MODULE 3

EXPLANATION OF SQL CODE FOR QUERIES

All SQL queries referenced here are implemented in query_data.py.

Q0 — How many GradCafe entries are in your database?

Explanation:

SELECT COUNT(*) counts every row in the **applicants** table.

FROM applicants specifies the table being counted.

This returns a single number: total GradCafe records stored.

Q1 — How many entries do you have in your database who have applied for Fall 2026?

Explanation:

SELECT COUNT(*) counts rows that match the filter.

FROM **applicants** specifies the table being counted

WHERE term ILIKE '%fall%' matches any term containing "fall" (case-insensitive).

AND term **ILIKE '%2026%'** further requires the same term string to contain "2026".

Using two substring filters makes the match tolerant of term formatting variations (e.g., "Fall 2026", "fall 2026", etc.).

Q2 — What percentage of entries are from international students (not American or other) (to two decimal places)?

Explanation:

SELECT COUNT (*) gets the total number of rows in **applicants**. This is the denominator.

Second **SELECT COUNT (*)** counts rows whose us_or_international contains "international" (case-insensitive), while excluding rows that also contain "american" or "other".

Then Python computes: (intl / total) * 100, with a guard to avoid divide-by-zero if total is 0. Result is formatted to two decimal places (e.g., 50.02%).

Q3 — What is the average GPA, GRE (Total), GRE (Section), and GRE AW of applicants who provide these metrics?

Explanation:

AVG(gpa), AVG(gre), AVG(gre_v), AVG(gre_aw) compute averages for each metric.

AVG ignores **NULL** automatically, so each average is computed only on rows where that column has a value.

The **WHERE** clause filters out obviously invalid GRE values while still allowing NULLs:

- gre must be 300–340 if present (or NULL allowed),
- gre_v must be 130–170 if present (or NULL allowed),
- gre_aw must be 0–6 if present (or NULL allowed).

This reduces the impact of scraped/dirty values (e.g., garbled GRE totals) on the averages. Rows with missing GRE values are still included; rows with implausible GRE values are excluded.

Q4 — What is the average GPA of American students in Fall 2026?

Explanation:

AVG(gpa) computes mean GPA for rows that meet the filters.

term **ILIKE '%fall%'** AND term **ILIKE '%2026%'** selects Fall 2026 applicants (case-insensitive).

us_or_international **ILIKE '%american%'** restricts to American applicants (case-insensitive substring match).

gpa IS NOT NULL ensures the average is based only on applicants who provided GPA.

If no matching rows exist, AVG returns NULL and code displays N/A.

Q5 — What percent of entries for Fall 2026 are Acceptances (to two decimal places)?

Explanation:

First **SELECT COUNT (*)** query counts all Fall 2026 entries (denominator).

Second **SELECT COUNT (*)** query counts Fall 2026 entries whose status contains “accept” (numerator) – e.g. **ILIKE ‘%accept%’**

Python computes: (f26_accept / f26_total) * 100, guarded against divide-by-zero.

This yields the percent of Fall 2026 records that are acceptances, formatted to two decimals.

Q6 — What is the average GPA of applicants who applied for Fall 2026 who are Acceptances?

Explanation:

AVG(gpa) averages GPA values for accepted Fall 2026 applicants.

WHERE term filters to Fall 2026 (case-insensitive substring match).

AND status **ILIKE ‘%accept%’** filters to accepted outcomes

AND **gpa IS NOT NULL** ensures only rows with GPA contribute.

If none match, AVG returns NULL and the code displays “N/A”.

Q7: How many entries are from applicants who applied to JHU for a master’s degree in Computer Science?

Explanation:

This query counts how many applicant records match a specific profile: students applying to Johns Hopkins for a master’s degree in Computer Science.

SELECT COUNT(*) returns the total number of rows that satisfy all filters from table **applicants**

WHERE university ILIKE ANY (...) checks whether the university field contains any common variations of “Johns Hopkins” (including misspellings and abbreviations like JHU). **ILIKE** makes the match case-insensitive, improving robustness against inconsistent scraped text.

AND degree ILIKE ANY (ARRAY['%master%', (…) uses a case-insensitive match against common master’s degree variants to handle inconsistent formatting in scraped text (e.g., “MS” vs “M.S.” vs “Master”).

AND (program ILIKE ANY (ARRAY['%computer science%', (…) identifies Computer Science using multiple common variants/abbreviations, and checks **both** the raw scraped program field and the LLM-normalized llm_generated_program field to improve robustness when the scraped program name is messy or inconsistent.

Together, these filters isolate applicants who plausibly applied to JHU for a Master's in Computer Science, and the query returns the total count of those matches.

