Database Systems

SOEN 363 – Fall 2022

Project - Phase 2

Marwa Khalid (40155098) – khalidmarwa786@gmail.com

William Wells (40111253) - williamwells2013@hotmail.com

Lucas Catchlove (27145640) - lucascatchlove@gmail.com

Shawn Gorman (40157925) - gorman.shawn23@hotmail.com

Professor: Essam Mansour

Group ID: 10

December 14th, 2022

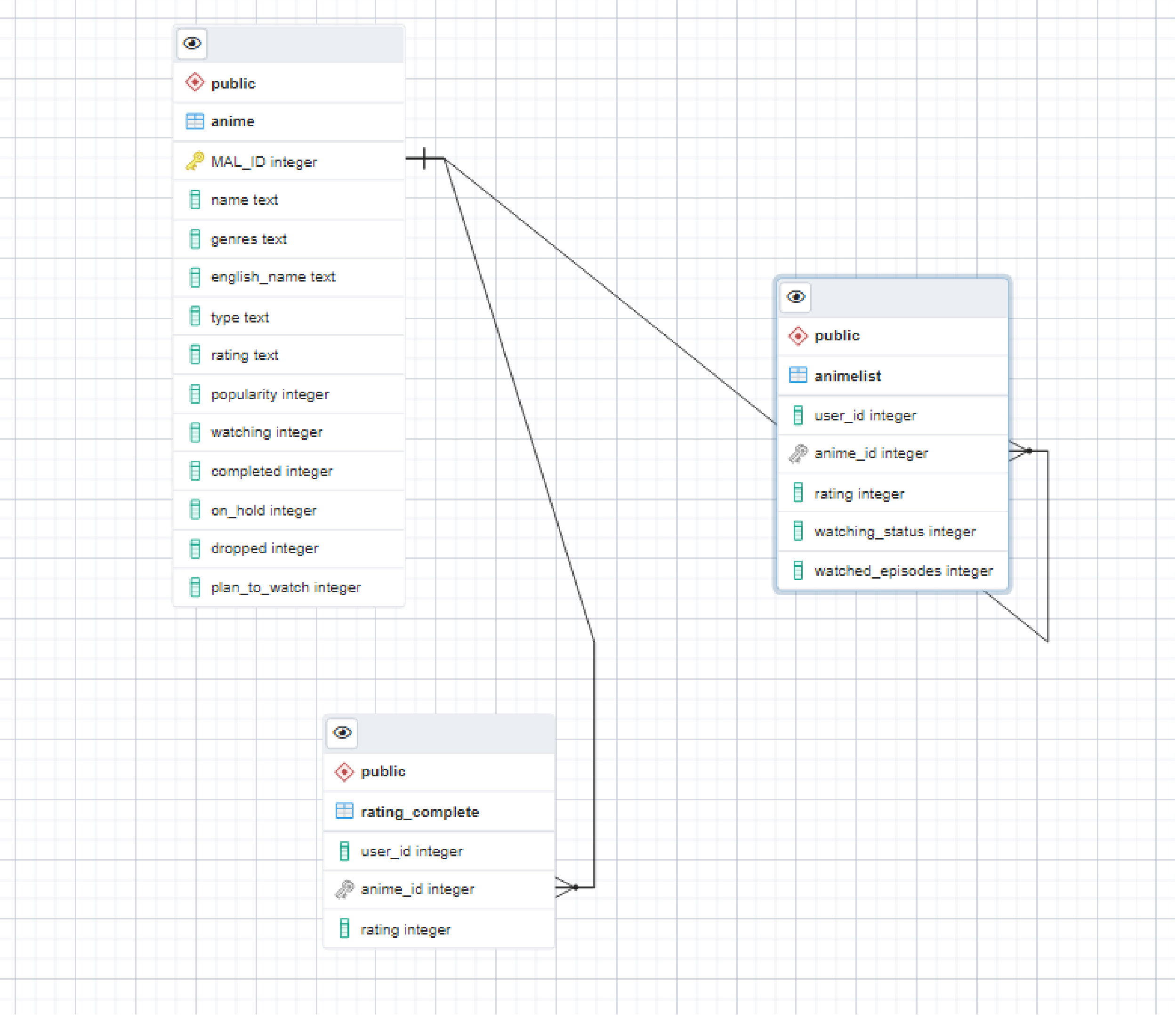
3 Analyzing Big Data Using SQL and RDBMS

