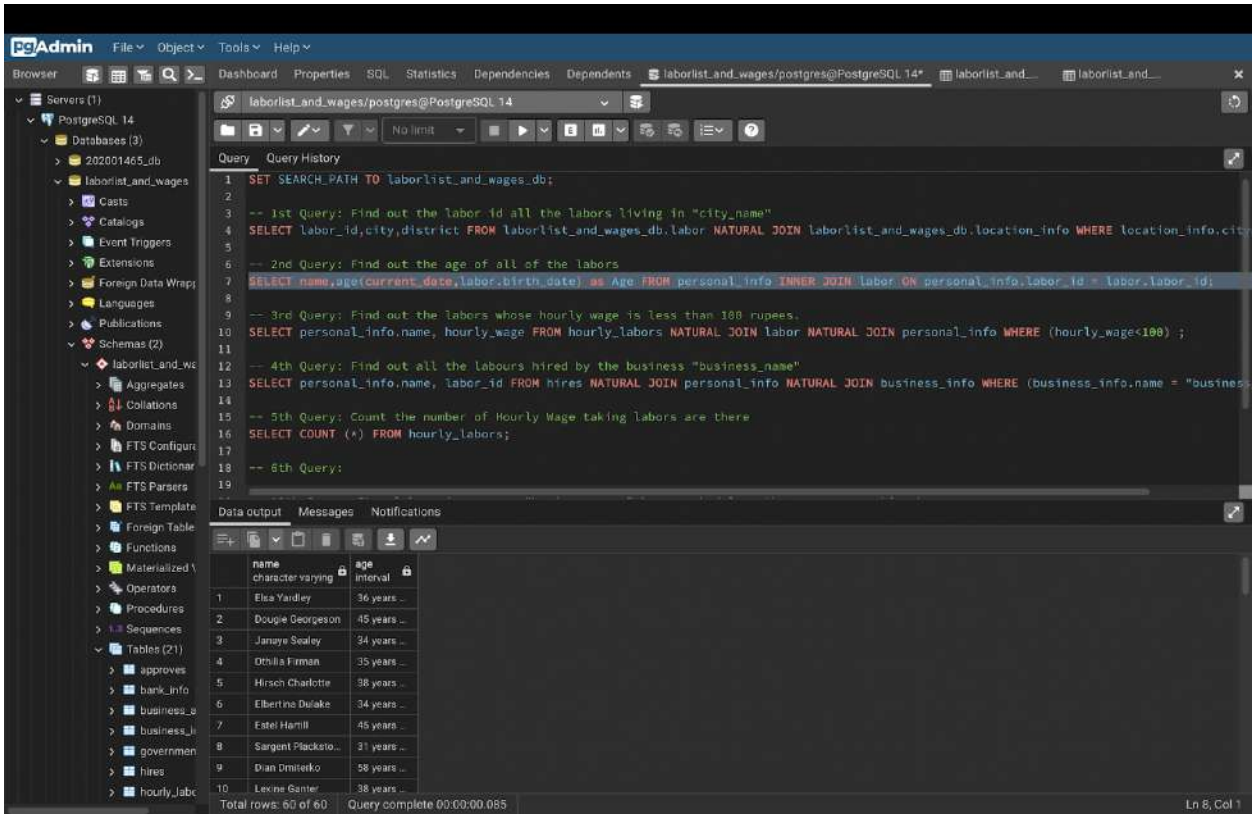
1st Query: Find out the labor id all the labors living in "city_name"

```
SELECT labor_id,city,district FROM laborlist_and_wages_db.labor NATURAL JOIN
laborlist_and_wages_db.location_info WHERE location_info.city='Ludvika';
```



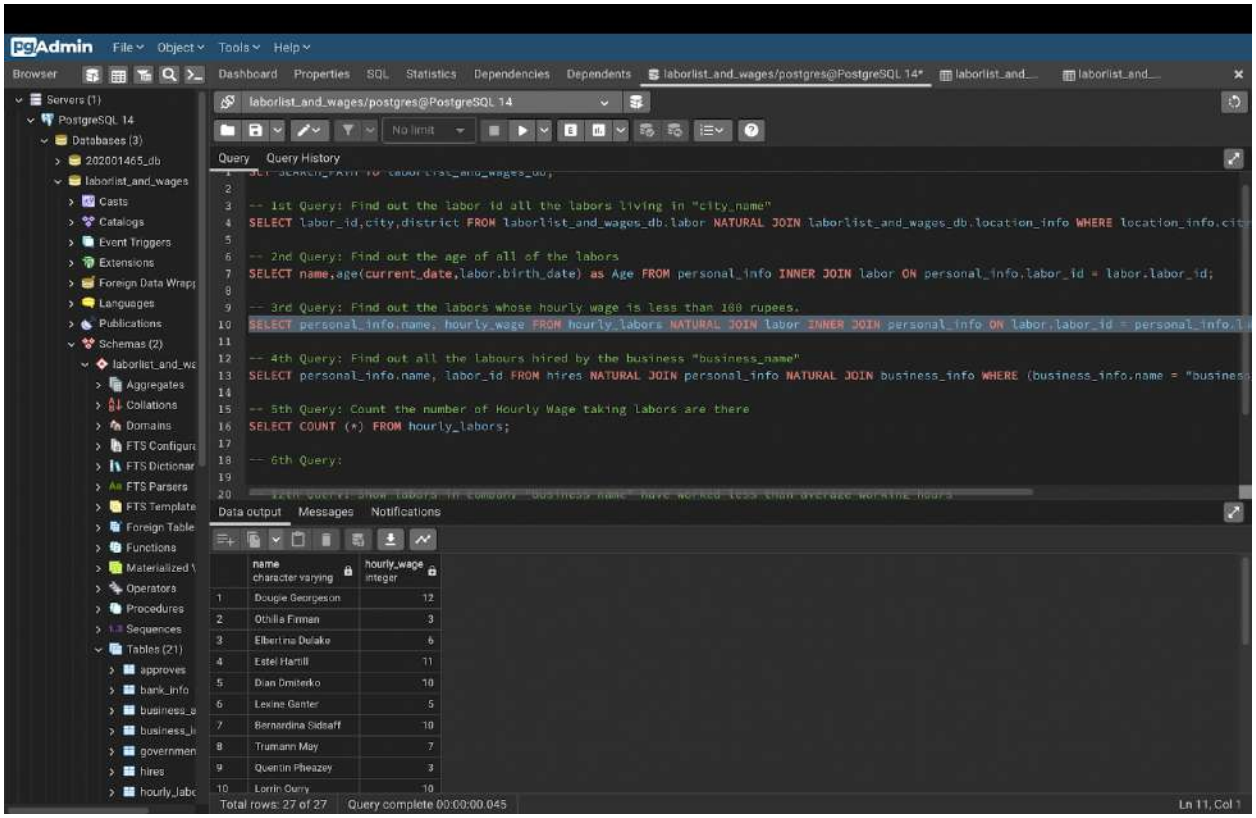
2nd Query: Find out the age of all of the labors

```
SELECT name,age(current_date,labor.birth_date) as Age FROM personal_info INNER
JOIN labor ON personal_info.labor_id = labor.labor_id;
```



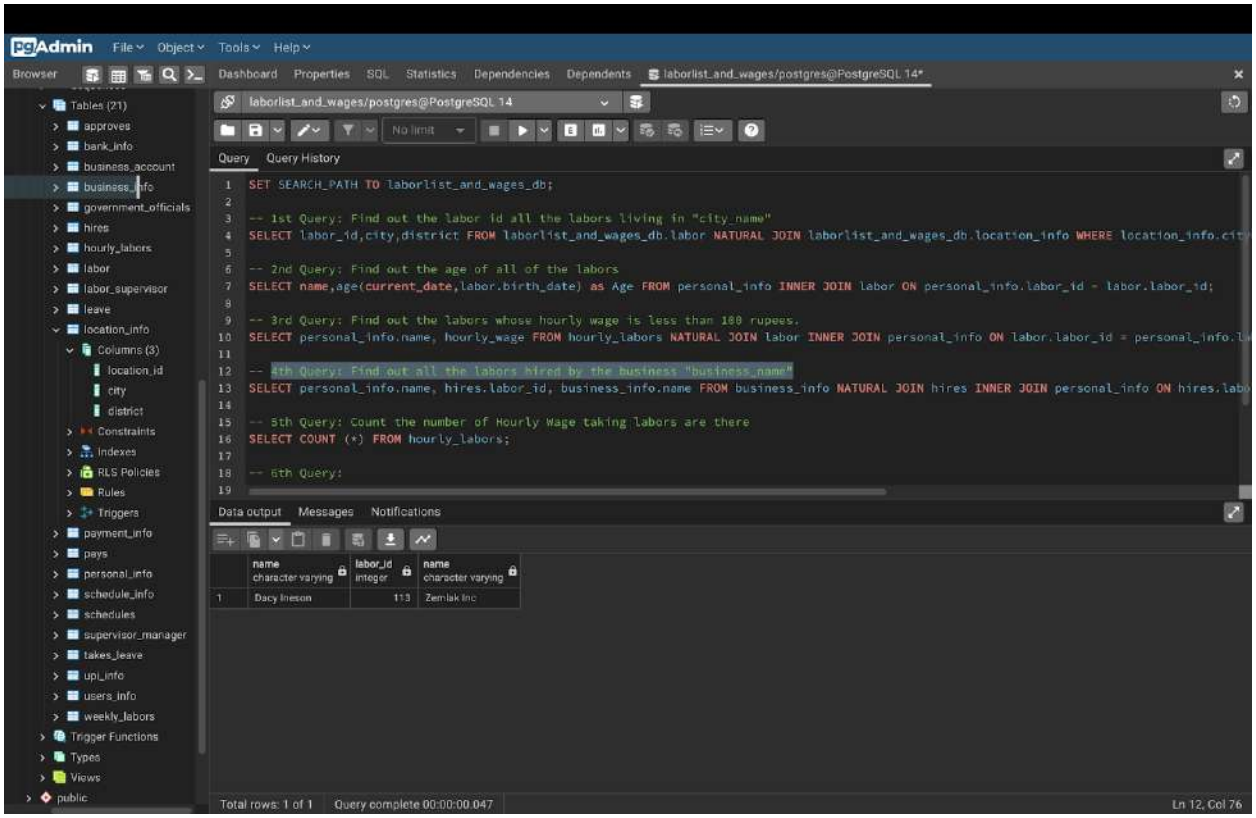
3rd Query: Find out the labors whose hourly wage is less than 100 rupees.

```
SELECT name,age(current_date,labor.birth_date) as Age FROM personal_info INNER
JOIN labor ON personal_info.labor_id = labor.labor_id;
```



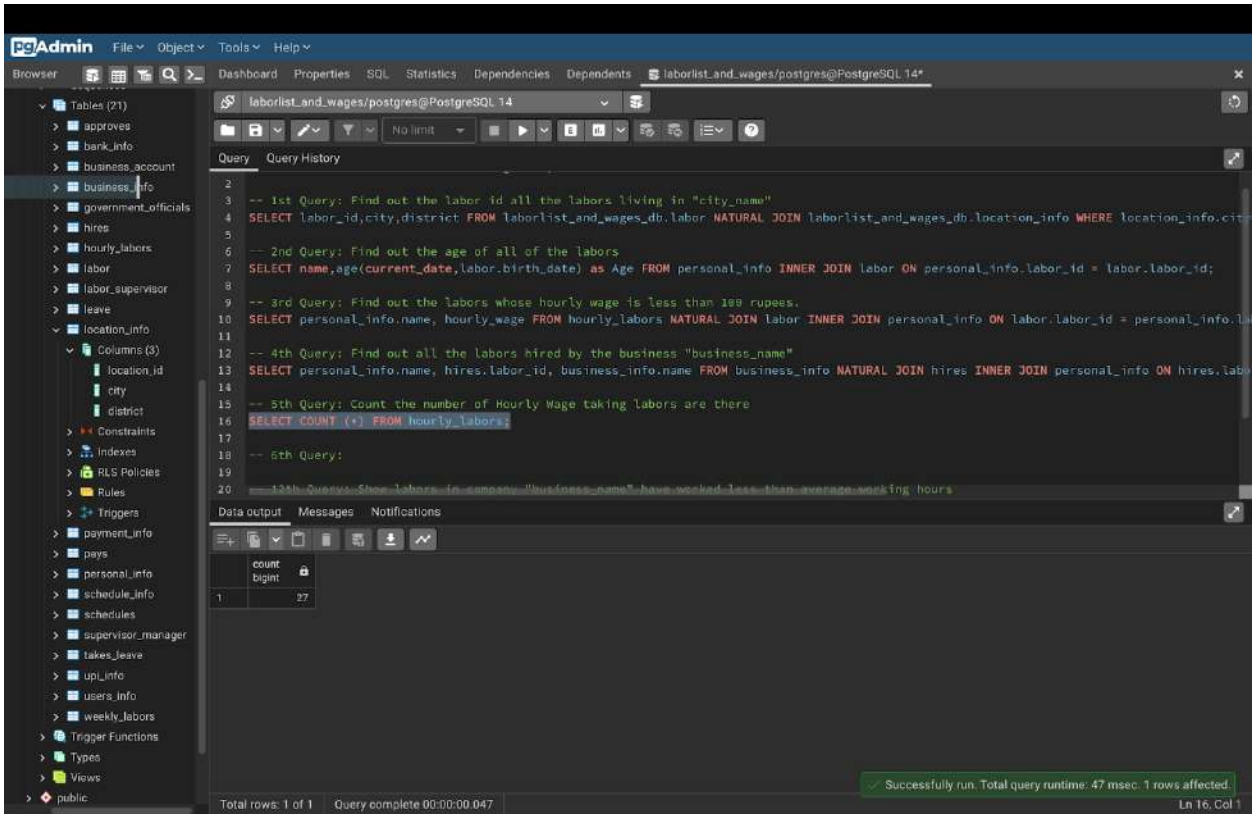
4th Query: Find out all the labors hired by the business "business_name"

```
SELECT personal_info.name, hires.labor_id, business_info.name FROM
business_info NATURAL JOIN hires INNER JOIN personal_info ON hires.labor_id =
personal_info.labor_id WHERE (business_info.name = 'Zemlak Inc');
```



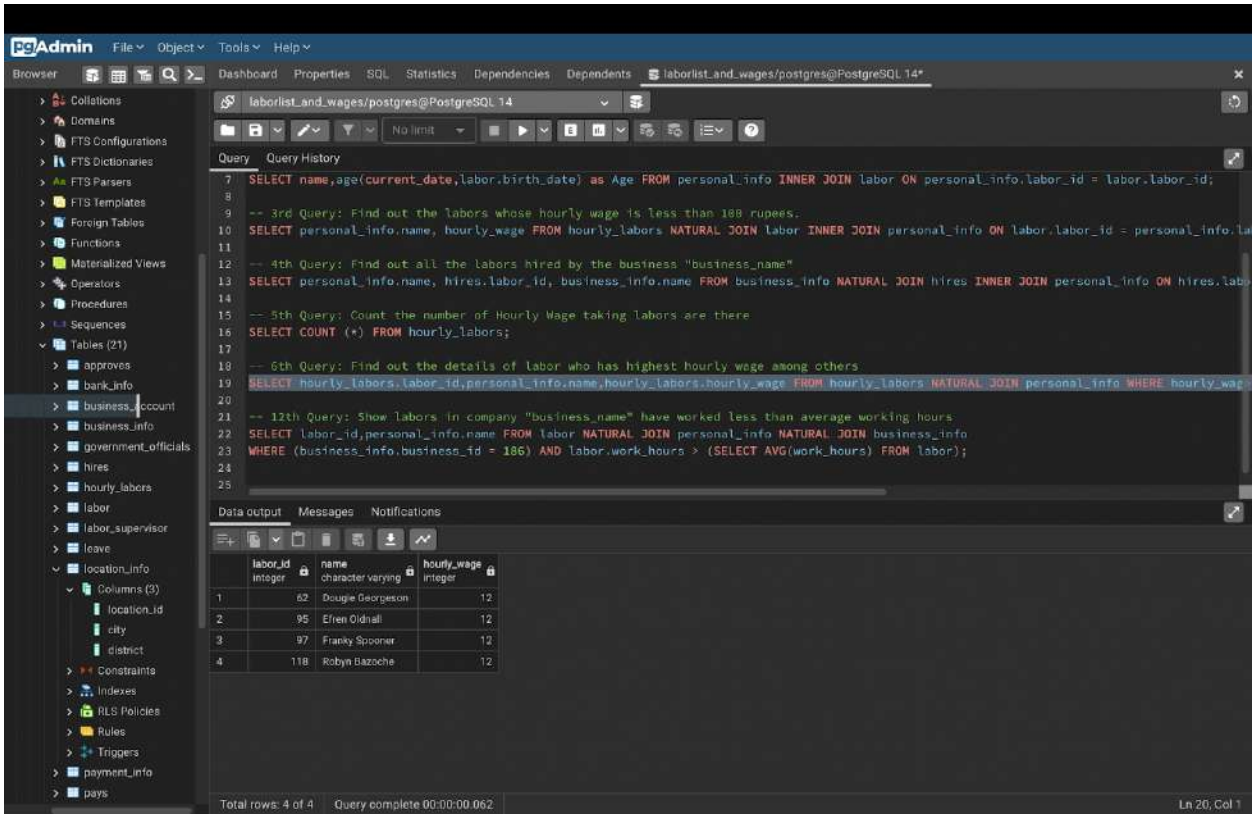
5th Query: Count the number of Hourly Wage taking labors are there

```
SELECT COUNT (*) FROM hourly_labors;
```



