SQL Project Portfolio

Welcome to my SQL Projects Portfolio. I am Ambili AN, a data enthusiast passionate about unlocking the potential within raw data to tell powerful stories that inform decisions and provide deep insights across various sectors. This portfolio aims to display a curated collection of projects that showcase my skills in utilizing SQL for data analysis, management, and insight generation.

Each project featured in this portfolio demonstrates my dedication to excellence in data management and analysis. The projects highlighted here include real work done during my tenure at companies like Emerge , True Value Cars, Plumcot IT. It's important to note that while these projects are based on real experiences, I have altered data and project titles to comply with confidentiality agreements previously signed. These modifications are carefully done to ensure the privacy and confidentiality policies of the involved companies are upheld, while still preserving the core of the projects. This approach allows me to showcase my expertise and accomplishments without compromising sensitive information.

As you go through this document, you'll find in-depth descriptions of the SQL projects I've led or contributed to, including the obstacles encountered, the strategies employed to overcome them, and the outcomes realized. My aim is not just to highlight my technical proficiency with SQL but also to illustrate my ongoing journey of learning and development in the realm of data analytics.

Thank you for investing the time to review my portfolio. I am keen to apply my skills and experience towards contributing to data-centric projects and initiatives. For any inquiries or to discuss potential collaborations,

Kind Regards

Ambili AN
0274083831 /ambiliarangathnarayanan@gmail.com

1

Project -1

Project Title: Netflix Shows and Movies Project

About Dataset

Netflix - TV Shows and Movies

This data set was created to list all shows available on Netflix streaming, and analyze the data to find interesting facts. This data was acquired in July 2022 containing data available in the United States.

Content

This dataset has two files containing the titles (**titles.csv**) and the cast (**credits.csv**) for the title.
This dataset contains **+5k** unique **titles on Netflix** with 15 columns containing their information, including:

- id: The title ID on JustWatch.
- title: The name of the title.
- show_type: TV show or movie.
- description: A brief description.
- release_year: The release year.
- age_certification: The age certification.
- runtime: The length of the episode (SHOW) or movie.
- genres: A list of genres.
- production_countries: A list of countries that produced the title.
- seasons: Number of seasons if it's a SHOW.
- imdb_id: The title ID on IMDB.
- imdb_score: Score on IMDB.
- imdb_votes: Votes on IMDB.
- tmdb_popularity: Popularity on TMDB.
- tmdb_score: Score on TMDB.

Business Problem: Netflix wants to gather useful insights on their shows and movies for their subscribers through their datasets. The issue is, they are working with too much data (approximately 82k rows of data combined) and are unsure how to effectively analyze and extract meaningful insights from it. They need a robust and scalable data analytics solution to handle the vast amount of data and uncover valuable patterns and trends.

How I Plan On Solving the Problem: In helping Netflix gather valuable insights from their extensive movies and shows dataset, I will be utilizing SQL and a data visualization tool like Tableau to extract relevant information, and conduct insightful analyses. By leveraging SQL's functions, I can uncover key metrics such as viewer ratings, popularity trends, genre preferences, and viewership patterns. Once the data has been extracted and prepared, I will leverage Tableau to present the findings. This will allow for interactive exploration of the data, enabling stakeholders at Netflix to gain actionable insights through visually appealing charts, graphs, and interactive visualizations. I plan on creating a dynamic dashboard in Tableau that enables users to delve into specific movie genres, viewer demographics, or geographical regions.

1. Which movies and shows on Netflix ranked in the top 10 and bottom 10 based on their IMDB scores?

- Top 10 Movies

```sql
SELECT title,
type,
imdb_score
FROM shows_movies.titles
WHERE imdb_score >= 8.0
AND type = 'MOVIE'
ORDER BY imdb_score DESC
LIMIT 10
```

Result:

title	type	imdb_score
Chhota Bheem & Krishna vs Zimbara	MOVIE	9.1
Major	MOVIE	9.1
David Attenborough: A Life on Our Planet	MOVIE	8.9
C/o Kancharapalem	MOVIE	8.9
Inception	MOVIE	8.8
Forrest Gump	MOVIE	8.8
Chhota Bheem & Krishna in Mayanagari	MOVIE	8.7
Bo Burnham: Inside	MOVIE	8.7
A Lion in the House	MOVIE	8.7
Chhota Bheem Neeli Pahaadi	MOVIE	8.7

- Top 10 Shows

```
SELECT title,
type,
imdb_score
FROM shows_movies.titles
WHERE imdb_score >= 8.0
AND type = 'SHOW'
ORDER BY imdb_score DESC
LIMIT 10
```

Result:

title	type	imdb_score
#ABtalks	SHOW	9.6
Khawatir	SHOW	9.5
Breaking Bad	SHOW	9.5
Our Planet	SHOW	9.3
Avatar: The Last Airbender	SHOW	9.3
Reply 1988	SHOW	9.2
Kota Factory	SHOW	9.1
The Last Dance	SHOW	9.1
My Mister	SHOW	9.1
Okupas	SHOW	9

- Bottom 10 Movies

```
SELECT title,
type,
imdb_score
FROM shows_movies.titles
WHERE type = 'MOVIE'
ORDER BY imdb_score ASC
LIMIT 10
```

Result:

title	type	imdb_score
He's Expecting	SHOW	2
Thomas & Friends: All Engines Go!	SHOW	2
Hype House	SHOW	2.1
A House of Blocks	SHOW	2.3
Until Dawn	SHOW	2.4
The Goop Lab	SHOW	2.5
Byron Baes	SHOW	2.6
First Class	SHOW	2.8
Bonus Family	SHOW	2.9
Richie Rich	SHOW	3

An IMDB score is a widely recognized measure of the overall quality and popularity of a movie or show. The top 10 movies and shows stood out for their exceptional IMDB scores, indicating that they are highly regarded by viewers. These titles have likely garnered significant acclaim and positive reviews, contributing to their high rankings within the Netflix library. Viewers who are seeking quality content would find these selections very appealing. On the other hand, the bottom 10 movies and shows had lower IMDB scores. While these entries may not have resonated as strongly with audiences, it's important to note that many factors influence these rankings such as individual preferences, weak plot, poor acting, and low-quality production. By uncovering the top and bottom performers based on IMDB scores, this project sheds light on the varying levels of audience reception and highlights titles that are likely to be well-received and those that may have room for improvement. These findings can provide valuable insights for viewers seeking highly-rated content and can serve as a basis for further analysis and decision-making for Netflix's audience recommendations.

2. How many movies and shows fall in each decade in Netflix's library?

```
SELECT CONCAT(FLOOR(release_year / 10) * 10, 's') AS decade,
        COUNT(*) AS movies_shows_count
FROM shows_movies.titles
WHERE release_year >= 1940
GROUP BY CONCAT(FLOOR(release_year / 10) * 10, 's')
ORDER BY decade;
```

Result:

decade	movies_shows_count
1940s	1
1950s	5
1960s	8
1970s	18
1980s	52
1990s	121
2000s	369
2010s	3304
2020s	1972

The results of the SQL query provide a fascinating insight into the distribution of movies and shows across different decades in Netflix's library. The data reveals a significant shift in content availability over time, with a notable increase in the number of titles from the 2000s onwards. Starting from the earlier decades, the 1940s-1980s showcase a small fraction of the total entries, suggesting that Netflix's collection from these decades is relatively limited. The 1990s demonstrate a large surge in offerings, with 121 titles. However, the true turning point in Netflix's library occurs in the 2010s with a remarkable 3,304 movies and shows from this decade. This abundance highlights Netflix's dedication to featuring contemporary content that aligns with current trends and audience preferences.

Even though the 2020s are still in progress, the dataset reveals an impressive count of 1,972 movies and shows, indicating a strong focus on acquiring and producing content from recent years. Overall, these findings shed light on Netflix's strategy of curating library that covers a wide range of decades. The significant increase in content availability from the 2000s onwards suggests a concerted effort to offer a diverse selection of titles. This collection spanning multiple decades allows Netflix's audience to explore a variety of movies and shows that reflect different eras.

