**Initial Setup**:

To effectively scrape data from the Zauba Corp website leveraging of various sorting and filtering options provided by the site. These filters allow to categorize companies into different financial and operational brackets.

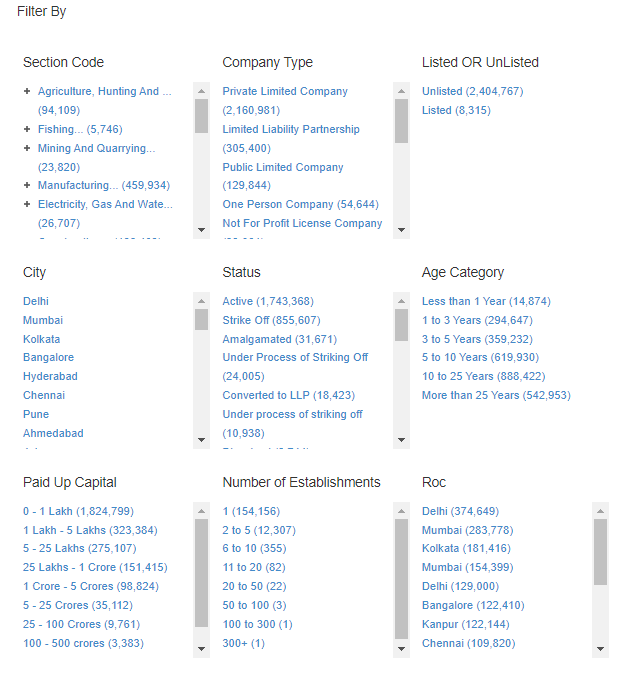


Figure 1

After examining the available sorting options and compiling the company counts for each category clear that Paid Up Capital, Status, and Age Categoryare categories that give highest number of results respectively. But if we see Paid Up Capital (0 – 1 Lakh) gives us 1824799 (Figure 1) which is more than 399990 so we have to add more filters in to make it under 399990.

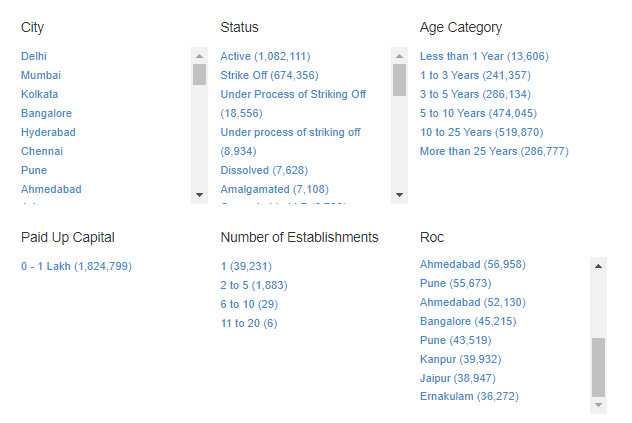


Figure 2

After selecting the 0-1 Lakh option (Figure 2) we know, Status has more total companies than the Age Category, so we use the filter of Status as Active and Strike Off rather than Age Category.

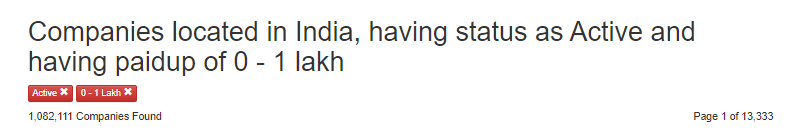


Figure 3

After selecting status (active) on the page where Paid-Up Capital (0-1 Lakh) already selected we got 1082111 results (Figure 3). Here total number of companies coming to 1082111 is still greater than 399990(13,333\*30). We still can’t get all the companies.

