NoSQL Assignment 1

Shashank Tippanavar IMT2022014
Kushal Suvan Jenamani IMT2022057

February 2025

> **Using the code.** The GitHub repository provides both the code and instructions for setting it up on your machine. If the link does not work, we have also submitted a ZIP file on LMS. You can access the repository here: GitHub Repository

Section A

1. **Answer:** (a) Marines
   Explanation: This query finds the teams that won their games but by no more than a margin of 2 runs

2. **Answer:** (d) Bay Stars, Monday
   Explanation: This query retrieves teams and the day they played when either they had no opponent or their runs are missing (NULL or negative)

3. **Answer:** (a) Bay Stars, Tigers, 2
   Explanation: This query retrieves each team's first opponent (when all its opponents are ordered lexicographically) and the highest score that team scored against any other team.

4. **Answer:** (a) Giants, Sunday, NULL, NULL could appear seventh through twelfth
   Explanation: Since we are sorting by runs in descending order, entries with NULL values for runs can appear either at the start or the bottom of the table, depending on the database's default behavior.

5. **Answer:** (c) Dragons, Giants
   Explanation: This selects teams from the table that have the same opponent, and NOT EQUAL TO operator ensures that we do not select the same team twice.

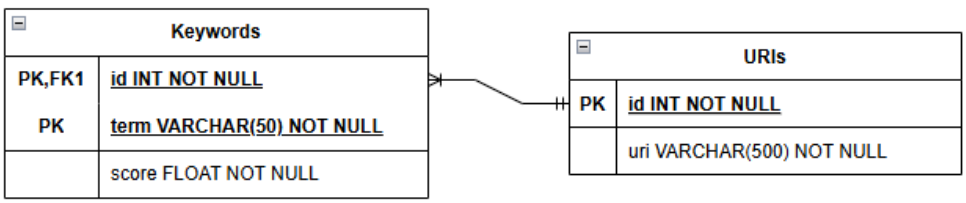
.

Section B

The following commands assume that you are connected to a nascent database. Refer to the GitHub repository for instructions on how to run the scripts.

Problem 1 - Bulkloading Data into a PostgreSQL Database

1. The following ER diagram shows the schema constraints.



2. We enable `\timing on` to estimate the time taken for our queries to run, for the next part of the problem.

   The `\COPY` commands are as follows-

```
-- The script must be run with the root of the project as the root of
    the psql for the file paths to work

-- Enabling timing to show duration of query execution
\timing on

\COPY uris(id, uri) FROM 'Wikipedia-EN-20120601_REVISION_URIS.TSV' WITH
    (FORMAT CSV, DELIMITER E'\t', ENCODING 'UTF8');

\COPY keywords(id, term, score) FROM
    'Wikipedia-EN-20120601_KEYWORDS.TSV' WITH (FORMAT CSV, DELIMITER
    E'\t', ENCODING 'UTF8');
```

3. **Results**

	w/o constraints	**w/ constraints**
keywords	19113.258 ms	56266.843 ms
uris	16.059 ms	17.032 ms

Added constraints **increased** the query execution time. Increase was 4× in **keywords** relation due to the added foreign key. The increase in **uris** bulkloading was minimal.

Query

We proceed by creating two schemas. We will not enforce the key constraints yet. Instead, we will test the time taken for the `\COPY` command without the constraints.

```
     CREATE TABLE keywords (
       id        INT NOT NULL,
       term      VARCHAR(50) NOT NULL,
       score     FLOAT NOT NULL
     );
     CREATE TABLE uris (
       id        INT NOT NULL,
       url       VARCHAR(500) NOT NULL
     );
```

The same command was timed after `TRUNCATE`ing both relations and adding the following `FOREIGN KEY` constraint.

```
TRUNCATE TABLE keywords;
TRUNCATE TABLE uris;

CREATE TABLE keywords (
    id INTEGER NOT NULL,
    term VARCHAR(50) NOT NULL,
    score FLOAT NOT NULL,
    PRIMARY KEY (id, term)
);
CREATE TABLE uris (
    id INTEGER NOT NULL,
    url VARCHAR(500) NOT NULL,
    PRIMARY KEY(id)
);
ALTER TABLE keywords
ADD FOREIGN KEY (id) REFERENCES uris(id);
```

Problem 2 - Running Keyword Queries over Wikipedia

1. Below we GROUP BY uris.url and set a condition that its selected iff it contains one of these 4 given terms: 'infantri', 'reinforc', 'brigad', 'fire'. Now we apply an additional filter on the aggregated values (uris.url) to give only those url which have 4 distinct terms (since we had only 4 options to pick from this naturally implies all 4 terms are supposed to be picked for URL to be selected).

   The query is as follows-

```
SELECT uris.url
FROM keywords
JOIN uris ON keywords.id = uris.id
WHERE keywords.term = 'infantri' OR keywords.term = 'reinforc' OR
    keywords.term = 'brigad' OR keywords.term = 'fire'
GROUP BY (uris.url)
HAVING COUNT (DISTINCT keywords.term) = 4;
```

2. Below we GROUP BY uris.url and set a condition that its selected iff it contains one of these 4 given terms: 'infantri', 'reinforc', 'brigad', 'fire'. Now we apply an additional filter on the aggregated values (uris.url) to give only those url which has 1 distinct term (this automatically implies that a url gets picked only if one of those 4 terms occur only once in the url).