3. How did age-certifications impact the dataset?

```
SELECT DISTINCT age_certification,
ROUND(AVG(imdb_score),2) AS avg_imdb_score,
ROUND(AVG(tmdb_score),2) AS avg_tmdb_score
FROM shows_movies.titles
GROUP BY age_certification
ORDER BY avg_imdb_score DESC
```

Result:

age_certification	avg_imdb_score
TV-14	6.71
TV-MA	6.62
TV-PG	6.37
TV-Y7	6.32
PG-13	6.2
TV-G	6.01
R	6
TV-Y	6
PG	5.96
NC-17	5.76
N/A	5.58
G	5.09

```sql
SELECT age_certification,
COUNT(*) AS certification_count
FROM shows_movies.titles
WHERE type = 'Movie'
AND age_certification != 'N/A'
GROUP BY age_certification
ORDER BY certification_count DESC
LIMIT 5;
```

Results:

age_certification	certification_count
R	556
PG-13	451
PG	233
G	124
NC-17	16

The first query focused on the average IMDB scores associated with each age certification, revealing interesting trends in audience ratings. According to the data, TV-14 emerges as the age certification with the highest average IMDB score of 6.71. This suggests that content designated for viewers aged 14 and older tends to receive relatively favorable ratings. The age certification TV-G obtains an average score of 6.01, signaling the appreciation for content suitable for all audiences. On the other hand, the age certifications R and TV-Y, each with average scores of 6, demonstrate that while they may have lower ratings, there is still a substantial audience that finds enjoyment in these respective categories.

When examining the distribution of movies and shows across age certifications, the second query showcases the varying prevalence of different certifications within Netflix's dataset. R emerges as the most prevalent age certification, with 556 titles falling under this category. PG-13 closely follows with 451 titles, reflecting a significant number of movies and shows targeted at mature audiences. The age certification PG accounts for 233 titles, indicating a considerable selection suitable for general audiences. The dataset also includes 124 titles classified as G, which mostly caters to a younger audience. Lastly, the least represented certification is NC-17, with only 16 titles available. These findings highlight the diverse range of age certifications present in Netflix's movies and shows dataset and provide valuable insights into both audience preferences and content distribution. The higher average scores associated with

TV-14, TV-MA, and TV-PG certifications suggest that content aligned with these age categories tends to resonate positively with viewers.

4. Which genres are the most common?

- Top 10 most common genres for MOVIES

```
SELECT genres,
COUNT(*) AS title_count
FROM shows_movies.titles
WHERE type = 'Movie'
GROUP BY genres
ORDER BY title_count DESC
LIMIT 10;
```

Result:

genres	title_count
['comedy']	384
['documentation']	230
['drama']	224
['comedy', 'documentation']	100
['comedy', 'drama']	84
['drama', 'romance']	76
['drama', 'comedy']	65
['comedy', 'romance']	60
['romance', 'comedy']	48
['comedy', 'drama', 'romance']	44

- Top 10 most common genres for SHOWS

```
SELECT genres,
COUNT(*) AS title_count
FROM shows_movies.titles
WHERE type = 'Show'
GROUP BY genres
ORDER BY title_count DESC
LIMIT 10;
```

Result:

genres	title_count
['reality']	113
['drama']	104
['comedy']	100
['documentation']	99
['comedy', 'drama']	51
['drama', 'romance']	48
['drama', 'comedy']	44
['documentation', 'crime']	35
['animation']	30
[]	25

- Top 3 most common genres OVERALL

```
SELECT t.genres,
COUNT(*) AS genre_count
FROM shows_movies.titles AS t
WHERE t.type = 'Movie' or t.type = 'Show'
GROUP BY t.genres
ORDER BY genre_count DESC
LIMIT 3;
```

Result:

genres	genre_count
['comedy']	484
['documentation']	329
['drama']	328

By analysing the frequency of genres, we can gain a better understanding of the content that dominates the platform and the preferences of its audience. Starting with movies, the first query reveals the top 10 most common genres. Comedy emerges as the most popular genre with a total of 384 movies, reflecting its widespread appeal. Following closely behind are documentation with 230 movies and drama with 224 movies, indicating the significance of these genres in Netflix's movie collection. Combinations of genres also feature prominently, with comedy + documentation and comedy + drama occupying the fourth and fifth positions respectively. The presence of drama + romance, drama + comedy, and comedy + romance further emphasizes the audience's likeness for movies that blend multiple genres. These findings highlight the diverse range of movie genres available on Netflix and the platform's commitment to catering to a wide array of preferences.

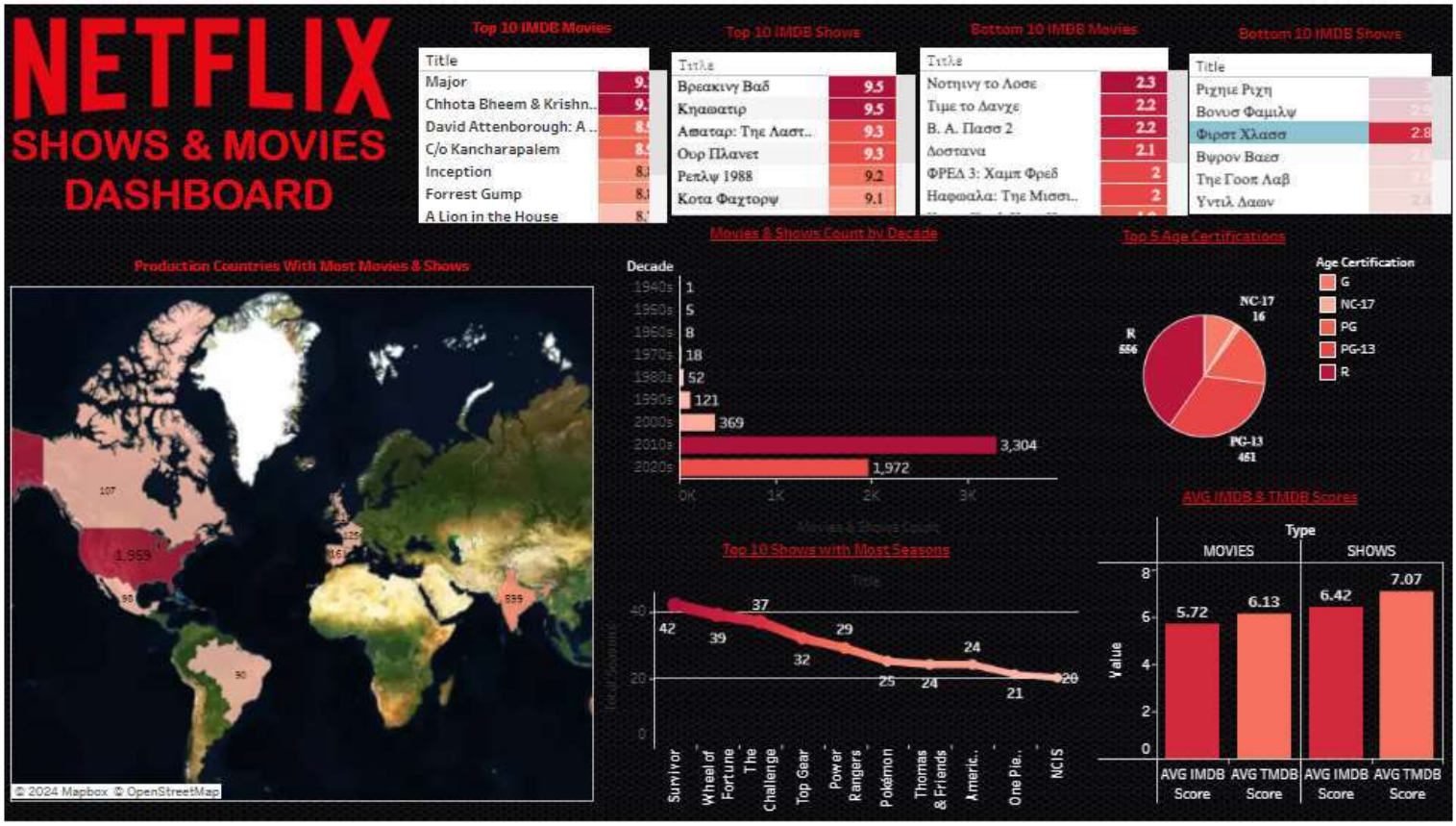
Shifting the focus, the second query presents the top 10 most common genres for shows. Reality takes the lead with 113 shows, showcasing the popularity of this genre among Netflix viewers. Drama follows closely behind with 104 shows. Comedy and documentation also emerge as prevalent genres with 100 shows each. Similar to movies, combinations of genres such as comedy + drama and drama + romance are present, indicating viewer interest in multi-genre shows.