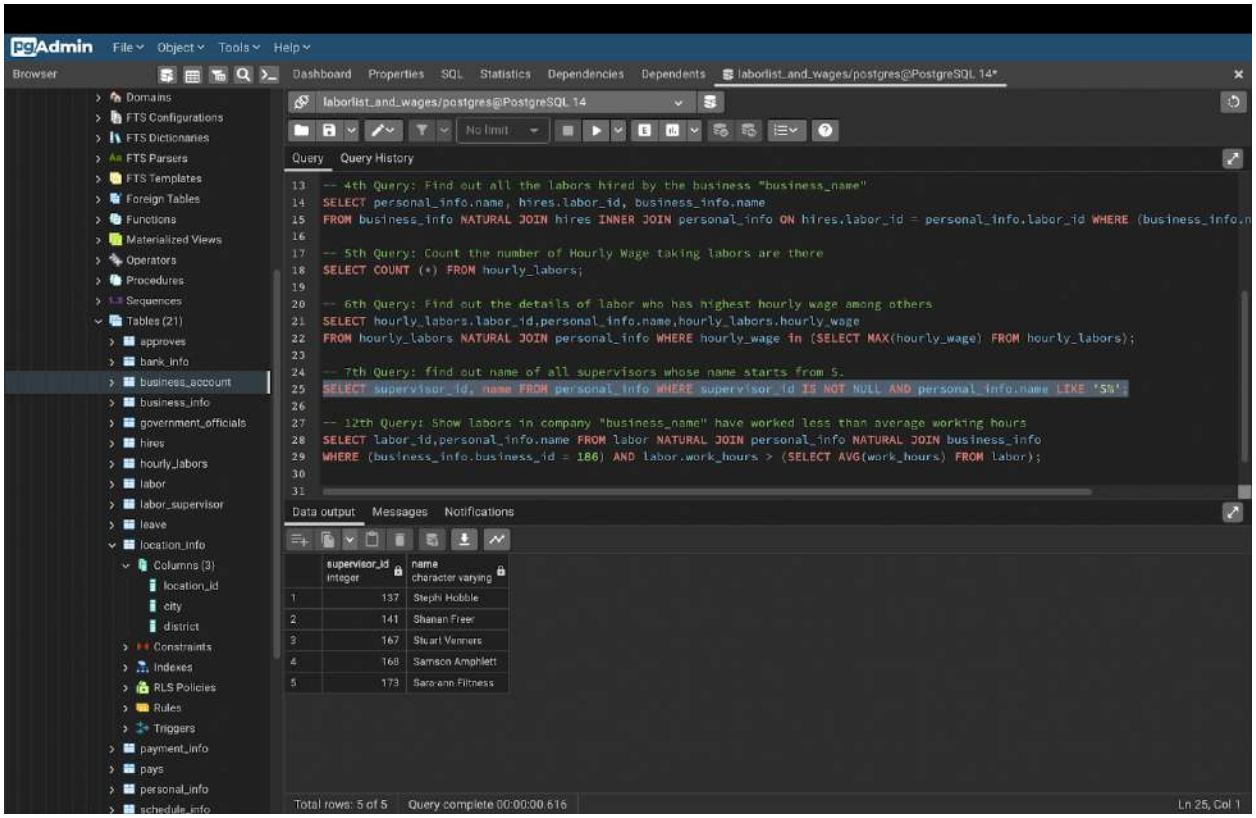
6th Query: Find out the details of labor who has highest hourly wage among others

```
SELECT hourly_labors.labor_id,personal_info.name,hourly_labors.hourly_wage
FROM hourly_labors NATURAL JOIN personal_info WHERE hourly_wage in (SELECT
MAX(hourly_wage) FROM hourly_labors);
```



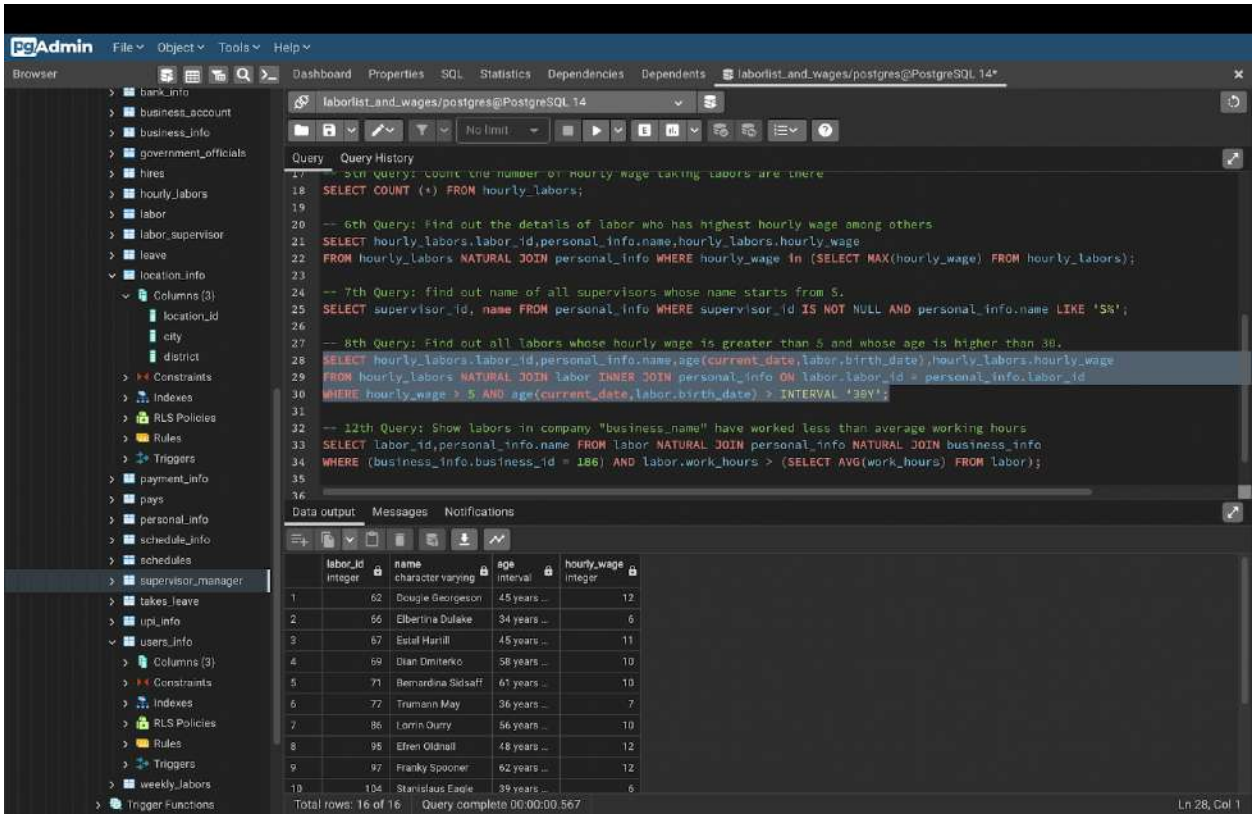
7th Query: find out the name of all supervisors whose name starts from S.

```
SELECT supervisor_id, name FROM personal_info WHERE supervisor_id IS NOT NULL
AND personal_info.name LIKE 'S%';
```



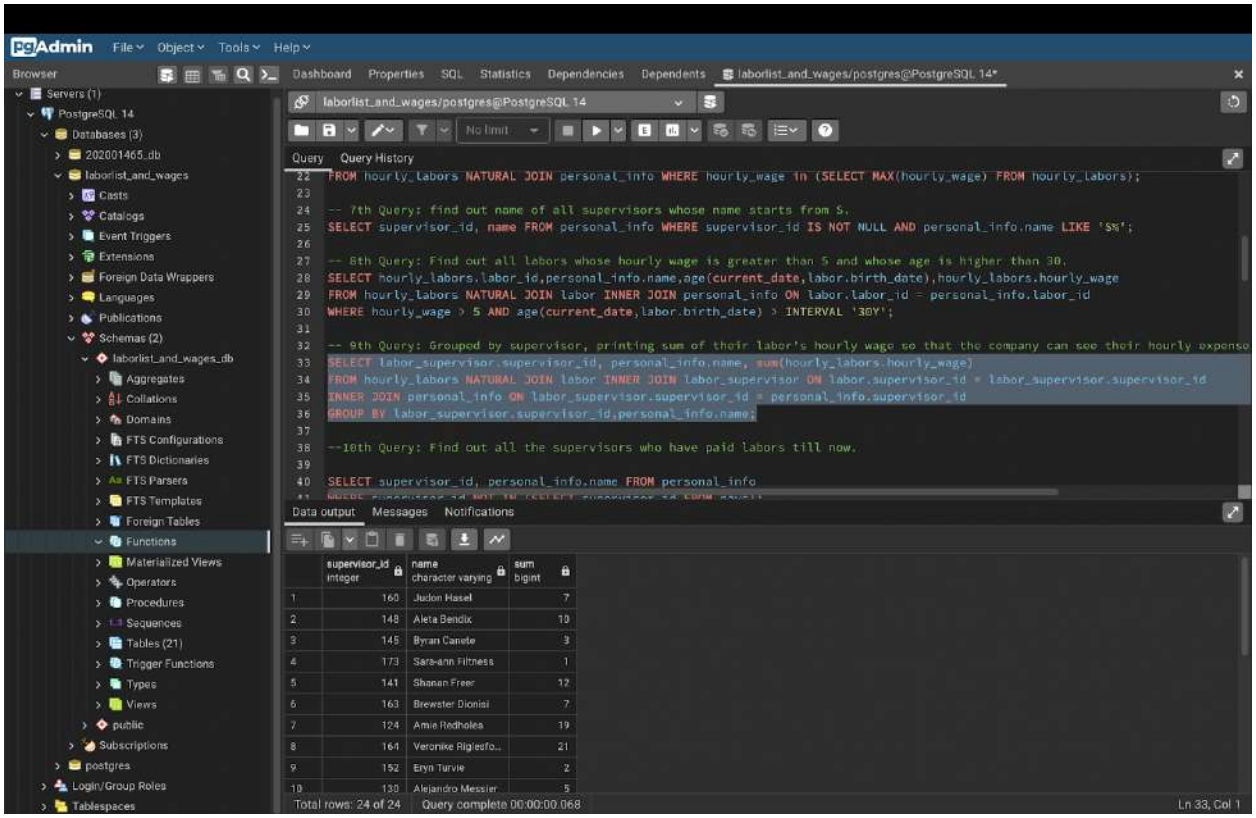
8th Query: Find out all laborers whose hourly wage is greater than 5 and whose age is higher than 30.

```
SELECT
hourly_labors.labor_id,personal_info.name,age(current_date,labor.birth_date),h
ourly_labors.hourly_wage
FROM hourly_labors NATURAL JOIN labor INNER JOIN personal_info ON
labor.labor_id = personal_info.labor_id
WHERE hourly_wage > 5 AND age(current_date,labor.birth_date) > INTERVAL '30Y';
```



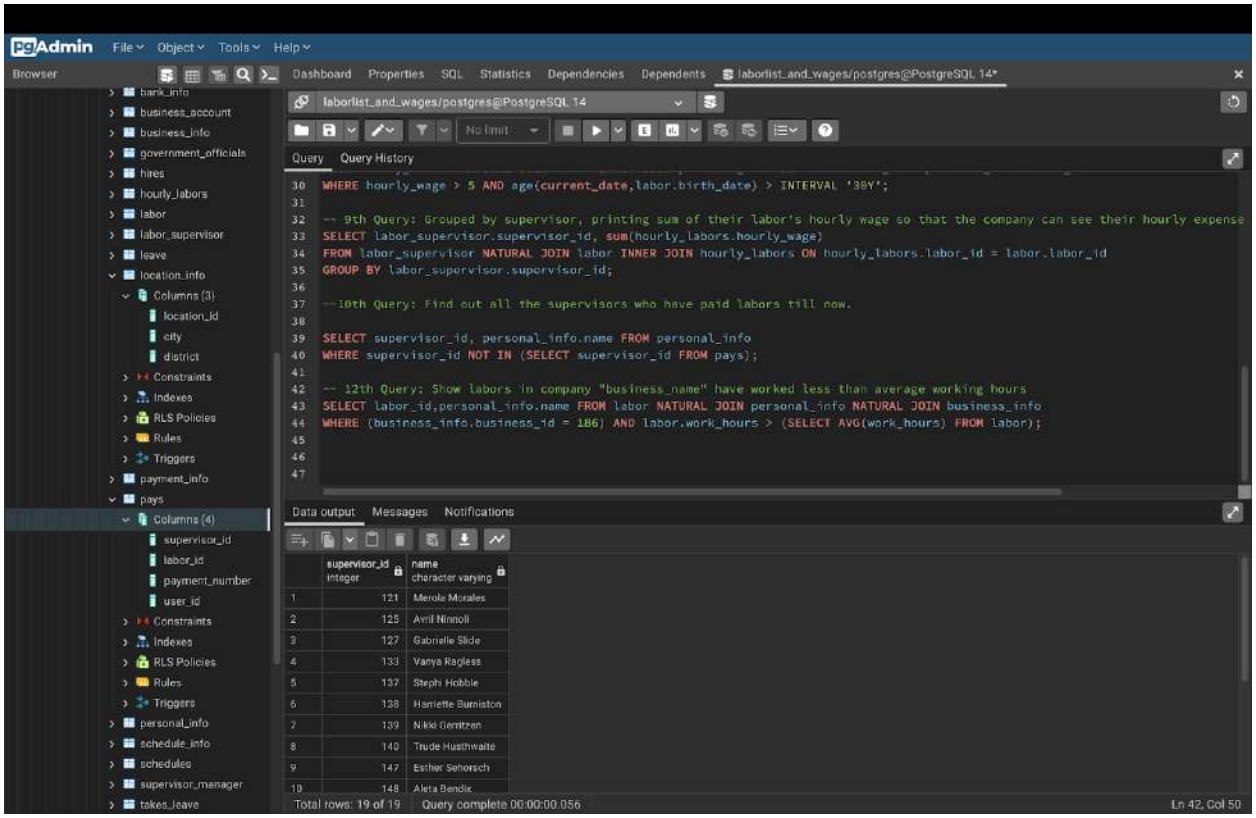
9th Query: Grouped by supervisor, printing sum of their labor's hourly wage so that the company can see their hourly expense on laborers per supervisor.

```
SELECT labor_supervisor.supervisor_id, personal_info.name,
sum(hourly_labors.hourly_wage)
FROM hourly_labors NATURAL JOIN labor INNER JOIN labor_supervisor ON
labor.supervisor_id = labor_supervisor.supervisor_id
INNER JOIN personal_info ON labor_supervisor.supervisor_id =
personal_info.supervisor_id
GROUP BY labor_supervisor.supervisor_id,personal_info.name;
```



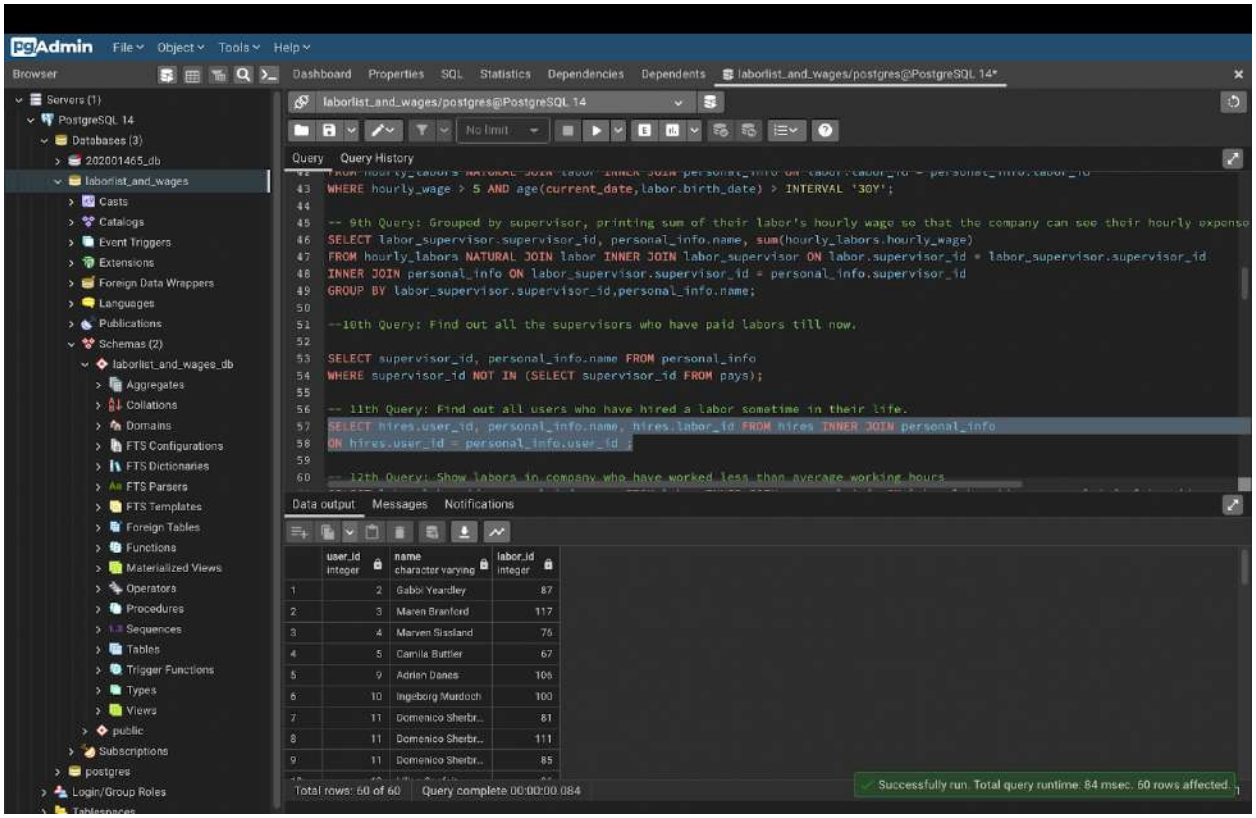
10th Query: Find out all the supervisors who have paid laborers till now.

```
SELECT supervisor_id, personal_info.name FROM personal_info
WHERE supervisor_id NOT IN (SELECT supervisor_id FROM pays);
```



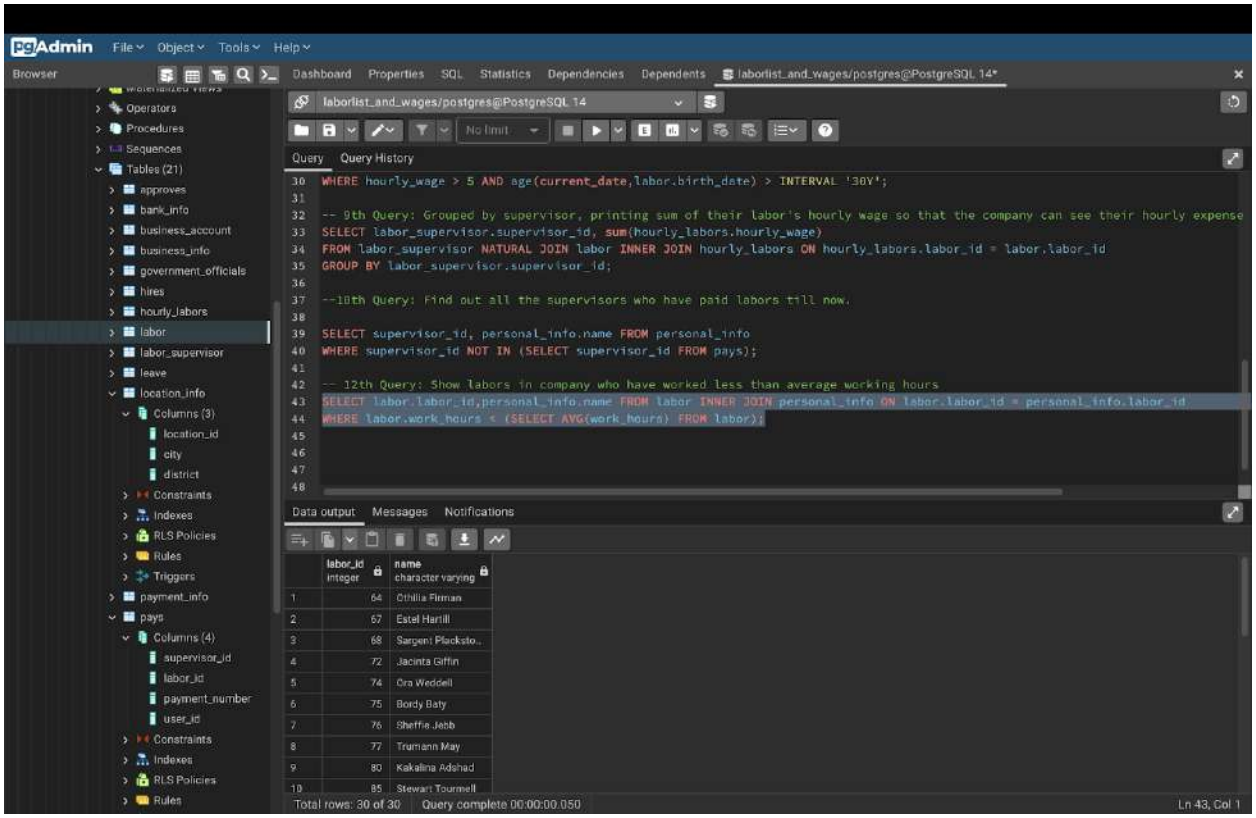
11th Query: Find out all users who have hired a labor sometime in their life.

```
SELECT hires.user_id, personal_info.name, hires.labor_id FROM hires INNER JOIN
personal_info
ON hires.user_id = personal_info.user_id ;
```



