*University Applications Database Management System*

Sai Kumar Thoppae Sethu Raman   
*School of Engineering and Applied Sciences University at  
Buffalo, USA*  
*sthoppae@buffalo.edu*

Vykunth Premnath   
*School of Engineering and Applied Sciences University at  
Buffalo, USA*  
*vykunthp@buffalo.edu*

Monica Evangelin Kaki  
*School of Engineering and Applied SciencesUniversity at  
Buffalo, USA  
monicaev@buffalo.edu*

*Abstract*—This project aims to address the challenge of managing the vast amount of data generated during the university application process by proposing a database system over an Excel file. The database system offers several advantages, including enforcing data integrity, scalability to handle large volumes of data, security and access controls, real-time collaboration and sharing, and advanced query and analysis capabilities. In this phase, the project focuses on normalizing the previous entity-relationship diagram and deploying it into a website. Additionally, the response of indexing, stored procedures, and triggers is shown. These features improve the efficiency and reliability of the university applications database system. The normalization process ensures that the data is organized into appropriate tables and that redundant data is minimized, which helps to maintain data integrity. The deployment of the system to a website enables users to access the database remotely, enhancing real-time collaboration and data sharing. The use of indexing, stored procedures, and triggers improves the system's performance, security, and reliability.

# Introduction

The university application process generates a vast amount of data, including information on students, their application materials, test scores, transcripts, and other relevant documents. Managing this data efficiently and accurately is a critical challenge for universities. Traditionally, universities have used Excel files to manage this data. However, Excel files are limited in their ability to manage large volumes of data, enforce data integrity, provide advanced query and analysis capabilities, and ensure secure access control.

A database system is a reliable and efficient solution for managing the complex data requirements of a university applications database. Compared to an Excel file, a database system offers several advantages, such as enforcing data integrity, scalability to handle large volumes of data, security and access controls, real-time collaboration and sharing, and advanced query and analysis capabilities. Therefore, a database system is a more preferable solution for managing a university applications database.

A database system can ensure data integrity by imposing constraints that ensure consistent, accurate, and reliable data. This is important in university application databases because inaccuracies or errors can have serious consequences for both the applicants and the university. Moreover, it is scalable and can handle large volumes of data without compromising performance, making it suitable for managing the substantial amount of data generated during the university application process.

In addition, a database system can provide security and access controls that ensure only authorized users can access sensitive data. This ensures that confidential information is protected from unauthorized access and that the integrity of the data is maintained. Furthermore, a database system enables real-time collaboration and data sharing among multiple users, allowing for efficient collaboration. This is particularly useful for universities with multiple campuses or for applicants who are not on campus.

Finally, a database system provides complex query and analysis capabilities that offer insights and trends that may be difficult or impossible to obtain from an Excel file. Advanced query and analysis capabilities enable universities to gain a better understanding of applicant data and make data-driven decisions that are critical to their operations.

In summary, the proposed university applications database system is an essential tool for managing the complex data requirements of a university applications database. The system offers several advantages over an Excel file, including enforcing data integrity, scalability, security and access controls, real-time collaboration and sharing, and advanced query and analysis capabilities. By using advanced query and analysis capabilities, the system can provide valuable insights and trends that are critical to decision-making processes..

# Advantages Of Database Over Excel

A university applications database management system offers several advantages over Excel. In this section, I will explain each of these advantages in detail and how they apply to managing a university applications database.

## Security

Security is a top priority when managing university application data. Databases provide secure access control systems that ensure that only authorized users can access and modify data. Administrators can define user roles and permissions, which limit access to sensitive data. Additionally, databases provide data encryption, which protects data from unauthorized access. Excel, on the other hand, is not secure, and it can be easily copied and modified, making it challenging to maintain data confidentiality and security.

## Data Integrity

Data integrity is crucial when managing university application data. A university applications database management system must ensure that the data entered into the system is complete, accurate, and consistent. Databases provide data validation mechanisms that ensure data integrity. These mechanisms check data for completeness, consistency, and accuracy before entering it into the database. Additionally, databases enforce data constraints, which minimize the risk of errors and inconsistencies.

## Data visualization and analytics

Databases provide advanced data analytics and visualization tools, enabling administrators to analyze data efficiently. These tools include data mining, machine learning, and predictive modeling, among others, which are critical for managing university application data. Excel, while useful for analyzing small datasets, lacks advanced data analytics features and cannot handle large datasets efficiently.

## Search

A university applications database management system must have a robust search function. Databases provide advanced search capabilities that enable administrators to search for data using specific criteria. These search capabilities allow administrators to filter and sort data efficiently, saving time and improving accuracy. Excel, on the other hand, has limited search capabilities and cannot handle large datasets efficiently.

## Data Insertion

Data insertion is a critical part of any university applications database management system. As applications come in, they must be entered into the database accurately and efficiently. Databases provide a structured way to input data, ensuring consistency and accuracy. Administrators can create forms and templates that ensure consistent data entry and minimize the risk of errors. In contrast, Excel relies on manual input, which is prone to errors and inconsistencies.

## Views

Databases allow administrators to create different views of the data, which can be customized based on user needs. These views enable administrators to analyze data efficiently, identify trends, and make informed decisions. Excel, on the other hand, has limited views and cannot handle large datasets efficiently.

## Duplication

Databases provide mechanisms to prevent duplicate data entries, which minimize the risk of errors and inconsistencies. These mechanisms ensure that each record is unique, eliminating the risk of duplication. Excel, on the other hand, lacks built-in systems for detecting and preventing duplicate data entries, making it challenging to ensure data consistency.

## Interactive UI

Databases provide interactive user interfaces that make it easy to input, view, and modify data. These interfaces enable administrators to navigate the data efficiently, minimizing the risk of errors and inconsistencies. Excel, on the other hand, has limited user interfaces and can be challenging to navigate, especially with large datasets.

## Availability

Databases are available 24/7, providing administrators with instant access to data. This availability ensures that administrators can access data when needed, improving productivity and efficiency. Excel, on the other hand, is limited by the availability of the file, making it challenging to access data when needed.

## Recommendations

Databases can provide recommendations based on data analytics, enabling administrators to make informed decisions. These recommendations can be used to improve the admission process, identify trends, and predict outcomes. Excel, on the other hand, lacks advanced data analytics features and cannot provide recommendations based on data analysis.

# Target Users

In the context of a university application process, a database serves as a central repository for managing and storing all relevant information related to the application process. Multiple groups of people, including applicants, admission officers, faculty members, and administrative staff, rely on this database for various purposes.

To ensure that the application process runs smoothly, applicants will have access to the database to submit their applications, upload additional materials, and track their application status. Admission officers will use the database to evaluate applications, make admission decisions, and communicate with applicants. Faculty members will be able to use the database to review applicants' academic qualifications and provide recommendations. Administrative staff will use the database to manage the application process, such as scheduling interviews and campus visits.

The database will be maintained by a dedicated team of database administrators, IT professionals, and support staff. The database administrators will be responsible for designing, implementing, and maintaining the database infrastructure, ensuring its security, and optimizing its performance. This may include tasks such as creating and managing user accounts, designing the database schema, and performing regular backups and maintenance.

IT professionals will provide technical support to ensure that the database runs smoothly and is accessible to all users. They will be responsible for troubleshooting any technical issues that arise, such as problems with network connectivity or hardware failures. In addition, they will provide training and support to users who may not be familiar with the database or its functionality.

The support staff will assist users with various tasks related to the database, such as troubleshooting, data entry, and maintenance. They will also provide customer service to users who may have questions or concerns about the application process. This may involve answering phone calls or emails, scheduling appointments, and providing guidance on how to use the database.

In summary, the database will serve as an essential tool in managing the university's application process, and the dedicated team of administrators, IT professionals, and support staff will work together to ensure that it runs smoothly and efficiently.

# Database

## Schema

Database used in this project is MySQL and the important relations used are listed below,

1. Country
2. Region
3. University\_Type
4. University\_Size
5. GRE\_waiver
6. Research\_output
7. Degree\_Table
8. University\_requirements
9. Applicants\_General\_Details
10. Applicants\_Application\_Status
11. University\_admins
12. Applicants\_login

The database presented here is a relational database designed to manage information related to university admissions. It is composed of 12 tables, each with a specific purpose and containing relevant data. The "Country" table stores information about countries, including a unique identifier (CID) and the name of the country. This table is used to maintain a list of countries and associate them with universities and applicants. The "Region" table stores information about regions, including a unique identifier (RID) and the name of the region. This table is used to maintain a list of regions and associate them with universities and applicants.

The "University\_Type" table stores information about types of universities, including a unique identifier (TID) and the name of the university type. This table is used to maintain a list of university types and associate them with universities. The "University\_Size" table stores information about university sizes, including a unique identifier (SID) and the size of the university. This table is used to maintain a list of university sizes and associate them with universities. The "GRE\_waiver" table stores information about GRE waiver options, including a unique identifier (GRE\_ID) and the name of the GRE waiver option. This table is used to maintain a list of GRE waiver options and associate them with universities. The "Research\_output" table stores information about research outputs, including a unique identifier (RO\_ID) and the name of the research output. This table is used to maintain a list of research outputs and associate them with universities.

The "Degree\_Table" table stores information about degree levels, including a unique identifier (Degree\_ID) and the name of the degree level. This table is used to maintain a list of degree levels and associate them with applicants and universities. The "University\_requirements" table stores information about university requirements, including a unique identifier (UID), university name, university type, country, region, research output, university size, rank display, student-faculty ratio, GRE waiver option, GRE score, IELTS score, TOEFL score, GPA, and link to university website. This table is used to associate universities with their specific requirements. The "Applicants\_General\_Details" table stores information about applicants, including a unique identifier (Applicant\_Id), applicant name, CGPA, GRE score, TOEFL score, IELTS score, research experience, degree level, number of universities applied to, and country. This table is used to maintain a list of applicants and associate them with their specific details. The "Applicants\_Application\_Status" table stores information about an applicant's application status for a specific university, including their unique identifier (Applicant\_Id), the university's unique identifier (UID), the course name, degree level, and status of the application. This table is used to maintain a list of applicants and their application status for each university.

The "University\_admins" table stores information about university administrators, including a unique identifier (UID), email, and password. This table is used to maintain a list of university administrators and their login information. The "Applicants\_login" table stores information about applicant login credentials, including their unique identifier (Applicant\_id), email, and password. This table is used to maintain a list of applicant login credentials.

Overall, this database provides a structured and organized way to manage university admissions data, which is critical in ensuring that the application process runs smoothly and efficiently.

## Dataset

To populate the tables with data, the Python Faker module was used to generate random data that closely resembles real-world data. This ensures that the database can handle real-world scenarios and can be used for testing and analysis purposes. The generated data was then inserted into the respective tables in the database.

To make the data accessible and shareable, it was exported to CSV files with the corresponding table names. These CSV files contain all the data that was inserted into the tables, including the primary keys and foreign keys. The files are organized and labeled appropriately to ensure easy access and management. The CSV files can be easily imported into the database, making it simple to migrate the data to a different database or to create backups. Overall, storing the data in CSV files provides a simple and effective way to manage and store the data for future use.