Combining the results from both movies and shows, the third query provides an overview of the top three most common genres overall. Comedy takes the lead with a total of 484 entries, reaffirming its position as a very popular genre among Netflix subscribers. Documentation follows closely behind with 329 entries, reflecting the popularity of informative content. Finally, drama secures the third

spot with 328 entries. Overall, these findings shed light on the genres that dominate Netflix's library, showcasing the platform's efforts to cater to a diverse range of viewer preferences.

Conclusion

By exploring various aspects of the dataset, a comprehensive understanding of Netflix's content landscape was gained. The analysis revealed the top 10 and bottom 10 movies and shows based on their IMDB scores, which highlighted the titles that garnered high praise and those that received lower ratings. This information can assist viewers in making informed choices and highlight areas for potential improvement in content quality. The examination of movies and shows distributed across different decades showed significant shifts in content availability over time. Notably, the dataset showcased a substantial increase in offerings from the 2000s onwards, emphasizing Netflix's commitment to featuring newer content that resonates with current trends and audience preferences.



2

Project -2

Project Title: Netflix Shows and Movies Project

```
/*
	Data Analyst Job Postings
	SQL Author: Jaime M. Shaker
	Dataset Creator: Luke Barousse (https://www.linkedin.com/in/luke-b/) (https://www.youtube.com/@LukeBarousse)
	Dataset Location: https://www.kaggle.com/datasets/lukebarousse/data-analyst-job-postings-google-search
	Email: jaime.m.shaker@gmail.com or jaime@shaker.dev
	Website: https://www.shaker.dev
	LinkedIn: https://www.linkedin.com/in/jaime-shaker/

	File Name: build_tables.sql
	Description:  This script will import data from the CSV files and create the
	schema, tables and table relationships for this project.  Once it is complete,
	it will drop any unecessary schemas and tables.
*/

-- Create an schema and table for importing data
DROP SCHEMA import_data CASCADE;
CREATE SCHEMA IF NOT EXISTS import_data;

/*
 *  Github only allows file sizes smaller than 100mb.  This dataset is at over 130mb on 2023-09-09.
 *  I split the CSV files into standard calendar quarters (Q1, Q2, Q3, Q4) for those that wish to use the CSV files on my repo.
 *
 */

-- Import data from CSV files
-- Create Data Analyst Jobs Table
DROP TABLE IF EXISTS import_data.jobs;
CREATE TABLE import_data.jobs (
	data_job_id TEXT,
	idx TEXT,
	title TEXT,
	company_name TEXT,
	job_location TEXT,
	via TEXT,
	description TEXT,
	extensions TEXT,
	job_id TEXT,
	thumbnail TEXT,
	posted_at TEXT,
	schedule_type TEXT,
	work_from_home TEXT,
	salary TEXT,
	search_term TEXT,
	date_time TEXT,
```

```
        search_location TEXT,
        commute_time TEXT,
        salary_pay TEXT,
        salary_rate TEXT,
        salary_avg TEXT,
        salary_min TEXT,
        salary_max TEXT,
        salary_hourly TEXT,
        salary_yearly TEXT,
        salary_standardized TEXT,
        description_tokens TEXT,
        PRIMARY KEY (data_job_id)
);
-- gsearch_jobs 2022 CSV file.
COPY import_data.jobs (
        data_job_id,
        idx,
        title,
        company_name,
        job_location,
        via,
        description,
        extensions,
        job_id,
        thumbnail,
        posted_at,
        schedule_type,
        work_from_home,
        salary,
        search_term,
        date_time,
        search_location,
        commute_time,
        salary_pay,
        salary_rate,
        salary_avg,
        salary_min,
        salary_max,
        salary_hourly,
        salary_yearly,
        salary_standardized,
        description_tokens
)
FROM '/var/lib/postgresql/source_data/csv/gsearch_jobs_2022.csv'
WITH DELIMITER ',' HEADER CSV;

-- gsearch_jobs_2023_q1 CSV file.
COPY import_data.jobs (
        data_job_id,
        idx,
        title,
        company_name,
        job_location,
        via,
        description,
        extensions,
        job_id,
```

```
        thumbnail,
        posted_at,
        schedule_type,
        work_from_home,
        salary,
        search_term,
        date_time,
        search_location,
        commute_time,
        salary_pay,
        salary_rate,
        salary_avg,
        salary_min,
        salary_max,
        salary_hourly,
        salary_yearly,
        salary_standardized,
        description_tokens
)
FROM '/var/lib/postgresql/source_data/csv/gsearch_jobs_2023_q1.csv'
WITH DELIMITER ',' HEADER CSV;

-- gsearch_jobs_2023_q2 CSV file.
COPY import_data.jobs (
        data_job_id,
        idx,
        title,
        company_name,
        job_location,
        via,
        description,
        extensions,
        job_id,
        thumbnail,
        posted_at,
        schedule_type,
        work_from_home,
        salary,
        search_term,
        date_time,
        search_location,
        commute_time,
        salary_pay,
        salary_rate,
        salary_avg,
        salary_min,
        salary_max,
        salary_hourly,
        salary_yearly,
        salary_standardized,
        description_tokens
)
FROM '/var/lib/postgresql/source_data/csv/gsearch_jobs_2023_q2.csv'
WITH DELIMITER ',' HEADER CSV;

-- gsearch_jobs_2023_q3 CSV file.
COPY import_data.jobs (
```

```
        data_job_id,
        idx,
        title,
        company_name,
        job_location,
        via,
        description,
        extensions,
        job_id,
        thumbnail,
        posted_at,
        schedule_type,
        work_from_home,
        salary,
        search_term,
        date_time,
        search_location,
        commute_time,
        salary_pay,
        salary_rate,
        salary_avg,
        salary_min,
        salary_max,
        salary_hourly,
        salary_yearly,
        salary_standardized,
        description_tokens
)
FROM '/var/lib/postgresql/source_data/csv/gsearch_jobs_2023_q3.csv'
WITH DELIMITER ',' HEADER CSV;

-- Create working table with 'cleaned' data.

DROP SCHEMA IF EXISTS data_analyst CASCADE;
CREATE SCHEMA IF NOT EXISTS data_analyst;

-- Import data from CSV files
-- Create Data Analyst Jobs Table
DROP TABLE IF EXISTS data_analyst.jobs;
CREATE TABLE data_analyst.jobs (
        data_job_id int UNIQUE,
        idx int NOT NULL,
        title TEXT NOT NULL,
        company_name TEXT NOT NULL,
        job_location TEXT,
        via TEXT,
        description TEXT,
        extensions TEXT [],
        job_id TEXT,
        thumbnail TEXT,
        posted_at TEXT,
        schedule_type TEXT,
        work_from_home boolean,
        salary TEXT,
        search_term TEXT,
        date_time timestamp NOT NULL,
        search_location TEXT,
```

```
        commute_time TEXT,
        salary_pay TEXT,
        salary_rate TEXT,
        salary_avg NUMERIC,
        salary_min NUMERIC,
        salary_max NUMERIC,
        salary_hourly NUMERIC,
        salary_yearly NUMERIC,
        salary_standardized NUMERIC,
        description_tokens TEXT [],
        PRIMARY KEY (data_job_id)
);

INSERT INTO data_analyst.jobs (
        data_job_id,
        idx,
        title,
        company_name,
        job_location,
        via,
        description,
        extensions,
        job_id,
        thumbnail,
        posted_at,
        schedule_type,
        work_from_home,
        salary,
        search_term,
        date_time,
        search_location,
        commute_time,
        salary_pay,
        salary_rate,
        salary_avg,
        salary_min,
        salary_max,
        salary_hourly,
        salary_yearly,
        salary_standardized,
        description_tokens
) (
        SELECT
                data_job_id::int,
                idx::int,
                lower(trim(title)) AS title,
                lower(trim(company_name)) AS company_name,
                lower(trim(job_location)) AS job_location,
                -- Remove 'via ' from column
                lower(trim(RIGHT(via, length(via) - 4))) AS via,
                -- Remove Pilcrow character and replace with newline
                trim(regexp_replace(description, E'[\\n]+', chr(13), 'g' )) AS description,
                -- Format to a proper PostgreSQL Array
                string_to_array(REGEXP_REPLACE(extensions, '[\[\]'']', '', 'g'), ', ') AS extensions,
                job_id,
                thumbnail,
                posted_at,
```

```
                schedule_type,
                upper(work_from_home)::boolean AS work_from_home,
                salary,
                search_term,
                date_time::timestamp,
                search_location,
                commute_time,
                salary_pay,
                salary_rate,
                salary_avg::NUMERIC,
                salary_min::NUMERIC,
                salary_max::NUMERIC,
                salary_hourly::NUMERIC,
                salary_yearly::NUMERIC,
                salary_standardized::NUMERIC,
                -- Format to a proper PostgreSQL Array
                string_to_array(REGEXP_REPLACE(description_tokens,  '[\[\]'']',  '',  'g'),  ',  ')  AS
description_tokens
        FROM
                import_data.jobs
);

-- Remove the posted_at value from the extensions column

UPDATE data_analyst.jobs
SET
        extensions = array_remove(extensions, posted_at);

-- Drop import schema and table

DROP TABLE import_data.jobs;
DROP SCHEMA import_data CASCADE;
```