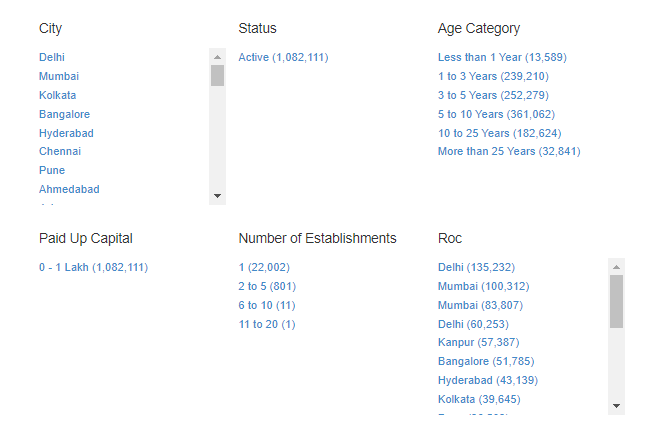


Figure 4

Now in the Age Category, we have total companies for any choice less than 399990 (Figure 4) so we click on them and get the final filter of 0-1 Lakh (Paid Up Capital) and Active (Status), and each option one by one (Age Category). Similarly goes 0-1 Lakh (Paid Up Capital) and Strike-off (Status) and each option one by one (Age Category). Now for other statuses, if not even given the filter of 0-1 Lakh they are already less than 399990 so we can take them independent of other filters, and the same goes for Paid Up Capital for all other options than 0-1 Lakh. This will give us this full union set.

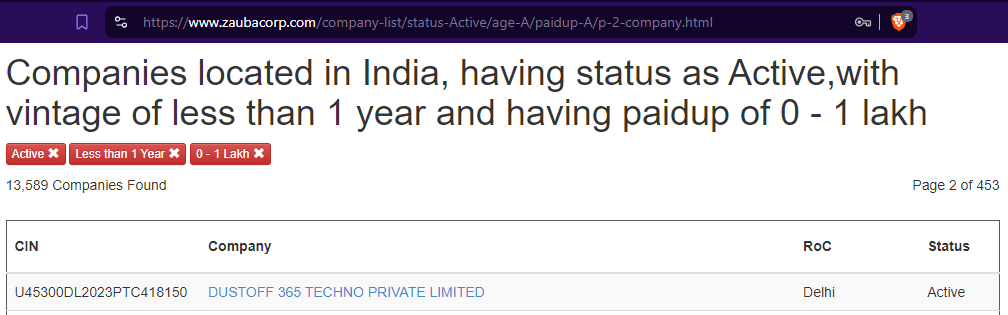


Figure 5

After putting all these filters open the page number 2 if exist and we got the total page and URL pattern (Figure 2) after replacing 2 to {i} in the URL. If page number 2 not exist then just use the default as then total page number will be 1 we just have to use it for once only.

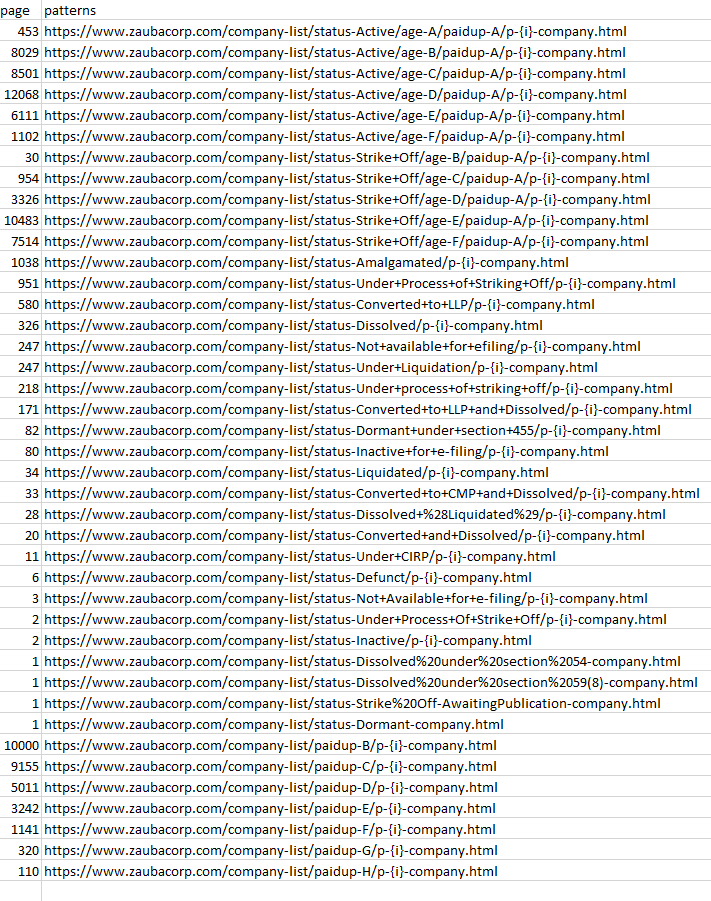


Figure 6

So that’s how I manually created a CSV file (Figure 6) that has the total page and URL pattern for all permutations and combinations of my approach.

