# *University Applications Database Management System*

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

## I. 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..

## II. 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.

### *A. 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.

### *B. 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.

### *C. 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.

### D. 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.

### E. 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.

### F. 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.

### G. 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.

### H. 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.

### I. 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.

### J. 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.

## III. 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.

## IV. Database

### A. 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.

### *B. 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.

### *C. 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.

### *D. 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

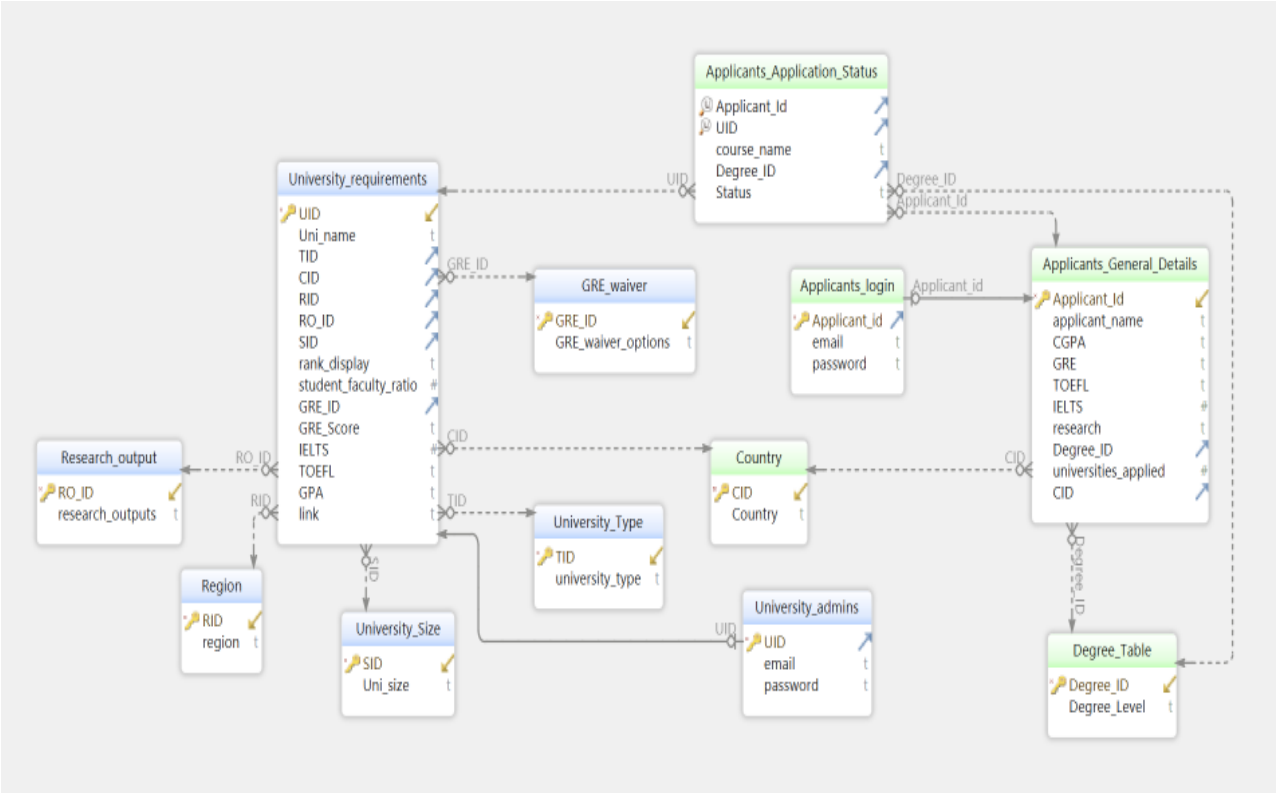
University_admins:

- 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

## V. ER DIAGRAM




## VI. 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.

8. 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.

9. 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.

10. 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:

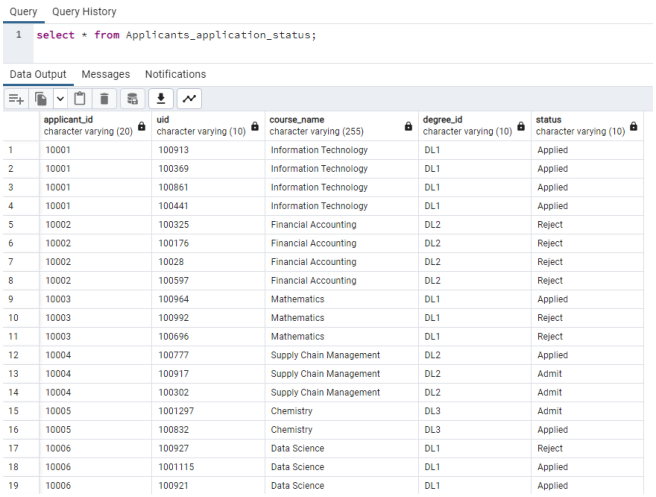
- (Applicant_Id, UID -> course_name, Degree_ID, Status)
- (Applicant_Id -> applicant_name, CGPA, GRE, TOEFL, IELTS, research, Degree_ID, universities_applied, CID)
- (UID -> Uni_name, TID, CID, RID, RO_ID, SID, rank_display, student_faculty_ratio, GRE_ID, GRE_Score, IELTS, TOEFL, GPA, link)
- (CID -> 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.