3

COSC444—Advanced Database Technologies—2023

Assignment 1: Relational Theory

Question

Consider a database relation of student records as follows:

Student (SID, GRADE, PCODE, PNAME, SEM, YEAR, ENROL, LECT, ROOM, RSIZE)
You can abbreviate this as **Student (I,G,P,M,S,Y,E,L,R,C)**

Attribute	Abbreviation	Description
SID	I	Student ID
GRADE	G	Student's grade for this paper
PCODE	P	Paper's 7-character code
PNAME	M	Paper's name
SEM	S	Semester in which the paper is taught
YEAR	Y	Year in which the paper was offered
ENROL	E	Number of students enrolled in the paper
LECT	L	Lecturer's name, for the lecturer that is the paper's coordinator
ROOM	R	The room in which lectures for this paper are held
RSIZE	C	The maximum capacity of the room in which lectures are held

Assume functional dependencies **F = {P→M, PMSY→LRCE, IPMSY→G, LSY→PME, R→C}**

(a) Show how to derive two candidate keys for **Student**, or justify why you cannot do so. (2)
(b) Find a minimal cover (i.e, an irreducible set of functional dependencies) for **Student**. (2)
(c) Does **F** imply **LSY→ C**? Show working that justifies your answer. (2)
(d) Is **Student** in the third normal form (3NF)? If not, find a decomposition of **Student** into 3NF. Show working that justifies your answer. (2)

Important points to note

- The total number of marks in this assignment is 8.
- This assignment makes up 8% of your COSC444 mark.
- You must provide clearly reasoned justifications to your answers in order to get full marks.
- Please submit via Blackboard. This assignment is **due at 4pm on April 18**.

Ambili Arangath Narayanan **COSC444-Advanced Database Technologies**

Assignment 1: Relational Theory Question

Answers to Question (a)

To derive candidate keys for the Student relation, we can follow a systematic approach that involves the following steps:

- ✓ Identify **all the attributes in the relation**. In this case, the attributes are related to the Student relation.
- ✓ Once we have identified all the attributes, we can determine which combinations of attributes could potentially form a **unique identifier**, or a candidate key, for the relation.
- ✓ To check whether a candidate key is valid, we need to verify that it satisfies certain criteria, such as **uniqueness and minimality.**
- ✓ We can **repeat the above steps** until we have identified all the possible candidate keys for the Student relation.

First let's understand the following terminologies

- ❖ **A super key** is a set of one or more attributes that can uniquely identify a tuple (row) in a relation (table).
- ❖ **A minimal superkey** is a superkey that has no unnecessary attributes.
- ❖ **A candidate key** is a minimal superkey that has been selected as the primary means of identifying tuples within a relation.
- ❖ **Uniqueness** refers to the property of a field or set of fields that ensures that each value in that field or set of fields is unique and appears only once in the database.
- ❖ **Minimality**, on the other hand, refers to the property of a set of fields that ensures that no unnecessary fields are included in the set.

We can then check for **superkeys** by combining attributes on the left-hand side of each functional dependency in F and the following are the ***superkeys that cannot be a candidate key.***

I, G, P, M, S, Y, E, L, R, C: This superkey is unique, but it is not minimal because we can remove some attributes and still have a functional dependency that determines all the other attributes.

IPM, IPS, IPY, IPG, IMS, IMY, IMG, ISY, ISG, IYG, IPMS, IPMY, IPMG, ISYG, IPMSY: All of these superkeys are not minimal because we can remove some attributes and still have a functional dependency that determines all the other attributes.

ILM, ILS, ILY, ILG, IPME, IPSE, IPYE, IPGE, IMSE, IMYE, IMGE, ISYE, ISGE, IYGE, IPMSE, IPMYE, IPMGE, IPSYE, ISYGE, IPMSYE, IPMYGE, IPSYGE: All of these superkeys are not minimal because we can remove some attributes and still have a functional dependency that determines all the other attributes.

(IPSY) is the only candidate key, as it satisfies the following conditions:

UNIQNESS CONDITION:

Each combination of values in IPSY is unique and does not appear more than once in the table. This is because IPSY is a combination of attributes that are themselves unique, namely I, P, S, and Y

Every combination of values for IPSY must be unique across all the tuples in the relation.

We can confirm this by checking the values in the table:

Student id	Grade	Paper Code	Subject Name	Semester	Year	Name of the student	Lecturer's name	Room	Capacity
I	**G**	**P**	**M**	**S**	**Y**	**E**	**L**	**R**	**C**
101	A	1234567	ENG	SEM1	2023	ABC	2	LG1	30
102	B	1234567	ENG	SEM1	2023	XYZ	2	LG1	30
102	C	1234568	BIO	SEM2	2022	LMN	2	BS1	40
101	A	1234568	BIO	SEM1	2023	ABC	2	BS1	40

Thus. we can see that for every tuple in the table, the combination of values for IPSY is unique. Therefore, IPSY satisfies the uniqueness condition.

MINIMALITY CONDITION:

IPSY cannot be further reduced without losing the uniqueness property. This means that none of the individual attributes can be removed from IPSY without creating non-unique combinations of values. Therefore, IPSY is a minimal superkey and qualifies as a candidate key.

- ✓ Based on the given functional dependencies, we can see that IPSY determines all other attributes in the relation.
- ✓ IP -> L, R, C (from the functional dependency IP → LRC) IS -> E (from the functional dependency IS → E) IY -> G (from the functional dependency IY → G)
- ✓ **So we can say that IPSY -> L, R, C, E, G**
- ✓ Furthermore, IPSY is unique because no other combination of attributes can determine all the attributes in the relation.
- ✓ Therefore, **IPSY is a candidate key** for the given relation and no other combination meets the criteria to become a candidate key.
- ✓ **{I, P, S, Y}: This is the best and only candidate key because** it is a superkey (its closure includes all the attributes in the relation) and no proper subset of it is a superkey while the others are superkeys but doesn't meet the minimality and uniqueness condition .

(b) Find a minimal cover (i.e, an irreducible set of functional dependencies) for Student.

Minimal cover is a process of simplifying a given set of functional dependencies in a relational database while preserving its meaning. It involves removing any redundant or extraneous functional dependencies and breaking down any composite dependencies into simpler ones.

To find a minimal cover for the functional dependencies in the Student relation, we need to eliminate redundant dependencies and extraneous attributes.

STEP 1 : Every right hand side has a single attribute

Starting with F={P→M, PMSY→LRCE, IPMSY→G, LSY→PME, R→C},

Apply the first step to check if every right-hand side has a single attribute:

P → M is already in the desired form.
PMSY → LRCE can be split into four functional dependencies:
PMSY → L PMSY → R PMSY → C PMSY → E
IPMSY → G is already in the desired form.
LSY → PME can be split into three functional dependencies:
LSY → P LSY → M LSY → E
R → C is already in the desired form.