   The query is as follows-

```
SELECT uris.url
FROM keywords
JOIN uris ON keywords.id = uris.id
WHERE keywords.term = 'infantri' OR keywords.term = 'reinforc' OR
    keywords.term = 'brigad' OR keywords.term = 'fire'
GROUP BY (uris.url)
HAVING COUNT (DISTINCT keywords.term) = 1;
```

3. This query retrieves URLs associated with the keyword 'reinforc' but excludes those that also contain any of the keywords 'infantri', 'brigad', or 'fire'. It first selects URLs from the uris table that have the term 'reinforc' and stores them in a temporary result (reinforc_table). Then, it filters out any URLs that appear in a second subquery, which retrieves URLs associated with 'infantri', 'brigad', or 'fire', ensuring that only URLs exclusive to 'reinforc' are returned

   The query is as follows-

```
SELECT url
FROM (
    SELECT uris.url AS url
    FROM keywords
    JOIN uris ON keywords.id = uris.id
    WHERE keywords.term = 'reinforc'
) AS reinforc_table
WHERE url NOT IN (
    SELECT uris.url AS url
    FROM keywords
    JOIN uris ON keywords.id = uris.id
    WHERE keywords.term = 'infantri' OR keywords.term = 'brigad' OR
        keywords.term = 'fire'
);
```

4. This query retrieves URLs that are associated with all four keywords: 'infantri', 'reinforc', 'brigad', and 'fire'. It first joins the keywords and uris tables based on their id and selects only the rows where the term matches one of the specified keywords. Then, it groups the results by url and calculates the total score for each URL. The HAVING clause ensures that only URLs containing all four distinct keywords are included in the final result. Finally, the output is sorted in descending order of total_score, hence the URLs with the highest total_score are ranked first. The query is as follows-

```
SELECT uris.url, SUM(keywords.score) AS total_score
FROM keywords
JOIN uris ON keywords.id = uris.id
WHERE keywords.term = 'infantri' OR keywords.term = 'reinforc' OR
    keywords.term = 'brigad' OR keywords.term = 'fire'
GROUP BY (uris.url)
HAVING COUNT(DISTINCT keywords.term) = 4
ORDER BY total_score DESC;
```

5. This query retrieves unique URLs that are associated with any of the keywords 'infantri', 'reinforc', 'brigad', or 'fire'. It joins the keywords and uris tables based on their id and filters rows where the term matches one of the specified keywords. The results are then grouped by url, and the total score for each URL is calculated by summing the score values. The DISTINCT ensures that each URL appears only once in the final output. Finally, the results are sorted in descending order of total_score, hence URLs with the highest combined score are ranked first. The query is as follows-

```
SELECT DISTINCT uris.url, SUM(keywords.score) AS total_score
FROM keywords
JOIN uris ON keywords.id = uris.id
WHERE keywords.term = 'infantri' OR keywords.term = 'reinforc' OR
    keywords.term = 'brigad' OR keywords.term = 'fire'
GROUP BY (uris.url)
ORDER BY total_score DESC;
```

6. This query identifies URLs associated with the keyword 'reinforc' while excluding those that also contain all three of 'infantri', 'brigad', and 'fire', then calculates their score differences.

   - table_A: Retrieves URLs that have the keyword 'reinforc'.
   - table_B: Retrieves URLs that contain all three keywords 'infantri', 'brigad', and 'fire' (ensured by HAVING COUNT(DISTINCT keywords.term) = 3).
   - table_C: Filters URLs from table_A, keeping only those not present in table_B, ensuring the URLs have 'reinforc' but do not have all three excluded terms.
   - table_D: Finds the corresponding id values for the URLs in table_C.
   - table_E: Retrieves the term and score for each id from keywords.
   - Then finally we calculate the score difference for each URL by subtracting the total score of the keywords 'infantri', 'brigad', and 'fire' from the total score of 'reinforc'. It first groups results by url, then computes three values: score_diff (the difference between 'reinforc' and the other three keywords), reinforc_score (the total score for 'reinforc'), and other_score (the combined score for 'infantri', 'brigad', and 'fire'). Finally, the results are sorted in descending order of score_diff, ensuring that URLs where 'reinforc' has the highest relative score are ranked first.

The query is as follows-

```
WITH table_A AS (
    SELECT uris.url
    FROM keywords
    JOIN uris ON keywords.id = uris.id
    WHERE keywords.term = 'reinforc'
    GROUP BY(uris.url)
),
table_B AS (
    SELECT uris.url
    FROM keywords
    JOIN uris ON keywords.id = uris.id
    WHERE keywords.term = 'infantri' OR keywords.term = 'brigad' OR
        keywords.term = 'fire'
    GROUP BY (uris.url)
    HAVING COUNT(DISTINCT keywords.term)=3
),
table_C AS (  --We now have all the necessary urls
    SELECT table_A.url
    FROM table_A
    WHERE table_A.url NOT IN
    (SELECT table_B.url FROM table_B)
),
table_D AS (  --We now have all necessary ids
    SELECT table_C.url, uris.id
    FROM uris
    JOIN table_C ON table_C.url = uris.url
),
table_E AS (
    SELECT table_D.url, keywords.term, keywords.score
    FROM table_D
    JOIN keywords ON keywords.id = table_D.id
)
SELECT table_E.url,
            SUM(CASE WHEN table_E.term = 'reinforc' THEN table_E.score
                ELSE 0 END) -
            SUM(CASE WHEN table_E.term IN ('infantri', 'brigad',
                'fire') THEN table_E.score ELSE 0 END) AS score_diff,
            SUM(CASE WHEN table_E.term = 'reinforc' THEN table_E.score
                ELSE 0 END) AS reinforc_score,
            SUM(CASE WHEN table_E.term IN ('infantri', 'brigad',
                'fire') THEN table_E.score ELSE 0 END) AS other_score
       FROM table_E
       GROUP BY table_E.url
       ORDER BY score_diff DESC;
```