12th Query: Show labors in company who have worked less than average working hours

```
SELECT labor.labor_id,personal_info.name FROM labor INNER JOIN personal_info
ON labor.labor_id = personal_info.labor_id
WHERE labor.work_hours < (SELECT AVG(work_hours) FROM labor);
```



13th Query: Trigger to check and preventing inserting into the labor table if duplicate labor is found

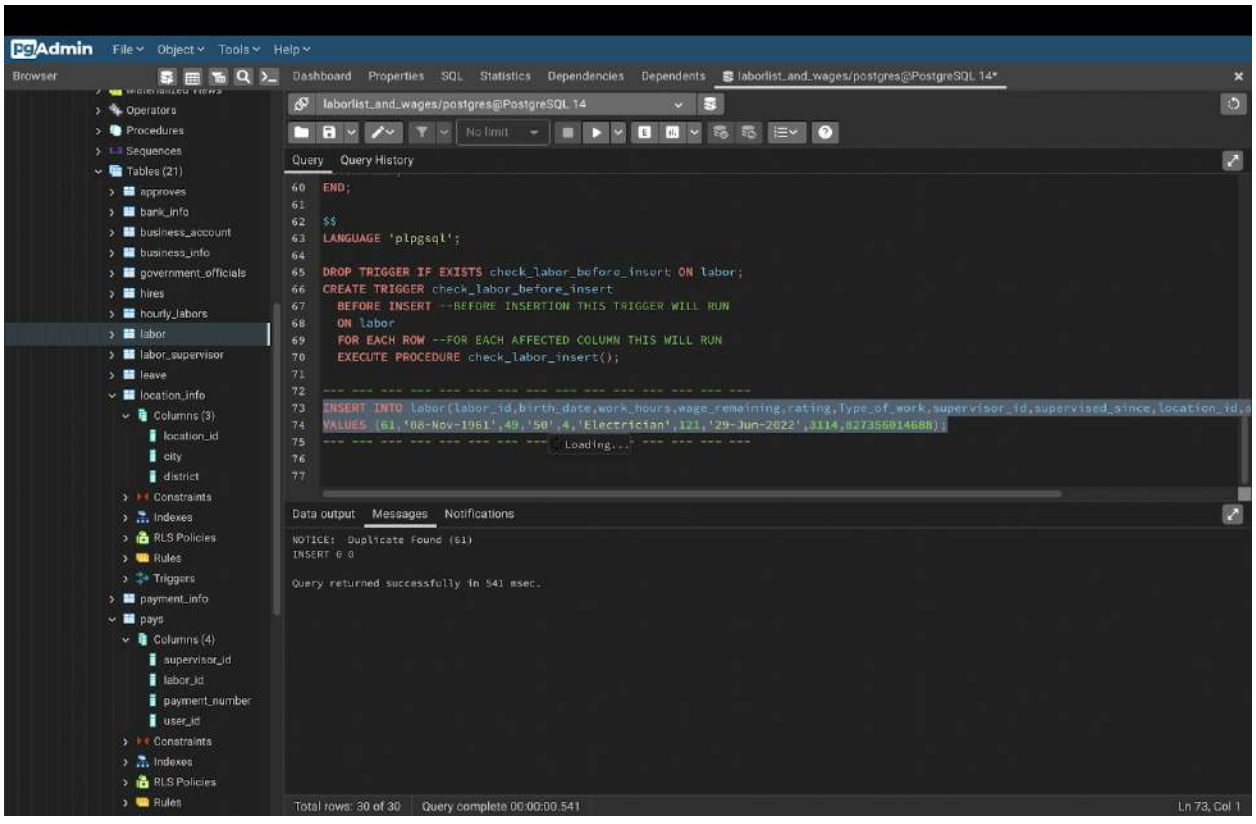
SQL Query:

```
CREATE OR REPLACE FUNCTION check_labor_insert()
  RETURNS trigger AS
$$
BEGIN
IF EXISTS(SELECT COUNT(*) FROM labor WHERE labor_id = NEW.labor_id) THEN
      RAISE NOTICE'Duplicate Found (%)', NEW.labor_id;
      RETURN OLD;
      --IF DUPLICATE WILL BE FOUND THEN IT WILL RAISE NOTICE AND THEN RETURN
OLD TABLE
END IF;
--ELSE BY DEFAULT IT WILL RETURN NEW TABLE
RETURN NEW;
END;

$$
```

```
LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS check_labor_before_insert ON labor;
CREATE TRIGGER check_labor_before_insert
  BEFORE INSERT --BEFORE INSERTION THIS TRIGGER WILL RUN
  ON labor
  FOR EACH ROW --FOR EACH AFFECTED COLUMN THIS WILL RUN
  EXECUTE PROCEDURE check_labor_insert();
```

Checked with the insert query:

```
INSERT INTO
labor(labor_id,birth_date,work_hours,wage_remaining,rating,Type_of_work,supervisor_id,supervised_since,location_id,account_number)
VALUES
(61,'08-Nov-1961',49,'50',4,'Electrician',121,'29-Jun-2022',3114,827356014688)
;
```



As we can see it prevented inserting these and duplicate labor id 61 was found.

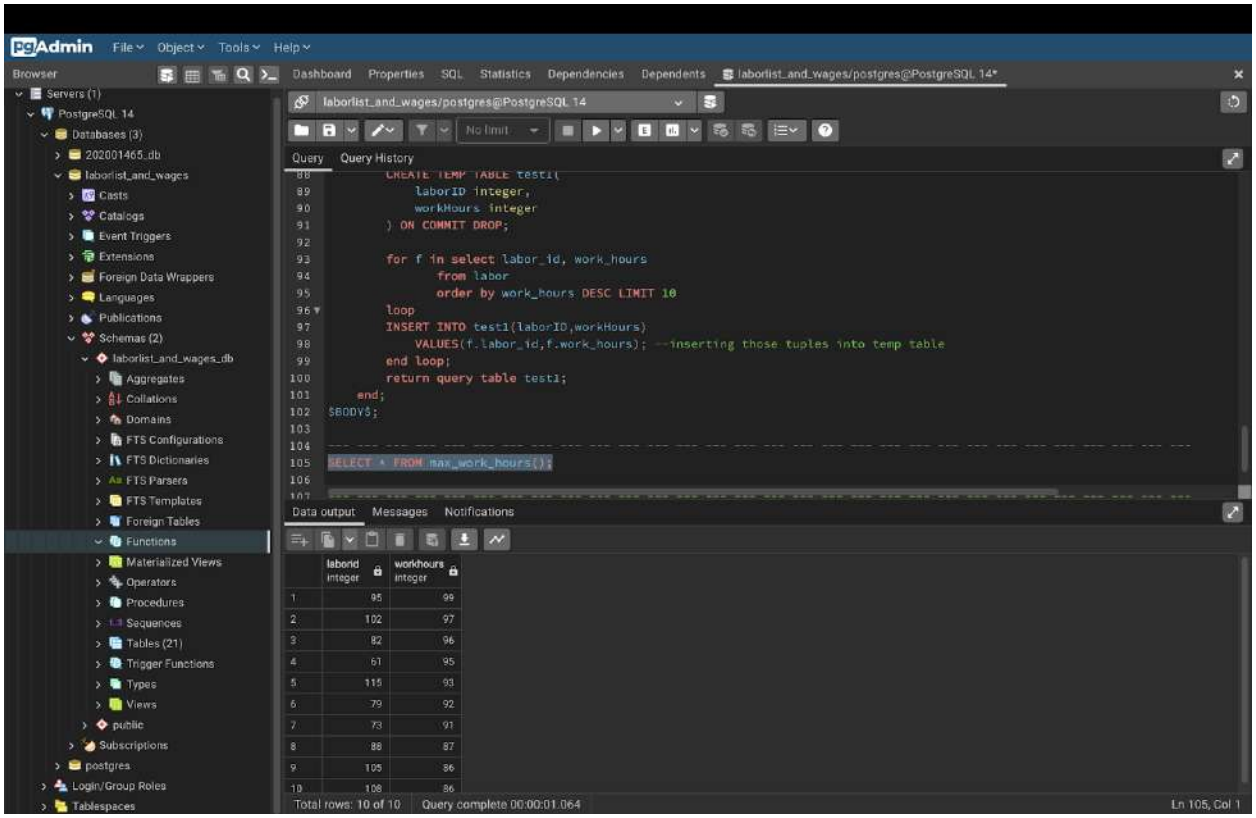
14th Query: Function to return 10 labors with maximum work hours per week

Function Query:

```
CREATE OR REPLACE FUNCTION max_work_hours()
RETURNS TABLE(
            laborID integer,
            workHours integer
      )
LANGUAGE 'plpgsql'
AS $BODY$
      declare f record;
      begin
            CREATE TEMP TABLE test1(
                  laborID integer,
                  workHours integer
            ) ON COMMIT DROP;

            for f in select labor_id, work_hours
                     from labor
                     order by work_hours DESC LIMIT 10
            loop
            INSERT INTO test1(laborID,workHours)
                  VALUES(f.labor_id,f.work_hours); --inserting those tuples
into temp table
            end loop;
            return query table test1;
      end;
$BODY$;
```

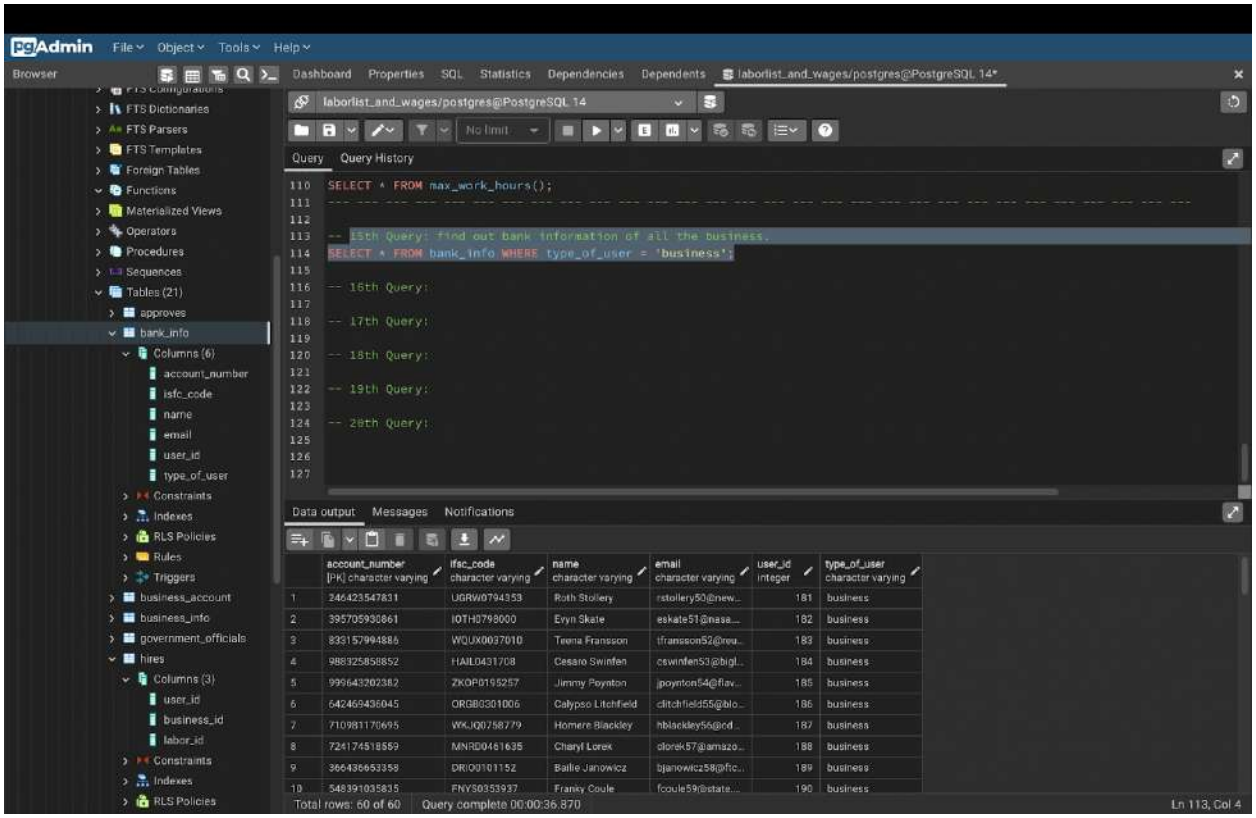
Checked with the query:

```
SELECT * FROM max_work_hours();
```



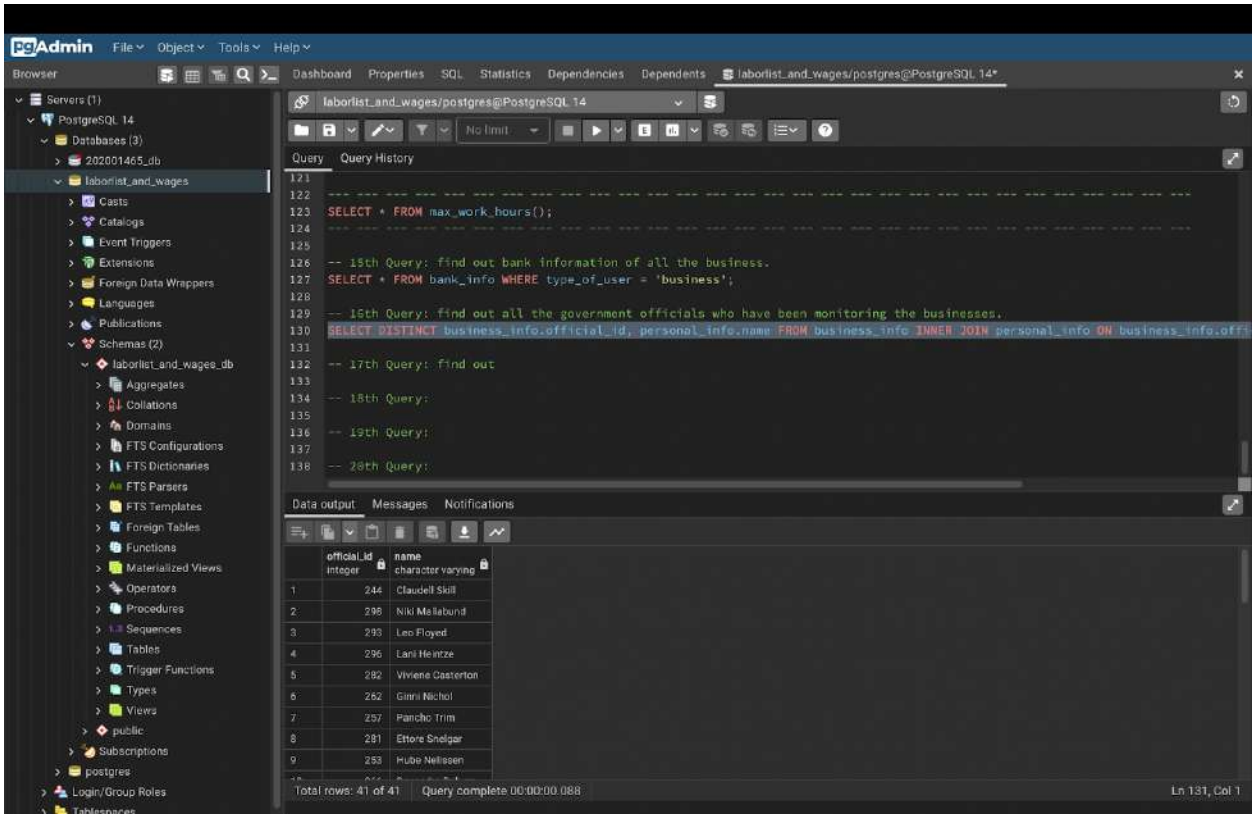
15th Query: find out bank information of all the business.

```
SELECT * FROM bank_info WHERE type_of_user = 'business';
```