Thus, the minimal cover for F is: P → M PMSY → L PMSY → R PMSY → C PMSY → E IPMSY → G LSY → P LSY → M LSY → E R → C

STEP 2 : Every left hand side is irreducible

Test P→M:
Calculate {P}+ = {P, M}.
Since P→M is already a single attribute dependency, **it is irreducible.**

Test PMSY→L:
Calculate {PMSY}+ = {PMSY, L, R, C, E}.
Since PMSY→LRCE is already a single attribute dependency, **it is irreducible.**

Test IPMSY→G:
Calculate {IPMSY}+ = {IPMSY, G, P, M}.
Since IPMSY→G is already a single attribute dependency, **it is irreducible.**

Test LSY→P:
Calculate {LSY}+ = {LSY, P, M, E, C}.
Since LSY→PME and LSY→C are already single attribute dependencies**, LSY→P is not redundant.**

Test LSY→M:
Calculate {LSY}+ = {LSY, P, M, E, C}.
Since LSY→PME and LSY→C are already single attribute dependencies, **LSY→M is not redundant.**

Test LSY→E:
Calculate {LSY}+ = {LSY, P, M, E, C}.
Since LSY→PME and LSY→C are already single attribute dependencies, **LSY→E is not redundant.**

Test LSY→C:
Calculate {LSY}+ = {LSY, P, M, E, C}.
Since LSY→PME is already a single attribute dependency, **LSY→C is redundant.**

Test R→C:
Calculate {R}+ = {R, C}.
Since R→C is already a single attribute dependency, **it is irreducible.**

After Step 2, we have the following minimal cover:

F = {P→M, PMSY→L, PMSY→R, IPMSY→G, LSY→PME, R→C}

Note that LSY→C was found to be redundant and removed from the original set of functional dependencies.

Step 3: Remove redundant FDs

To remove redundant functional dependencies (FDs) from

F = {P→M, PMSY→L, PMSY→R, IPMSY→G, LSY→PME, R→C},

we can follow these steps:

Test PMSY→L:
To test whether MSY can be removed for the left-hand side (LHS),
calculate {P}+ = {PM}, {P, M}+ = {PM, L, R, C, G, E} MSY is not in {P}+,
Hence, PMSY→L is not redundant.

Test PMSY→R:
To test whether MSY can be removed for the LHS,
calculate {P}+ = {PM}, {P, M}+ = {PM, L, R, C, G, E} MSY is not in {P}+,
Hence, PMSY→R is not redundant.

Test LSY→PME:
To test whether SY can be removed for the LHS,
calculate {L}+ = {L, D, M, K}, {L, S, Y}+ = {L, D, M, K, P, E} SY is in {L, S, Y}+,
Hence, LSY→PME is redundant.

After applying the above tests, the minimal cover of F is:

F = {P→M, PMSY→L, PMSY→R, IPMSY→G, R→C}

F= {P → M
PMSY → L
PMSY → R
~~PMSY → C~~ ~~PMSY → E~~
IPMSY →G
~~LSY → P LSY → M LSY → E~~
R → C}

c) Does F imply LSY→ C? Show working that justifies your answer

To determine whether F implies LSY → C, we need to check if the functional dependency LSY → C can be inferred from the set of dependencies F using **Armstrong's axioms.**

Armstrong's axioms are a set of inference rules used in relational database theory to derive all the functional dependencies that hold in a given relation. The three axioms are:

Reflexivity: If X is a set of attributes, then X → X.
Augmentation: If X → Y, then XZ → YZ for any set of attributes Z.
Transitivity: If X → Y and Y → Z, then X → Z.

Let us first examine the given functional dependencies in F:

- P → M
- PMSY → LRCE
- IPMSY → G
- LSY → PME
- R → C

- We can use the transitive rule to combine the functional dependencies P → M and LSY → PME to obtain LSY → PE.
- We can then use the augmentation rule to obtain LSYR → PE.
- Finally, we can use the transitive rule again to combine the functional dependencies LSYR → PE and PMSY → LRCE to obtain LSYR → CE.
- Since we have been able to derive the functional dependency LSYR → CE from F, it follows that LSY → C is also implied by F.

Using transitivity of functional dependencies, we can derive:
LSY → PME → E (from LSY → PME)
PMSY → LRCE → R → C (from PMSY → LRCE and R → C)
Then, combining the two derivations using transitivity again, we get:
LSY → PME → E → R → C

- This means that the maximum capacity of the room in which lectures are held (C) is functionally dependent on the lecturer's name, semester, and year (LSY), based on the given dependencies in F.

Is Student in the third normal form (3NF)? If not, find a decomposition of Student into 3NF. Show working that justifies your answer.

- The given Student relation is not in 3NF because it has a transitive dependency. Specifically, the Grade (G) is transitively dependent on the combination of (I,P,M,S,Y), which is not a candidate key.

- This means that there is a functional dependency chain that goes through non-key attributes, violating the third normal form.

- Student (I,G,P,M,S,Y,E,L,R,C) is not in 3NF because there are transitive dependencies where non-key attributes depend on non-superkey attributes. Specifically, ENROL, LECT, ROOM, and RSIZE depend on the composite key {P, M, S, Y}, which is not a superkey.

- To bring Student into 3NF, we need to decompose it into multiple relations.

  Step 1: Create a relation based on each FD

  R1(PM) {P→M}
  R2(PMSYLRCE) {PMSY→LRCE}
  R3(IPMSY) {IPMSY→G}
  R4(LSYPME) {LSY→PME}
  R5(RC) {R→C}

  Step 2: Merge FDs with the same left hand side

  R1(PM) {P→M}
  R2(PMSY) {PMSY→L, PMSY→R, PMSY→C, PMSY→E}
  R3(IPMSY) {IPMSY→G}
  R4(LSY) {LSY→P, LSY→M, LSY→E}
  R5(RC) {R→C}

  Step 3: Merge tables with equivalent keys

  R1(PM) {P→M} with key P
  R2(PMSY) {PMSY→L, PMSY→R, PMSY→C, PMSY→E} with key PMSY
  R3(IPMSY) {IPMSY→G} with key IPMSY
  R4(LSY) {LSY→P, LSY→M, LSY→E} with key LSY
  R5(RC) {R→C} with key R

- **Another approach** is to identify subsets of attributes that are functionally dependent on each other, and split them into separate relations. In this case, we can split the original Student relation into two new relations: Student1 and Student2.

- **Student1** has attributes related to the paper, lecturer, and student enrollment. Its candidate key is (I, P, S, Y), which includes the attributes that uniquely identify a student's enrollment in a particular paper. The functional dependencies in Student1 are {P → M, LSY → M}, which mean that a paper uniquely determines its associated lecturer, and a combination of semester, year, and lecturer uniquely determines the maximum enrollment capacity.

- Student2 has attributes related to the room and its capacity. Its candidate key is (P, S, Y), which includes the attributes that identify the paper, semester, and year of a lecture. The functional dependency in Student2 is {PMSY → LRCE}, which means that a combination of paper, semester, year, and lecturer uniquely determines the room, day, and time of a lecture, as well as its maximum capacity.

- The decomposition is lossless and dependency preserving because we can obtain the original Student relation by joining Student1 and Student2 on their common attributes (I,P,S,Y).

- The resulting relations are in 3NF because they do not have any transitive dependencies, and each relation has a candidate key that is a minimal superkey.

4

Assignment 2: Database Administration Due 25th May, 2023

Assignment scenario

For this assignment you are to consider yourself as a person who has just been employed by a retail company in order to supply IT services to employees (which does not have its own IT department - and does not wish to venture into that area). You have been employed in the position of software developer/systems admin to write scripts and applications to configure your company's servers which provide mail, web and database services - the company is hoping that you will be able to do pretty much any computer admin task they ask you to do!

You have come into a pre-exiting situation and find that the server for the task in hand has Linux installed, and the database being used is MySQL. It turns out that the person from whom you have taken over gave your employers the impression that his skills were far greater than they turned out to be in practice - and, sadly, when he had to leave suddenly, he left very little documentation about what work he had done already to configure the database you now find yourself having to manage. You must configure it appropriately to provide high performance, security and reliability. You should work your way through the tasks given as follows.