## Milestone 1 to Milestone 2

The project involved creating a database management system for university admissions, with a focus on managing the vast amounts of data related to applicants, universities, and their requirements. The project was completed in two milestones.

In milestone 1, the initial database schema was designed and implemented. Tables were created to store data related to countries, regions, university types, university sizes, GRE waiver options, research output, degree levels, student research, result status, applicants' general details, university requirements, and applicants' application status. Each table was designed with its respective primary key and foreign keys to establish relationships with other tables.

In milestone 2, the initial database schema was refined to improve its performance and optimize its storage. Some of the changes that were made included updating the table attributes, modifying primary keys, and renaming tables. For instance, the Applicants\_General\_Details table had some attribute changes, such as CGPA and GRE, which were changed from decimal to varchar. The Applicant\_Id attribute was also changed to varchar(20) instead of INT.

New tables were also created to support the system's functionalities. The University\_admins table was added to store the login credentials of university administrators, while the Applicants\_login table was added to store the login credentials of applicants. Both tables were designed with their respective primary keys and foreign keys to establish relationships with other tables.

Overall, the project's goal was to create a robust and scalable database management system that could handle the vast amounts of data related to university admissions. The system's design and implementation were focused on ensuring data integrity, security, scalability, and performance. The project's two milestones were crucial in achieving these goals and refining the system to ensure it meets the needs of users.

## Table’s Constraints and Description

Country:

* CID: primary key representing the country code
* Country: name of the country

Region:

* RID: primary key representing the region code
* region: name of the region

University\_Type:

* TID: primary key representing the university type code
* university\_type: name of the university type (e.g. public, private, etc.)

University\_Size:

* SID: primary key representing the university size code
* Uni\_size: name of the university size category (e.g. small, medium, large)

GRE\_waiver:

* GRE\_ID: primary key representing the GRE waiver code
* GRE\_waiver\_options: options for GRE waiver (e.g. not available, available for certain applicants, etc.)

Research\_output:

* RO\_ID: primary key representing the research output code
* research\_outputs: description of the research outputs for the university (e.g. high, medium, low)

Degree\_Table:

* Degree\_ID: primary key representing the degree code
* Degree\_Level: name of the degree level (e.g. bachelor's, master's, PhD)

University\_requirements:

* UID: primary key representing the university requirements code
* Uni\_name: name of the university
* TID: foreign key referencing University\_Type table
* CID: foreign key referencing Country table
* RID: foreign key referencing Region table
* RO\_ID: foreign key referencing Research\_output table
* SID: foreign key referencing University\_Size table
* rank\_display: the ranking of the university
* student\_faculty\_ratio: the ratio of students to faculty
* GRE\_ID: foreign key referencing GRE\_waiver table
* GRE\_Score: the minimum GRE score required
* IELTS: the minimum IELTS score required
* TOEFL: the minimum TOEFL score required
* GPA: the minimum GPA required
* link: the link to the university's admission requirements webpage

Applicants\_General\_Details:

* Applicant\_Id: primary key representing the applicant's ID
* applicant\_name: name of the applicant
* CGPA: the applicant's cumulative GPA
* GRE: the applicant's GRE score
* TOEFL: the applicant's TOEFL score
* IELTS: the applicant's IELTS score
* research: whether the applicant has research experience (yes/no)
* Degree\_ID: foreign key referencing Degree\_Table table
* universities\_applied: the number of universities the applicant has applied to
* CID: foreign key referencing Country table

Applicants\_Application\_Status:

* Applicant\_Id: foreign key referencing Applicants\_General\_Details table
* UID: foreign key referencing University\_requirements table
* course\_name: the name of the course applied to
* Degree\_ID: foreign key referencing Degree\_Table table
* Status: the current status of the application

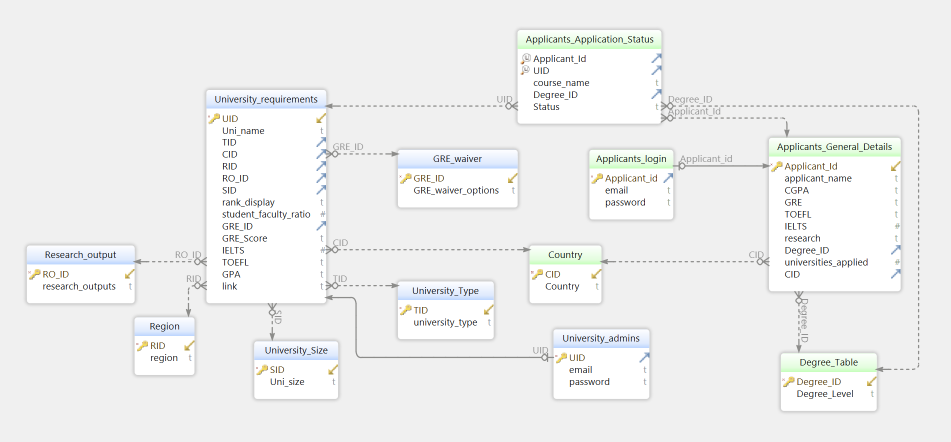
University\_admins:

* UID: primary key representing the university admin's ID
* email: the university admin's email address
* password: the university admin's password
* UID: foreign key referencing University\_requirements table

Applicants\_login:

* Applicant\_id: primary key referencing Applicants\_General\_Details table
* email: the applicant's email address
* password: the applicant's password
* Applicant\_id: foreign key referencing Applicants\_General\_Details table

# ER Diagram



# Normalization

Normalization is the process of organizing data in a database to eliminate redundancy and dependency. It involves dividing the data into smaller, more manageable tables, and establishing relationships between them to improve data consistency, integrity, and efficiency. Normalization helps prevent data anomalies and inconsistencies, making it easier to maintain and modify the database over time.

For a table to be in BCNF (Boyce-Codd Normal Form), it must satisfy the following conditions:

1. Every determinant (i.e., attribute that determines another attribute) must be a candidate key.
2. Every non-trivial functional dependency (i.e., dependency between non-prime attributes) must have a determinant that is a candidate key.

Let's analyze each table to determine whether it satisfies these conditions and can be in BCNF:

1. Country (CID - >Country)
2. Region (RID -> region)
3. University\_Type (TID -> university\_type)
4. University\_Size (SID -> Uni\_size)
5. GRE\_waiver (GRE\_ID -> GRE\_waiver\_options)
6. Research\_output (RO\_ID -> research\_outputs)
7. Degree\_Table (Degree\_ID -> Degree\_Level)

All of the tables mentioned above are in BCNF because they satisfy both conditions of BCNF. Specifically, each table has a single determinant or primary key that uniquely identifies each record, and there are no non-trivial functional dependencies in which a non-prime attribute depends on another non-prime attribute through a prime attribute. This means that all the tables are well-structured and free from data redundancy or update anomalies, which makes them suitable for efficient and accurate data storage and retrieval.

1. University\_requirements (UID -> Uni\_name, TID, CID, RID, RO\_ID, SID, rank\_display, student\_faculty\_ratio, GRE\_ID, GRE\_Score, IELTS, TOEFL, GPA, link):

The primary key of this table is UID, which uniquely determines all the other attributes. Therefore, it satisfies the first condition of BCNF. To determine if it satisfies the second condition, we need to check for non-trivial functional dependencies. The following dependencies exist:

* (UID -> Uni\_name, TID, CID, RID, RO\_ID, SID, rank\_display, student\_faculty\_ratio, GRE\_ID, GRE\_Score, IELTS, TOEFL, GPA, link)
* (CID -> Country)
* (RID -> region)
* (TID -> university\_type)
* (SID -> Uni\_size)
* (RO\_ID -> research\_outputs)
* (GRE\_ID -> GRE\_waiver\_options)

All of the above dependencies are trivial or depend on a candidate key. Therefore, the University\_requirements table satisfies both conditions and is in BCNF.

1. Applicants\_General\_Detail:(Applicant\_Id-> applicant\_name, CGPA, GRE, TOEFL, IELTS, research, Degree\_ID, universities\_applied, CID):

The primary key of this table is Applicant\_Id, which uniquely determines all the other attributes. Therefore, it satisfies the first condition of BCNF. To determine if it satisfies the second condition, we need to check for non-trivial functional dependencies. The following dependencies exist:

* (Applicant\_Id -> applicant\_name, CGPA, GRE, TOEFL, IELTS, research, Degree\_ID, universities\_applied, CID)
* (CID -> Country)
* (Degree\_ID -> Degree\_Level)

All of the above dependencies are trivial or depend on a candidate key. Therefore, the Applicants\_General\_Details table satisfies both conditions and is in BCNF.

1. Applicants\_Application\_Status ((Applicant\_Id, UID) -> course\_name, Degree\_ID, Status):

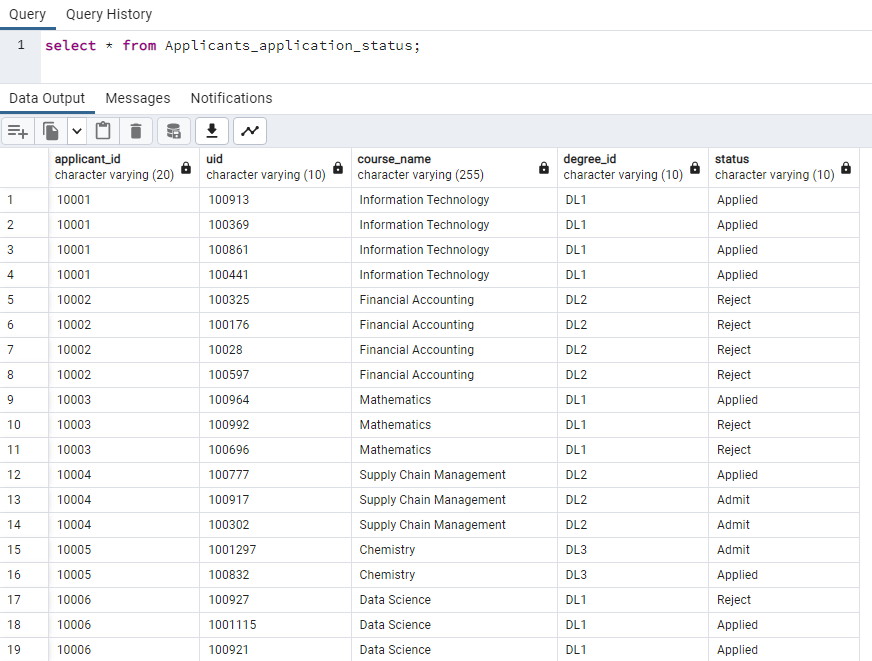
The primary key of this table is the composite key (Applicant\_Id, UID), which uniquely determines all the other attributes. Therefore, it satisfies the first condition of BCNF. To determine if it satisfies the second condition, we need to check for non-trivial functional dependencies. The following dependencies exist:

* (Applicant\_Id, UID -> course\_name, Degree\_ID, Status)
* (Applicant\_Id -> applicant\_name, CGPA, GRE, TOEFL, IELTS, research, Degree\_ID, universities\_applied, CID)
* (UID -> Uni\_name, TID, CID, RID, RO\_ID, SID, rank\_display, student\_faculty\_ratio, GRE\_ID, GRE\_Score, IELTS, TOEFL, GPA, link)
* (CID -> Country)
* (RID -> region)
* (TID -> university\_type)
* (SID -> Uni\_size)
* (RO\_ID -> research\_outputs)
* (GRE\_ID -> GRE\_waiver\_options)
* (Degree\_ID -> Degree\_Level)