Our team decided to use PostgreSQL with pgadmin4 as the technologies used to use SQL for extracting and analyzing useful information from real datasets.

a) This dataset contains information about 17 562 Animes and the preference from over 300 000 different users. In particular, this dataset contains:
   i) **anime_list.csv**: Contains the list of unique animes per user. For each anime the user has given a rating, watched episodes and ‘watching status’. (1 - Currently watching, 2 - Completed, 3 - On Hold, 4 - Dropped, 6 - Plan to watch)
   ii) **ratings_completed.csv:** CSV that contains the list of ratings given by the user to animes with watching status = 2 (complete)
   (note: this file is very large which caused us to disregard this table altogether at the end)
   iii) **anime.csv:** Information of anime scrapped of main page and stats page.

   Total size of dataset: (2.87GB)
   https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020?select=anime.csv

b) ERD Diagram



c) DDL Statements + Load

```
-- Table: public.anime

-- DROP TABLE IF EXISTS public.anime;

CREATE TABLE IF NOT EXISTS public.anime
(
    mal_id integer NOT NULL,
    name text COLLATE pg_catalog."default" NOT NULL,
    genres text COLLATE pg_catalog."default" NOT NULL,
    english_name text COLLATE pg_catalog."default" NOT NULL,
    type text COLLATE pg_catalog."default" NOT NULL,
    rating text COLLATE pg_catalog."default" NOT NULL,
    popularity integer NOT NULL,
    watching integer NOT NULL,
    completed integer NOT NULL,
    on_hold integer NOT NULL,
    dropped integer NOT NULL,
    plan_to_watch integer NOT NULL,
    CONSTRAINT anime_pkey PRIMARY KEY (mal_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.anime
    OWNER to postgres;
```

```
CREATE TABLE IF NOT EXISTS public.animelist
(
    user_id integer NOT NULL,
    anime_id integer,
    rating integer,
    watching_status integer,
    watched_episodes integer,
    CONSTRAINT anime_id FOREIGN KEY (anime_id)
        REFERENCES public.anime ("mal_id") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.animelist
    OWNER to postgres;
-- Index: fki_N

-- DROP INDEX IF EXISTS public."fki_N";

CREATE INDEX IF NOT EXISTS "fki_N"
    ON public.animelist USING btree
    (anime_id ASC NULLS LAST)
    TABLESPACE pg_default;

-- Table: public.rating_complete

-- DROP TABLE IF EXISTS public.rating_complete;
```

```
CREATE TABLE IF NOT EXISTS public.rating_complete
(
    user_id integer NOT NULL,
    anime_id integer NOT NULL,
    rating integer NOT NULL,
    CONSTRAINT anime_id FOREIGN KEY (anime_id)
        REFERENCES public.anime ("mal_id") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.rating_complete
    OWNER to postgres;
-- Index: fki_anime_id

-- DROP INDEX IF EXISTS public.fki_anime_id;

CREATE INDEX IF NOT EXISTS fki_anime_id
    ON public.rating_complete USING btree
    (anime_id ASC NULLS LAST)
    TABLESPACE pg_default;
```

d) Reports

1. Find all anime that do not have a rating of 1

```
SELECT A.name AS not_rating_of_1
FROM anime A
WHERE A.mal_id NOT IN(SELECT AL.anime_id
                                          FROM animelist AL
                                          WHERE AL.rating = 1)
```

```
Total rows: 14796
Query complete 00:00:00.146
```

2. Find all names of animes that have a higher popularity than anime called ‘Final Approach’

   ```
   SELECT A.name
   FROM anime A
   WHERE A.popularity > ALL
   (SELECT A2.popularity
   FROM anime a2
   WHERE A2.name = 'Final Approach')
   ```

   Total rows: 14671
   Query complete 00:00:40.838

3. Find all anime which contain the word “hack” in their name.

   ```
   SELECT name
   FROM anime
   WHERE name LIKE '%hack%'
   ORDER BY name
   ```

   Total rows: 16
   Query complete 00:00:00.051

4. Find all anime of type “TV” with a popularity higher than 100.

   ```
   SELECT name, type, popularity
   FROM anime
   WHERE type = 'TV' AND popularity > 100
   ORDER by popularity
   ```

   Total rows: 4901
   Query complete 00:00:00.049

5. Find the amount of episodes watched for each rating category.

   ```
   SELECT COUNT(watched_episodes) as watched_episodes, rating
   FROM animelist
   GROUP BY rating
   ORDER BY COUNT(watched_episodes) desc
   ```