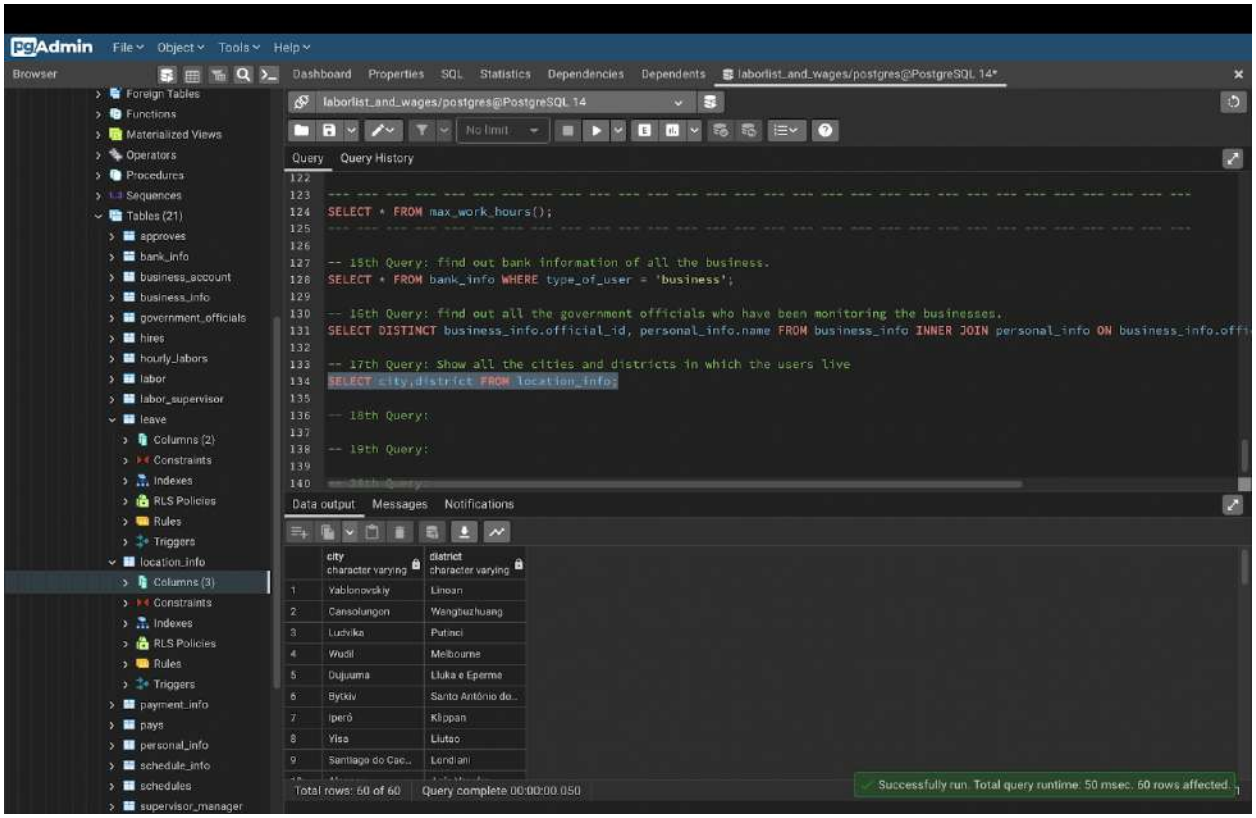
16th Query: find out all the government officials who have been monitoring the businesses.

```
SELECT DISTINCT business_info.official_id, personal_info.name FROM
business_info INNER JOIN personal_info ON business_info.official_id =
personal_info.official_id;
```



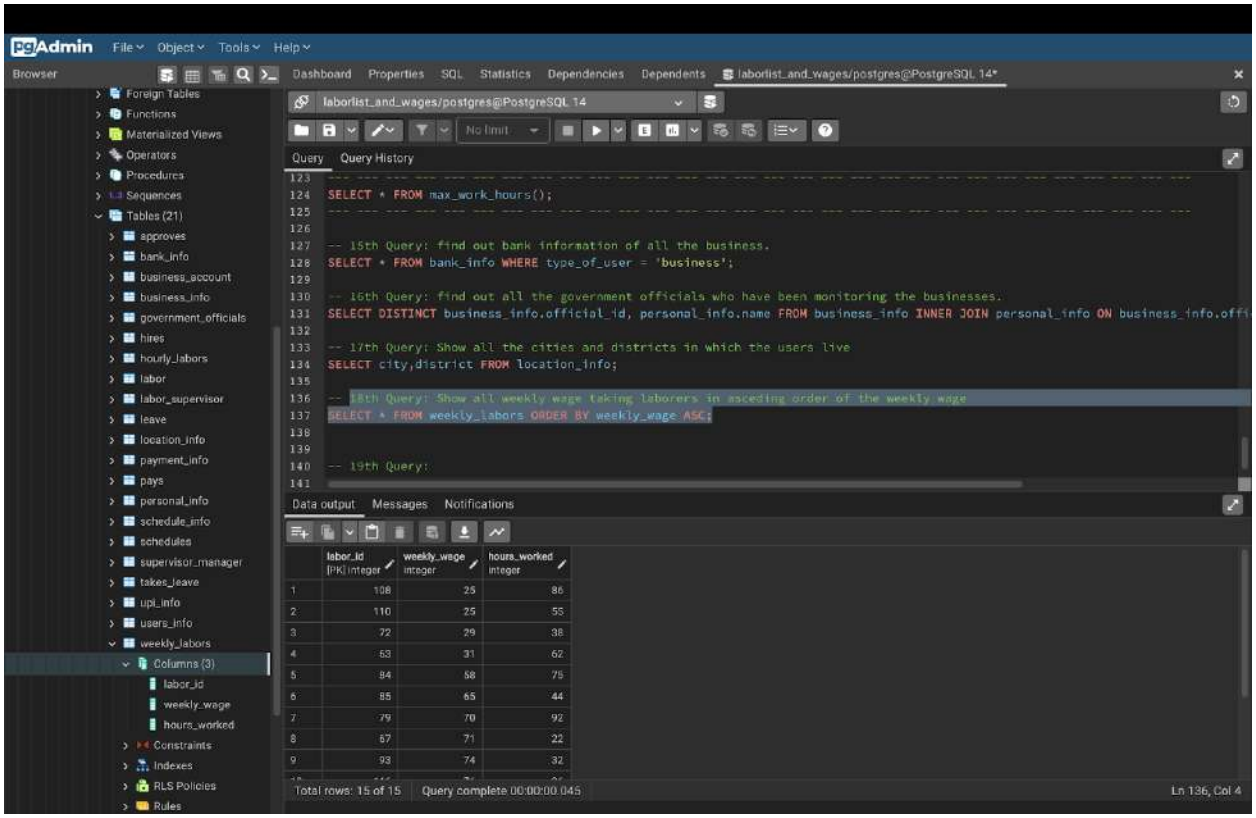
17th Query: Show all the cities and districts in which the users live

```
SELECT city,district FROM location_info;
```



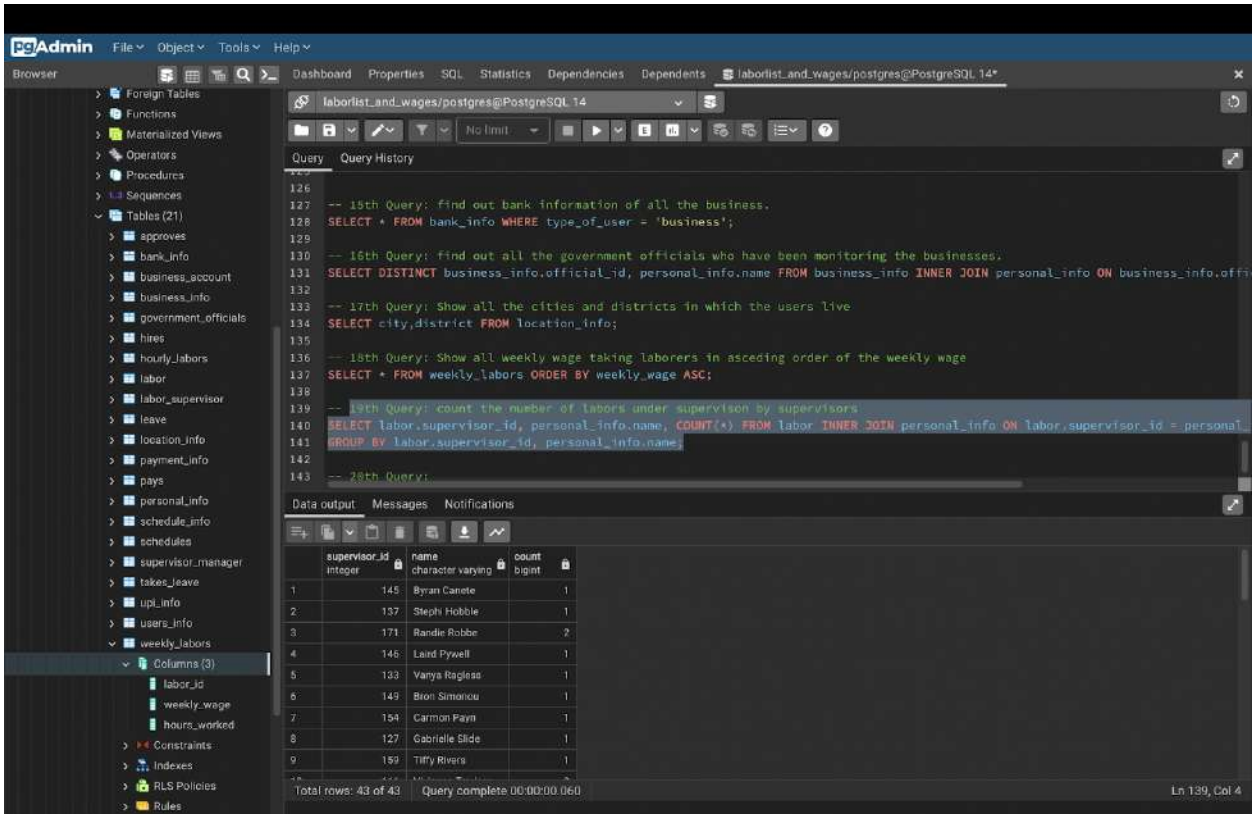
18th Query: Show all weekly wage taking laborers in asceding order of the weekly wage.

```
SELECT * FROM weekly_labors ORDER BY weekly_wage ASC;
```



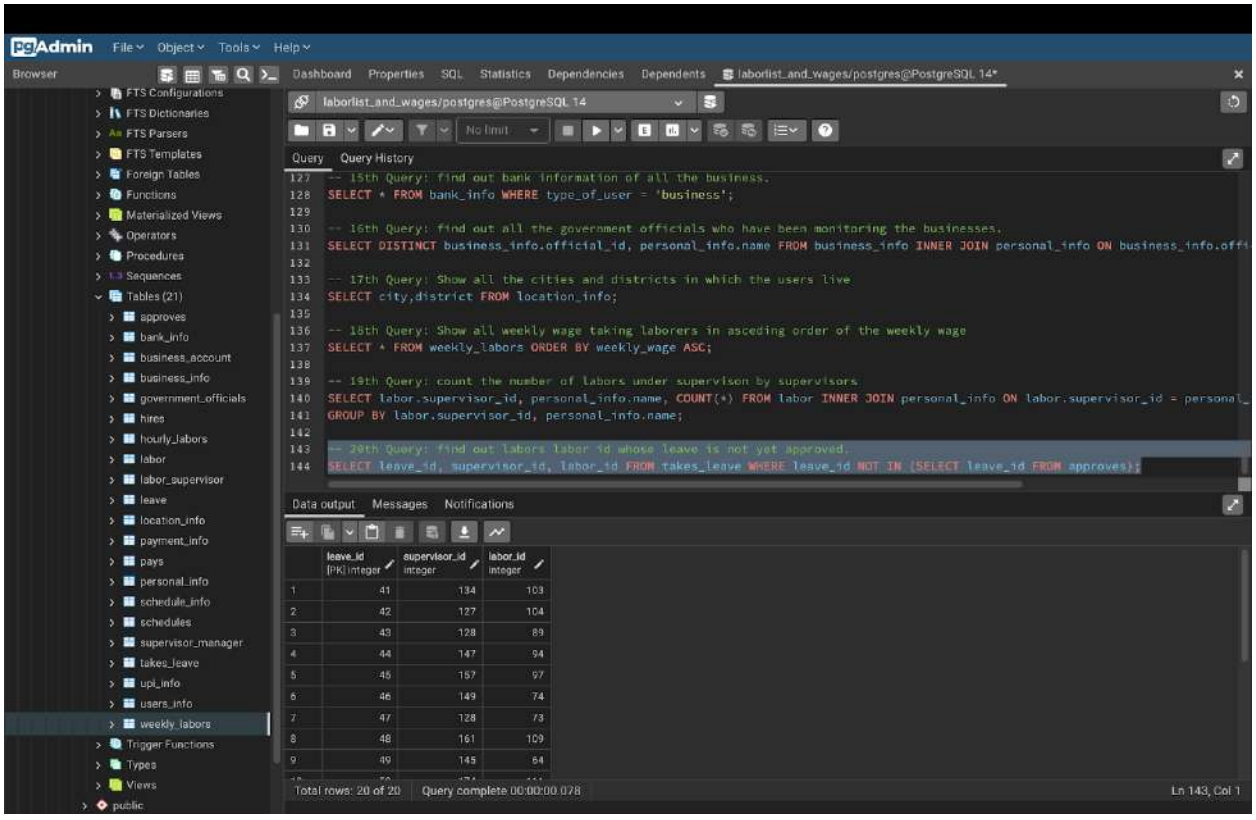
19th Query: count the number of laborers under supervision by supervisors.

```
SELECT labor.supervisor_id, personal_info.name, COUNT(*) FROM labor INNER JOIN
personal_info ON labor.supervisor_id = personal_info.supervisor_id
GROUP BY labor.supervisor_id, personal_info.name;
```



20th Query: -- 20th Query: find out labors labor id whose leave is not yet approved.

```
SELECT leave_id, supervisor_id, labor_id FROM takes_leave WHERE leave_id NOT IN (SELECT leave_id FROM approves);
```


pgAdmin
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
laborlist_and_wages/postgres@PostgreSQL 14*
laborlist_and_wages/postgres@PostgreSQL 14
No limit
Query
Query History
-- 15th Query: find out bank information of all the business.
SELECT * FROM bank_info WHERE type_of_user = 'business';
-- 16th Query: find out all the government officials who have been monitoring the businesses.
SELECT DISTINCT business_info.official_id, personal_info.name FROM business_info INNER JOIN personal_info ON business_info.offi
-- 17th Query: Show all the cities and districts in which the users live
SELECT city,district FROM location_info;
-- 18th Query: Show all weekly wage taking laborers in asceding order of the weekly wage
SELECT * FROM weekly_labors ORDER BY weekly_wage ASC;
-- 19th Query: count the number of labors under supervison by supervisors
SELECT labor.supervisor_id, personal_info.name, COUNT(*) FROM labor INNER JOIN personal_info ON labor.supervisor_id = personal_
GROUP BY labor.supervisor_id, personal_info.name;
-- 20th Query: find out labors labor id whose leave is not yet aproved.
SELECT leave_id, supervisor_id, labor_id FROM takes_leave WHERE leave_id NOT IN (SELECT leave_id FROM approves);
Data output
Messages
Notifications
leave_id [PK] integer
supervisor_id integer
labor_id integer
1 41 134 103
2 42 127 104
3 43 128 89
4 44 147 94
5 45 157 97
6 46 149 74
7 47 128 73
8 48 161 109
9 49 145 64
Total rows: 20 of 20
Query complete 00:00:00.078
Ln 143, Col 1
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (21)
approves
bank_info
business_account
business_info
government_officials
hires
hourly_labors
labor
labor_supervisor
leave
location_info
payment_info
pays
personal_info
schedule_info
schedules
supervisor_manager
takes_leave
upi_info
users_info
weekly_labors
Trigger Functions
Types
Views
public

IT214 - DBMS
(Front-end Development)

Case Study Database: Laborlist and Wages Database

Prepared by
Team No. 17 | Group 6 | Section 10

Group Members
Dhruv Patel - 202001271
Jaykumar Navadiya - 202001465

Under the guidance of
Prof. Chhaya
Teaching Assistant - Vinay Maharaaj

Date
November 18, 2022

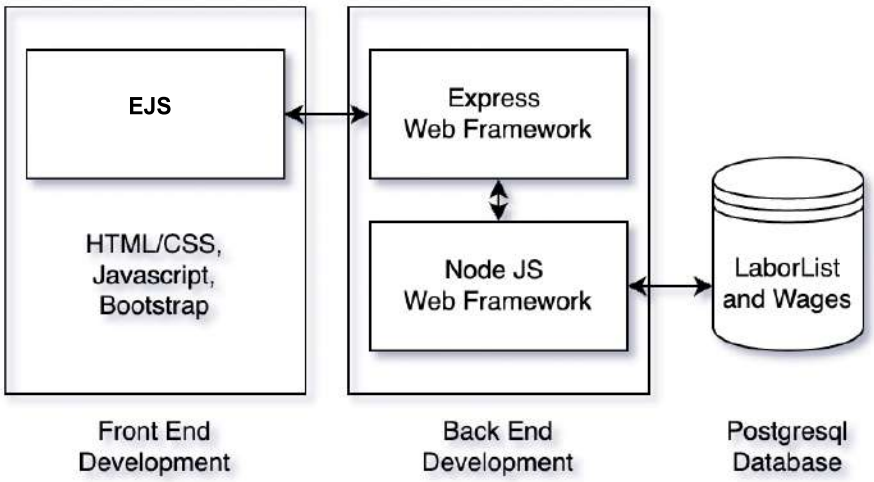
18. Website with CRUD functionalities and Other Features:

We have used Embedded Javascript (an alternative of HTML5 as we can write both HTML and Javascript code in one file itself in Embedded Javascript file, extension: .ejs)

For designing purposes we have used CSS and Javascript.

For the backend of the website, we have used Node.js and Express Module which are written in Javascript as well.

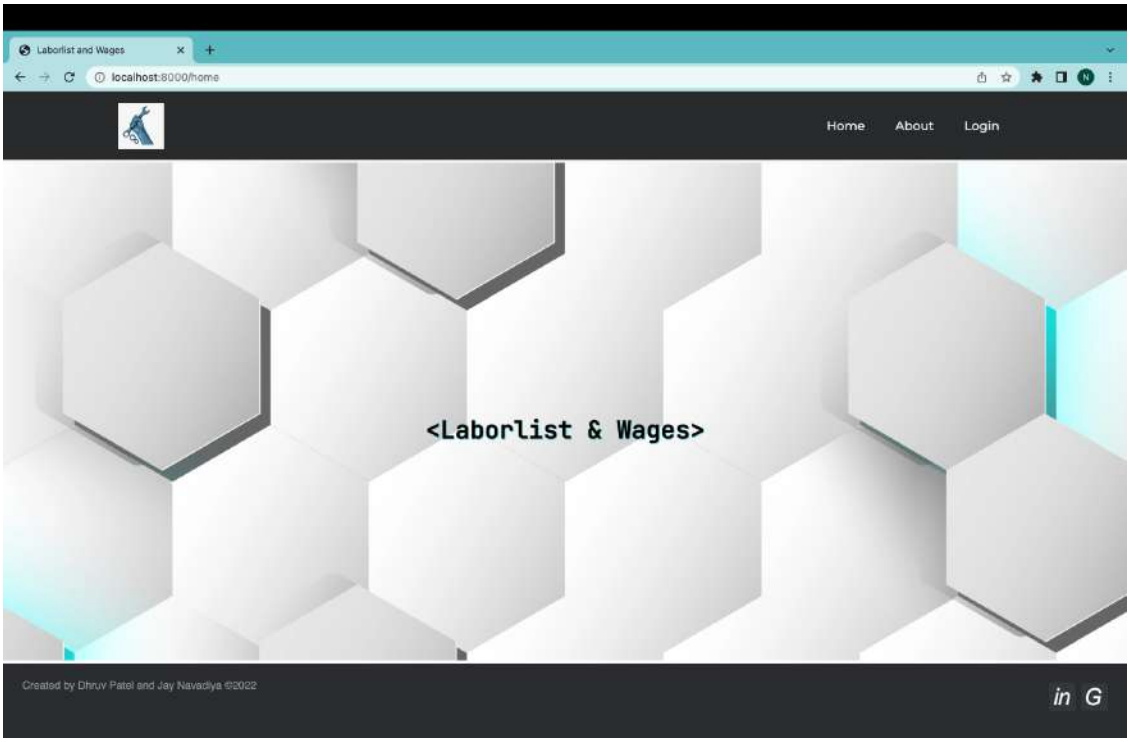
Below is the diagram explaining the PEEN Stack we have used for this website:



All programming files (Website Code Files) are there in the website folder included in the zip file.