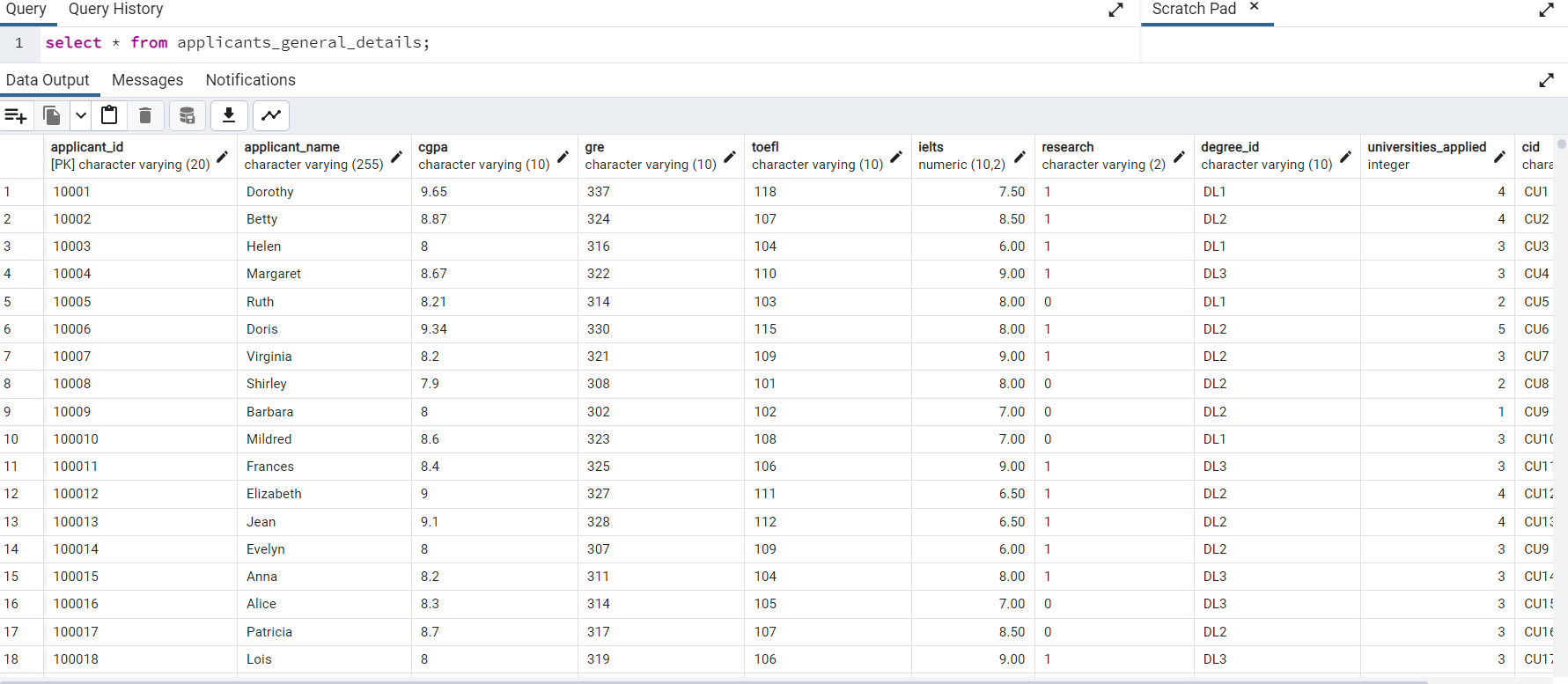
All of the above dependencies are either trivial or depend on a candidate key. Therefore, the Applicants\_Application\_Status table satisfies both conditions and is in BCNF.

1. University\_admins (UID -> email, password): This table has only one determinant (UID), which is also the primary key. Therefore, it satisfies both conditions and is already in BCNF.
2. Applicants\_login (Applicant\_Id -> email, password): Similar to the University\_admins table, this table has only one determinant (Applicant\_Id), which is also the primary key. Therefore, it satisfies both conditions and is already in BCNF.

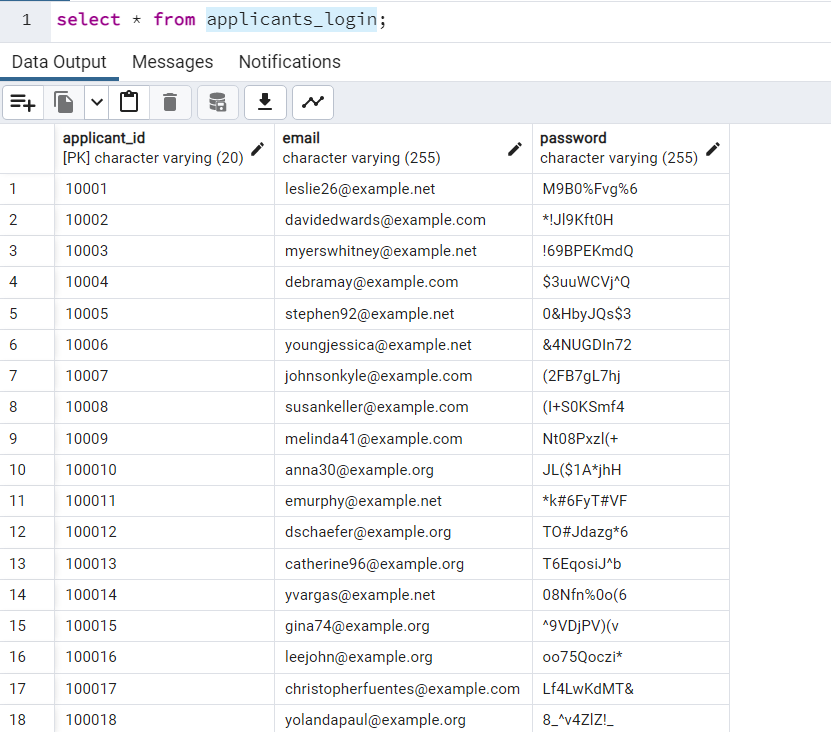
# Database Lookup

Applicants\_application\_status table:

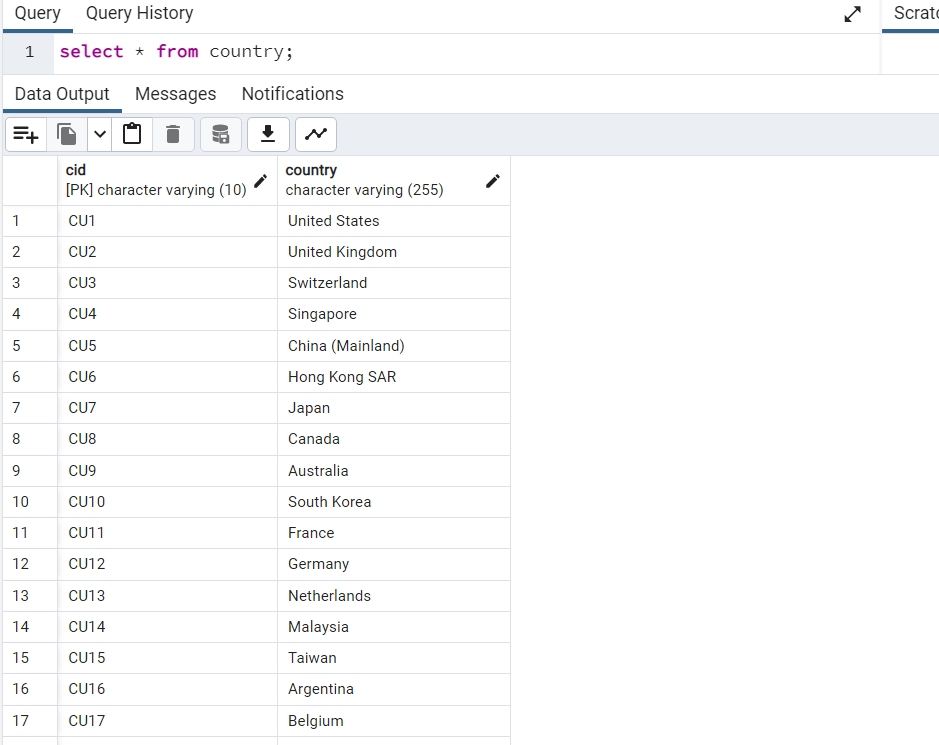
##### applicants\_general\_details table



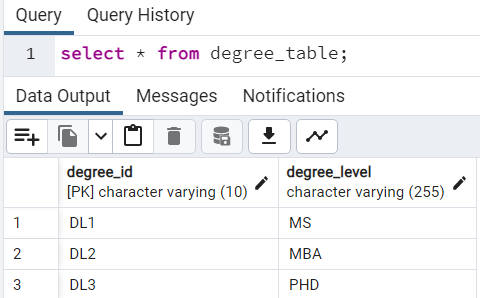
applicants\_login table:



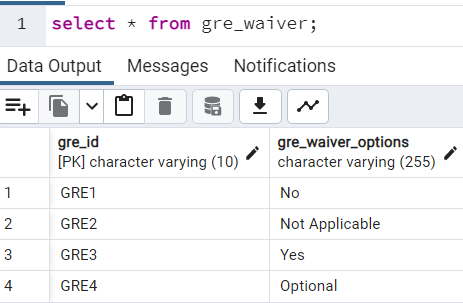
Country Table



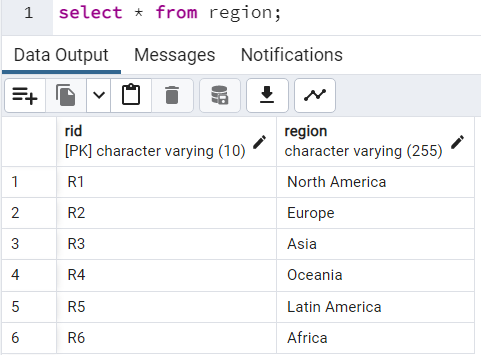
Degree Table



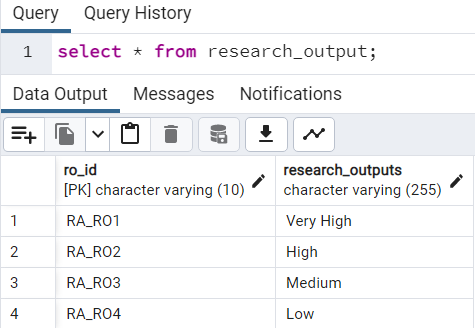
gre waiver table



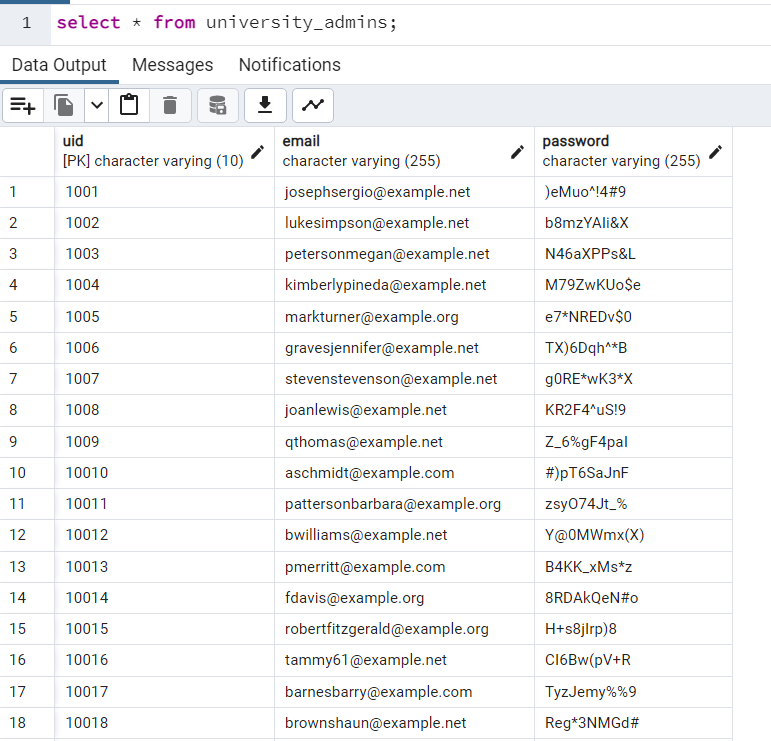
region table



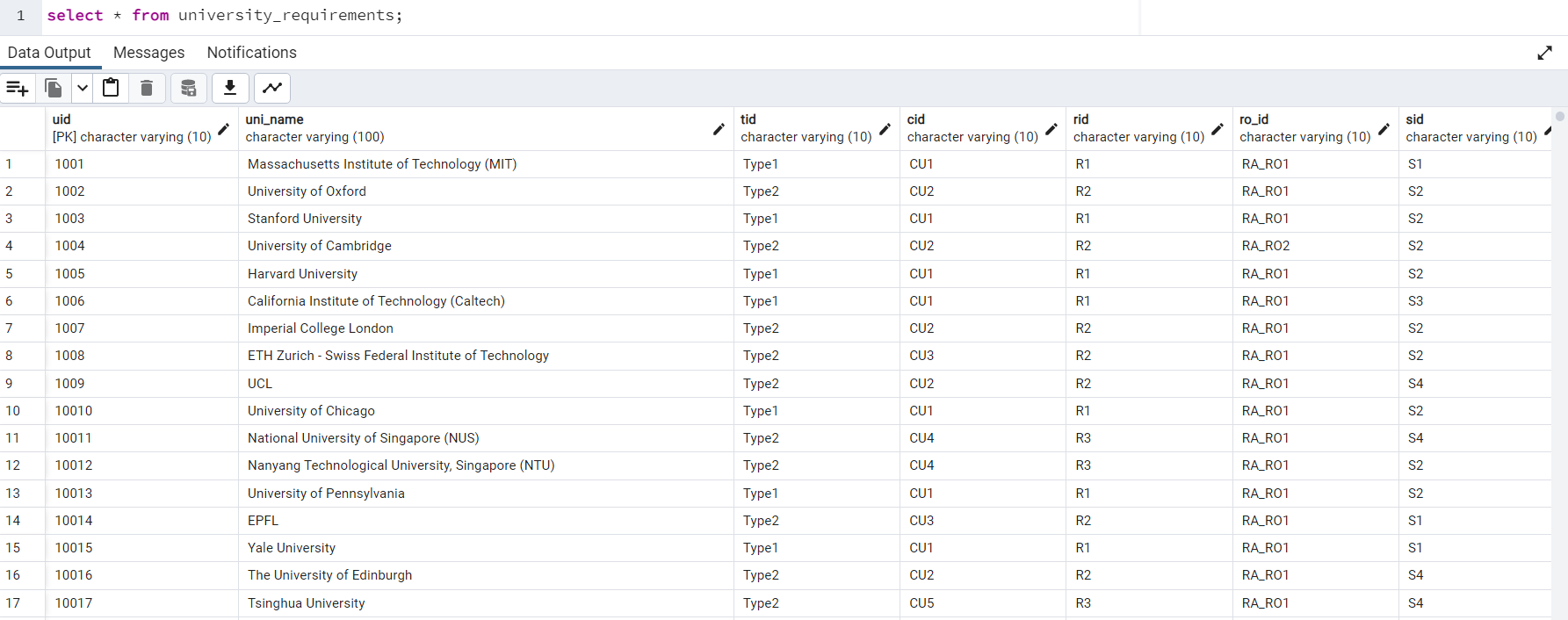
research ouput



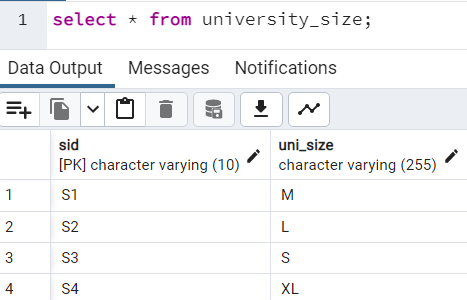
university admins



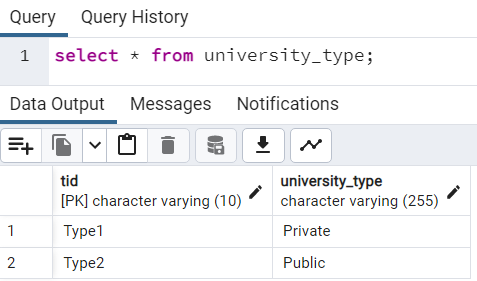
university requirements



university size

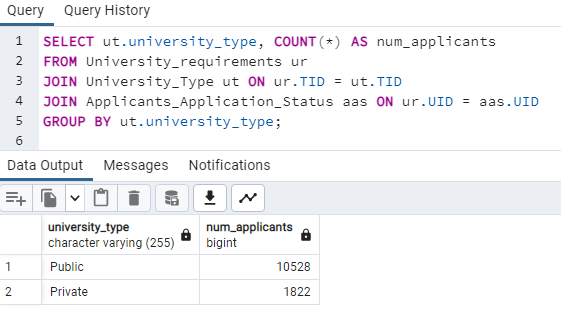


university type

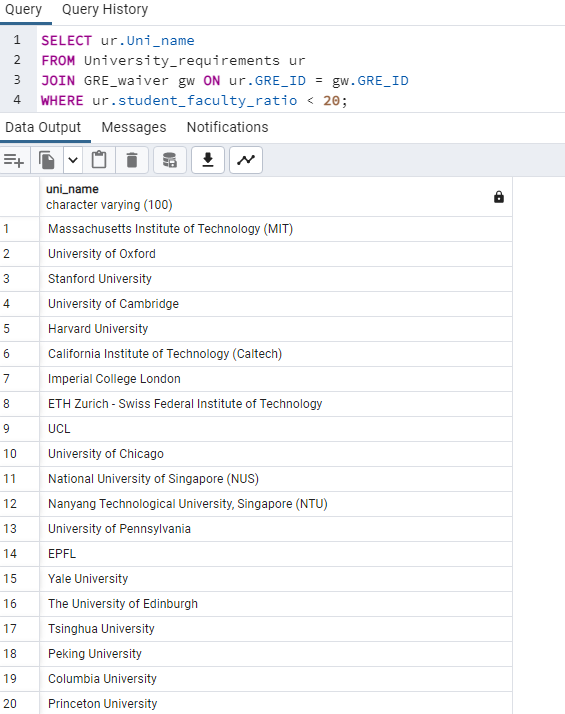


# Query Executions

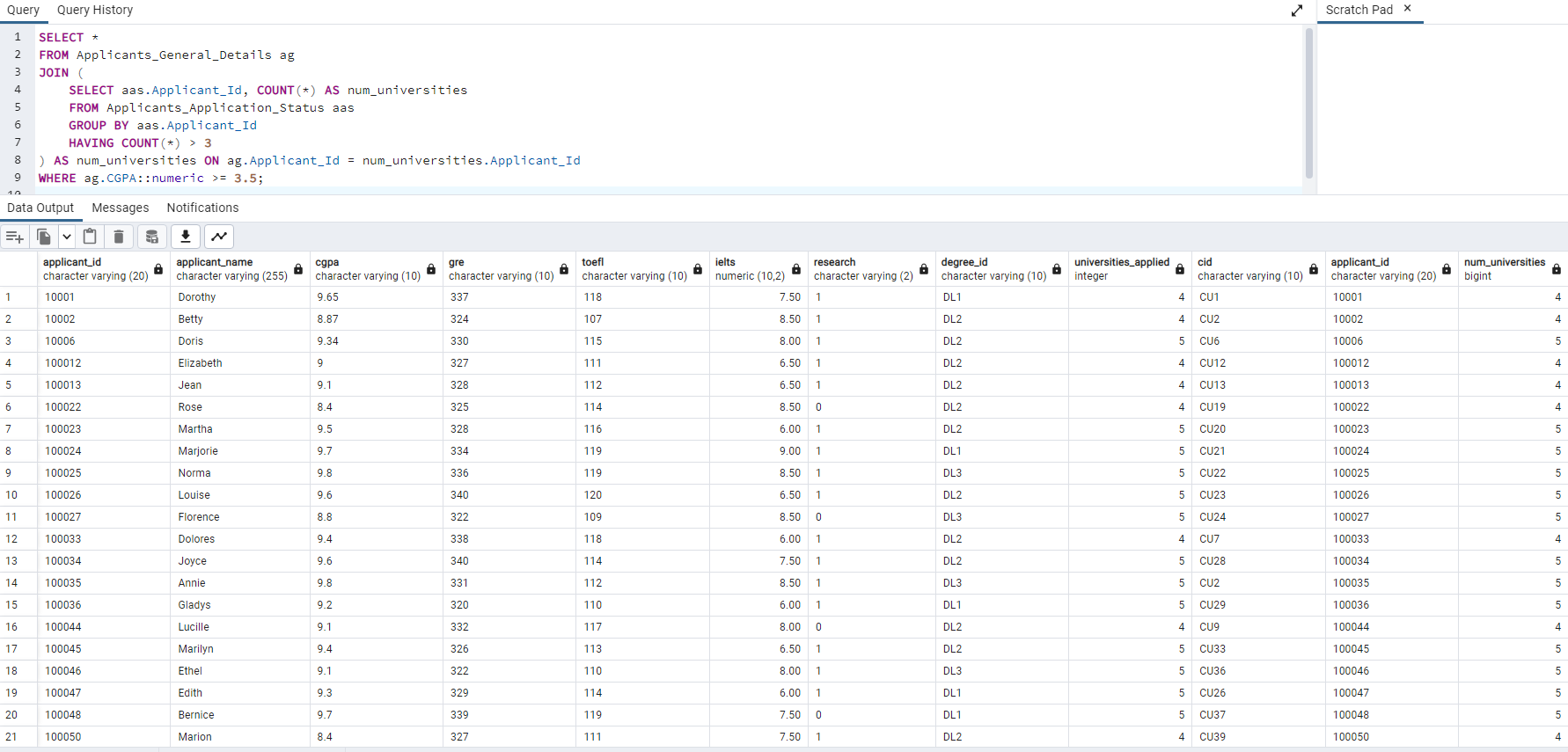
1. Find the number of applicants who have applied for each university type:



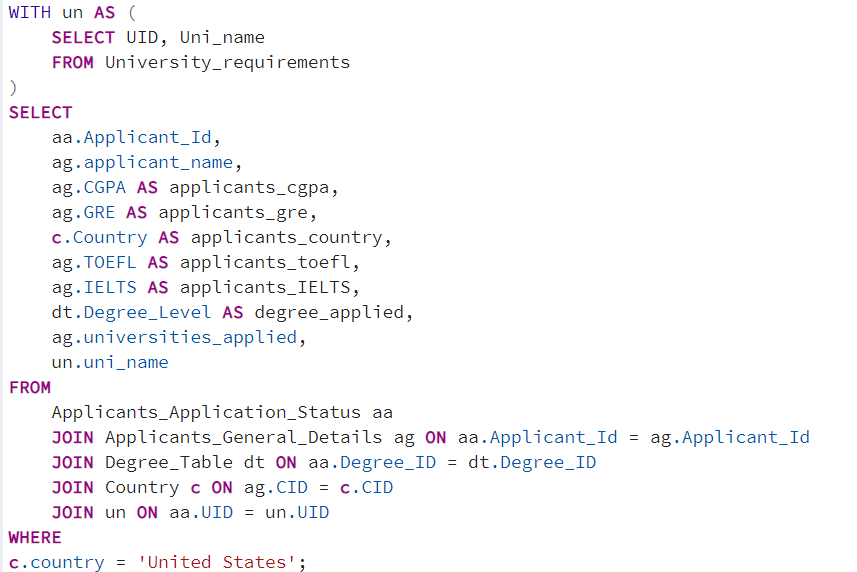
1. Find the universities that offer GRE waivers and have a student-faculty ratio less than 20



1. Find the applicants who applied to more than 3 universities and have a CGPA of 3.5 or higher:



1. Select Query Query

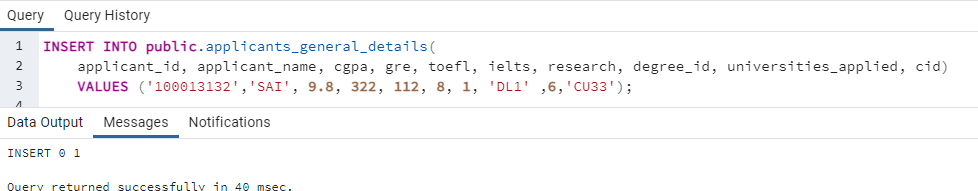


This query can be useful for getting information on applicants who have applied to universities in the United States, and their relevant details such as test scores and degrees applied for.

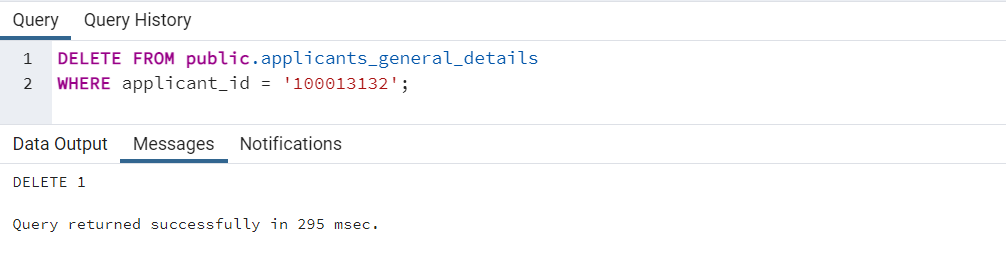
1. Update Query



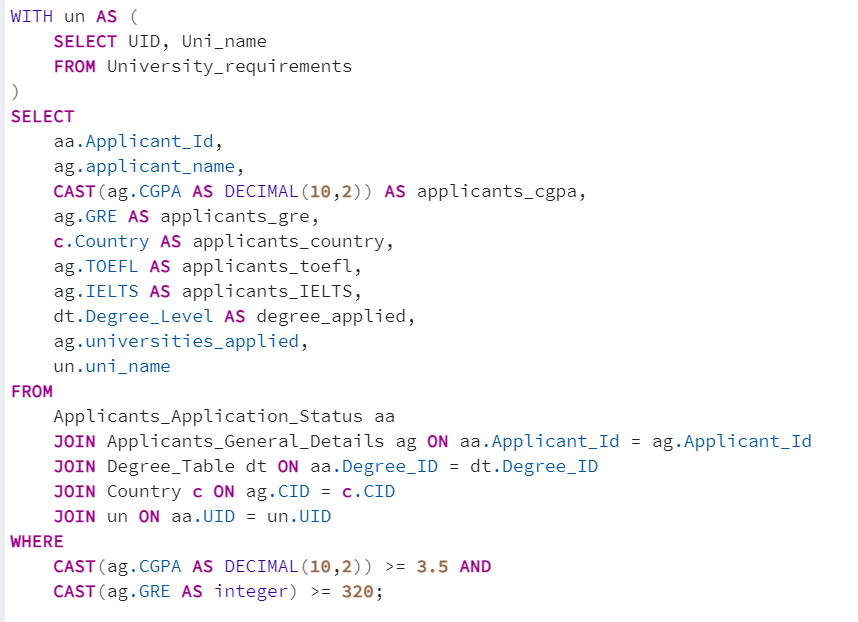
1. Insert query

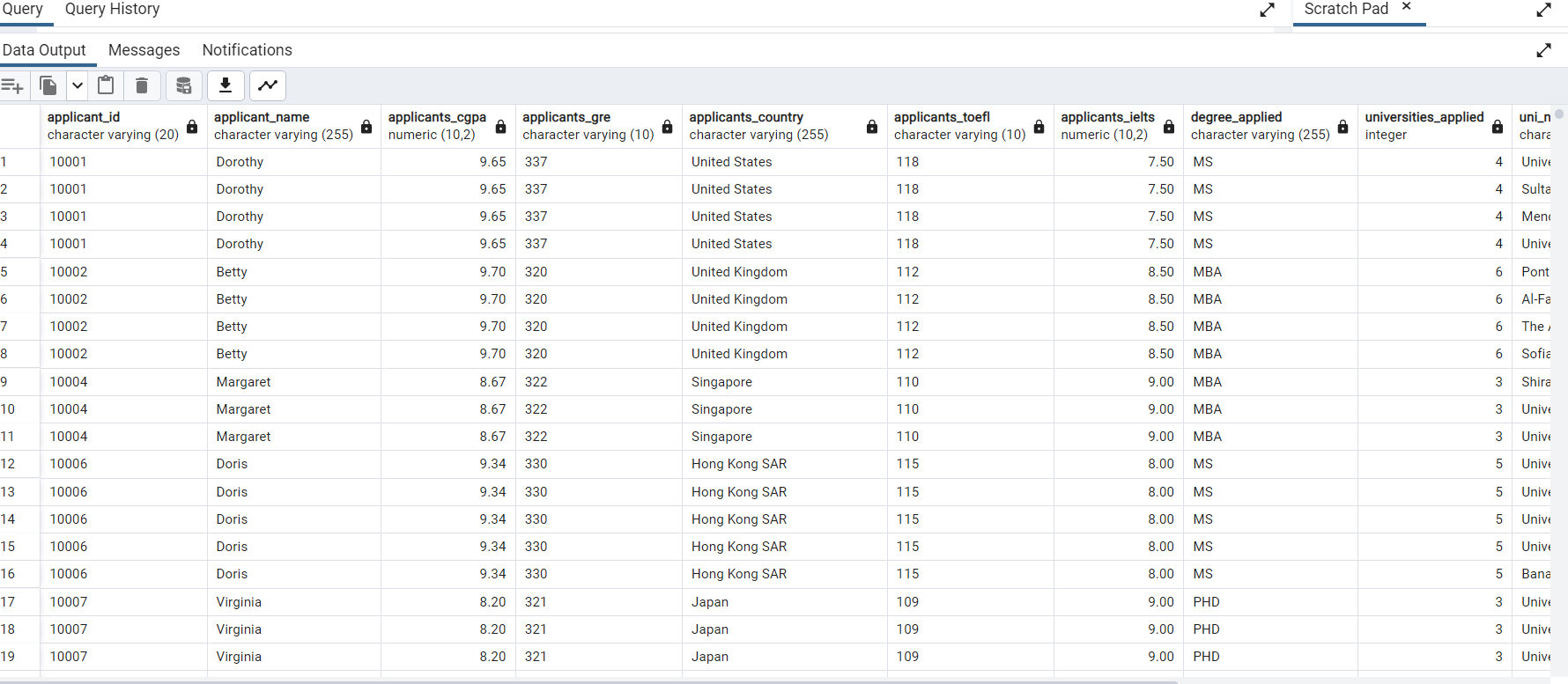


1. Delete query



1. The query is joining multiple tables to retrieve applicant details, including their CGPA, GRE score, TOEFL score, IELTS score, and the universities they applied to. It also filters the results to only include applicants with a CGPA of 3.5 or higher and a GRE score of 320 or higher.





1. Creating Temporary Table for the website Fetching:

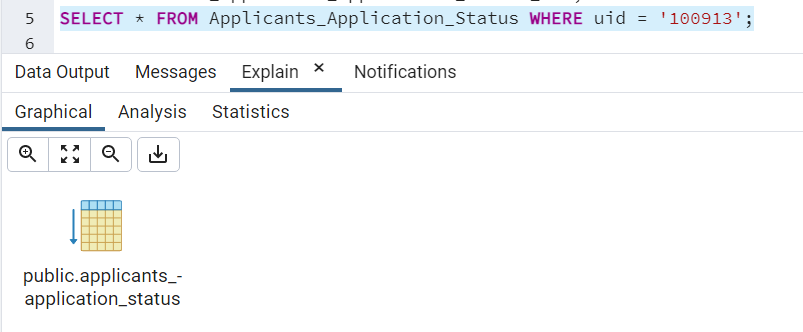


Once this statement is executed, a temporary table named applicants\_status is created with the same columns and data types as the result of the query. This table can be used within the current session or transaction for further queries, and will be automatically dropped once the session or transaction ends.

Using temporary tables can be useful for web applications when you need to perform complex queries or intermediate calculations that require temporary storage of data. It can also be used to improve performance by reducing the number of times a query needs to be executed. However, it's important to note that temporary tables should be used with caution and not relied on as a long-term storage solution.