Q8: How many 2026 acceptances are from applicants who applied to Georgetown, MIT, Stanford, or Carnegie Mellon University for a PhD in Computer Science?

Explanation:

This query counts how many applicants meet a very specific academic profile: 2026 acceptances to selected universities for a PhD in Computer Science.

COUNT(*) returns the total number of rows that satisfy all filtering criteria from table **applicants**

The term **ILIKE '%2026%'** restricts results to applicants associated with the 2026 admissions cycle.

The status **ILIKE ANY (...)** captures records indicating an offer outcome, matching variations like "accept" or "admit" in a case-insensitive way.

The degree **~* '\\m(phd|ph\\.d\\.)\\M'** uses a case-insensitive regular expression to match PhD designations while enforcing word boundaries, preventing accidental partial matches.

The program **~* '(...)'** applies regex pattern matching to recognize multiple spelling and abbreviation variations of Computer Science, including compact forms like "CS".

The university **ILIKE ANY (...)** limits results to applicants targeting Georgetown, MIT, Stanford, or Carnegie Mellon, accounting for alternate naming formats and abbreviations.

Together, these conditions isolate accepted 2026 PhD Computer Science applicants from the specified universities and return the total count.

Q9: Do the numbers for Q8 change if you use the LLM generated fields (rather than downloaded fields)?

Explanation:

This query counts how many applicants match the same acceptance criteria as Q8, but uses the LLM-normalized fields instead of the raw scraped fields. The goal is to see whether data normalization changes the result.

COUNT(*) returns the number of rows meeting all filters from the table **applicants**

The term **ILIKE '%2026%'** restricts results to the 2026 admissions cycle.

The status **ILIKE ANY (...)** captures records indicating an admission or acceptance outcome.

The degree **~* '\\m(phd|ph\\.d\\.)\\M'** uses a case-insensitive regular expression with word boundaries to reliably identify PhD designations.

The Computer Science program (LLM field) **llm_generated_program ~* '(...)'** matches normalized Computer Science labels, handling spacing and abbreviation variation.

The **llm_generated_university ILIKE ANY (...)** limits results to Georgetown, MIT, Stanford, or Carnegie Mellon, using standardized university names produced by the LLM.

Together, these filters isolate accepted 2026 PhD Computer Science applicants from the selected universities using cleaned, normalized text fields, allowing comparison with the raw-field query.

Q10: What are the top 5 most popular programs applied to in gradcafe.com?

Explanation:

Question 10 groups the full dataset by program and counts frequency to find the top 5 most common programs applied to. Here is what the code does specifically:

SELECT program, COUNT(*) AS count – SELECT returns the values of program name as output; COUNT (*) counts how many rows are in each group with the * designating to count every row. AS count renames the output to count and makes it readable. This line answers how many applicants applied to each program.

FROM applicants – instructs to read from the table “applicants”

WHERE program IS NOT NULL AND program <> '' – this line filters out junk and/or missing data. “Program is not null” removes rows where program is missing entirely and **‘program <>’** removes rows where program is an empty string

GROUP BY program – this line groups all rows that share the same program together; this is required because SQL cannot return program alongside COUNT (*) unless program is grouped

ORDER BY count DESC – this line sorts the output in “count” in descending order (largest number of programs applied to are at the top)

LIMIT 5; - this returns only the first 5 results after sorting

Q11: What are the top 5 universities applied to for Physics PhD?

Explanation:

Question 11 filters the dataset to Physics PhD only and then groups by university; it then counts the frequency to find the top 5 target universities for Physics PhD applicants. Here is what the code does specifically:

SELECT university, COUNT(*) AS count – this selects the university field (the school the applicant applied to) and COUNT (*) counts how many rows are in each group with the * designating to count every row. AS count renames the output to count and makes it readable.

FROM applicants – instructs to read from the table **applicants**

WHERE program

~*'(\\mphysics\\M.*\\m(phd|ph\\.d\\.)\\M|\\m(phd|ph\\.d\\.)\\M.*\\mphysics\\M)'

This uses a **case-insensitive regular expression** (~*) to match entries where the program contains both **“physics”** and **“phd” / “ph.d.”**. The pattern is written to allow **either order**: “physics … phd” **or** “phd … physics” so it captures variants like “Physics PhD”, “PhD Physics”, “Physics – Ph.D.”, etc.

AND university IS NOT NULL AND university <> '' – similar to Q10, filters out missing universities and prevents null from appearing in the data

GROUP BY university – this groups all Physics PhD rows by university allowing **COUNT (*)** per university

ORDER BY count DESC – this sorts the data in descending order

LIMIT 5; - this shows only the top 5 results