11. 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.
12. 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.

## VII. Database Lookup

APPLICANTS APPLICATION STATUS TABLE:

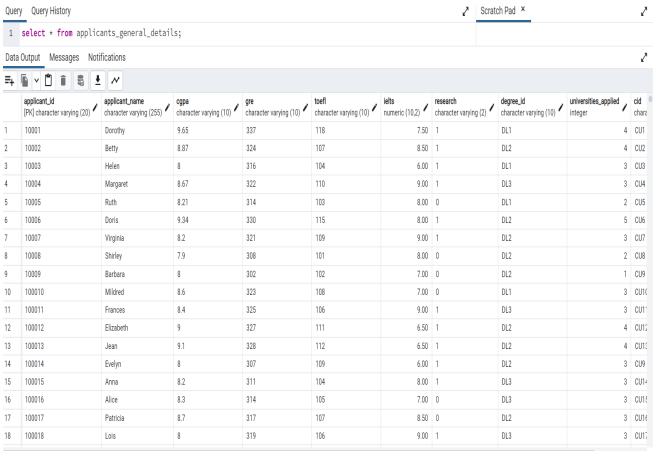

Query Query History

```
select * from Applicants_application_status;
```

Data Output Messages Notifications

| | applicant_id character varying (20) | uid character varying (10) | course_name character varying (255) | degree_id character varying (10) | status character varying (10) |
|---|---|---|---|---|---|
| 1 | 10001 | 100913 | Information Technology | DL1 | Applied |
| 2 | 10001 | 100369 | Information Technology | DL1 | Applied |
| 3 | 10001 | 100861 | Information Technology | DL1 | Applied |
| 4 | 10001 | 100441 | Information Technology | DL1 | Applied |
| 5 | 10002 | 100325 | Financial Accounting | DL2 | Reject |
| 6 | 10002 | 100176 | Financial Accounting | DL2 | Reject |
| 7 | 10002 | 10028 | Financial Accounting | DL2 | Reject |
| 8 | 10002 | 100597 | Financial Accounting | DL2 | Reject |
| 9 | 10003 | 100964 | Mathematics | DL1 | Applied |
| 10 | 10003 | 100992 | Mathematics | DL1 | Reject |
| 11 | 10003 | 100696 | Mathematics | DL1 | Reject |
| 12 | 10004 | 100777 | Supply Chain Management | DL2 | Applied |
| 13 | 10004 | 100917 | Supply Chain Management | DL2 | Admit |
| 14 | 10004 | 100302 | Supply Chain Management | DL2 | Admit |
| 15 | 10005 | 1001297 | Chemistry | DL3 | Admit |
| 16 | 10005 | 100832 | Chemistry | DL3 | Applied |
| 17 | 10006 | 100927 | Data Science | DL1 | Reject |
| 18 | 10006 | 1001115 | Data Science | DL1 | Applied |
| 19 | 10006 | 100921 | Data Science | DL1 | Applied |

APPLICANTS_GENERAL_DETAILS TABLE

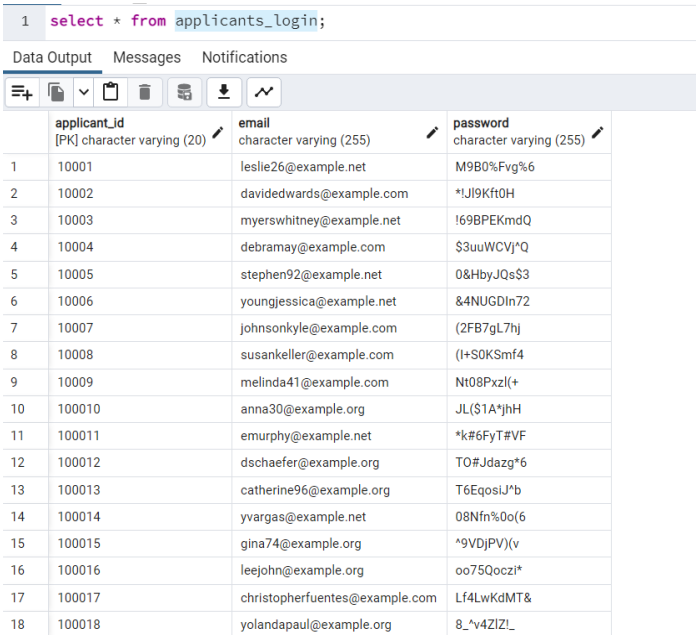

Query Query History Scratch Pad

```
select * from applicants_general_details;
```