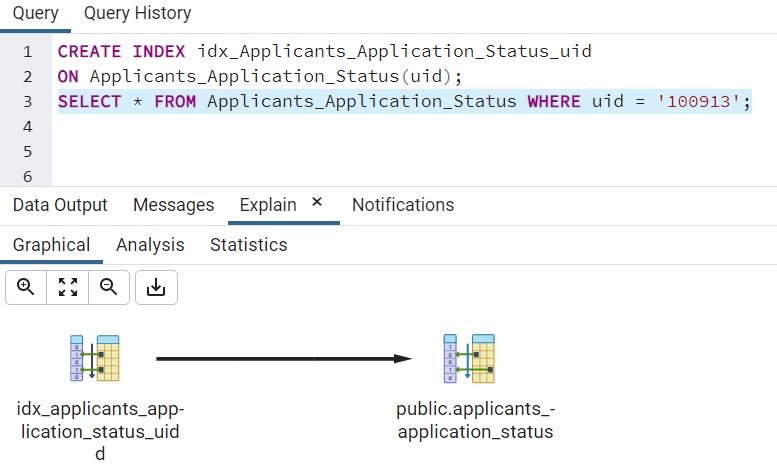
# Indexing and Tiggers (Complex queries)

Select query without Indexing

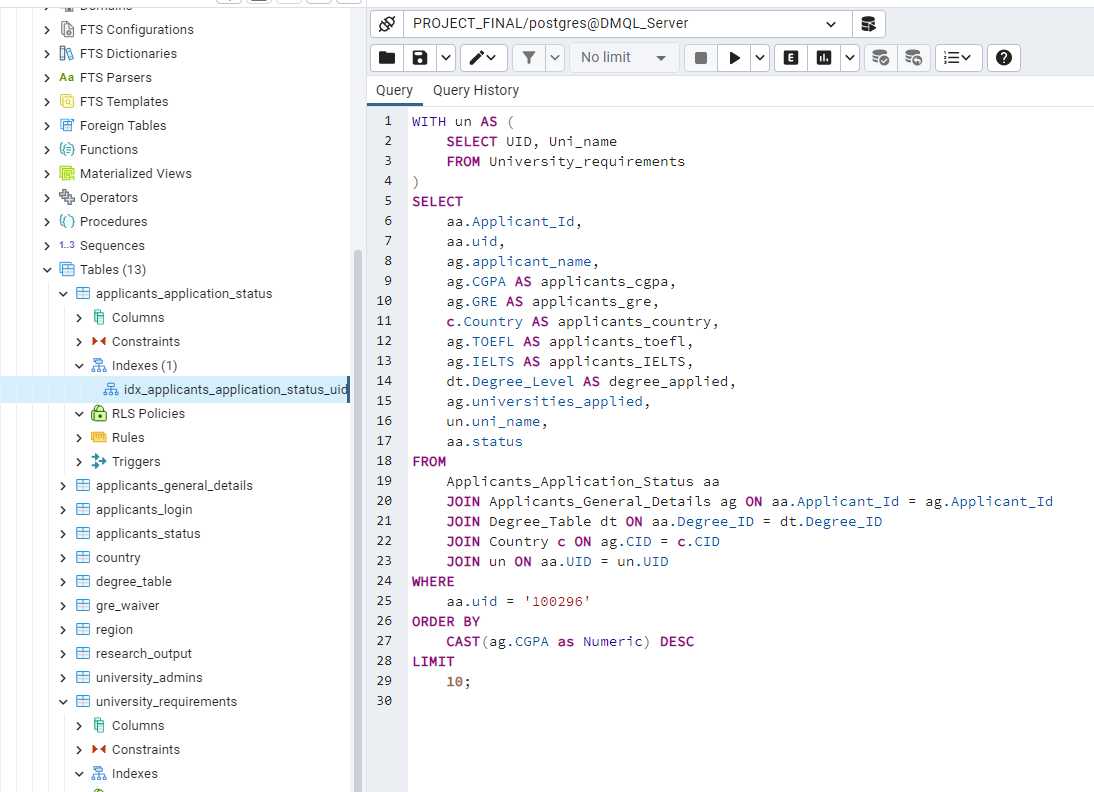


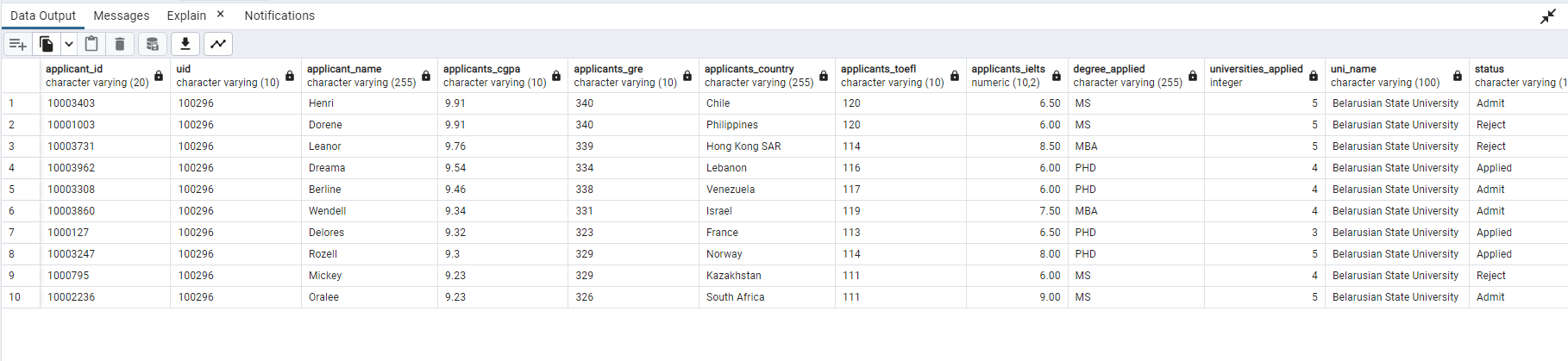


With Indexing:









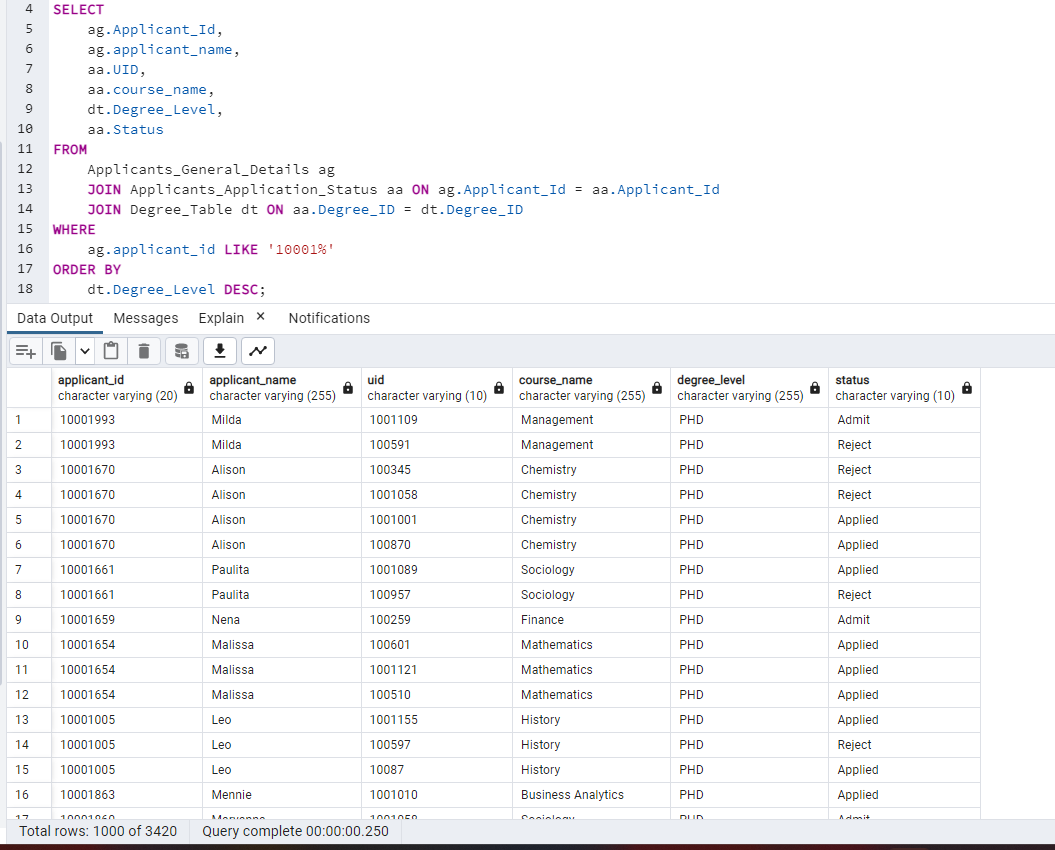
Without indexing, the query has to scan the entire table Applicants\_Application\_Status to find the rows that match the condition aa.uid = '100296'. This can be slow if the table is large and there are many rows that match the condition.

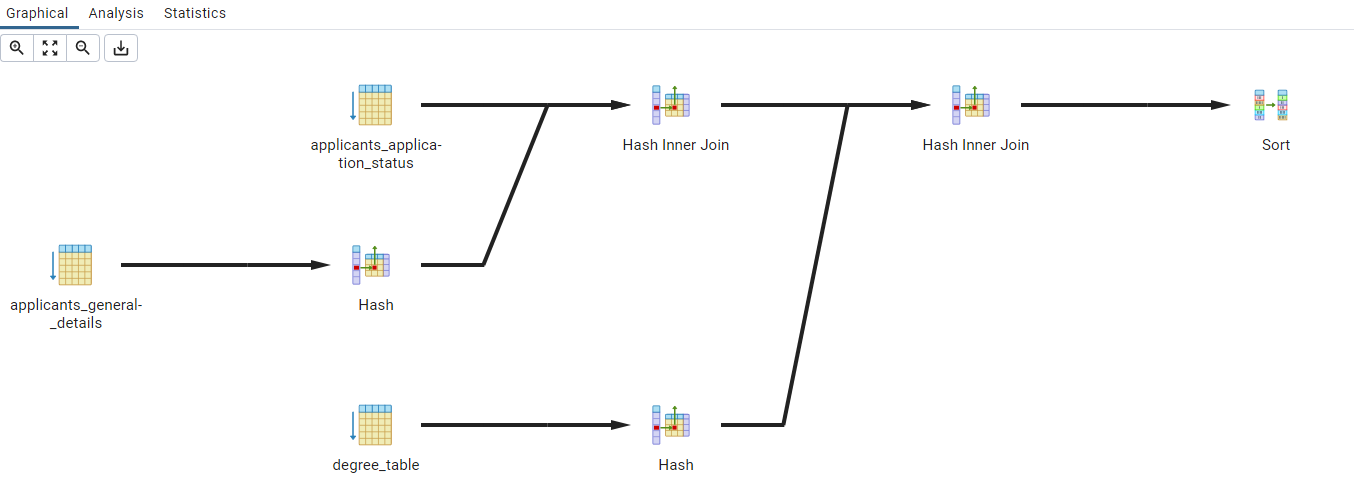
However, if an index is created on the column uid in the Applicants\_Application\_Status table, the database can use the index to quickly locate the rows that match the condition, instead of scanning the entire table. This can significantly improve the performance of the query.

Therefore, by using indexing for the uid column, the query can execute much faster as it does not have to scan the entire table to find the matching rows.