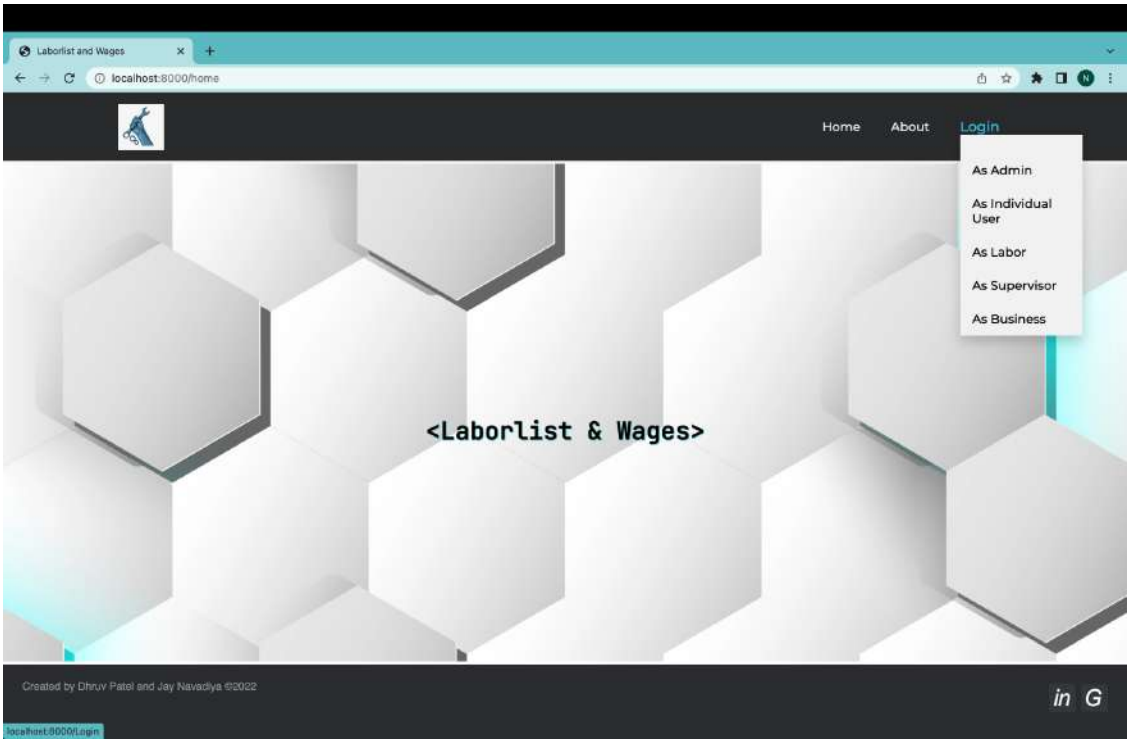
18.1 Functionalities:

1. Homepage



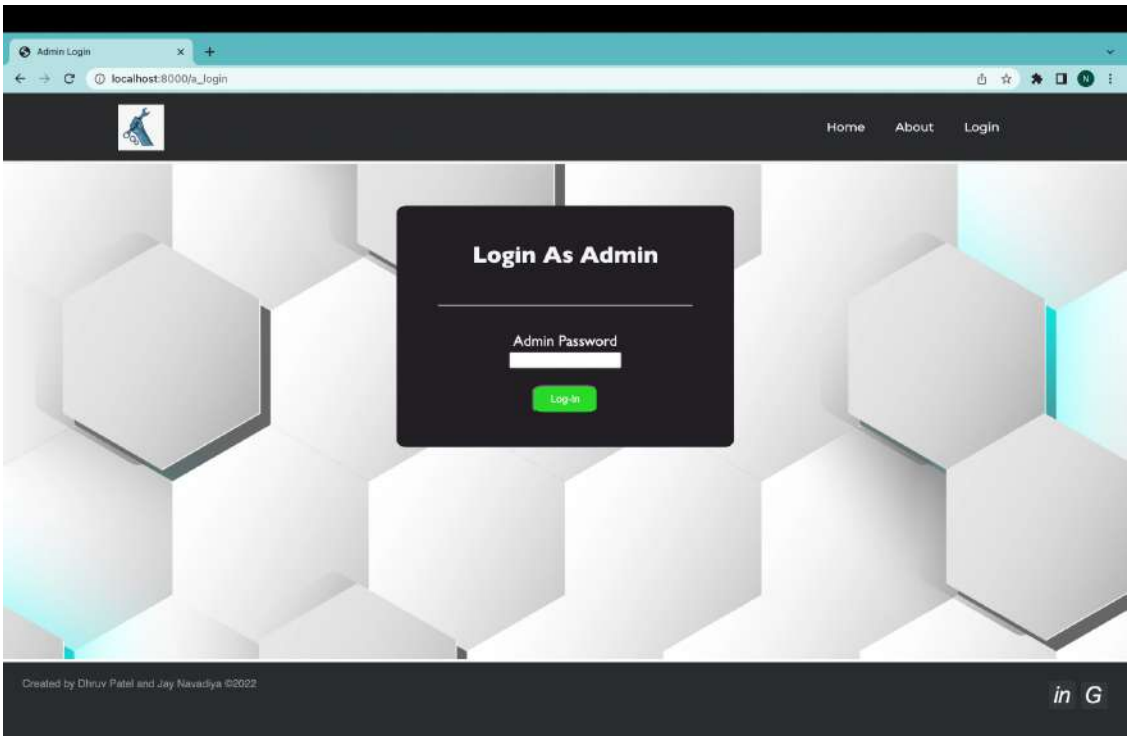
There is an animation in the Laborlist & Wages text which can be explored in detail with the code given.

2. Login Function



This opens a hidden dropdown menu which contains various types of users that our database supports.

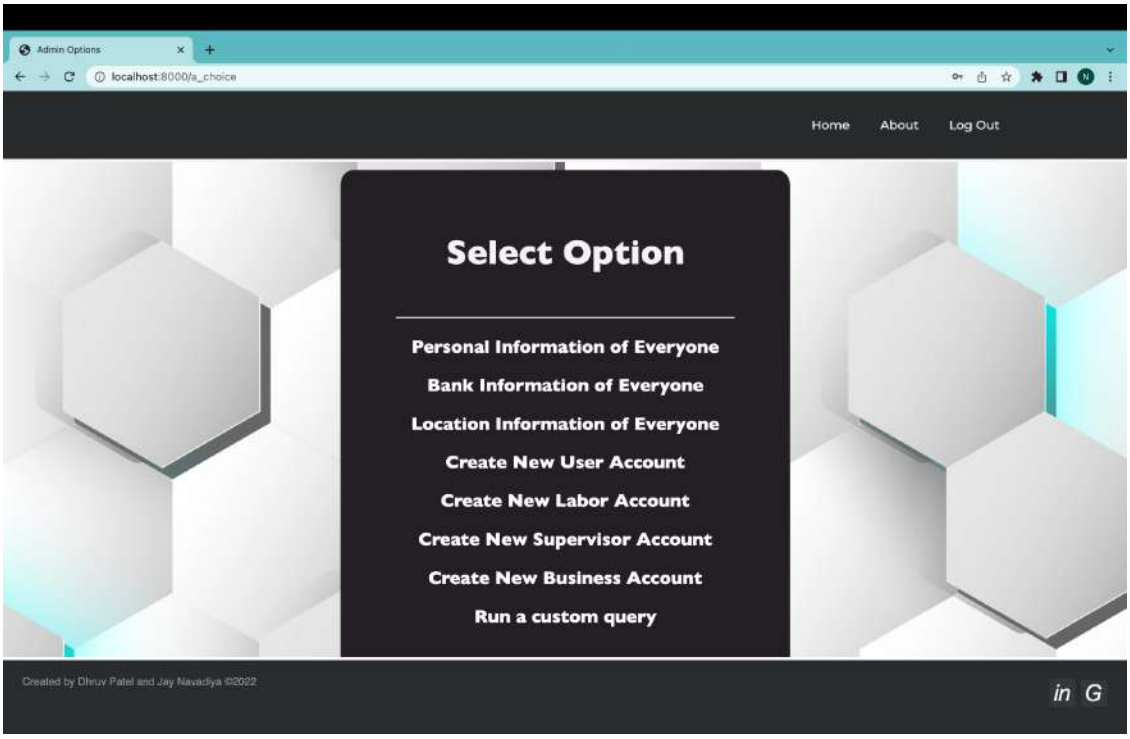
3. Admin Login



This only contains a password field. The password is admin.

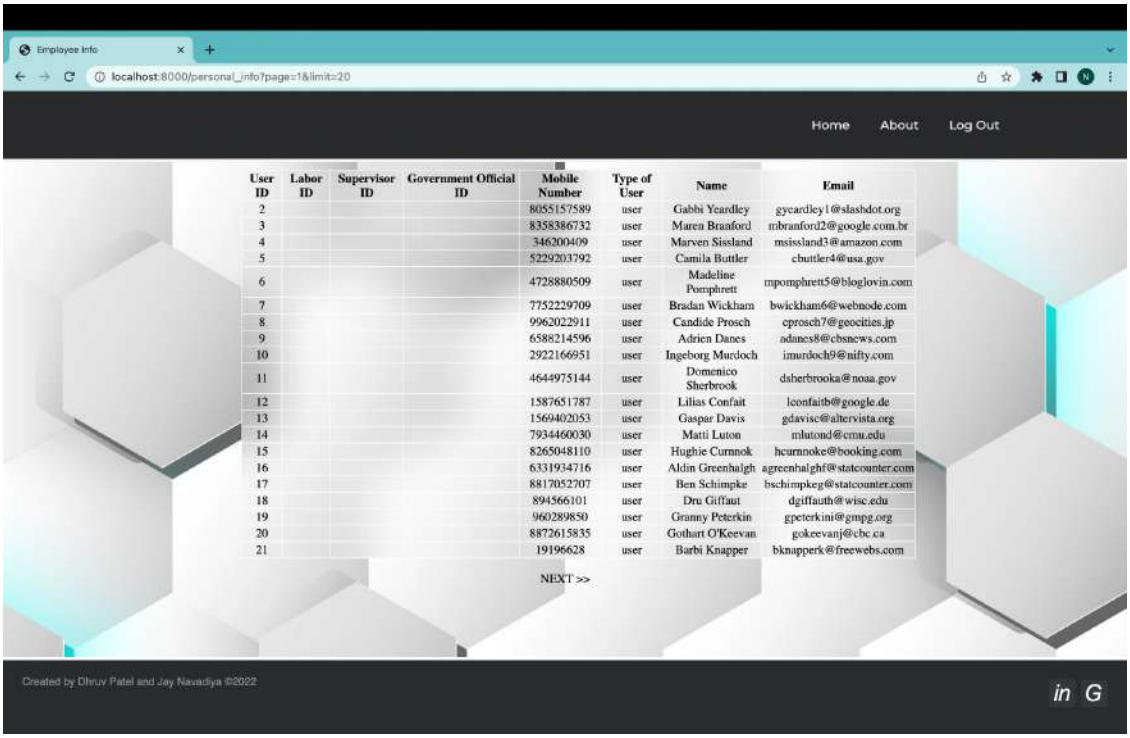
After logging into the Admin, there are various options to choose from for the admin.

4. Admin Choices



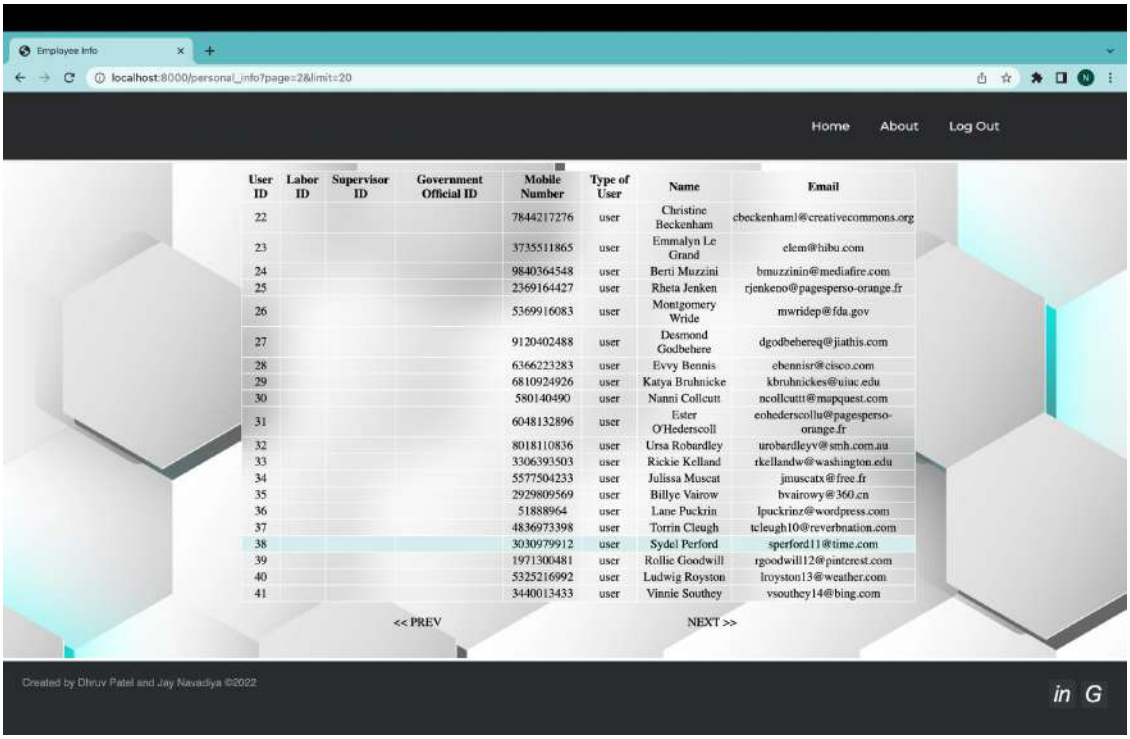
5. Option 1: Personal Information of Everyone

It shows the table of personal_info of our database with pages and prev and next options.



User ID	Labor ID	Supervisor ID	Government Official ID	Mobile Number	Type of User	Name	Email
2				8055157589	user	Gabbi Yeardley	gyeardley1@slashdot.org
3				8358386732	user	Maren Branford	mbranford2@google.com.br
4				346200409	user	Marven Sissland	msissland3@amazon.com
5				5229203792	user	Camila Buttler	cbuttler4@usa.gov
6				4728880509	user	Madeline Pomphrett	mpomphrett5@bloglovin.com
7				7752229709	user	Bradan Wickham	bwickham6@webnode.com
8				9962022911	user	Candide Prosch	cprosch7@geocities.jp
9				6588214596	user	Adrien Danes	adanes8@cbsnews.com
10				2922166951	user	Ingeborg Murdoch	imurdoch9@nifty.com
11				4644975144	user	Domenico Sherbrook	dsherbrooka@noaa.gov
12				1587651787	user	Lilias Confait	lconfaitb@google.de
13				1569402053	user	Gaspar Davis	gdavisc@altervista.org
14				7934460030	user	Matti Luton	mlutond@cmu.edu
15				8265048110	user	Hughie Curnnok	hcurnnoke@booking.com
16				6331934716	user	Aldin Greenhalgh	agreenhalghf@statcounter.com
17				8817052707	user	Ben Schimpke	bschimpkeg@statcounter.com
18				894566101	user	Dru Giffaut	dgiffauth@wisc.edu
19				960289850	user	Granny Peterkin	gpeterkini@gmpg.org
20				8872615835	user	Gothart O'Keevan	gokeevanj@cbc.ca
21				19196628	user	Barbi Knapper	bknapperk@freewebs.com

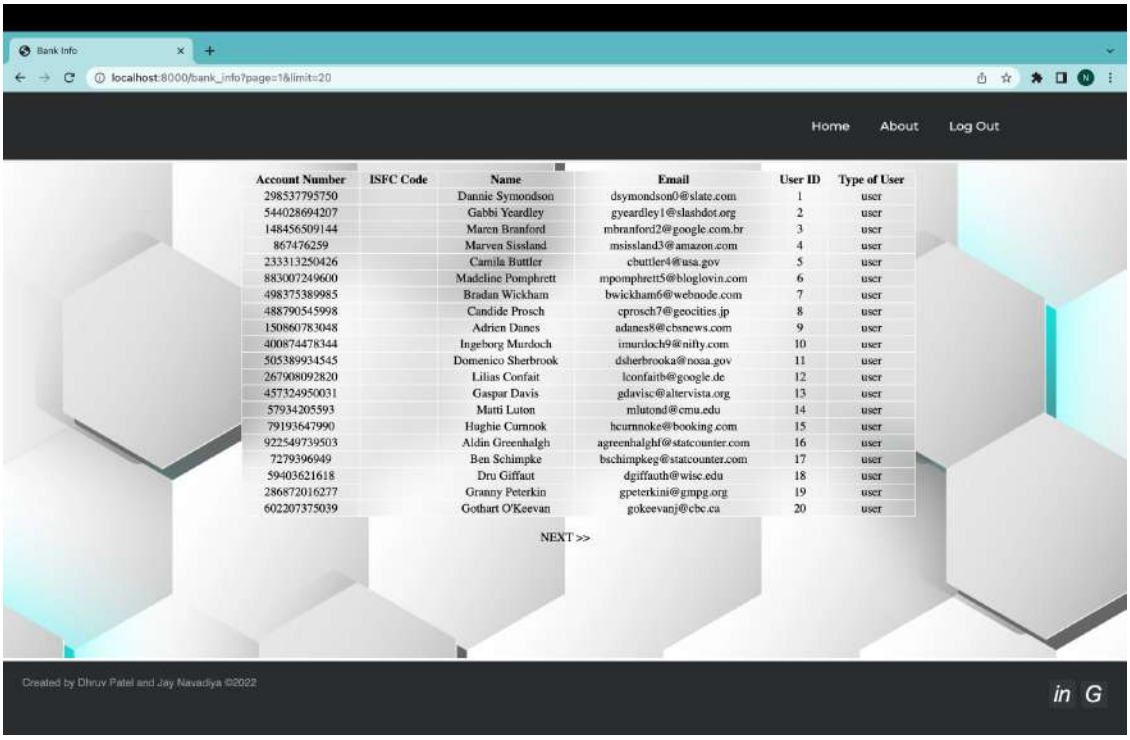
NEXT >>



User ID	Labor ID	Supervisor ID	Government Official ID	Mobile Number	Type of User	Name	Email
22				7844217276	user	Christine Beckenham	cbeckenham1@creativecommons.org
23				3735511865	user	Emmalyn Le Grand	elem@hibu.com
24				9840364548	user	Berti Muzzini	bmuzzinin@mediafire.com
25				2369164427	user	Rheta Jenken	rjenkeno@pagesperso-orange.fr
26				5369916083	user	Montgomery Wride	mwridep@fda.gov
27				9120402488	user	Desmond Godbehere	dgodbehereq@jiathis.com
28				6366223283	user	Evvy Bennis	ebennisr@cisco.com
29				6810924926	user	Katya Bruhnicke	kbruhnickes@uiuc.edu
30				580140490	user	Nanni Collcutt	ncollcuttt@mapquest.com
31				6048132896	user	Ester O'Hederscoll	eohederscollu@pagesperso-orange.fr
32				8018110836	user	Ursa Robardley	urobardleyv@smh.com.au
33				3306393503	user	Rickie Kelland	rkellandw@washington.edu
34				5577504233	user	Julissa Muscat	jmuscatx@free.fr
35				2929809569	user	Billye Vairow	bvairowy@360.cn
36				51888964	user	Lane Puckrin	lpuckrinz@wordpress.com
37				4836973398	user	Torrin Cleugh	tcleugh10@reverbnation.com
38				3030979912	user	Sydel Perford	sperford11@time.com
39				1971300481	user	Rollie Goodwill	rgoodwill12@pinterest.com
40				5325216992	user	Ludwig Royston	lroyston13@weather.com
41				3440013433	user	Vinnie Southey	vsouthey14@bing.com