Data Output Messages Notifications

| | applicant_id [PK] character varying (20) | applicant_name character varying (255) | cgpa character varying (10) | gre character varying (10) | toefl character varying (10) | ielts numeric (10,2) | research character varying (2) | degree_id character varying (10) | universities_applied integer | cid |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10001 | Dorothy | 9.65 | 337 | 118 | 7.50 | 1 | DL1 | 4 | CU1 |
| 2 | 10002 | Betty | 8.87 | 324 | 107 | 8.50 | 1 | DL2 | 4 | CU2 |
| 3 | 10003 | Helen | 8 | 316 | 104 | 6.00 | 1 | DL1 | 3 | CU3 |
| 4 | 10004 | Margaret | 8.67 | 322 | 110 | 9.00 | 1 | DL3 | 3 | CU4 |
| 5 | 10005 | Ruth | 8.21 | 314 | 103 | 8.00 | 0 | DL1 | 2 | CU5 |
| 6 | 10006 | Doris | 9.34 | 330 | 115 | 8.00 | 1 | DL2 | 5 | CU6 |
| 7 | 10007 | Virginia | 8.2 | 321 | 109 | 9.00 | 1 | DL2 | 3 | CU7 |
| 8 | 10008 | Shirley | 7.9 | 308 | 101 | 8.00 | 0 | DL2 | 2 | CU8 |
| 9 | 10009 | Barbara | 8 | 302 | 102 | 7.00 | 0 | DL2 | 1 | CU9 |
| 10 | 100010 | Mildred | 8.6 | 323 | 108 | 7.00 | 0 | DL1 | 3 | CU10 |
| 11 | 100011 | Frances | 8.4 | 325 | 106 | 9.00 | 1 | DL3 | 3 | CU11 |
| 12 | 100012 | Elizabeth | 9 | 327 | 111 | 6.50 | 1 | DL2 | 4 | CU12 |
| 13 | 100013 | Jean | 9.1 | 328 | 112 | 6.50 | 1 | DL2 | 4 | CU13 |
| 14 | 100014 | Evelyn | 8 | 307 | 109 | 6.00 | 1 | DL2 | 3 | CU9 |
| 15 | 100015 | Anna | 8.2 | 311 | 104 | 8.00 | 1 | DL3 | 3 | CU14 |
| 16 | 100016 | Alice | 8.3 | 314 | 105 | 7.00 | 0 | DL3 | 3 | CU15 |
| 17 | 100017 | Patricia | 8.7 | 317 | 107 | 8.50 | 0 | DL2 | 3 | CU16 |
| 18 | 100018 | Lois | 8 | 319 | 106 | 9.00 | 1 | DL3 | 3 | CU17 |

APPLICANTS_LOGIN TABLE:


```
select * from applicants_login;
```

Data Output Messages Notifications

| | applicant_id [PK] character varying (20) | email character varying (255) | password character varying (255) |
|---|---|---|---|
| 1 | 10001 | leslie26@example.net | M9B0%Fvg%6 |
| 2 | 10002 | davidedwards@example.com | *!JI9Kft0H |
| 3 | 10003 | myerswhitney@example.net | !69BPEKmdQ |
| 4 | 10004 | debramay@example.com | $3uuWCVj^Q |
| 5 | 10005 | stephen92@example.net | 0&HbyJQs$3 |
| 6 | 10006 | youngjessica@example.net | &4NUGDIn72 |
| 7 | 10007 | johnsonkyle@example.com | (2FB7gL7hj |
| 8 | 10008 | susankeller@example.com | (I+S0KSmf4 |
| 9 | 10009 | melinda41@example.com | Nt08Pxzl(+ |
| 10 | 100010 | anna30@example.org | JL($1A*jhH |
| 11 | 100011 | emurphy@example.net | *k#6FyT#VF |
| 12 | 100012 | dschaefer@example.org | TO#Jdazg*6 |
| 13 | 100013 | catherine96@example.org | T6EqosiJ^b |
| 14 | 100014 | yvargas@example.net | 08Nfn%0o(6 |
| 15 | 100015 | gina74@example.org | ^9VDjPV)(v |
| 16 | 100016 | leejohn@example.org | oo75Qoczi* |
| 17 | 100017 | christopherfuentes@example.com | Lf4LwKdMT& |
| 18 | 100018 | yolandapaul@example.org | 8_^v4ZlZ!_ |

