Finally, provide your answers within a PDF file with a description of the query you used (and why). Please provide the code used to create the queries within a file called query_data.py.

1. How many entries do you have in your database who have applied for Fall 2026?

```
cur.execute("SELECT COUNT(*) FROM applicants WHERE term = 'Fall 2026';")
        q1 = cur.fetchone()[0]
```

Query Description: Uses `SELECT COUNT(*)` with a `WHERE` clause to filter the `term` column for the string 'Fall 2026'.

Why: This provides a total count of all records associated with the upcoming Fall 2026 admission cycle.

2. What percentage of entries are from international students (not American or Other) (to two decimal places)?

```
cur.execute("""
            SELECT ROUND(
                (COUNT(*) FILTER (WHERE us_or_international =
'International')::numeric /
                NULLIF(COUNT(*), 0)::numeric) * 100, 2
            ) FROM applicants;
        """)
```

Query Description: Counts entries with `us_or_international = 'International'`, divides by the total count, and applies `ROUND(..., 2)`. It uses `::numeric` casting to avoid integer division errors.

Why: This calculates the proportion of international applicants relative to the whole dataset, formatted to the required two decimal places.

3. What is the average GPA, GRE, GRE V, GRE AW of applicants who provide these metrics?

```
cur.execute("""
            SELECT
                AVG(gpa) FILTER (WHERE gpa <= 4.0),
                AVG(gre) FILTER (WHERE gre >= 130 AND gre <= 170),
                AVG(gre_v) FILTER (WHERE gre_v >= 130 AND gre_v <= 170),
                AVG(gre_aw) FILTER (WHERE gre_aw >= 0 AND gre_aw <= 6)
            FROM applicants;
        """)
```

Query Description: Uses `AVG()` on GPA and GRE columns. Critically, it uses `FILTER` clauses to restrict GPA to , GRE Q/V to , and GRE AW to .

Why: Real-world data often contains typos (e.g., GPA of "34") or old scoring scales (GRE out of 800). Filtering ensures the averages represent the current standard scales accurately.

4. What is their average GPA of American students in Fall 2026?

```
cur.execute("""
            SELECT AVG(gpa) FROM applicants
            WHERE us_or_international = 'American'
            AND term = 'Fall 2026'
            AND gpa <= 4.0;
        """)
```

Query Description: Calculates the average GPA specifically for rows where `us_or_international` is 'American' and `term` is 'Fall 2026'.

Why: This allows for a demographic-specific academic benchmark for the Fall 2026 cycle while excluding outliers above 4.0.

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

```
 cur.execute("""
            SELECT ROUND(
                (COUNT(*) FILTER (WHERE status = 'Accepted' AND term = 'Fall
2026')::numeric /
                NULLIF(COUNT(*) FILTER (WHERE term = 'Fall 2026'), 0)::numeric) *
100, 2
            ) FROM applicants;
        """)
```

Query Description: Divides the number of 'Accepted' statuses by the total number of entries for the 'Fall 2026' term.

Why: This provides the statistical "selectivity" of the programs recorded in the database for that specific year.

6. What is the average GPA of applicants who applied for Fall 2026 who are Acceptances?

```
cur.execute("""
            SELECT AVG(gpa) FROM applicants
            WHERE term = 'Fall 2026'
            AND status = 'Accepted'
            AND gpa <= 4.0;
        """)
```

Query Description: Filters for entries that are both 'Accepted' and 'Fall 2026', then calculates the average of the `gpa` column.

Why: This identifies the academic "profile" of a successful applicant in the most recent data.

7. How many entries are from applicants who applied to JHU for a masters degrees in Computer Science?

```
cur.execute("""
            SELECT COUNT(*) FROM applicants
            WHERE llm_generated_university ILIKE '%Johns Hopkins%'
            AND degree = 'Masters' AND llm_generated_program ILIKE '%Computer
Science%';
        """)
```

Query Description: Uses `ILIKE '%Johns Hopkins%'` and `ILIKE '%Computer Science%'` with a filter for `degree = 'Masters'`.