Preparation

You need to create an instance of the database based on a backup. You can download the backup file named `retail_databackup.sql` from Blackboard. It is a full backup of the database called "classicmodels" that consists of the following tables:

- **Customers**: stores customer's data.
- **Products**: stores a list of scale model cars.
- **ProductLines**: stores a list of product line categories.
- **Orders**: stores sales orders placed by customers.
- **OrderDetails**: stores sales order line items for each sales order.
- **Payments**: stores payments made by customers based on their accounts.
- **Employees**: stores all employee information as well as the organization structure such as who reports to whom.
- **Offices**: stores sales office data.

Task 1: Manage user accounts and privileges

1. For all users, configure MySQL to expire the password every 90 days, and every password change needs to provide the current password for verification. Explain how you make the configurations. (1 mark)

2. Suppose the company recruits a new finance administrator Ms Alice Jones.
   - Create a MySQL account for Alice. Give the SQL statement you used to create the account for Alice, and explain how you will give her the username and password securely. (1 mark)
   - What privileges will you grant to Alice? Give the SQL statements for granting privileges and justify your answer. (2 marks)

- Configure Alice's account so that any more than 4 incorrect login attempts will result in a 24-hour suspension of the account. Explain how you make this configuration.

(0.5 mark)

- Alice got her account locked after 4 incorrect login attempts. She came to ask you to unlock his account immediately. Explain how you will unlock her account.

(0.5 mark)

Task 2: backup

1. Design a backup plan to protect the data in the "classicmodels" database in the event of data corruption, system crash, or physical hardware failure. Your backup plan should guarantee that the database can be recovered to the state just before the error happened. Your plan should consider the cost, reliability, and convenience.

(5 marks)

2. Implement your backup plan. Explain your implementation step by step. If certain methods cannot be implemented due to constraints such as limited resources, describe how they can be implemented once these constraints have been resolved. If a method can be implemented, include the commands you used, the scripts you developed and even screenshots if necessary.

(4 marks)

Task 3: recovery

Make sure you already have a full backup of the "classicmodels" database. Add a new customer in the "customers" table, and insert three records for this new customer in the "orders" table. Suppose the database crashed after these changes (you can mimic the crash by dropping the database). Demonstrate how you recover the database to the state just before the database was crashed.

(3 marks)

Tips on working this assignment

1. To work on Task 1, you might need to refer to access control and account management in the MySQL 8.0 reference manual.
2. The lab on database security in Week 9 and the lab on database backup and recovery in Week 11 will prepare you to work on Task 2 and Task 3.
3. Task 2 needs more critical thinking and research. Your plan should consider all possible failures.

Important points to note

- The total number of marks for this assignment is 17, which makes up 17% of your COSC444 final mark.
- You must provide clearly reasoned justifications or explanation in order to get full marks.
- Please submit via Blackboard.

ASSN 2 – COSC444 AMBILI AN

AFTER EXECUTING THE NECESSARY SCRIPT FILE with MYSQL BENCH, we get the following ……

#To verify if the data has been successfully uploaded, switch to the classicmodels database by executing

use classicmodels;

#Then, display the list of tables using the command

show tables;

We can see a list of tables, it means that your database has been imported into the MySQL server.

```
C:\Users\l530>docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
328ba678bf27: Pull complete
f3f5ff088d73: Pull complete
dd7054d6d8c7: Pull complete
70b5d4e8750e: Pull complete
cdc4a7b43bdd: Pull complete
a8608f8959e0: Pull complete
5823e721608f: Pull complete
a564ada930a9: Pull complete
539565d00e89: Pull complete
a11a06843fd5: Pull complete
92f6d4aa041d: Pull complete
Digest: sha256:a43f6e7e7f3a5e5b90f857fbed4e3103ece771b19f0f75880f767cf66bbb6577
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest

C:\Users\l530>docker run -d --name mysql-container -v D:/Freelancing/retail_databackup.sql:/docker-entrypoint-initdb.d/retail_databackup.sql -e MYSQL_ROOT_PASSWORD=12345 mysql
49c797a9ac3b45377f73bc6631f0f8ca9ff4b795db72abab8179b8573398f305

C:\Users\l530>docker exec -it mysql-container mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.33 MySQL Community Server - GPL
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

```
Database changed
mysql> show tables;
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| customers               |
| employees               |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
+-------------------------+
8 rows in set (0.00 sec)
```

```
mysql> use classicmodels;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Task 1: Manage user accounts and privileges

1. For all users, configure MySQL to expire the password every 90 days, and every password change needs to provide the current password for verification. Explain how you make the configurations.

ANSWER:

MySQL to expire password every 90days

- Task 1 involves managing user accounts and privileges in MySQL. Firstly, to set the password expiry to 90 days.
- We can use the Commands:-
- **SET GLOBAL default_password_lifetime = 90"**
- **SET GLOBAL password_require_current=ON;**
- along with the commands
- **ALTER USER 'root'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY**
  ALTER USER 'root'@'localhost' PASSWORD REQUIRE CURRENT
- To confirm whether the password policy has been applied in our MySQL server, we can execute the command
- **SHOW VARIABLES LIKE 'validate_password%';**
- This will display a table containing information related to the password validation policy

```
mysql> ALTER USER 'root'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
Query OK, 0 rows affected (0.07 sec)
```

```
mysql> SET GLOBAL default_password_lifetime = 90;
Query OK, 0 rows affected (0.00 sec)

mysql>  select plugin_name, plugin_status from information_schema.plugins where plugin_name like 'valid
ate%';
+-------------------+---------------+
| plugin_name       | plugin_status |
+-------------------+---------------+
| validate_password | ACTIVE        |
+-------------------+---------------+
1 row in set (0.00 sec)

mysql> SHOW VARIABLES LIKE 'validate_password%'
    ->
    -> ;
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password_check_user_name    | OFF    |
| validate_password_dictionary_file    |        |
| validate_password_length             | 8      |
| validate_password_mixed_case_count   | 1      |
| validate_password_number_count       | 1      |
| validate_password_policy             | MEDIUM |
| validate_password_special_char_count | 1      |
+--------------------------------------+--------+
7 rows in set (0.00 sec)

mysql>
```

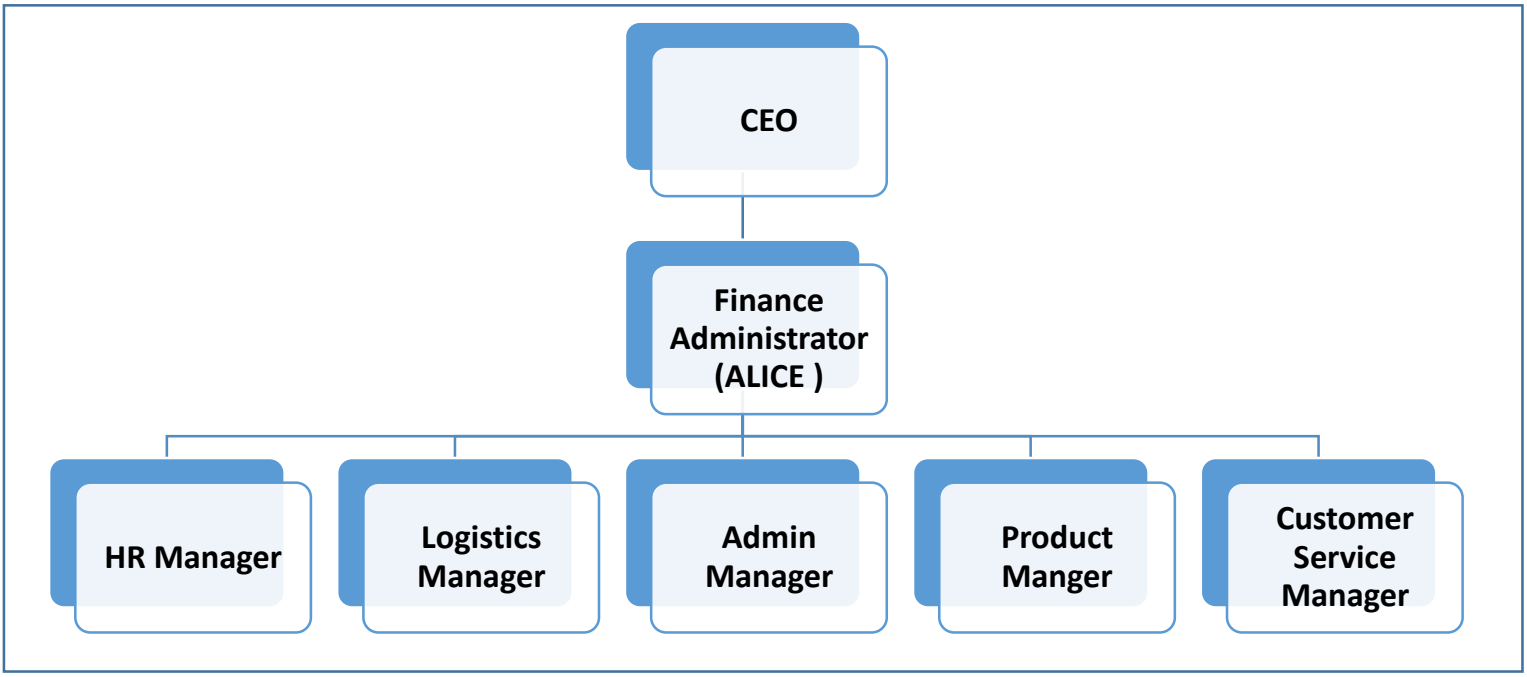
2.Suppose the company recruits a new finance administrator Ms Alice Jones • Create a MySQL account for Alice. Give the SQL statement you used to create the account for Alice, and explain how you will give her the username and password securely.