COUNTRY TABLE

```
select * from country;
```

Data Output | Messages | Notifications

| | cid [PK] character varying (10) | country character varying (255) |
|---|---|---|
| 1 | CU1 | United States |
| 2 | CU2 | United Kingdom |
| 3 | CU3 | Switzerland |
| 4 | CU4 | Singapore |
| 5 | CU5 | China (Mainland) |
| 6 | CU6 | Hong Kong SAR |
| 7 | CU7 | Japan |
| 8 | CU8 | Canada |
| 9 | CU9 | Australia |
| 10 | CU10 | South Korea |
| 11 | CU11 | France |
| 12 | CU12 | Germany |
| 13 | CU13 | Netherlands |
| 14 | CU14 | Malaysia |
| 15 | CU15 | Taiwan |
| 16 | CU16 | Argentina |
| 17 | CU17 | Belgium |

DEGREE TABLE

```
select * from degree_table;
```

Data Output | Messages | Notifications

| | degree_id [PK] character varying (10) | degree_level character varying (255) |
|---|---|---|
| 1 | DL1 | MS |
| 2 | DL2 | MBA |
| 3 | DL3 | PHD |

GRE WAIVER TABLE

```
select * from gre_waiver;
```

Data Output | Messages | Notifications

| | gre_id [PK] character varying (10) | gre_waiver_options character varying (255) |
|---|---|---|
| 1 | GRE1 | No |
| 2 | GRE2 | Not Applicable |
| 3 | GRE3 | Yes |
| 4 | GRE4 | Optional |

REGION TABLE

```
select * from region;
```

Data Output | Messages | Notifications

| | rid [PK] character varying (10) | region character varying (255) |
|---|---|---|
| 1 | R1 | North America |
| 2 | R2 | Europe |
| 3 | R3 | Asia |
| 4 | R4 | Oceania |
| 5 | R5 | Latin America |
| 6 | R6 | Africa |

RESEARCH OUPUT

```
select * from research_output;
```

Data Output | Messages | Notifications

| | ro_id [PK] character varying (10) | research_outputs character varying (255) |
|---|---|---|
| 1 | RA_RO1 | Very High |
| 2 | RA_RO2 | High |
| 3 | RA_RO3 | Medium |
| 4 | RA_RO4 | Low |

UNIVERSITY ADMINS

```
select * from university_admins;
```

Data Output | Messages | Notifications

| | uid [PK] character varying (10) | email character varying (255) | password character varying (255) |
|---|---|---|---|
| 1 | 1001 | josephsergio@example.net | )eMuo^!4#9 |
| 2 | 1002 | lukesimpson@example.net | b8mzYAIi&X |
| 3 | 1003 | petersonmegan@example.net | N46aXPPs&L |
| 4 | 1004 | kimberlypineda@example.net | M79ZwKUo$e |
| 5 | 1005 | markturner@example.org | e7*NREDv$0 |
| 6 | 1006 | gravesjennifer@example.net | TX)6Dqh^*B |
| 7 | 1007 | stevenstevenson@example.net | g0RE*wK3*X |
| 8 | 1008 | joanlewis@example.net | KR2F4^uS!9 |
| 9 | 1009 | qthomas@example.net | Z_6%gF4pal |
| 10 | 10010 | aschmidt@example.com | #)pT6SaJnF |
| 11 | 10011 | pattersonbarbara@example.org | zsyO74Jt_% |
| 12 | 10012 | bwilliams@example.net | Y@0MWmx(X) |
| 13 | 10013 | pmerritt@example.com | B4KK_xMs*z |
| 14 | 10014 | fdavis@example.org | 8RDAkQeN#o |
| 15 | 10015 | robertfitzgerald@example.org | H+s8jIrp)8 |
| 16 | 10016 | tammy61@example.net | CI6Bw(pV+R |
| 17 | 10017 | barnesbarry@example.com | TyzJemy%%9 |
| 18 | 10018 | brownshaun@example.net | Reg*3NMGd# |

UNIVERSITY REQUIREMENTS

UNIVERSITY SIZE

```
select * from university_size;
```

Data Output | Messages | Notifications

| | sid [PK] character varying (10) | uni_size character varying (255) |
|---|---|---|
| 1 | S1 | M |
| 2 | S2 | L |
| 3 | S3 | S |
| 4 | S4 | XL |