**Deploying Server.py**:

**Suppose we have a dummy email id as** [dummy@gmail.com](mailto:dummy@gmail.com)**. Using this create a git hub account.** Then create a public git-hub repository and upload the required files as server.py and fetchAndExtract.js and templates folder with an index.html in it. Create a requirements.txt and added all versions carefully to avoid version conflicts. Now you have the public git-hub repository similar to <https://github.com/black-raven-001/zauba-scrape>.

Create a dummy render account using mail id as [dummy+1@gmail.com](mailto:dummy+1@gmail.com). Using +1 or +anything we can create multiple account on render using same email id by just adding anything after +. All email come to [dummy@gmail.com](mailto:dummy@gmail.com) but on render they will all act as different. Then in a new web service, gave the public git-hub repository(Figure 7).

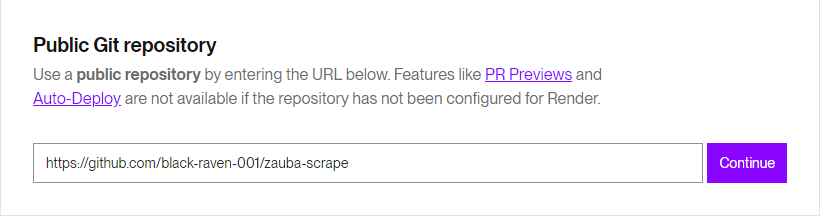


Figure 7

After that use a sequence name with numbers in it as in all other accounts just numbers change everything remains the same. Use “pip install -r requirements.txt && npm install” as build command then after deploying web services and removing all errors and conflicts finally the server.py is successfully deployed. (Figure 8) Now we just have to create 99 more render accounts to create 99 more web services using the same method and git-hub repository. As all errors were already resolved in 1st deploy now there will be no waste of debugging time. The only thing that will be different is the name as in ith account its name will “scrape{i}.

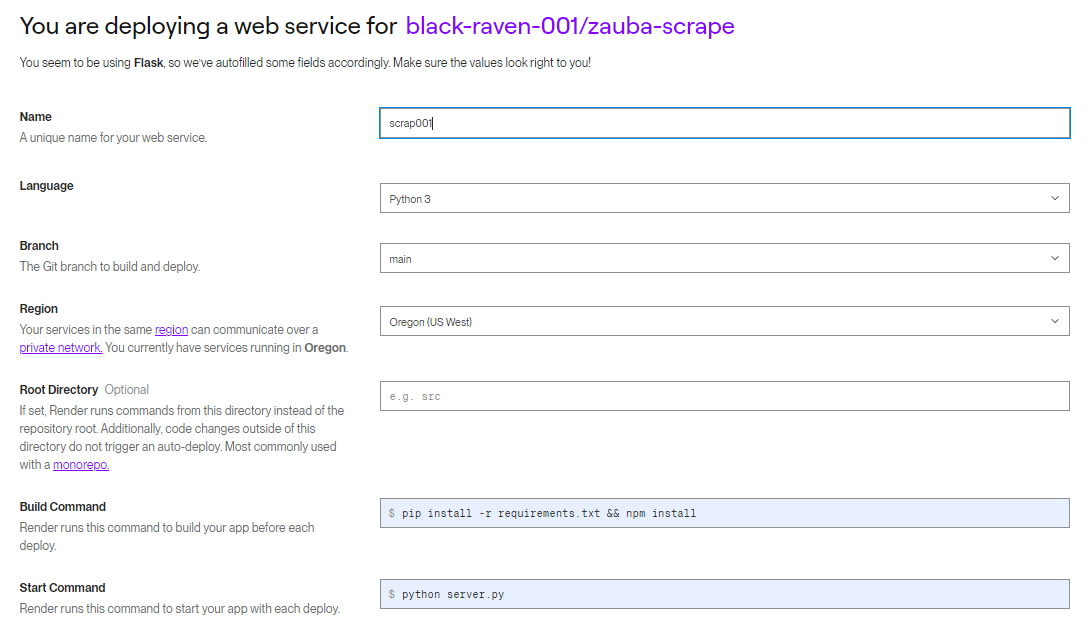


Figure 8

These 100 names will go from scrap001 to scrap100 for 100 accounts for the same <https://github.com/black-raven-001/zauba-scrape>. Now we have 100 working server.py at a different address. so for these, we have to create cron-jobs that can give them a webpage request after every 10 minutes. so we can create 50 cron-job accounts for this 100 deployed server.py, 1 on every 2 (Figure 9).