ASSUMPTIONS:

1) Let's assume that the under mentioned organisation chart for the Question# 2.

2) Alice is in the Top Management reporting to the CEO and she has HR, Logistics, Product, Administration and Customer Service manager reporting to her.

3) In nut shell, she is the second in command for the organisation.



Role of a Financial Administrator

- A financial administrator oversees the financial activities of a business and is responsible for the accounts of the organisation.
- The financial administrator is one who is responsible for any financially related task of the organisation.
- Gathering, analyzing, and interpreting relevant financial data. Evaluating and optimizing financial controls and procedures.
- Updating daily transaction records and assisting with payroll administration. Managing accounts receivable and payable, as well as expenses.

User name & Password

- To create a new user account for alice, we can use the SQL statement
- **CREATE USER 'alice'@'localhost' IDENTIFIED BY 'password';**

- *Here are some tips on how to give Alice the username and password securely:*

- Do not send the username and password in an email that is not encrypted.
- Do not write the username and password on a piece of paper that is easily accessible.
- Do not tell Alice the username and password over the phone.
- If you are giving Alice the username and password in person, make sure that you are in a private place where no one else can overhear you and the undermnetioned style cab be adopted.
- **Encrypted email:** Send an encrypted email containing the username and password to Alice if a face-to-face meeting is not feasible. Emails can be encrypted using a variety of encryption techniques, including S/MIME (Secure/Multipurpose Internet Mail Extensions) and PGP (Pretty Good Privacy).
- **Password manager:** Utilising a password manager to safely store the login and password is an additional choice. The password manager can be set up to provide Alice access to the credentials, and she can do so by entering a master password or another kind of authentication. LastPass, 1Password, and KeePass are a few of the well-known password managers.
- Whichever approach is taken, it's crucial to emphasise to Alice the significance of keeping the login information secret and not disclosing it to anyone else.

```
mysql> CREATE USER 'alice'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.02 sec)
```

What privileges will you grant to Alice? Give the SQL statements for granting privileges and justify your answer

ANSWER:

Priviiges for Alice

- As a finance administrator, Alice may need access to the financial data stored in the database. This could include the Customers, Orders, OrderDetails, Payments, and Employees tables. The following are commands grant her the necessary privileges using SQL statements.

- **GRANT SELECT, INSERT, UPDATE, DELETE ON classicmodels.Customers TO 'alice'@'localhost';**
- **GRANT SELECT, INSERT, UPDATE, DELETE ON classicmodels.Payments TO 'alice'@'localhost';**
- **GRANT SELECT, INSERT, UPDATE, DELETE ON classicmodels.Products TO 'alice'@'localhost';**
- **GRANT SELECT ON classicmodels.Employees TO 'alice'@'localhost';**
- **GRANT SELECT ON classicmodels.Productlines TO 'alice'@'localhost';**
- **GRANT SELECT ON classicmodels.offices TO 'alice'@'localhost';**
- **GRANT SELECT ON classicmodels.orderdetails TO 'alice'@'localhost';**
- **GRANT SELECT ON classicmodels.orders TO 'alice'@'localhost';**

- These statements grant Alice the ability to view **(SELECT), add (INSERT), modify (UPDATE), and delete (DELETE)** data in the Customers, Orders, OrderDetails, and Payments tables.
- She is also granted the ability to view data in the Employees table.
- These privileges should allow her to perform her duties as a finance administrator while also ensuring that she only has access to the data she needs.
- **ALTER and DROP Privileges** are not given and it will be with the CEO of the company.

Justification

- As per the organisational Chart, Alice has been appointed as Finance Administartor is consdiered as the second most powerful designation after CEO.

- **CUSTOMER TABLE :** Alice needs to access this table as she has to decide on the CREDIT LIMIT to be fixed for each customer.
- **EMPLOYEE TABLE :**Alice needs limited access to this table as HR MANAGER will handle the employee details. She can ONLY View (SELECT) all details of the employee table for salary and reimbursement purpose.
- **OFFICE TABLE :** Alice needs limited access to this table . She can ONLY View (SELECT) all details of the office table ad ADMIN MANAGER will handle this table.
- **ORDER DETAILS TABLE :** Alice needs limited access to this table as CUSTOMER SERVICE MANAGER will handle the day to day activities pertaining to ORDERS. She can ONLY View (SELECT) all details of the Orderdetails table. This table in general gets information from the Order and Product Table and Hence, ALice needs only Select Priviliage.
- **ORDERS TABLE :** Alice needs limited access to this table as LOGISTICS DEPARTMENT handles the day to day activities pertaining to ORDERS .
- **PRODUCT LINES TABLE :** Alice needs limited access to this table as MARKETING DEPARTMENT handles the day to day activities pertaining to Product lines. She can ONLY View (SELECT) all details of the Productlines table.
- **PRODUCTS TABLE :** Alice needs access to this table as she has to decide on the BUYPRICE and MSRP for each product lines. ALice should be able to update BUYPRICE & MSRP as she is responsibile and decision makers pertaining to the profit margin of each products.

Configure Alice's account so that any more than 4 incorrect login attempts will result in a 24-hour suspension of the account. Explain how you make this configuration.

Configuration

- We can enable temporary account locking after too many consecutive incorrect-password login failures by setting the **FAILED_LOGIN_ATTEMPTS** and **PASSWORD_LOCK_TIME** options for an account to be nonzero1.
- **FAILED_LOGIN_ATTEMPTS** is an option for MySQL accounts that specifies the maximum number of consecutive failed login attempts that are permitted before the account is temporarily locked.
- **PASSWORD_LOCK_TIME** is an option for MySQL accounts that specifies the number of days for which an account will be locked after reaching the maximum number of failed login attempts specified by the FAILED_LOGIN_ATTEMPTS option.
- To configure Alice's account so that any more than 4 incorrect login attempts will result in a 24-hour suspension of the account, we can use the following command:
- **ALTER USER 'alice'@'localhost'**
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 1;
- This will set the maximum number of failed login attempts to 4 and the lock time to 1 day for Alice's account1.

```
mysql> ALTER USER 'alice'@'localhost' FAILED_LOGIN_ATTEMPTS 4 PASSWORD_LOCK_TIME 1;
Query OK, 0 rows affected (0.01 sec)
```

Alice got her account locked after 4 incorrect login attempts. She came to ask you to unlock his account immediately. Explain how you will unlock her account

ANSWER:

Unlock Alice Account

- To immediately unlock Alice's account, we can use the ALTER USER statement with the ACCOUNT UNLOCK option. The command that will unlock Alice's account
- **ALTER USER 'alice'@'localhost' ACCOUNT UNLOCK;**
- This command will immediately unlock Alice's account and allow her to log in again.

```
mysql> ALTER USER 'alice'@'localhost' FAILED_LOGIN_ATTEMPTS 4 PASSWORD_LOCK_TIME 1;
Query OK, 0 rows affected (0.01 sec)
```