<< PREV NEXT >>

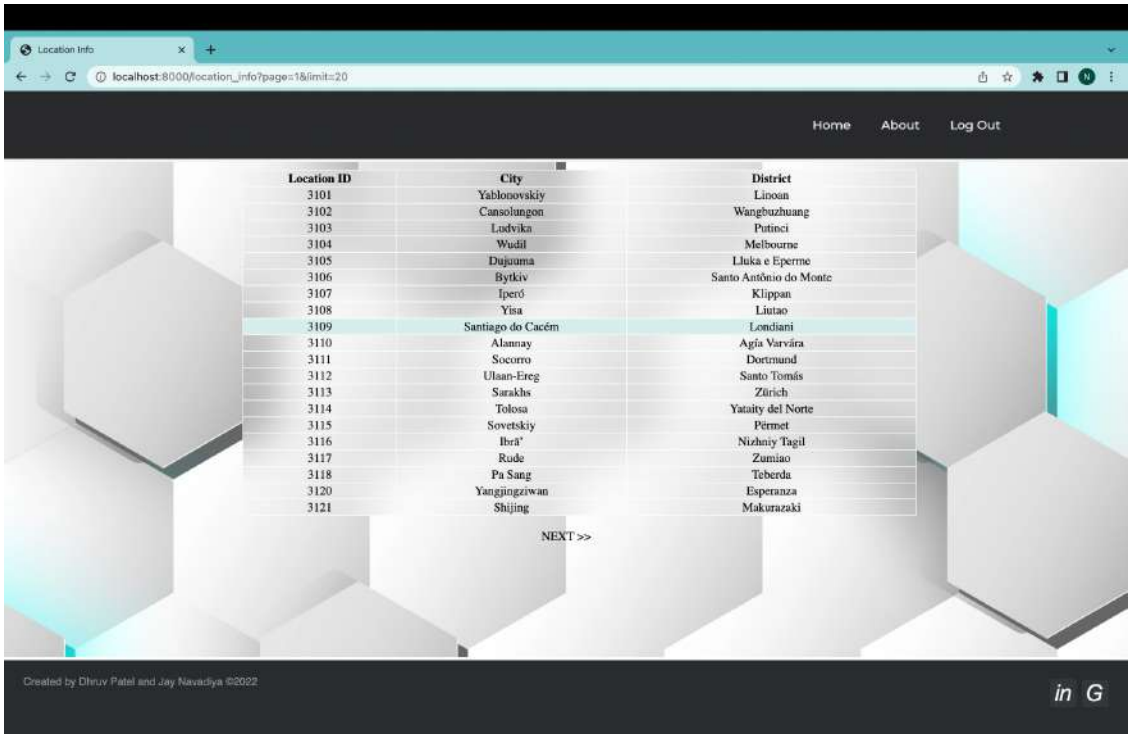
6. Option 2: Shows the bank information for every account in various pages as before.



7. Option 3: Shows the location table.

The pagination logic is given in the code and 20 entries will be shown at once. Also, by using the blur filter we made the table information more readable.

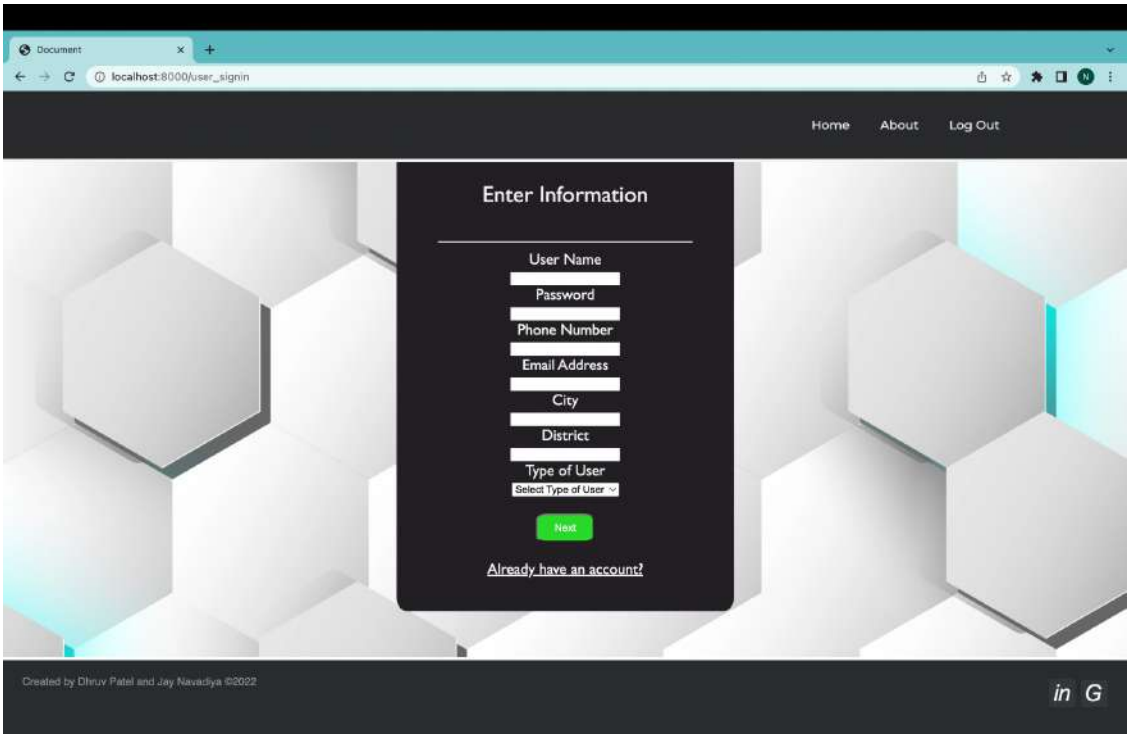
Hovering is also implemented in the table as well. Have a look at the screenshot given below. The row turns blue while hovering over it.



8. Option 4: Create new user account

This option lands you to the /user_signin page. Similarly for other options given after, such as creating a new supervisor, business and labor account does the same.

After clicking, Create a new user account it lands you to the /user_signin page as shown below.

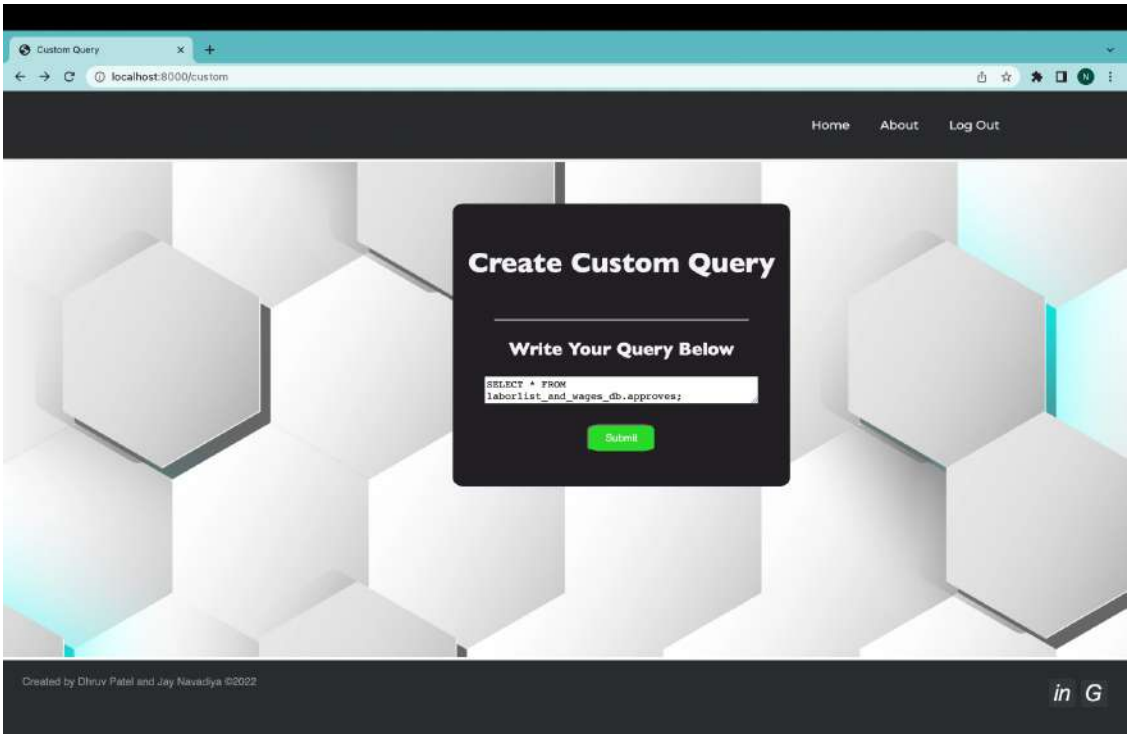


Here, if the user already has an account and the admin wants to access it, the Already have an account? option will land you on the login page and they can login into the account.

9. Option 5: Run a Custom Query

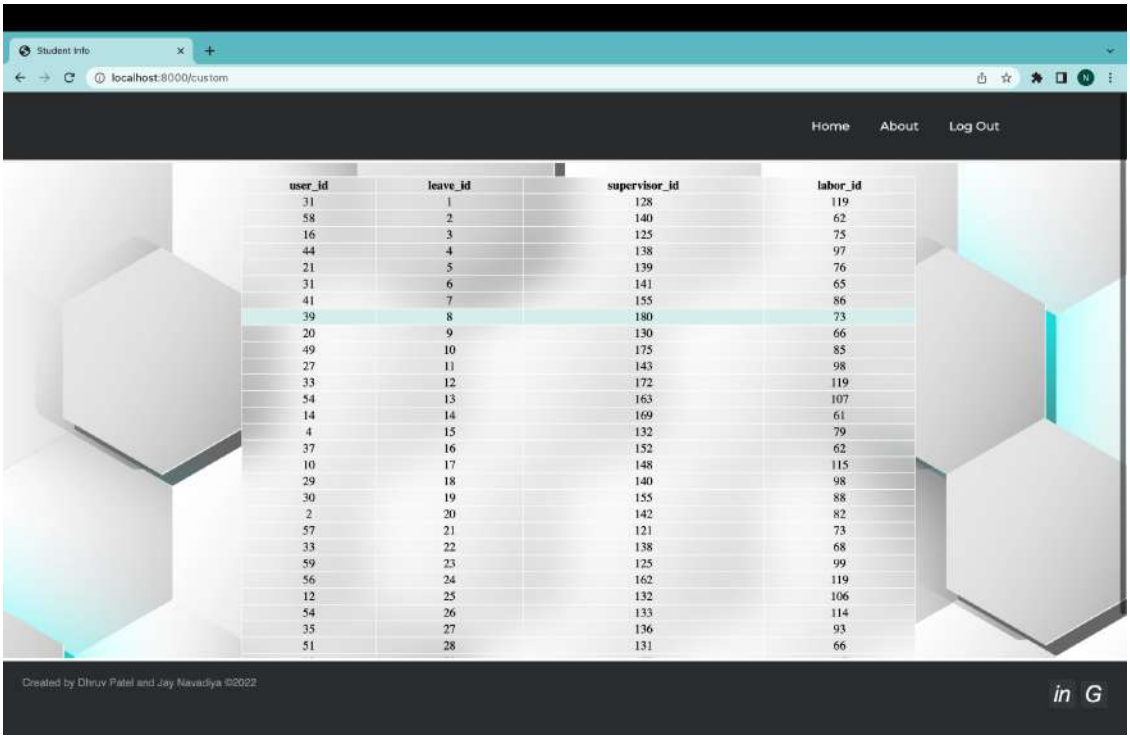
It allows the admin to run a custom query directly to the database. It is implemented with the help of Javascript and JSON.

Example: Query for seeing the leaves that are approved by the higher authorities.



Result:

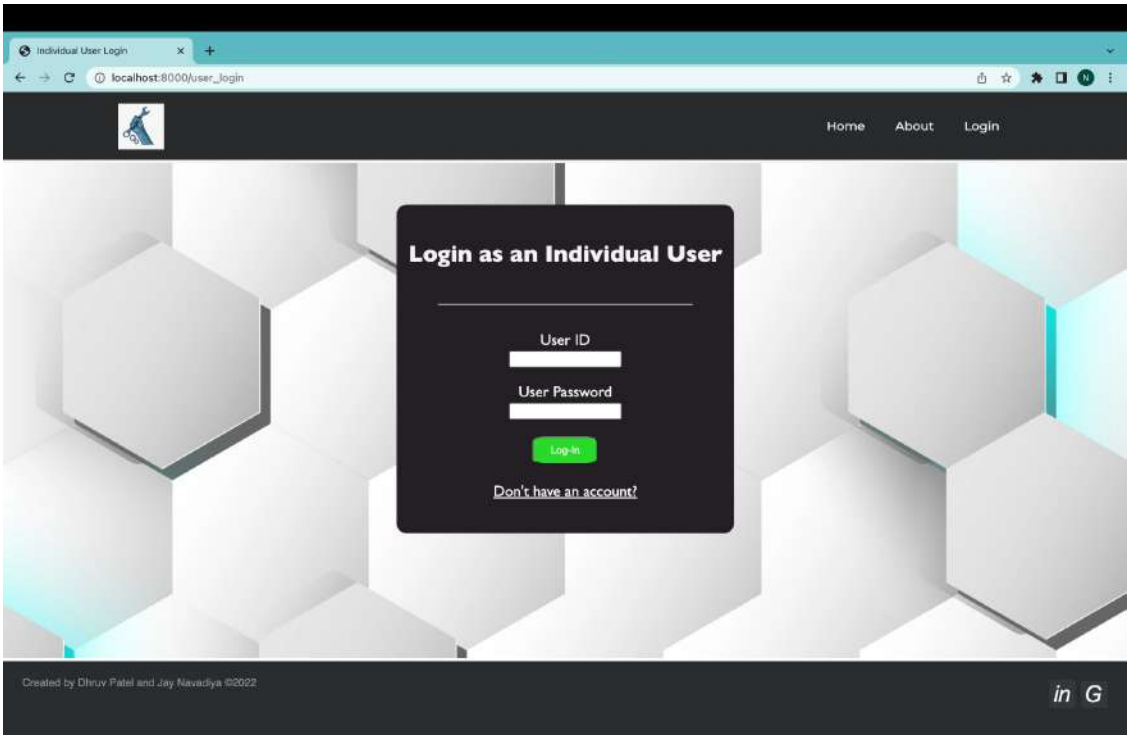
The scrolling feature is also implemented here as you can see on the right of the screenshot given below.



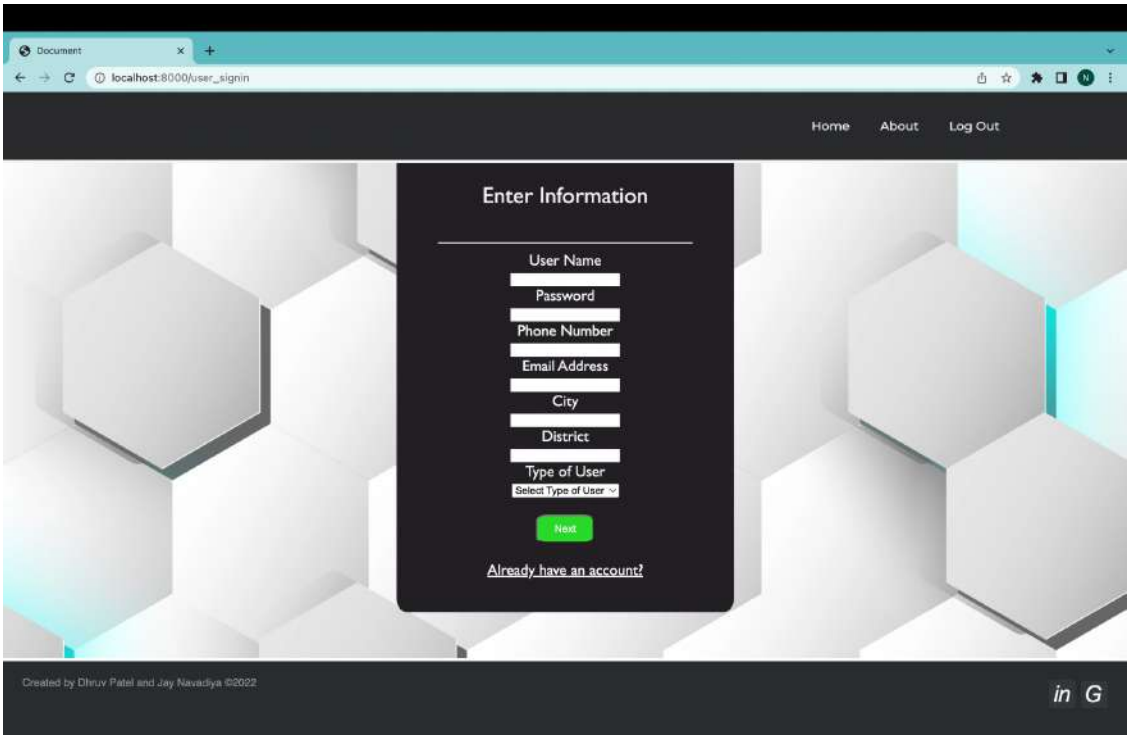
user_id	leave_id	supervisor_id	labor_id
31	1	128	119
58	2	140	62
16	3	125	75
44	4	138	97
21	5	139	76
31	6	141	65
41	7	155	86
39	8	180	73
20	9	130	66
49	10	175	85
27	11	143	98
33	12	172	119
54	13	163	107
14	14	169	61
4	15	132	79
37	16	152	62
10	17	148	115
29	18	140	98
30	19	155	88
2	20	142	82
57	21	121	73
33	22	138	68
59	23	125	99
56	24	162	119
12	25	132	106
54	26	133	114
35	27	136	93
51	28	131	66

Now, let's have a look at the login as an individual user feature.