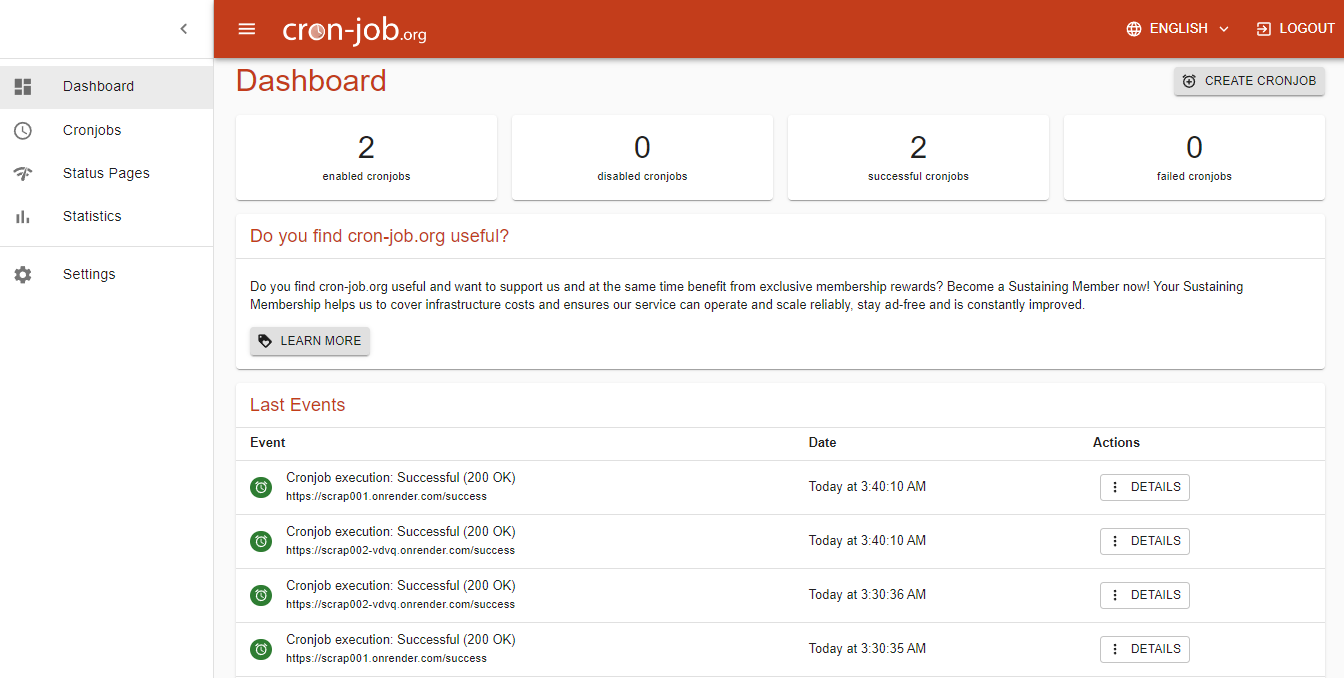


Figure 9

<https://scrap001.onrender.com/success> this will return an HTML page with just One Piece written on it which says that server.py is working. If there is no ping on server.py for 15 minutes then there is a chance of getting it down so we may have to restart it again.

**Batch processing**:

We start by preparing the data, where continuous Serial Numbers (SN) are generated to uniquely identify each request. A dataframe is then created, consisting of two columns SN and URL. The SN serves as a unique identifier for each request, and the URL is the target endpoint for data retrieval.

Next, we form batches, setting the batch size to either 400 or 500 requests based on the server capacity. This batch is further split into smaller groups of four URLs, distributed across different servers to optimize parallel processing. Utilizing Python's concurrent.futures.ThreadPoolExecutor, multiple threads are initiated to handle these requests simultaneously, improving efficiency and speed. Requests are allocated to different servers using a round-robin approach based on the SN, ensuring balanced load distribution.

During concurrent execution, each thread sends a request to the designated server and waits for a response. Once all requests in a batch are processed, results are collected and validated. Valid data is integrated into the database using the add\_new\_urls\_to\_detail(), add\_entry\_to\_full\_detail() and add\_director\_from\_dataframe() functions which takes dictionary or dataframe as an input parameter. If any errors occur during processing, such as network issues or server errors, the entire batch is discarded. The system then waits for a specified time (usually 1 minute) before retrying the same batch again. This retry process is limited to a maximum of five attempts to prevent indefinite looping.