   Total rows: 11

```
Query complete 00:00:00.179
```

6. Find the top ten animes which have been dropped.

   ```
   SELECT name, dropped
   FROM anime
   ORDER BY dropped DESC
   LIMIT 10

   Total rows: 10
   Query complete 00:00:00.048
   ```

7. Find all anime which not only having a rating of 10 but also are PG-13 friendly.

   ```
   SELECT name, rating
   FROM anime
   WHERE mal_id IN(SELECT anime_id
                   FROM animelist
                   WHERE rating = 10) AND rating LIKE '%PG-13%'

   Total rows: 2911
   Query complete 00:00:00.135
   ```

8. Find the 5 lowest rated (and completely watched) anime along with their name and popularity.

   ```
   SELECT name, popularity
   FROM anime
   WHERE mal_id IN (SELECT anime_id
                          FROM rating_complete A
                          ORDER BY A.rating
                          LIMIT 5)
   Total rows: 5
   Query complete 00:00:03.733
   ```

9. Find the names of the top 10 highest rated (and completely watched) anime.

   ```
   SELECT name
   FROM anime
   WHERE mal_id IN (
   ```

```
        SELECT anime_id
        FROM rating_complete
        ORDER BY rating DESC
        LIMIT 10
        )
```

Total rows: 10
Query complete 00:00:03.619

10. Find all anime that have a different english name.

```
SELECT name, english_name
FROM anime
WHERE name != english_name
```

Total rows: 16292
Query complete 00:00:00.082

e) Explore indexes to enhance the performance of these queries

1. Find all anime that do not have a rating of 1

```
SELECT A.name AS not_rating_of_1
FROM anime A
WHERE A.mal_id NOT IN(SELECT AL.anime_id
                      FROM animelist AL
                      WHERE AL.rating = 1)

CREATE INDEX idx_anime ON anime(mal_id, name)
CREATE INDEX idx_animelist ON animelist(anime_id, rating)
```

Total rows: 14796
Query complete 00:00:00.054

2. Find all names of animes that have a higher popularity than anime called ‘Final Approach’

```
SELECT A.name
FROM anime A
WHERE A.popularity > ALL
```

```
(SELECT A2.popularity
FROM anime a2
WHERE A2.name = 'Final Approach')
```

```
CREATE INDEX idx_anime ON anime(popularity, name)
```

```
Total rows: 14671
Query complete 00:00:00.078
```

3. Find all anime which contain the word “hack” in their name.

```
SELECT name
FROM anime
WHERE name LIKE '%hack%'
ORDER BY name
```

```
CREATE INDEX idx_anime ON anime(name)
```

```
Total rows: 16
Query complete 00:00:00.085
```

4. Find all anime of type “TV” with a popularity higher than 100.

```
SELECT name, type, popularity
FROM anime
WHERE type = 'TV' AND popularity > 100
ORDER by popularity
```

```
CREATE INDEX idx_anime ON anime(name, type, popularity)
```

```
Total rows: 4901
Query complete 00:00:00.033
```

5. Find the amount of episodes watched for each rating category.

```
SELECT COUNT(watched_episodes) as watched_episodes, rating
FROM animelist
GROUP BY rating
ORDER BY COUNT(watched_episodes) desc
```

```
CREATE INDEX idx_animelist ON animelist(watched_episodes, rating)
```

```
Total rows: 11
Query complete 00:00:00.148
```

6. Find the top ten animes which have been dropped.

```
SELECT name, dropped
FROM anime
ORDER BY dropped DESC
LIMIT 10

CREATE INDEX idx_anime ON anime(name, dropped)

Total rows: 10
Query complete 00:00:00.043
```

7. Find all anime which not only having a rating of 10 but also are PG-13 friendly.

```
SELECT name, rating
FROM anime
WHERE mal_id IN(SELECT anime_id
                 FROM animelist
                 WHERE rating = 10) AND rating LIKE '%PG-13%'

CREATE INDEX idx_anime ON anime(mal_id, name, rating)
CREATE INDEX idx_animelist ON animelist(anime_id, rating)

Total rows: 2911
Query complete 00:00:00.093
```

8. Find the 5 lowest rated (and completely watched) anime along with their name and popularity.

```
SELECT name, popularity
FROM anime
WHERE mal_id IN (SELECT anime_id
                        FROM rating_complete A
                        ORDER BY A.rating
                        LIMIT 5)
```

```
CREATE INDEX idx_anime ON anime(mal_id, name, popularity)
CREATE INDEX idx_ratingcomplete ON rating_complete(anime_id, rating)
```

```
Total rows: 5
Query complete 00:00:02.676
```