10. Login Window

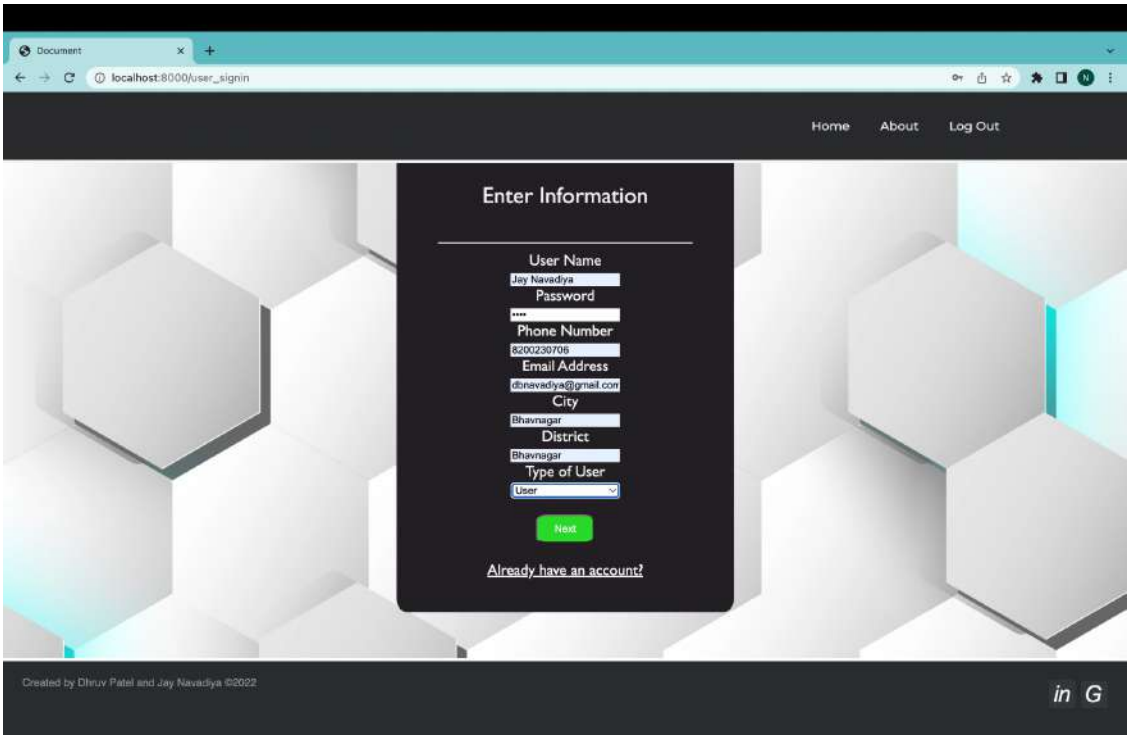


In case of no account, it can be created using Don't have an account?



A new user account can be created from here.

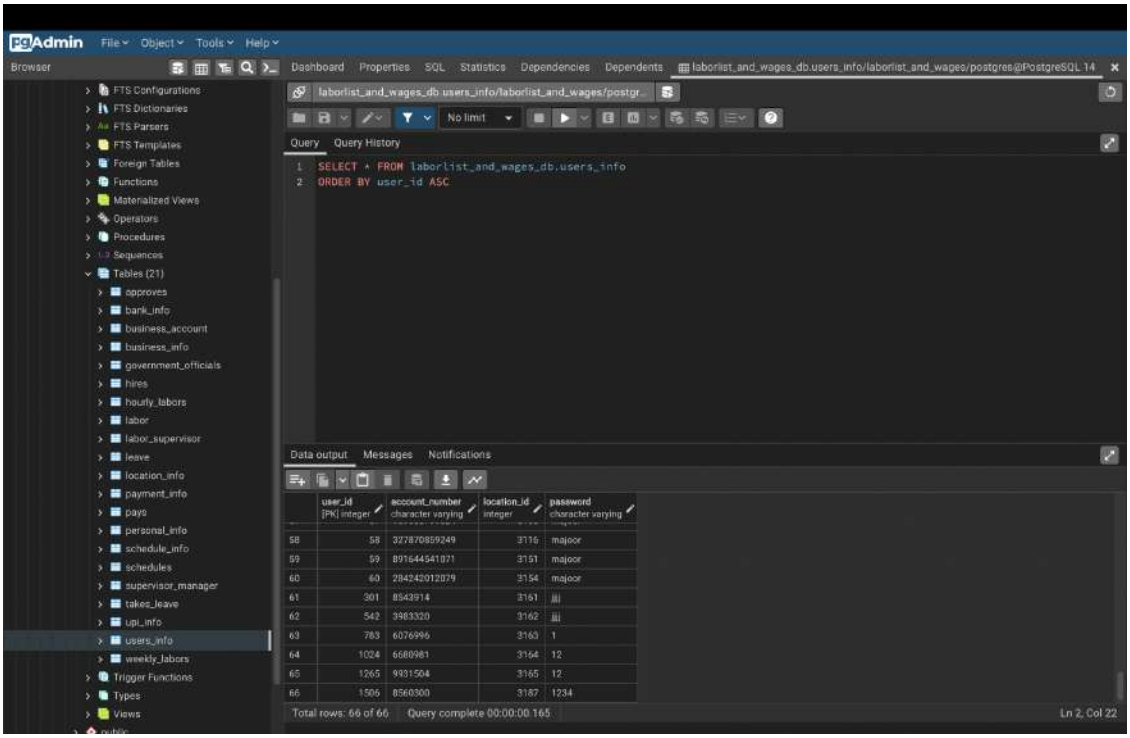
Let’s create an account by filling up the details shown in the screenshot given below.



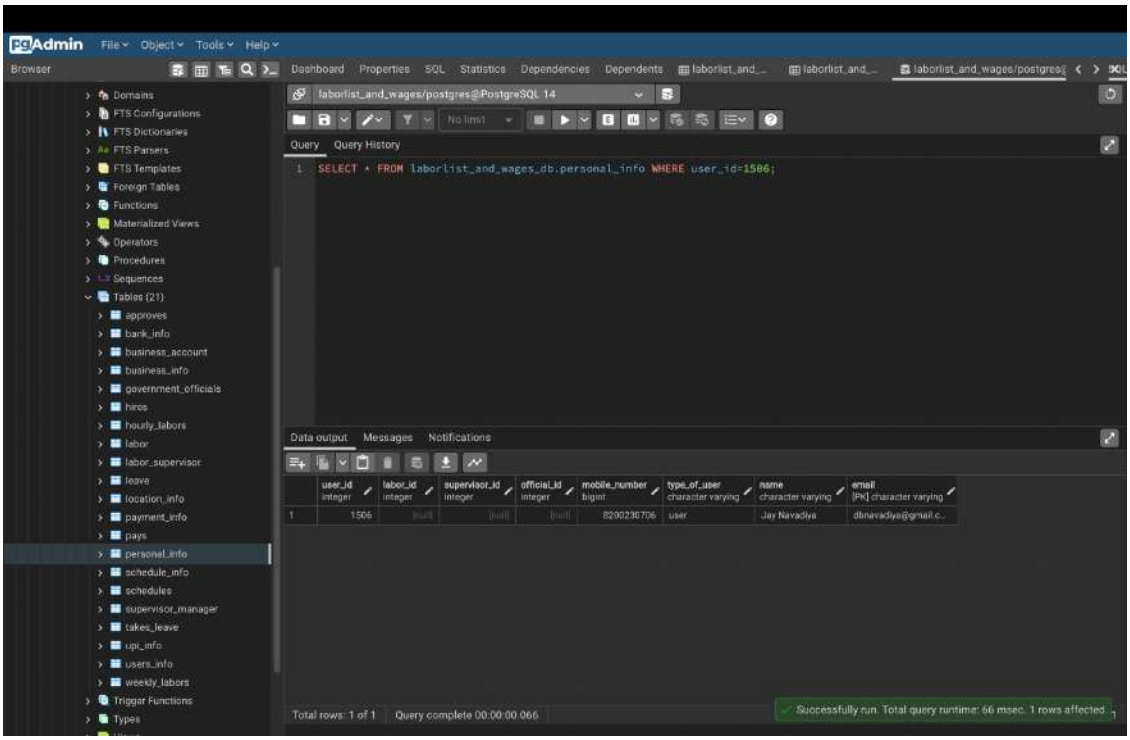
If the user account has been created successfully, the website will take you to the homepage.

Let’s check if the user has been created or not.

Pgadmin screenshot of users_info table:



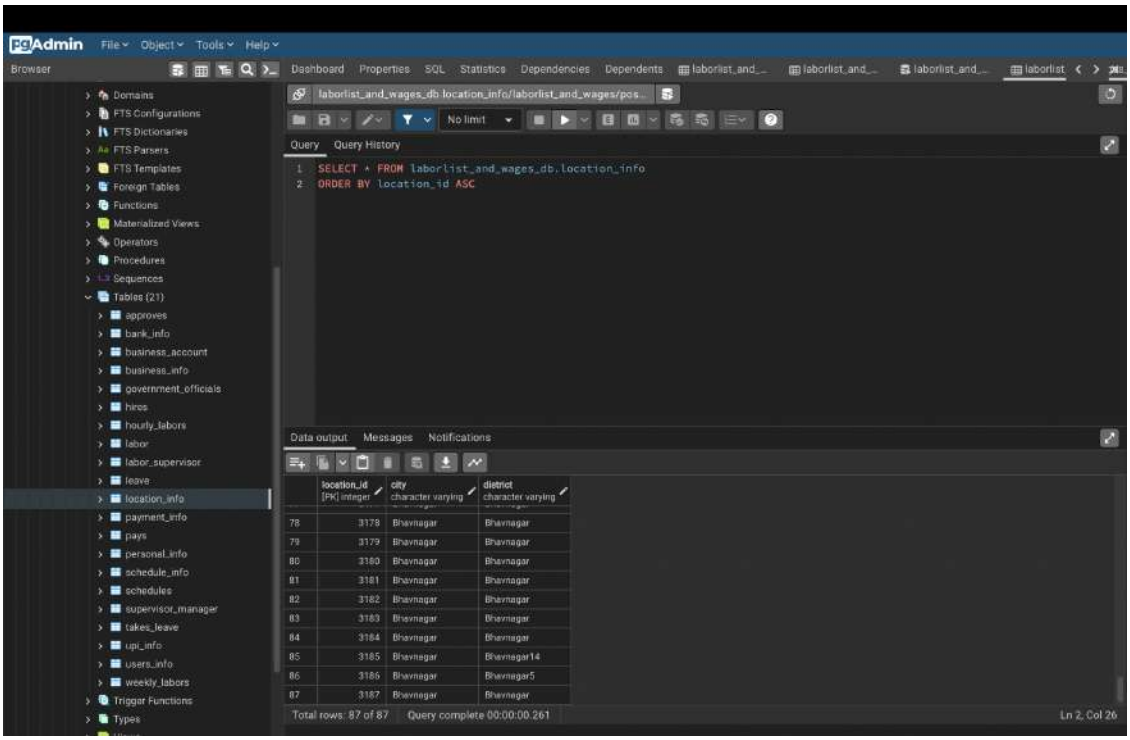
As we can see that user id 1506 has been created in the users_info table. Let's see the personal_info of user_id 1506.



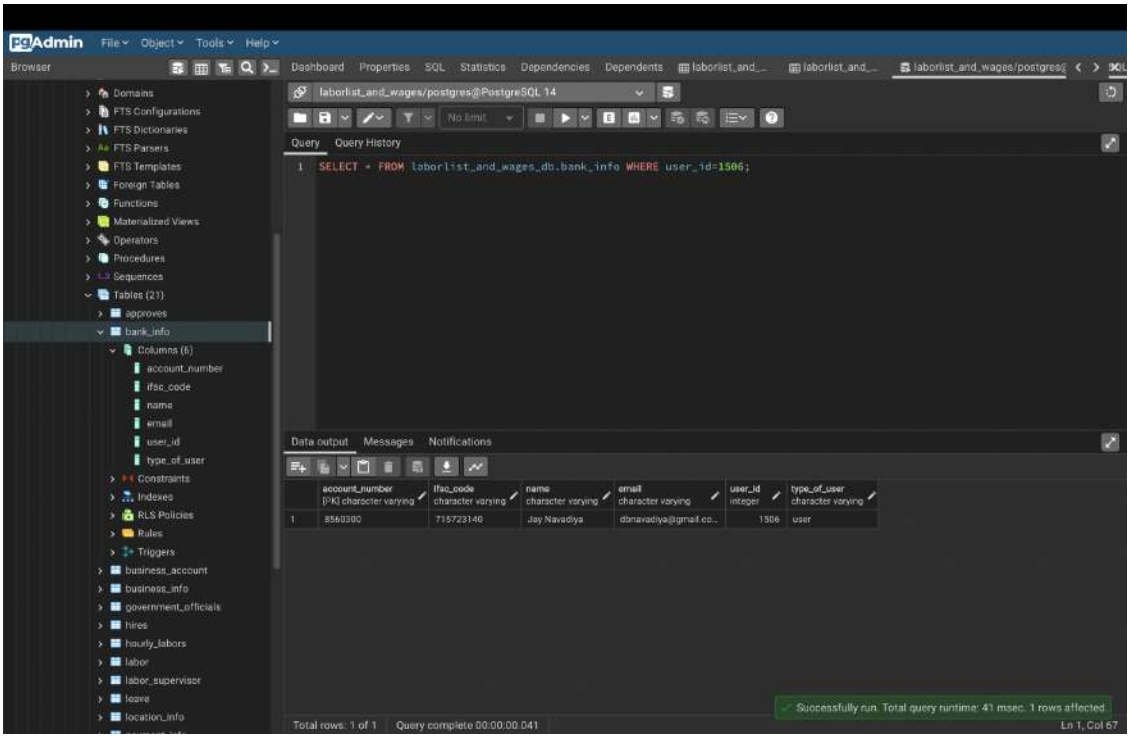
As we can see, the name, type-of-user and every other field have been successfully created.

The location_info table:

As we can see, The city, Bhavnagar and the district, Bhavnagar has been successfully created with location_id 3187. Ignore other entries as they were used for testing purposes.

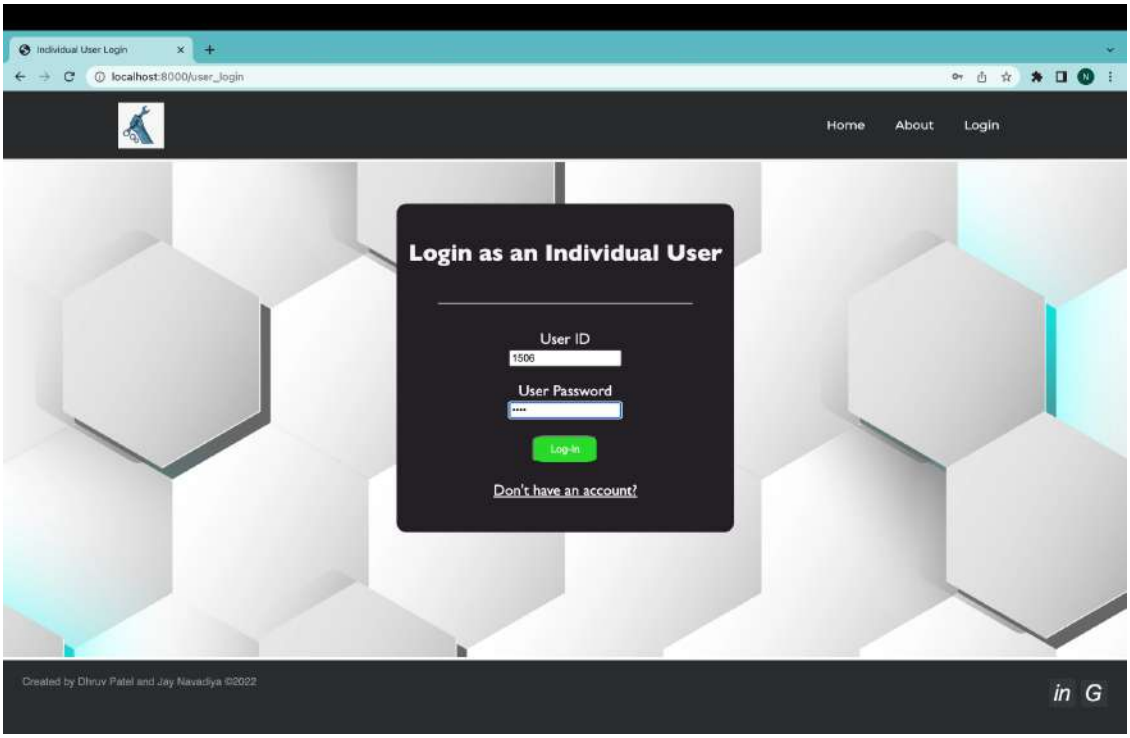


Let's look at the bank_info entry for user_id = 1506.

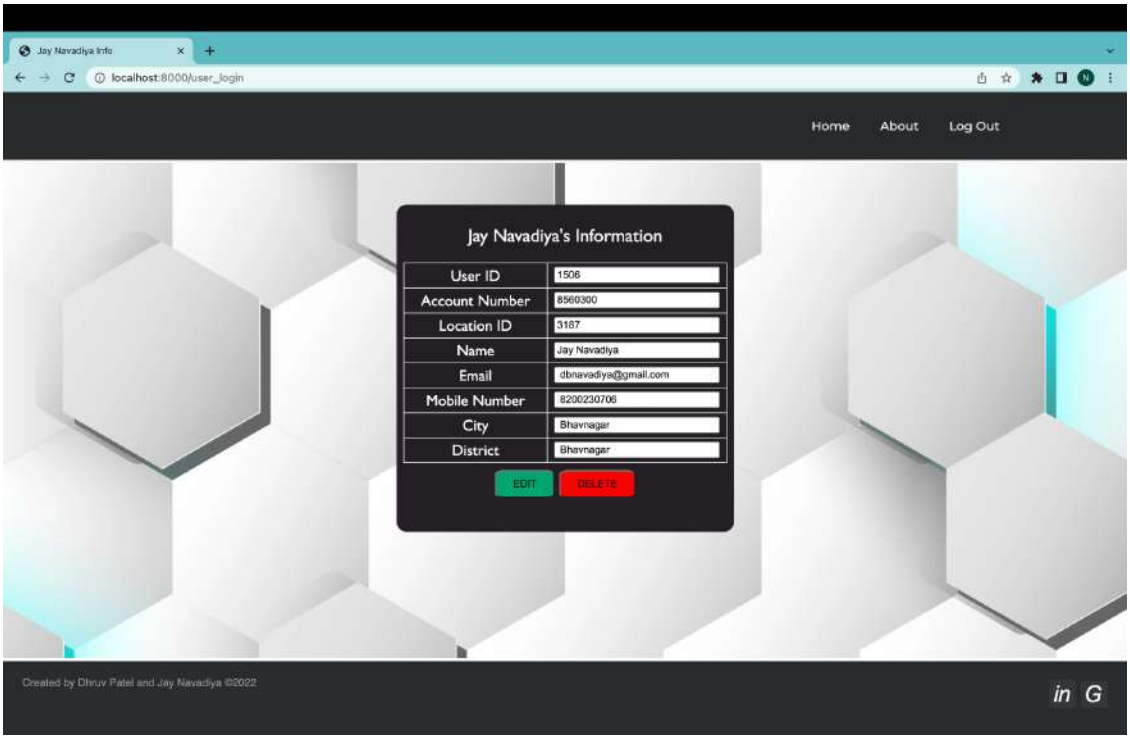


As we can see that the account_number and the ifsc code have been inserted successfully randomly and they are unique too.