Throughout the process, detailed logs are maintained to monitor progress, including the number of requests processed, the time taken for each batch, and any errors encountered. The system estimates the remaining time for processing based on the progress of the current batch, providing a percentage of completion and time remaining. This ensures transparency and allows for effective monitoring of the batch processing.

Finally, once all batches are successfully processed, the system logs the completion of the process, and the database connection is gracefully closed to ensure data integrity and resource management. This robust batch-processing mechanism enables efficient, reliable, and scalable data retrieval, allowing for the seamless integration of large datasets into our system.

1. **Storing Scraped Data into MySQL Database:**

The DBConnection class is designed to handle the storage of scraped data into a MySQL database efficiently and robustly. It manages connections, dynamically adjusts the schema as needed, and ensures data integrity through comprehensive error handling.

**Dynamic Query Generation**: The add\_new\_urls\_to\_detail(), add\_entry\_to\_full\_detail() and add\_director\_from\_dataframe() method constructs SQL queries dynamically from a provided dictionary, where keys are column names, and values are the data to be inserted.

**Handling New Columns**: If a new key (column) is encountered, the method checks for "Unknown column" errors. If detected, it alters the table to add the new column with a default value of NULL, thereby redesigning the schema on-the-fly.

**Managing Long Column Names**: MySQL limits column names to 64 characters. If a key exceeds this limit, it is shortened using a hashing mechanism to ensure uniqueness, and the original-to-shortened name mapping is maintained in a runtime dictionary.

**Data Length Issues**: When a "Data too long for column" error is encountered, the method dynamically increases the column size and retries the insertion, ensuring all data is accommodated without truncation.

This robust mechanism ensures that all scraped data is inserted into the database accurately, efficiently, and with minimal manual intervention, even as new data fields and variations are encountered.

**5 Limitations:**

**Website Performance**: The Zauba Corp website was sometimes slow, taking a considerable amount of time to load pages. This latency impacted the overall efficiency and speed of the data collection process.

**Data Availability**: Although there are approximately 27 lakh companies listed on Zauba Corp, the website only provides access to 13,333 pages, each containing 30 company URLs. This limitation restricts access to about 4 lakh companies through this approach. However, different filtering techniques were employed to bypass this and access more company data.

**Server Management**: The project utilized 100 servers to distribute the scraping load. If any server became unresponsive or was temporarily blocked by Zauba Corp, it required manual intervention to restart the server and resume scraping. Monitoring and maintaining these servers added to the complexity of the project.

**Data Inconsistencies**: Despite collecting URLs using details.py, some URLs led to pages with the message "Company not found." This issue required additional handling to ensure data completeness and accuracy.

**Volume of Data**: Dealing with 27 lakh companies involved fetching over 27 lakh pages and storing the data in a MySQL database. This massive volume of data made the process time-consuming, requiring the code to run continuously for 4 to 5 days, depending on the speed and reliability of the servers.

**Dynamic Database**: The Zauba Corp database is frequently updated with new companies, meaning the data collected could quickly become outdated. At any given time, it was impossible to capture all companies. For example, after completing the scraping process, 27,19,728 companies were scraped out of the currently available 27,23,070, leading to a small percentage of missing data. Nevertheless, the accuracy achieved was more than 99.8%.

**Inconsistent Data Fields**: The data fields for companies were not consistent. Some companies had a Company Identification Number (CIN), while others had an LLP Identification Number or a Registration Number. This inconsistency required additional handling to ensure all relevant data fields were captured correctly.

**Database Structure Changes**: The variability in company data required dynamic changes to the database structure during runtime. New types of data necessitated adding new columns to the database, which was challenging given the large volume of existing data. Additionally, if a new value exceeded the predefined length for a column, the table structure had to be modified accordingly, complicating the data management process.

**Resource Intensive**: Even with 100 servers running parallely. the project required significant computational resources and careful orchestration. Any minor delay or change in server speed could extend the total processing time, illustrating the resource-intensive nature of large-scale web scraping projects.