9. Find the names of the top 10 highest rated (and completely watched) anime.

```
SELECT name
FROM anime
WHERE mal_id IN (
        SELECT anime_id
        FROM rating_complete
        ORDER BY rating DESC
        LIMIT 10
        )
```

```
CREATE INDEX idx_anime ON anime(mal_id, name)
CREATE INDEX idx_ratingcomplete ON rating_complete(anime_id, rating)
```

```
Total rows: 10
Query complete 00:00:02.659
```

10. Find all anime that have a different english name.

```
SELECT name, english_name
FROM anime
WHERE name != english_name
```

```
CREATE INDEX idx_anime ON anime(english_name, name)
```

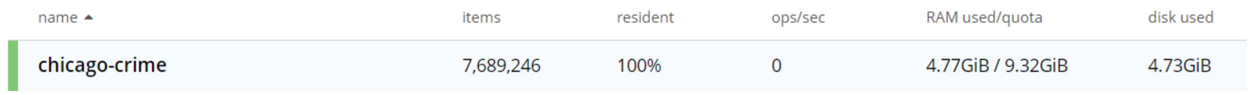
```
Total rows: 16292
Query complete 00:00:00.048
```

5 Analyzing Big Data Using NoSQL Systems

Note: Couchebase was used as the NoSQL database solution

a)

The dataset used pertained to reported incidents of crime in the City of Chicago from 2001 and onward. The actual csv file is approximately 1.5GB while its json counterpart is 4.7GB. The size of the bucket within Couchbase Server is approximately ~4.7GB (unsurprisingly about the size of the json file).



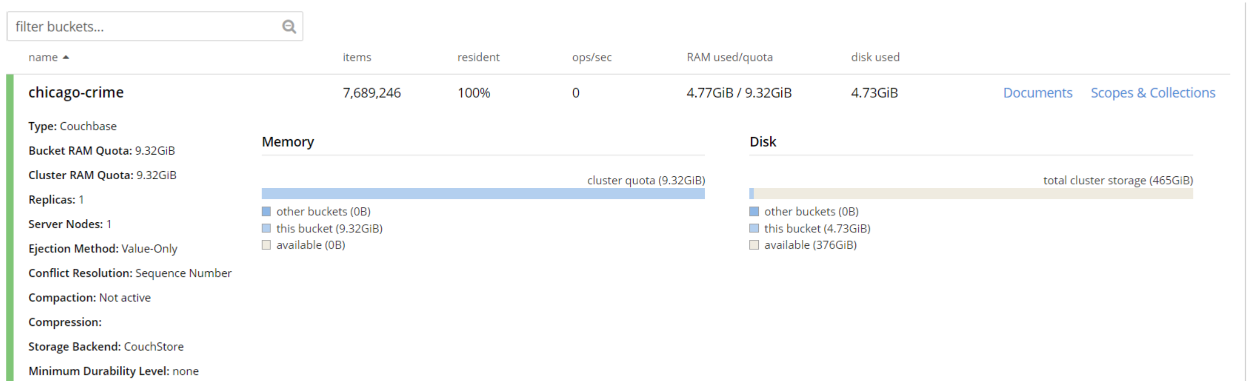
name ▲	items	resident	ops/sec	RAM used/quota	disk used
chicago-crime	7,689,246	100%	0	4.77GiB / 9.32GiB	4.73GiB

b)

Couchebase is a document database. All crimes that took place in the City of Chicago since 2001 are documented with the following attributes:

- Arrest (boolean)
- Beat (number)
- Block (string)
- Case Number (string)
- Community Area (number,string)
- Date (string)
- Description (string)
- District (number)
- Domestic (boolean)
- FBI Code (number,string)
- ID (number)
- IUCR (number,string)
- Latitude (number,string)
- Location (string)
- Location Description (string)
- Longitude (number,string)
- Primary Type (string)
- Updated On (string)
- Ward (number,string)
- X Coordinate (number,string)
- Y Coordinate (number,string)
- Year (number)

c) and d)



name ▲	items	resident	ops/sec	RAM used/quota	disk used
chicago-crime	7,689,246	100%	0	4.77GiB / 9.32GiB	4.73GiB