UNIVERSITY TYPE

Query  Query History

```
1  select * from university_type;
```

Data Output  Messages  Notifications

| | tid [PK] character varying (10) | university_type character varying (255) |
|---|---|---|
| 1 | Type1 | Private |
| 2 | Type2 | Public |

## VIII. Query Executions

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

Query  Query History

```
1  SELECT ut.university_type, COUNT(*) AS num_applicants
2  FROM University_requirements ur
3  JOIN University_Type ut ON ur.TID = ut.TID
4  JOIN Applicants_Application_Status aas ON ur.UID = aas.UID
5  GROUP BY ut.university_type;
6
```

Data Output  Messages  Notifications

| | university_type character varying (255) | num_applicants bigint |
|---|---|---|
| 1 | Public | 10528 |
| 2 | Private | 1822 |

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

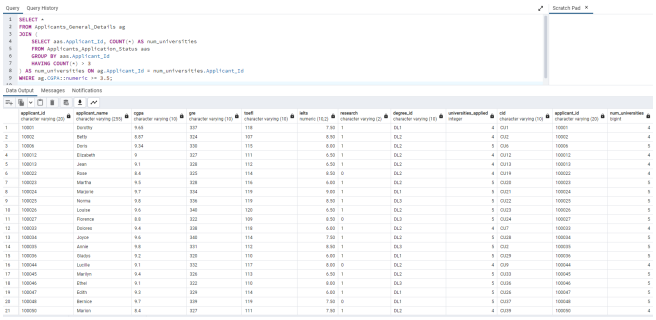
Query  Query History

```
1  SELECT ur.Uni_name
2  FROM University_requirements ur
3  JOIN GRE_waiver gw ON ur.GRE_ID = gw.GRE_ID
4  WHERE ur.student_faculty_ratio < 20;
```

Data Output  Messages  Notifications

| | uni_name character varying (100) |
|---|---|
| 1 | Massachusetts Institute of Technology (MIT) |
| 2 | University of Oxford |
| 3 | Stanford University |
| 4 | University of Cambridge |
| 5 | Harvard University |
| 6 | California Institute of Technology (Caltech) |
| 7 | Imperial College London |
| 8 | ETH Zurich - Swiss Federal Institute of Technology |
| 9 | UCL |
| 10 | University of Chicago |
| 11 | National University of Singapore (NUS) |
| 12 | Nanyang Technological University, Singapore (NTU) |
| 13 | University of Pennsylvania |
| 14 | EPFL |
| 15 | Yale University |
| 16 | The University of Edinburgh |
| 17 | Tsinghua University |
| 18 | Peking University |
| 19 | Columbia University |
| 20 | Princeton University |

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



4.Select Query Query

```
WITH un AS (
    SELECT UID, Uni_name
    FROM University_requirements
)
SELECT
    aa.Applicant_Id,
    ag.applicant_name,
    ag.CGPA AS applicants_cgpa,
    ag.GRE AS applicants_gre,
    c.Country AS applicants_country,
    ag.TOEFL AS applicants_toefl,
    ag.IELTS AS applicants_IELTS,
    dt.Degree_Level AS degree_applied,
    ag.universities_applied,
    un.uni_name
FROM
    Applicants_Application_Status aa
    JOIN Applicants_General_Details ag ON aa.Applicant_Id = ag.Applicant_Id
    JOIN Degree_Table dt ON aa.Degree_ID = dt.Degree_ID
    JOIN Country c ON ag.CID = c.CID
    JOIN un ON aa.UID = un.UID
WHERE
c.country = 'United States';
```

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.

5.Update Query

6.Insert query

Query  Query History

```
1  INSERT INTO public.applicants_general_details(
2      applicant_id, applicant_name, cgpa, gre, toefl, ielts, research, degree_id, universities_applied, cid)
3      VALUES ('100013132','SAI', 9.8, 322, 112, 8, 1, 'DL1' ,6,'CU33');
```

Data Output  Messages  Notifications

```
INSERT 0 1

Query returned successfully in 40 msec.
```

7.Delete query

Query  Query History

```
1  DELETE FROM public.applicants_general_details
2  WHERE applicant_id = '100013132';
```

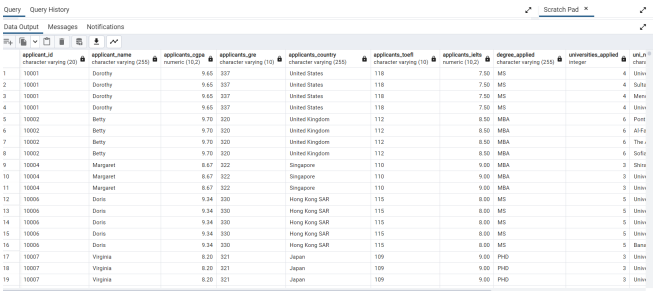
Data Output  Messages  Notifications

```
DELETE 1

Query returned successfully in 295 msec.
```