1. **Output:**

After completing the data scraping, processing, and storing procedures, we organized the information into four primary tables: details, full\_details, director, and shorted. However, we decided to retain only the most comprehensive tables for our final SQL Server database. Here is a summary of the final output:

**Final Tables:**

**Full\_Details Table**:

**Content**: Contains comprehensive information about company.

**Rows**: 2,719,728 unique companies.

**Primary Key**: Unique Serial Number (SN) for each company.

**Director Table**:

**Content**: Includes information about company directors and their relationships to companies.

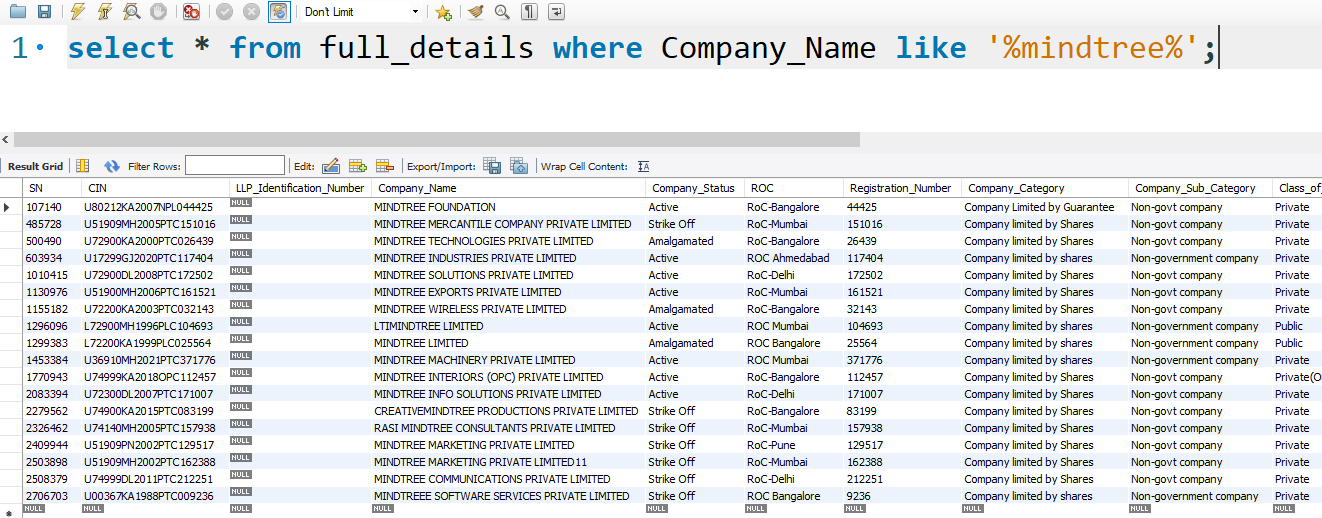
**Rows**: 5,844,094 unique company-director pairs.

**Mapping**: Directors are mapped to companies using the company's SN.

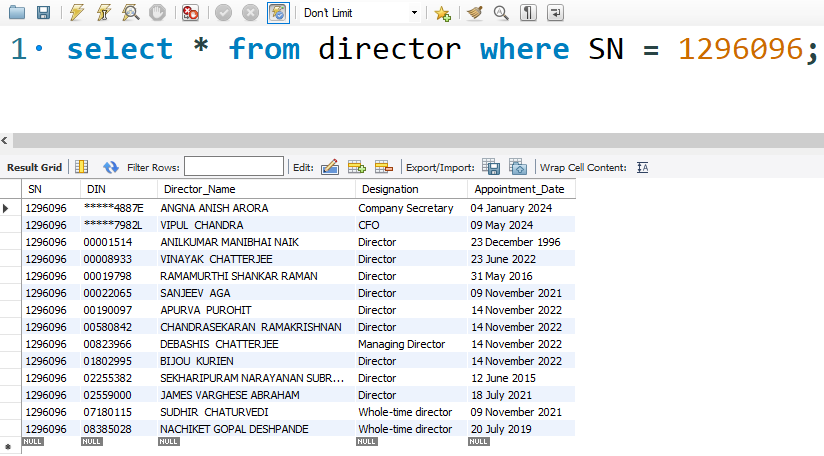
**Query Capabilities:**

With the streamlined database structure, users can efficiently run queries to retrieve specific information:

**Search Company by Name**: Users can search for a company by name in the full\_details table.



**Retrieve Director Details by SN:** Given a company's SN, users can query the director table to get all directors associated with that company.



**Find Companies by Director:** Users can search for a director by name in the join of director and full\_details on SN to retrieve all companies where the individual serves as a director.