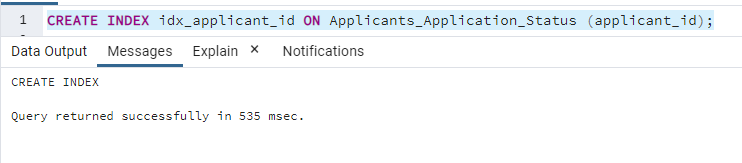
Index-2

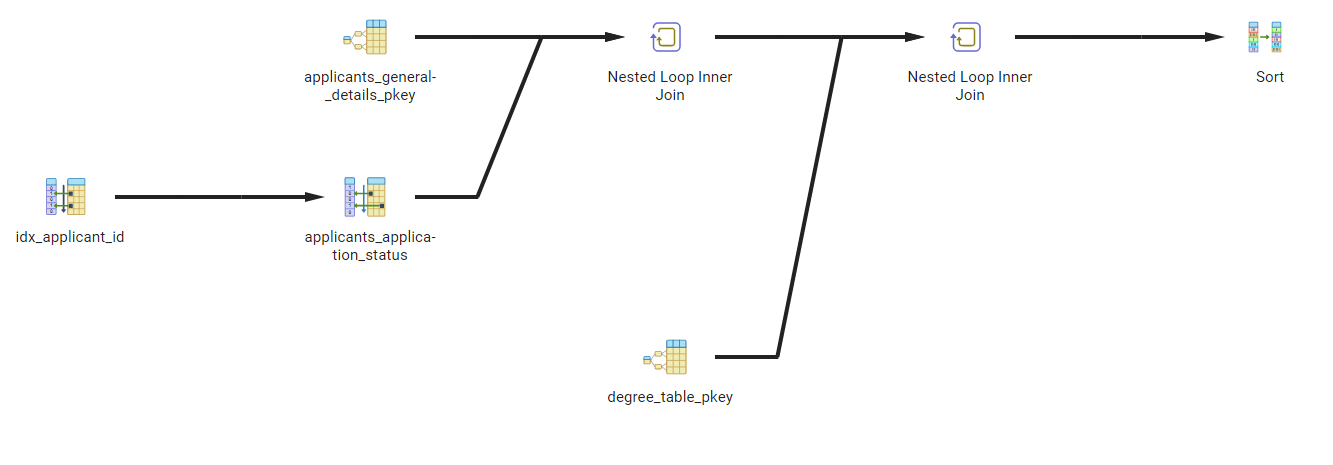
Running without Indexing:

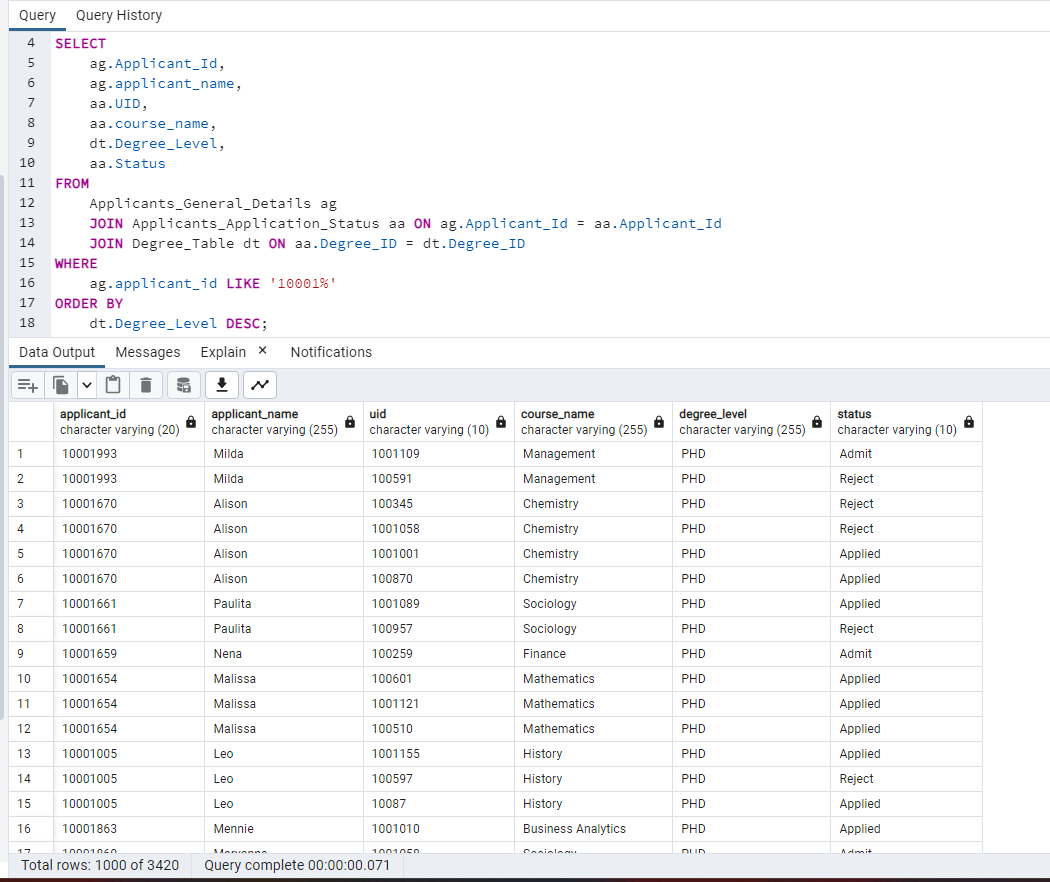




Running with Indexing:





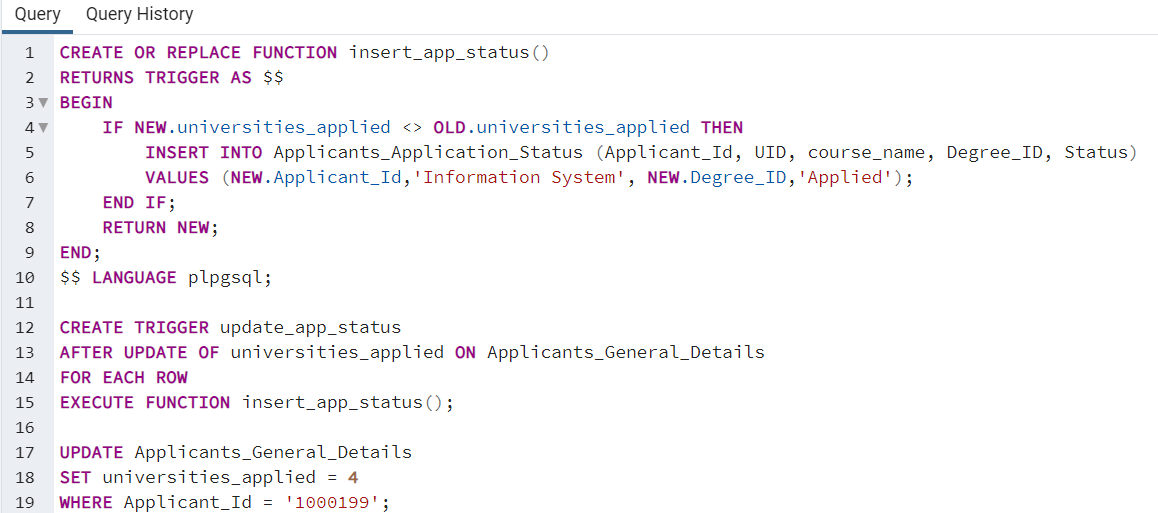


With the indexing on the Applicants\_Application\_Status table for the applicant\_id column, the database can efficiently search for rows that match the condition in the WHERE clause (ag.applicant\_id LIKE '10001'). The index allows the database to quickly locate the rows that satisfy the condition and retrieve the necessary data from the other tables. This results in faster query execution time.

Without the index, the database has to scan the entire Applicants\_Application\_Status table to find the matching rows, which can be time-consuming and resource-intensive, especially for larger tables. This can result in slower query execution time.

In summary, the index on the applicant\_id column improves the performance of the query by allowing the database to quickly find the relevant rows, while the lack of an index can lead to slower query execution time due to the need for a full table scan.

Tiggers:

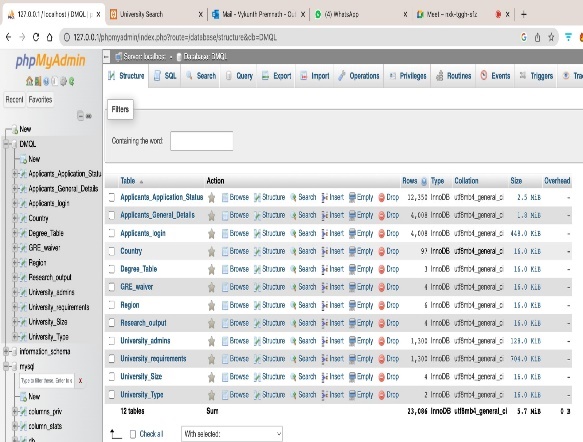


This code creates a trigger named update\_app\_status that is executed after any update to the universities\_applied attribute in the Applicants\_General\_Details table.

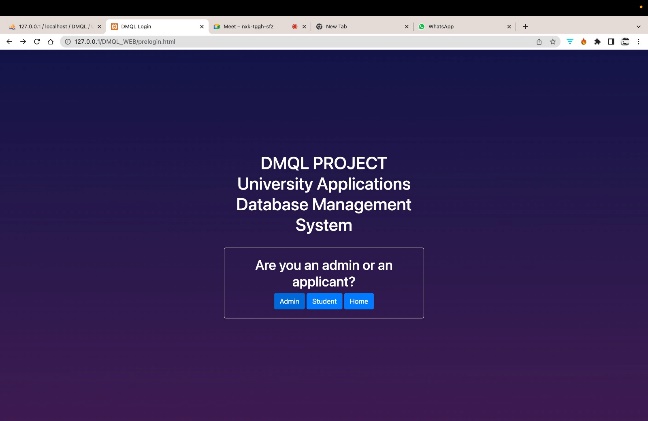
The trigger calls a PL/pgSQL function named insert\_app\_status which first checks if the value of the universities\_applied attribute has been modified. If it has been

modified, then the trigger inserts a new row in the Applicants\_Application\_Status table with the Applicant\_Id, UID, course\_name, Degree\_ID, and Status attributes of the updated row in Applicants\_General\_Details

# Website Screenshots

**Phpmyadmin for dataset**

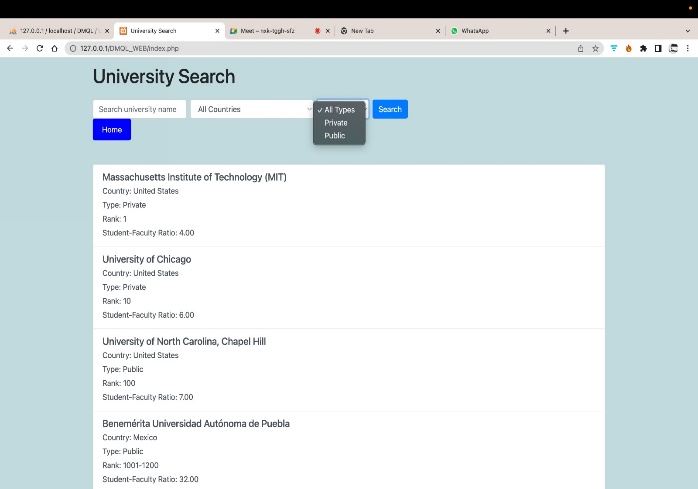
**Prelogin:** <http://127.0.0.1/DMQL_WEB/prelogin.html>