8. 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.

```
WITH un AS (
    SELECT UID, Uni_name
    FROM University_requirements
)
SELECT
    aa.Applicant_Id,
    ag.applicant_name,
    CAST(ag.CGPA AS DECIMAL(10,2)) AS applicants_cgpa,
    ag.GRE AS applicants_gre,
    c.Country AS applicants_country,
    ag.TOEFL AS applicants_toefl,
    ag.IELTS AS applicants_IELTS,
    dt.Degree_Level AS degree_applied,
    ag.universities_applied,
    un.uni_name
FROM
    Applicants_Application_Status aa
    JOIN Applicants_General_Details ag ON aa.Applicant_Id = ag.Applicant_Id
    JOIN Degree_Table dt ON aa.Degree_ID = dt.Degree_ID
    JOIN Country c ON ag.CID = c.CID
    JOIN un ON aa.UID = un.UID
WHERE
    CAST(ag.CGPA AS DECIMAL(10,2)) >= 3.5 AND
    CAST(ag.GRE AS integer) >= 320;
```



9. Creating Temporary Table for the website Fetching:

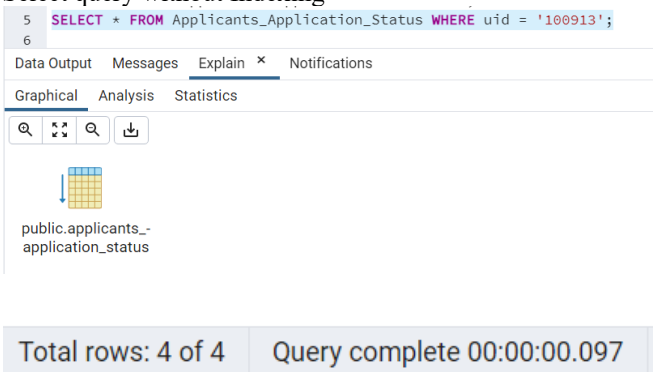
```
CREATE TABLE applicants_status AS
WITH un AS (
    SELECT UID, Uni_name
    FROM University_requirements
)
SELECT
    aa.Applicant_Id,
    ag.applicant_name,
    ag.CGPA AS applicants_cgpa,
    ag.GRE AS applicants_gre,
    c.Country AS applicants_country,
    ag.TOEFL AS applicants_toefl,
    ag.IELTS AS applicants_IELTS,
    dt.Degree_Level AS degree_applied,
    ag.universities_applied,
    un.uni_name,
    aa.status
FROM
    Applicants_Application_Status aa
    JOIN Applicants_General_Details ag ON aa.Applicant_Id = ag.Applicant_Id
    JOIN Degree_Table dt ON aa.Degree_ID = dt.Degree_ID
    JOIN Country c ON ag.CID = c.CID
    JOIN un ON aa.UID = un.UID;
```

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.

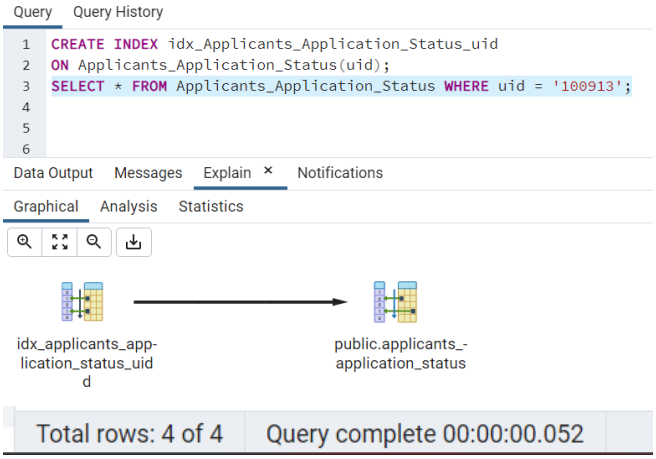
Using temporary tables can be useful for web applications when you need to perform complex queries or intermediate calculations that require temporary storage of data. It can also be used to improve performance by reducing the number of times a query needs to be executed. However, it's important to note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