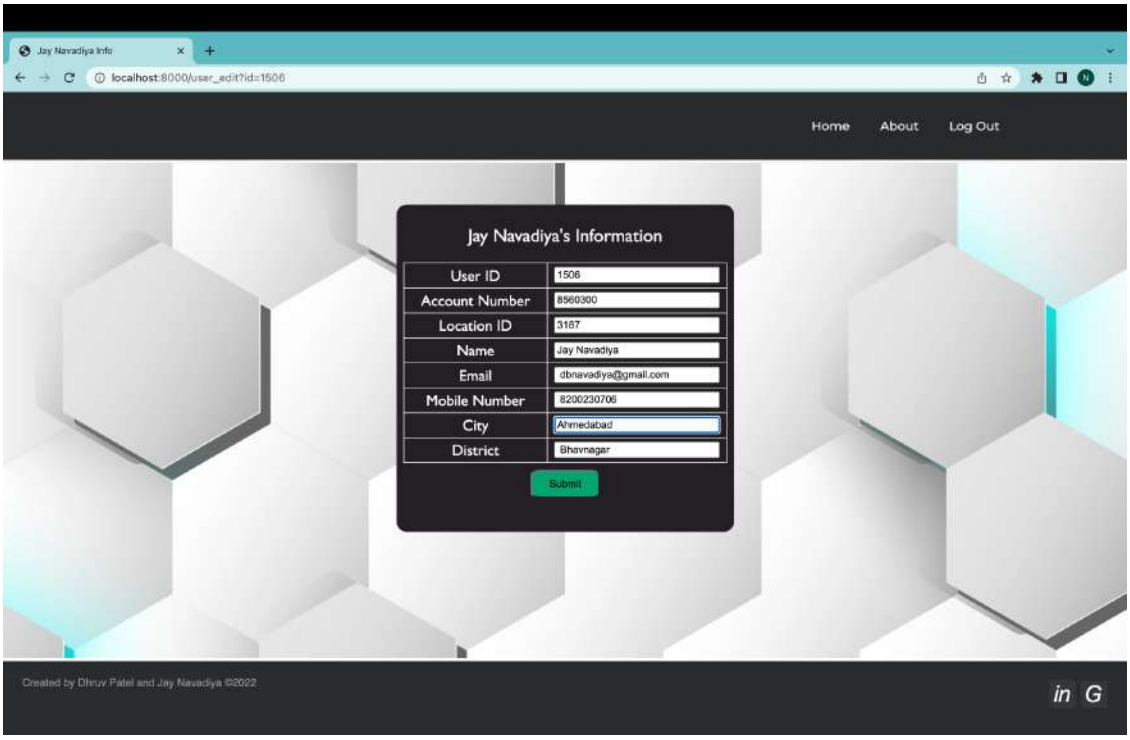
Now, let’s login into this account.



It shows their information and gives an option to delete their account or edit the editable allowed information.

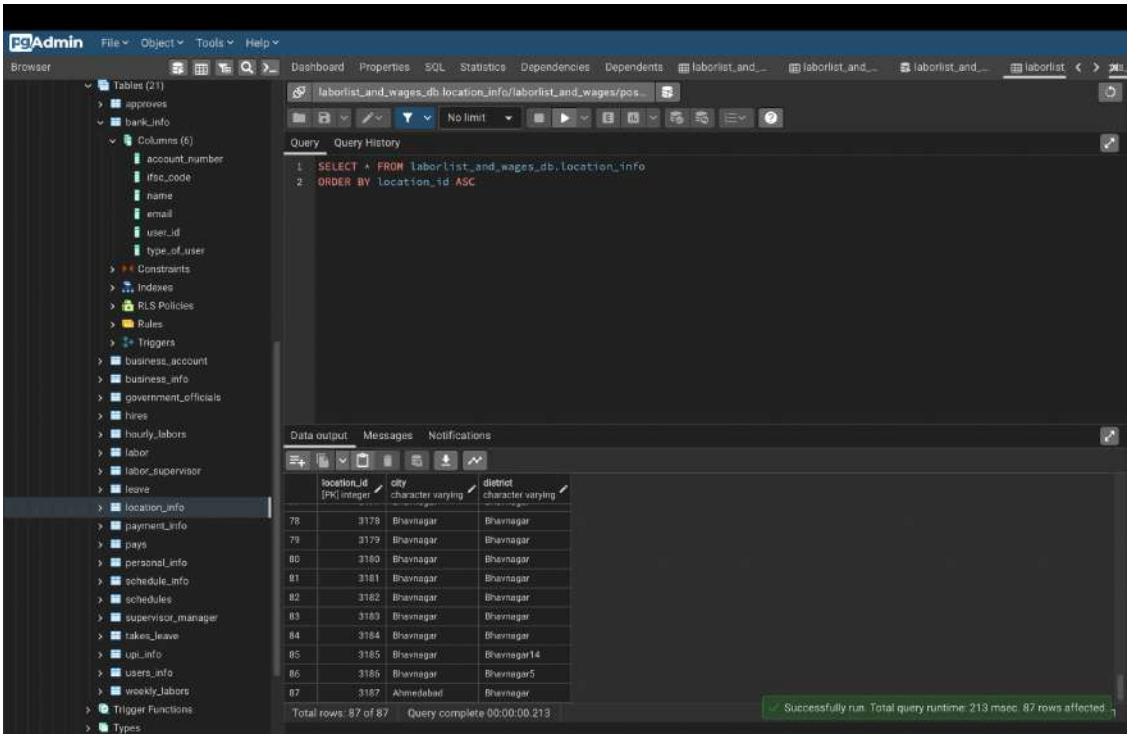


Let’s edit the location and change the city to Ahmedabad.



As we can see, the edit button lands the user on user_edit page.

Let’s check if the changes have been made in the database or not.

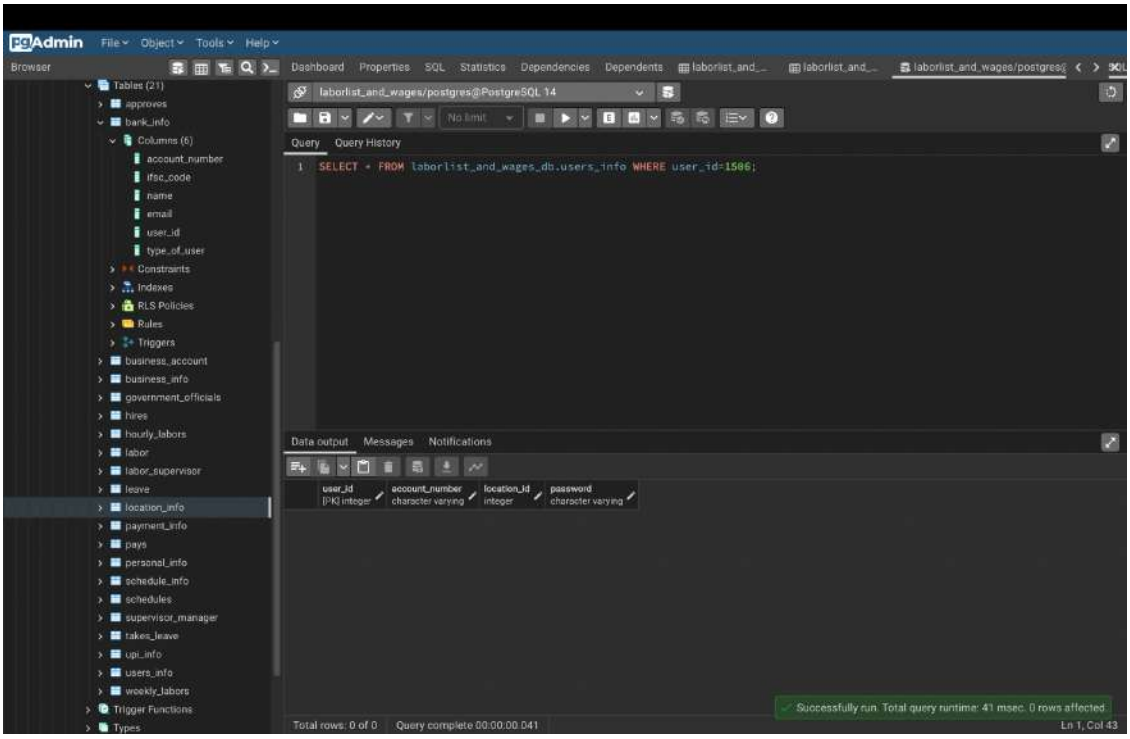


As we can see, the entry is updated as the location_id remains the same and the city has been changed to Ahmedabad.

Now, let's delete this account by again logging into the account and clicking the red delete button.

If any action has been taken and it's successful, you will be landed on the homepage.

After deletion, let's check if the user exists in the users_info table or not.

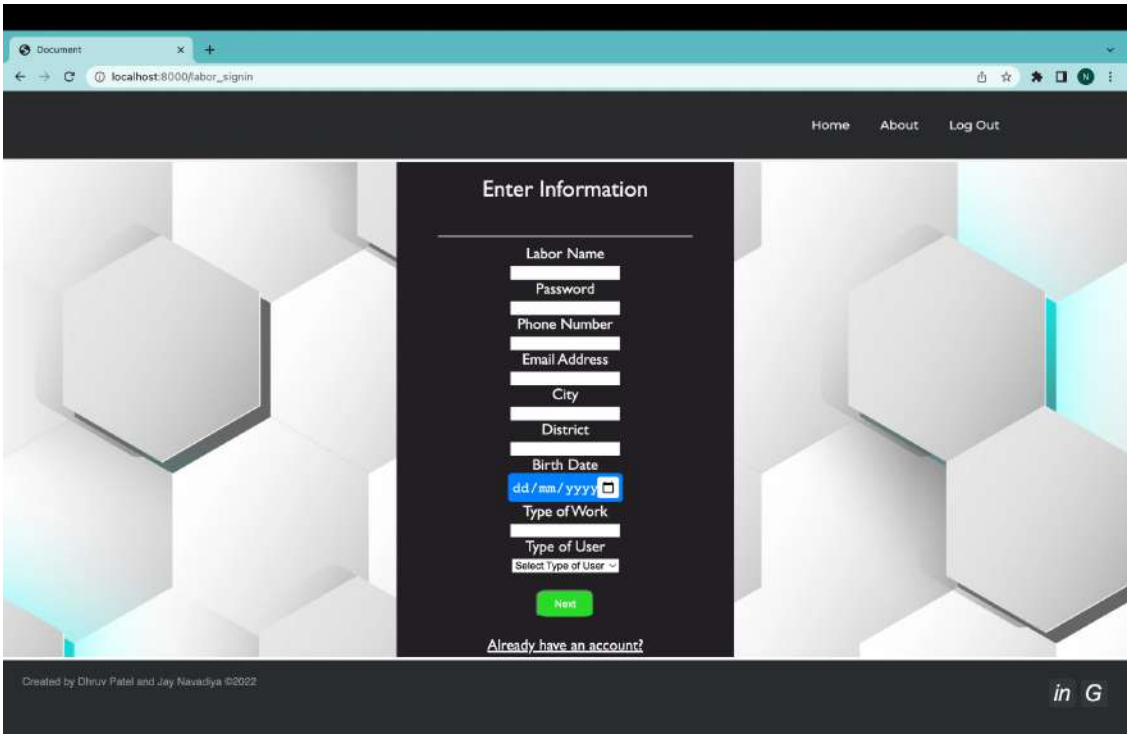


As we can see, there are no users with user_id=1506. By the foreign key constraints and the way we have run the queries it will be deleted from all the tables. I.e. the location_info, bank_info and personal_info tables.

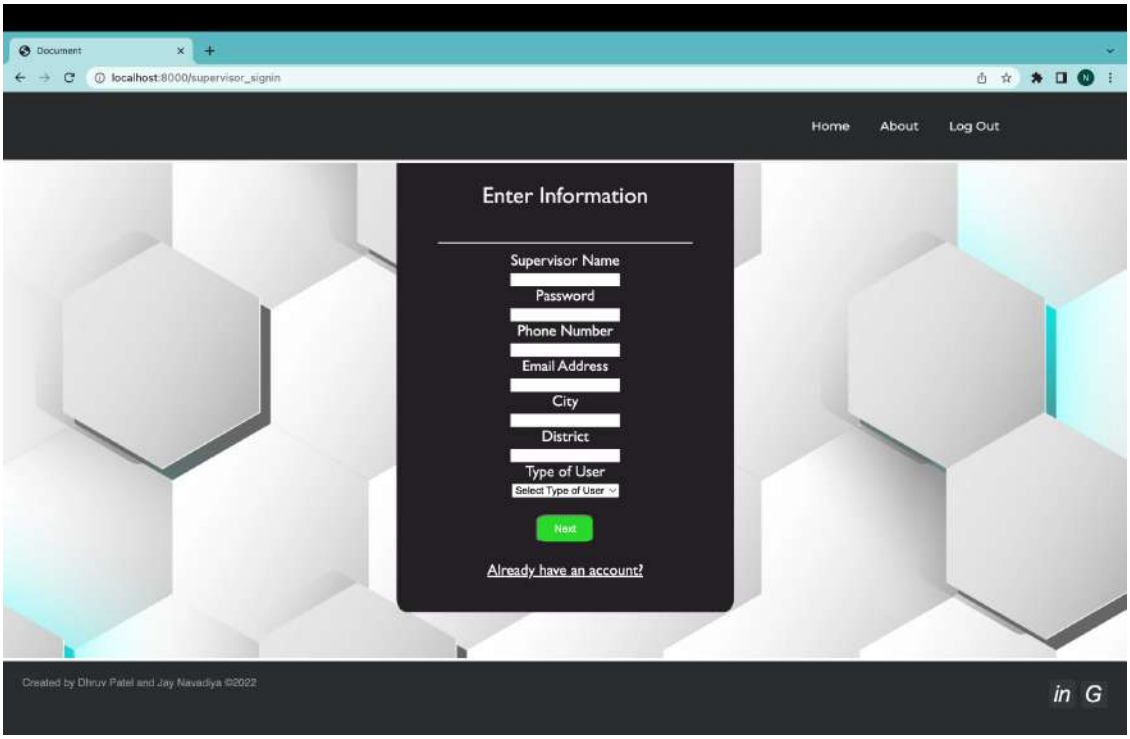
The same can be seen within the code file of server.js.

The same functionalities have been implemented for all types of users.

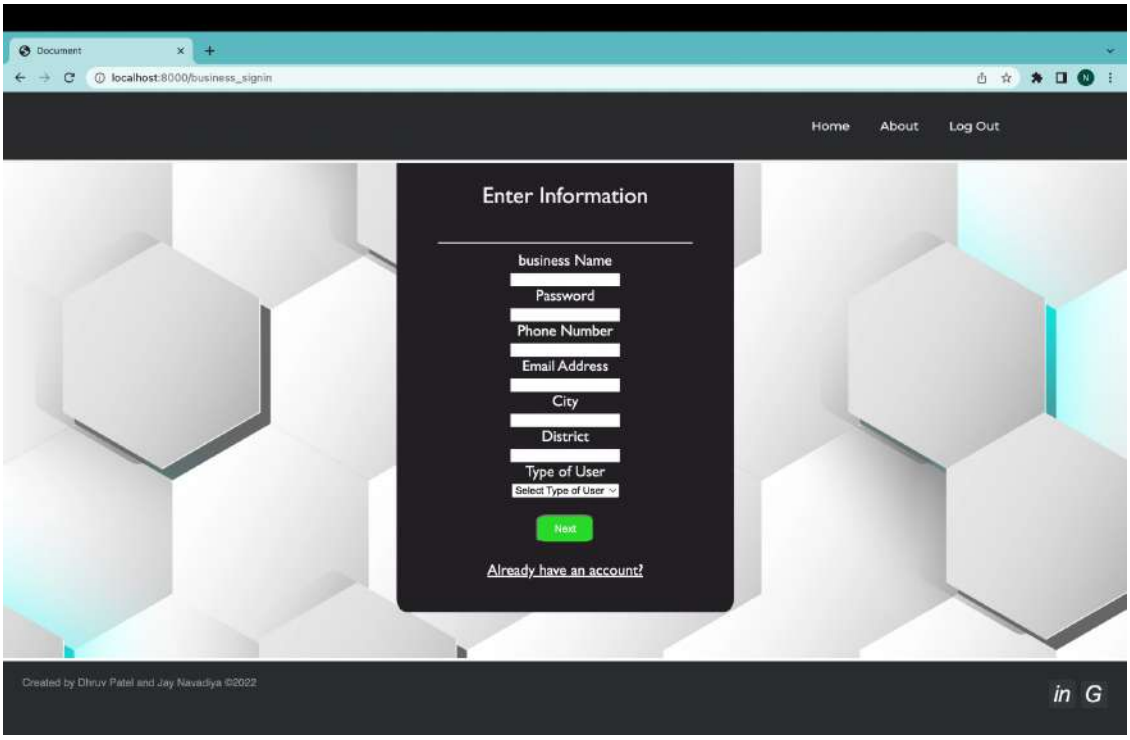
Labor registration form:



Supervisor Registration Form:



Business Registration Form:



Also, if someone enters the wrong password or any information it will be displayed as a text and he/she will not be able to log in.

These features are available for all types of users as previously explained. The website can be accessed when the database has been properly set in the local machine and node has been installed. Node server.js will allow the link localhost:8000 to be run in the browser.

The database can be made easily by using the DDL Scripts, database.sql for the whole database or schema.sql for making the schema in the database only.

Thank You!

Teaching Assistant Vinay Sir and Prof. Rachit Chhaya,
For the guidance and support throughout the project.