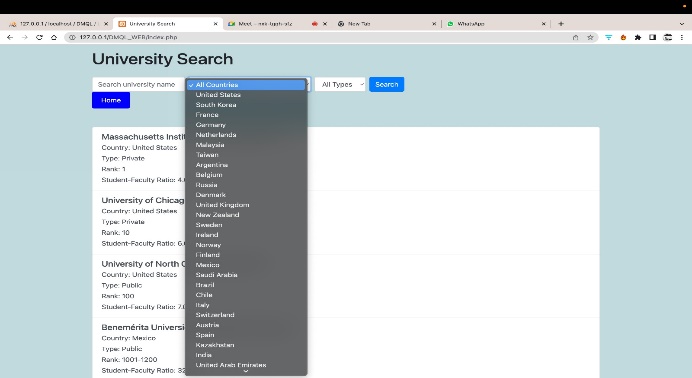


1st Page

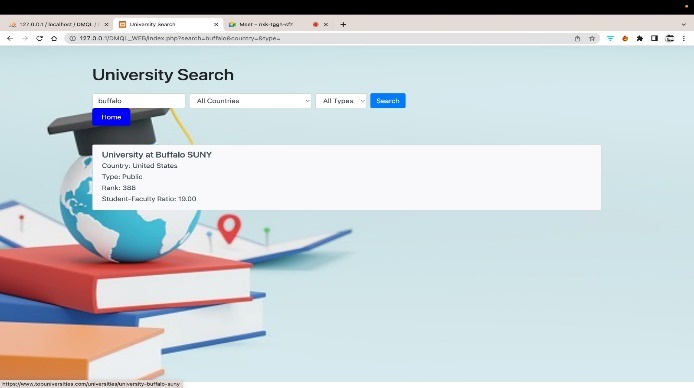
This page contains a function that redirects the user to the appropriate page based on their selection. If the user selects "applicant," they will be redirected to the applicant page. The same function is executed for the "admit" and "home" options, redirecting the user to the respective pages.

Home Page: http://127.0.0.1/DMQL\_WEB/index.php





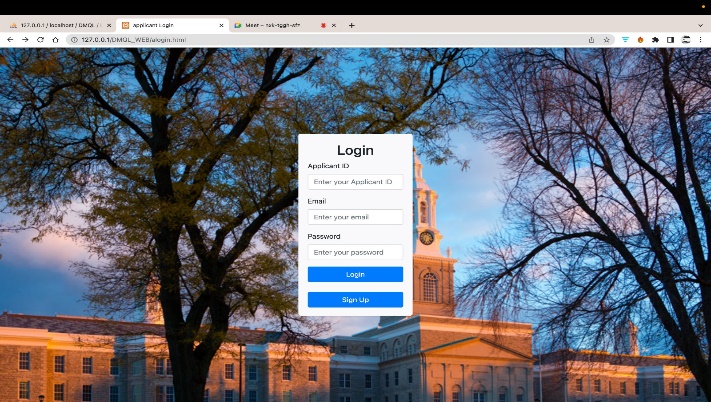
This page is dedicated to providing comprehensive information about universities. Users can utilize the three filter options to search for universities based on their name, country, and type. The search function allows for quick and efficient access to specific university details.



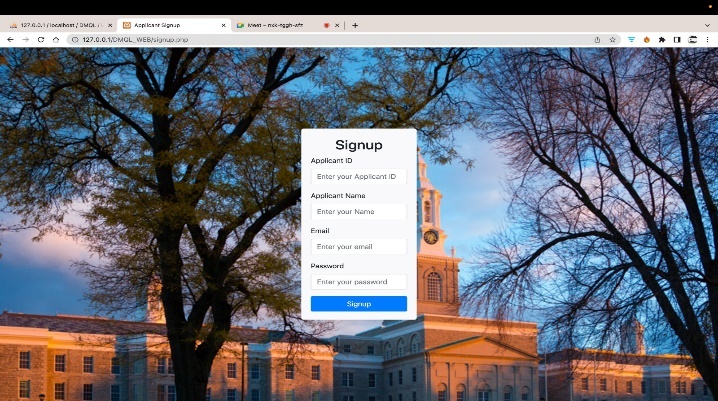
By clicking the university details card it will take you to the top-universities webpage.

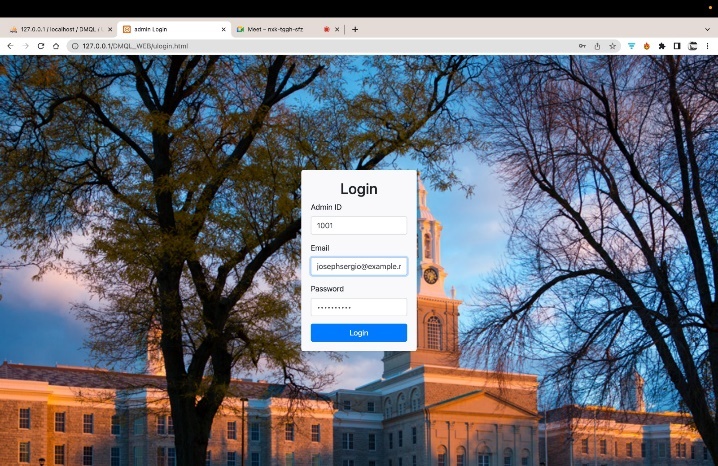
If you click on the "Home" button, you will be redirected to the first page.

Login/ Sign up : <http://127.0.0.1/DMQL_WEB/login.html>



Login or Sign-up page





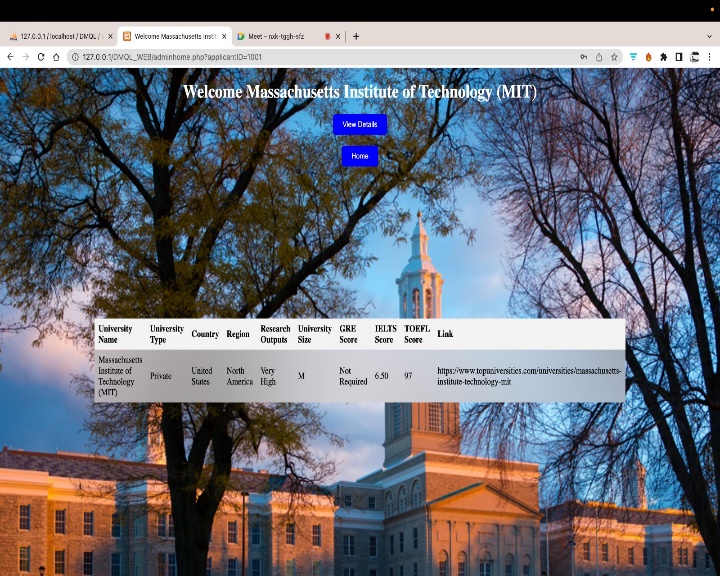
Admin login page <http://127.0.0.1/DMQL_WEB/ulogin.html>

Sample Data:

admin: 1001

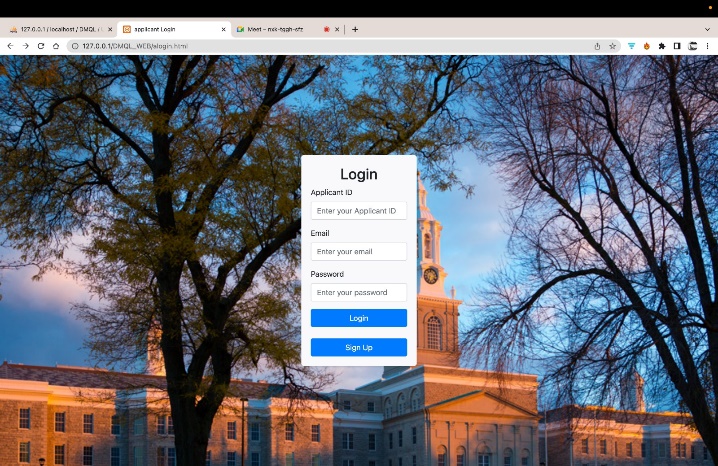
email: josephsergio@example.net

Password: )eMuo^!4#9



Next to the Admin Login Page

<http://127.0.0.1/DMQL_WEB/adminhome.php>



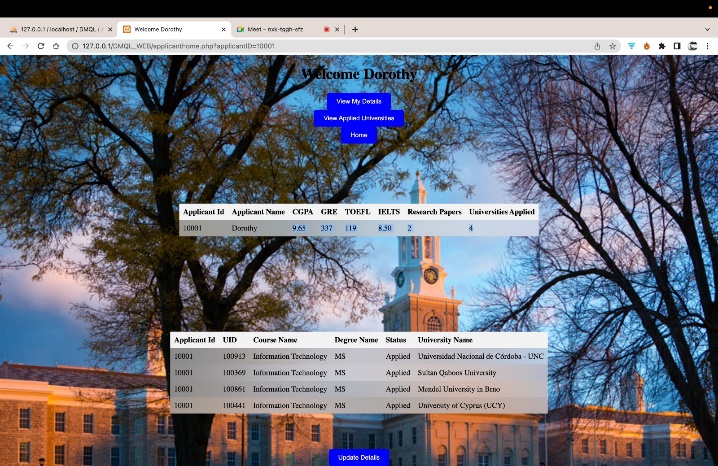
<http://127.0.0.1/DMQL_WEB/alogin.html>

Applicant Login

Applicant\_id:10001

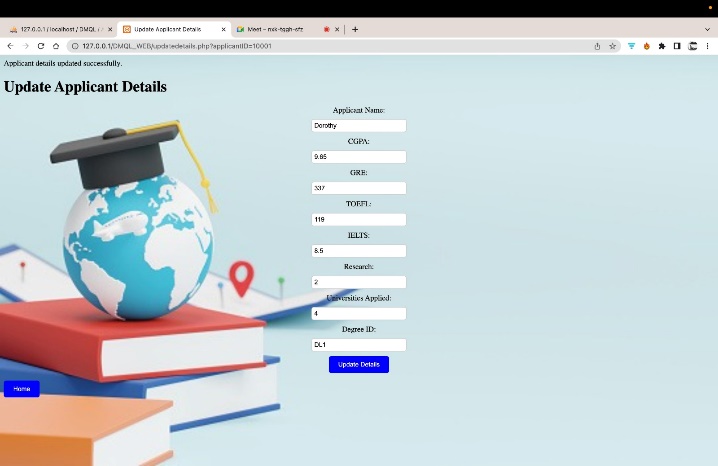
Email :v@b.edu

Password: 1234



http://127.0.0.1/DMQL\_WEB/applicanthome.php

To modify your information, click on the "Update" button. If you have just created a new account, please use the "Update" feature to input your personal details.



# References

1. <https://faker.readthedocs.io/en/master/>
2. <https://www.php.net/docs.php>
3. <https://www.kaggle.com/datasets/mylesoneill/world-university-rankings?resource=download&select=timesData.csv4>.
4. Database systems, Ulman
5. Professor’s lecture slides - Functional dependency and BCNF normalization