Why: `ILIKE` is case-insensitive, ensuring that variations in how users typed the university name or program do not result in missed data.

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

```
cur.execute("""
            SELECT COUNT(*) FROM applicants
            WHERE status = 'Accepted' AND term = 'Fall 2026' AND degree = 'PhD'
            AND program ILIKE '%Computer Science%'
            AND (
                  program ILIKE '%Georgetown%' OR
```

```
                program ILIKE '%MIT%' OR
                program ILIKE '%Massachusetts Institute of Technology%' OR
                program ILIKE '%Stanford%' OR
                program ILIKE '%Carnegie%' OR
                program ILIKE '%CMU%'
            );
        """)
```

Query Description: Searches the raw `program` text for a list of specific keywords (MIT, Stanford, Georgetown, Carnegie, CMU) for PhD applicants in CS.

Why: Because original data is "uncleaned," we must use multiple `OR` conditions and wildcards (`%`) to catch different ways users might have abbreviated these schools.

9. Do you numbers for question 8 change if you use LLM Generated Fields (rather than your downloaded fields)?

```
2. cur.execute(f"""
3.             SELECT COUNT(*) FROM applicants
4.             WHERE status = 'Accepted' AND term = 'Fall 2026' AND degree =
   'PhD'
5.             AND llm_generated_program ILIKE '%Computer Science%'
6.             AND llm_generated_university IN {elite_schools};
7.         """)
```

Query Description: Performs the same count but uses the `llm_generated_university` and `llm_generated_program` columns.

Why: This tests the effectiveness of data cleaning. Standardized fields should theoretically make querying easier and more accurate than searching raw, messy text.

10. Top 5 most applied to universities:

```
cur.execute("""
          SELECT llm_generated_university, COUNT(*) as apps
          FROM applicants
          WHERE llm_generated_university IS NOT NULL
          GROUP BY llm_generated_university
          ORDER BY apps DESC LIMIT 5;
      """)
```

Query Description: Uses `GROUP BY` on the university name, `COUNT(*)` to find volume, and `ORDER BY apps DESC` to sort from highest to lowest.

Why: This reveals which institutions are the most "popular" or have the most reported data points in the system.

11. Top 5 Lowest Application Counts

```
cur.execute("""
            SELECT
                TRIM(BOTH '[] '' ' FROM llm_generated_university) AS cleaned_uni,
                COUNT(*) as apps
            FROM applicants
            WHERE llm_generated_university IS NOT NULL
              AND llm_generated_university != ''
            GROUP BY cleaned_uni
            ORDER BY apps ASC, cleaned_uni ASC
            LIMIT 5;
        """)
```

Query Description: Uses `TRIM` to strip Python list formatting (brackets and quotes) from the university names, then groups and orders by count in ascending order (`ASC`).

Why: This identifies the "long tail" of the dataset—niche or smaller programs that are rarely reported by users on Grad Cafe.

Written Assignment Requirements

Analyzing anonymously submitted data, such as that from Grad Cafe, presents significant inherent limitations regarding selection bias and data integrity. Unlike official standardized reports, which include every test-taker, anonymous platforms are prone to self-selection bias, where individuals with exceptionally high scores are more motivated to share their successes publicly. This explains why the average GRE quantitative score in the dataset (nearly 165) deviates so sharply from the global average of 157. Furthermore, without a verification mechanism, there is no way to prevent users from accidentally entering incorrect data or intentionally inflating their scores to appear more competitive, leading to a "prestige bias" that skews the overall analytics.

The analytic responses were surprising in their deviation from established standards, illustrating how "crowdsourced" data creates a distorted view of the applicant pool. In professional

standards, data is collected through controlled environments with verified identities, ensuring a representative bell curve. In contrast, the Grad Cafe entries likely represent an "elite" subset of students applying to top-tier programs, who may also be more active in online communities. This suggests that while the data is useful for high-level sentiment and trends, it cannot be used as a definitive benchmark for the general population due to the lack of demographic weighting and the high probability of "social desirability" bias among anonymous contributors.