Type: Couchbase
Bucket RAM Quota: 9.32GiB
Cluster RAM Quota: 9.32GiB
Replicas: 1
Server Nodes: 1
Ejection Method: Value-Only
Conflict Resolution: Sequence Number
Compaction: Not active
Compression:
Storage Backend: CouchStore
Minimum Durability Level: none

Memory

cluster quota (9.32GiB)

- other buckets (0B)
- this bucket (9.32GiB)
- available (0B)

Disk

total cluster storage (465GiB)

- other buckets (0B)
- this bucket (4.73GiB)
- available (376GiB)

Queries so far:

1-

top_10_districts_most_domestic_assaults

```
SELECT District,
       COUNT(*) AS domestic_assault_incidents
FROM `chicago-crime`
WHERE `Primary Type` = "ASSAULT"
   AND Domestic=TRUE
GROUP BY District
ORDER BY domestic_assault_incidents DESC
LIMIT 10
```

```
CREATE INDEX idx_domestic_assault ON `chicago-crime` (District, `Primary Type`);
```

2-

```
SELECT Year,
       COUNT(*) AS number_of_burglaries
FROM `chicago-crime`
WHERE `Primary Type`="BURGLARY"
GROUP BY Year
ORDER BY Year DESC
```

```
CREATE INDEX idx_burglaries ON `chicago-crime` (Year, `Primary Type`);
```

3- arrest rate

```
SELECT (a.arrests / ta.total_arrests)*100 AS arrest_rate_percentage
FROM (
   SELECT COUNT(*) AS arrests
   FROM `chicago-crime`
   WHERE Arrest=TRUE) AS a,
(
   SELECT COUNT(*) AS total_arrests
   FROM `chicago-crime`) ta

CREATE INDEX idx_arrest ON `chicago-crime` (Arrest);
```

4- cannabis possession offenses by community area

```
SELECT `Community Area`,
     COUNT(Description) cannabis_possession_offenses
FROM `chicago-crime`
WHERE Description LIKE "%POSS: CANNABIS%"
GROUP BY `Community Area`

CREATE INDEX idx_cannabis_possession ON `chicago-crime` (`Community Area`,
Description);
```

5- crimes involving a firearm at a school

```
SELECT `Case Number`,
     Description
FROM `chicago-crime`
WHERE Description LIKE "%FIREARM%"
   AND `Location Description` LIKE "%SCHOOL%"

CREATE INDEX idx_firearm_school ON `chicago-crime` (Description, `Location
Description`);
```

6 - Primary type of crime and the number of crimes for each district

```
SELECT `Primary Type`, `District`, COUNT(*) AS `NumberOfCrimes`
FROM `chicago-crime`
GROUP BY `Primary Type`, `District`

CREATE INDEX idx_primary_type_district ON `chicago-crime` (`Primary Type`, District);
```

7 - Top 10 most common crime descriptions

```
SELECT `Description`, COUNT(*) AS `NumberOfOccurrences`
FROM `chicago-crime`
GROUP BY `Description`
ORDER BY `NumberOfOccurrences` DESC
LIMIT 10

CREATE INDEX idx_description ON `chicago-crime` (Description);
```

8 - All crimes that occurred on a specific date

```
SELECT *
FROM `chicago-crime`
WHERE `Date` LIKE '04/19/2004%'

CREATE INDEX idx_date ON `chicago-crime` (Date);
```

9 - All crimes that occurred within a specific location or block

```
SELECT *
FROM `chicago-crime`
WHERE `Block` = '004XX E 63RD ST'

CREATE INDEX idx_block ON `chicago-crime` (Block);
```

10 - Number of crimes that occurred in each community area

```
SELECT `Community Area`, COUNT(*) AS `NumberOfCrimes`
FROM `chicago-crime`
GROUP BY `Community Area`

CREATE INDEX idx_community_area ON `chicago-crime` (`Community Area`);
```

f)

Couchbase allows you to configure the level of consistency and availability for your data using a number of different options, such as the consistency mode, the replication factor, and the durability level.

This puts the balancing consistency and availability in the database completely into the hands of the database administrator and maintainer.

g)

Couchbase supports a variety of indexing techniques, including primary indexes, secondary indexes, and functional indexes.

Primary indexes are the default indexes that are automatically created for every bucket in Couchbase.

Secondary indexes allow you to index any attribute or combination of attributes in your documents, allowing you to perform efficient lookups and queries based on those attributes.

Functional indexes are special types of indexes that store the results of a particular function or expression, rather than the raw values of the indexed attributes.