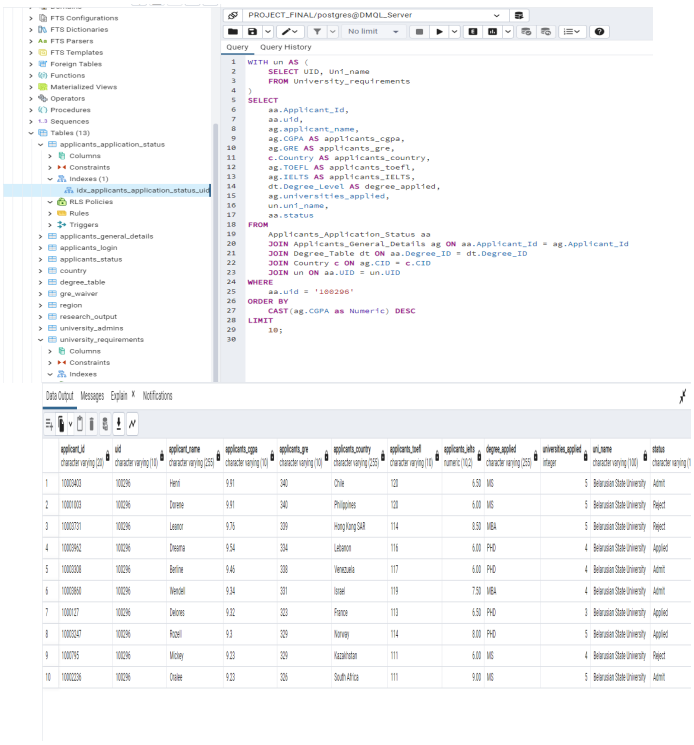
## IX. INDEXING AND TIGGERS (COMPLEX QUERIES)

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

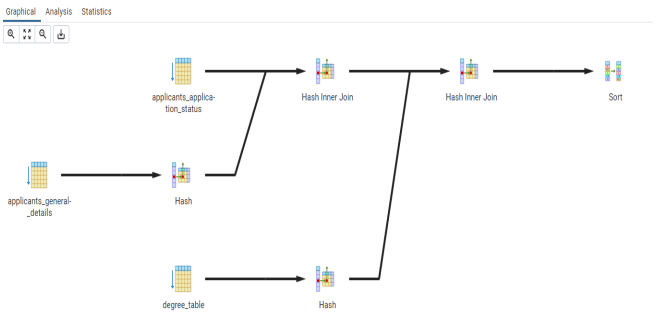
Running without Indexing:

```
SELECT
    ag.Applicant_Id,
    ag.applicant_name,
    aa.UID,
    aa.course_name,
    dt.Degree_Level,
    aa.Status
FROM
    Applicants_General_Details ag
    JOIN Applicants_Application_Status aa ON ag.Applicant_Id = aa.Applicant_Id
    JOIN Degree_Table dt ON aa.Degree_ID = dt.Degree_ID
WHERE
    ag.applicant_id LIKE '10001%'
ORDER BY
    dt.Degree_Level DESC;
```

Data Output | Messages | Explain | Notifications

| | applicant_id character varying (20) | applicant_name character varying (255) | uid character varying (10) | course_name character varying (255) | degree_level character varying (255) | status character varying (10) |
|---|---|---|---|---|---|---|
| 1 | 10001993 | Milda | 1001109 | Management | PHD | Admit |
| 2 | 10001993 | Milda | 100591 | Management | PHD | Reject |
| 3 | 10001670 | Alison | 100345 | Chemistry | PHD | Reject |
| 4 | 10001670 | Alison | 1001058 | Chemistry | PHD | Reject |
| 5 | 10001670 | Alison | 1001001 | Chemistry | PHD | Applied |
| 6 | 10001670 | Alison | 100870 | Chemistry | PHD | Applied |
| 7 | 10001661 | Paulita | 1001089 | Sociology | PHD | Applied |
| 8 | 10001661 | Paulita | 100957 | Sociology | PHD | Reject |
| 9 | 10001659 | Nena | 100259 | Finance | PHD | Admit |
| 10 | 10001654 | Malissa | 100601 | Mathematics | PHD | Applied |
| 11 | 10001654 | Malissa | 1001121 | Mathematics | PHD | Applied |
| 12 | 10001654 | Malissa | 100510 | Mathematics | PHD | Applied |
| 13 | 10001005 | Leo | 1001155 | History | PHD | Applied |
| 14 | 10001005 | Leo | 100597 | History | PHD | Reject |
| 15 | 10001005 | Leo | 10087 | History | PHD | Applied |
| 16 | 10001863 | Mennie | 1001010 | Business Analytics | PHD | Applied |

Total rows: 1000 of 3420 | Query complete 00:00:00.250



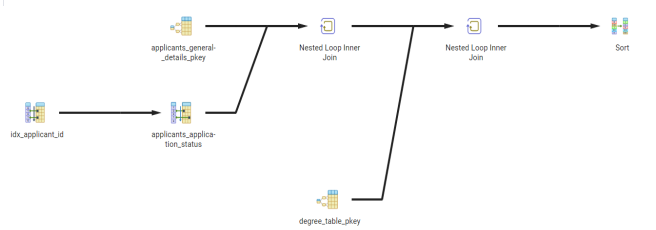

Running with Indexing:

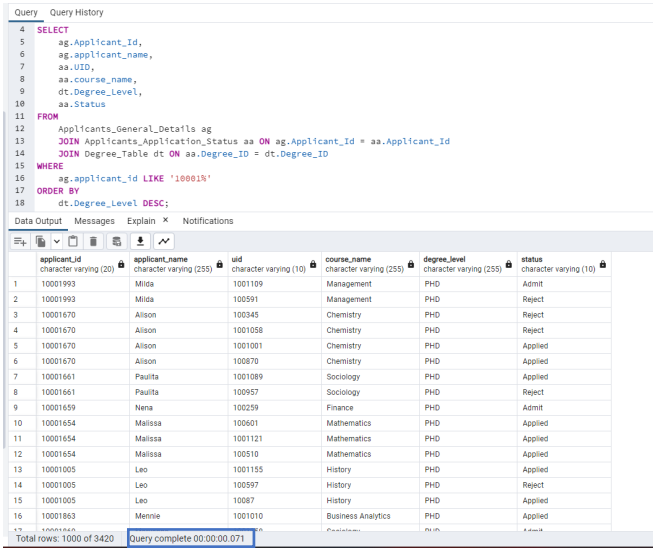
```
CREATE INDEX idx_applicant_id ON Applicants_Application_Status (applicant_id);
```

Data Output | Messages | Explain | Notifications

```
CREATE INDEX

Query returned successfully in 535 msec.
```



```
SELECT
    ag.Applicant_Id,
    ag.applicant_name,
    aa.UID,
    aa.course_name,
    dt.Degree_Level,
    aa.Status
FROM
    Applicants_General_Details ag
    JOIN Applicants_Application_Status aa ON ag.Applicant_Id = aa.Applicant_Id
    JOIN Degree_Table dt ON aa.Degree_ID = dt.Degree_ID
WHERE
    ag.applicant_id LIKE '10001%'
ORDER BY
    dt.Degree_Level DESC;
```

Data Output | Messages | Explain | Notifications

| | applicant_id character varying (20) | applicant_name character varying (255) | uid character varying (10) | course_name character varying (255) | degree_level character varying (255) | status character varying (10) |
|---|---|---|---|---|---|---|
| 1 | 10001993 | Milda | 1001109 | Management | PHD | Admit |
| 2 | 10001993 | Milda | 100591 | Management | PHD | Reject |
| 3 | 10001670 | Alison | 100345 | Chemistry | PHD | Reject |
| 4 | 10001670 | Alison | 1001058 | Chemistry | PHD | Reject |
| 5 | 10001670 | Alison | 1001001 | Chemistry | PHD | Applied |
| 6 | 10001670 | Alison | 100870 | Chemistry | PHD | Applied |
| 7 | 10001661 | Paulita | 1001089 | Sociology | PHD | Applied |
| 8 | 10001661 | Paulita | 100957 | Sociology | PHD | Reject |
| 9 | 10001659 | Nena | 100259 | Finance | PHD | Admit |
| 10 | 10001654 | Malissa | 100601 | Mathematics | PHD | Applied |
| 11 | 10001654 | Malissa | 1001121 | Mathematics | PHD | Applied |
| 12 | 10001654 | Malissa | 100510 | Mathematics | PHD | Applied |
| 13 | 10001005 | Leo | 1001155 | History | PHD | Applied |
| 14 | 10001005 | Leo | 100597 | History | PHD | Reject |
| 15 | 10001005 | Leo | 10087 | History | PHD | Applied |
| 16 | 10001863 | Mennie | 1001010 | Business Analytics | PHD | Applied |

Total rows: 1000 of 3420 | Query complete 00:00:00.071

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:

Query | Query History

```
CREATE OR REPLACE FUNCTION insert_app_status()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.universities_applied <> OLD.universities_applied THEN
        INSERT INTO Applicants_Application_Status (Applicant_Id, UID, course_name, Degree_ID, Status)
        VALUES (NEW.Applicant_Id,'Information System', NEW.Degree_ID,'Applied');
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_app_status
AFTER UPDATE OF universities_applied ON Applicants_General_Details
FOR EACH ROW
EXECUTE FUNCTION insert_app_status();

UPDATE Applicants_General_Details
SET universities_applied = 4
WHERE Applicant_Id = '1000199';
```

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

## REFERENCES

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].


[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989**.**