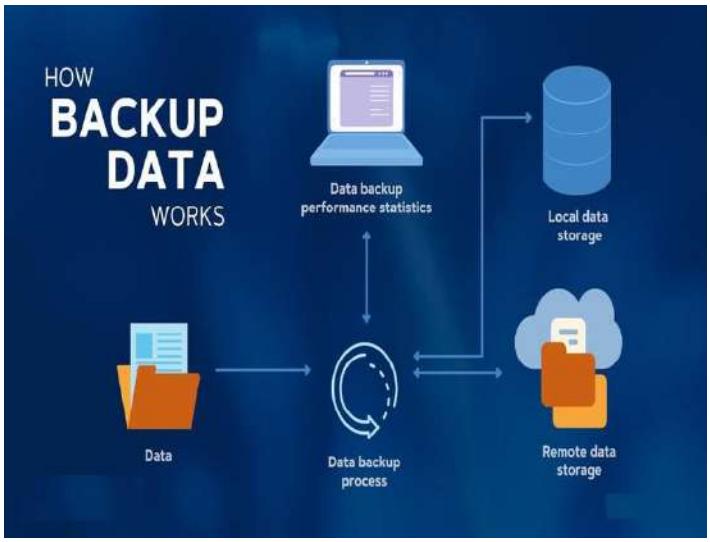
Task 2: backup

1. Design a backup plan to protect the data in the "classicmodels" database in the event of data corruption, system crash, or physical hardware failure. Your backup plan should guarantee that the database can be recovered to the state just before the error happened. Your plan should consider the cost, reliability, and convenience

ANSWER :

What Is Data Backup

- **Data backup** is a common term used in the domain of information technology. It means making a copy of the existing data and storing it somewhere else. The primary purpose of backing up data is to use it if the original data is inaccessible, lost, deleted, corrupted, or stolen.
- We can also store it in an external data storage device like an external HDD, SSD, and USB. Else, data backup on **a cloud storage platform** is another viable option.
- The objective of the data backup is not to stop data loss but to restore the backup copy if it happens. Hence, it is the most critical step during any large-scale edit to a database, computer, or website.



Importance of Data Backup

- **Data Loss Prevention** - One of the primary reasons behind data loss is accidental or intentional deletion, and a backup copy can save your back in such situations. Any employee may accidentally delete any crucial data without knowing its importance or requirement in the future.
- **Protection Against Physical Disasters** - Contrary to popular belief, you can lose your business-critical data due to different physical disasters such as fire, floods, earthquakes, or tornados. Such disasters can completely wipe out the data and make data recovery practically impossible.
- **Answer to Hacking Attacks** - Even with the most advanced protection against hacking, your business can become a victim of hacking and ransomware attacks. These prevalent outbreaks can cause heavy financial damage to your business if you do not have the data backed up somewhere else.
- **Useful for Audits and Archives** - Business companies need to go through audits, and not having necessary documents and information at the end of the financial year will not look good.
- **Quickly Resume Operation** - In any event of data loss, the ability to recover quickly and resume operations is crucial to an organization's success.

It is a must to have a backup plan to protect the data in the "classicmodels" database in the event of data corruption, system crash, or physical hardware failure. The backup plan should guarantee that the database can be recovered to the state just before the error happened. The plan is based on the factors like cost, reliability, and convenience.

Backup frequency: The backup frequency is determined by the amount of data that is being stored and the level of risk that is acceptable. For example, if the data is critical and cannot be lost, then the backup frequency should be daily or even hourly. If the data is less critical, then the backup frequency can be weekly or monthly.

Backup method: There are two main methods for backing up data: full backups and incremental backups. A full backup copies all of the data in the database. An incremental backup copies only the data that has changed since the last backup. Full backups are more time-consuming and expensive than incremental backups, but they provide a complete copy of the data. Incremental backups are less time-consuming and expensive than full backups, but they do not provide a complete copy of the data.

Backup location: The backup location should be secure and accessible. For example, the backup can be stored on a local hard drive, a network drive, or an external hard drive. The backup can also be stored in the cloud.

Backup restoration: The backup restoration process should be tested regularly to ensure that it works properly. The restoration process should be documented so that it can be followed in the event of a disaster..

Back up plan

- To protect the "classicmodels" database in the event of data corruption, system crash or physical hardware failure, we can implement the following backup plan:

- 1. **Backup Frequency:**
  - Perform a daily backup to ensure recent data is protected. Additionally, consider performing incremental backups throughout the day tominimize data loss in case of a failure.

- 2. **Backup Method:**
- Use a combination of full backups and incremental backups for efficient and reliable data protection.
- **Full Backup:** Take a complete backup of the "classicmodels" database periodically (e.g., once a week).
- mysqldump -uusername -p > database_backupfile.sql (Back up for a Single File)
- mysqldump -uusername -p databasename tablename > tablebackup.sql (Segmented Files)

- **Incremental Backup:** Take incremental backups that capture only the changes since the last full backup or incremental backup.
- mysqlbinlog [binarylogfile] | mysql . This command is commonly used for database replication, where you have a master-slave configuration. The binary log events from the master server are read using mysqlbinlog and then fed into the slave server using mysql, ensuring that the slave server stays synchronized with the changes happening on the master server.
- mysqldump -uusername -p –flush-logs --all-databases > database_backupfile.sql
- The FLUSH BINARY LOGS statement saves the current binary log and creates a new binary log file. This is useful when doing a full backup, because it ensures that the backup includes all recent changes to the database. If you need to recover the database after a disaster, you can first restore the full backup and then use the binary logs to restore any changes that occurred after the backup.

- **3. Backup Storage Location:**
- Store the backups in a separate directory or mount a volume specifically designated for backup storage. Ensure that the backup storage location is secure and has sufficient space to accommodate regular backups.

- **4. Backup Retention:**
- Define a backup retention policy to determine how long backups should be retained. Keep multiple backup copies to provide options for data restoration from different points in time. For example, retain daily backups for one week, weekly backups for one month, and monthly backups for six months. Within 2 weeks: All daily backups will be retained.
- Older than 2 weeks: 1 backup per week will be retained.
- Older than 1 year: 1 backup every fortnight (every two weeks) will be retained.
- Older than 2 years: 1 monthly backup will be retained.
- Older than 5 years: 1 quarterly backup will be retained.

Back up plan

- **5. Automation and Scheduling:**
- Implement an automated backup script or tool to ensure consistent and reliable backups. Schedule the backup script to run at a suitable time when the system load is low, to minimize any impact on performance. Use a scheduling tool like cron to schedule regular backups according to the defined backup frequency. The time will be Midnight 12 as the servers will not be busy.

- **6. Data Encryption:**
- Implement encryption for backups to enhance security and protect sensitive data. Consider using tools or methods like Transparent Data Encryption (TDE) or backup encryption features provided by MySQL.

- **7. Testing and Monitoring:**
- Regularly test the backup process to ensure the backups are created correctly and can be restored successfully. Monitor the backup process to detect any failures or errors and take appropriate actions promptly.

- **8. Offsite Backup:**
- Consider creating an additional offsite backup to protect against physical hardware failure or disasters at the primary location. Replicate backups to an offsite location, such as cloud storage or a different physical location, for an added layer of protection

TASK 3. Implement your backup plan. Explain your implementation step by step. If certain methods cannot be implemented due to constraints such as limited resources, describe …….

ANSWER : (Part 1 - Implementation steps)

1. Make sure we already have a full backup of the "classicmodels" database.

2. To simulate a database crash:

Stop the MySQL container:

```
C:\Users\l530>docker stop mysql-container
```

Remove the MySQL container:

```
C:\Users\l530>docker rm mysql-container
```

3. Restore the database from the backup:

Create a new MySQL container and mount the backup file or directory:

```
PS C:\Users\l530> docker run -d --name mysql-container -v /path/to/backup:/docker-entrypoint-initdb.d -e MYSQL_ROOT_PASSWORD=12345 mysql
```

Replace "/path/to/backup" with the actual path to the backup file or directory.

4. Verify the database recovery:

Access the MySQL container's shell:

```
docker exec -it mysql-container mysql -u root -p
```

Enter the MySQL root password when prompted.

• Inside the MySQL container's shell, check the database and verify the recovered data:

```
USE classicmodels;
SELECT * FROM customers;
SELECT * FROM orders;
```

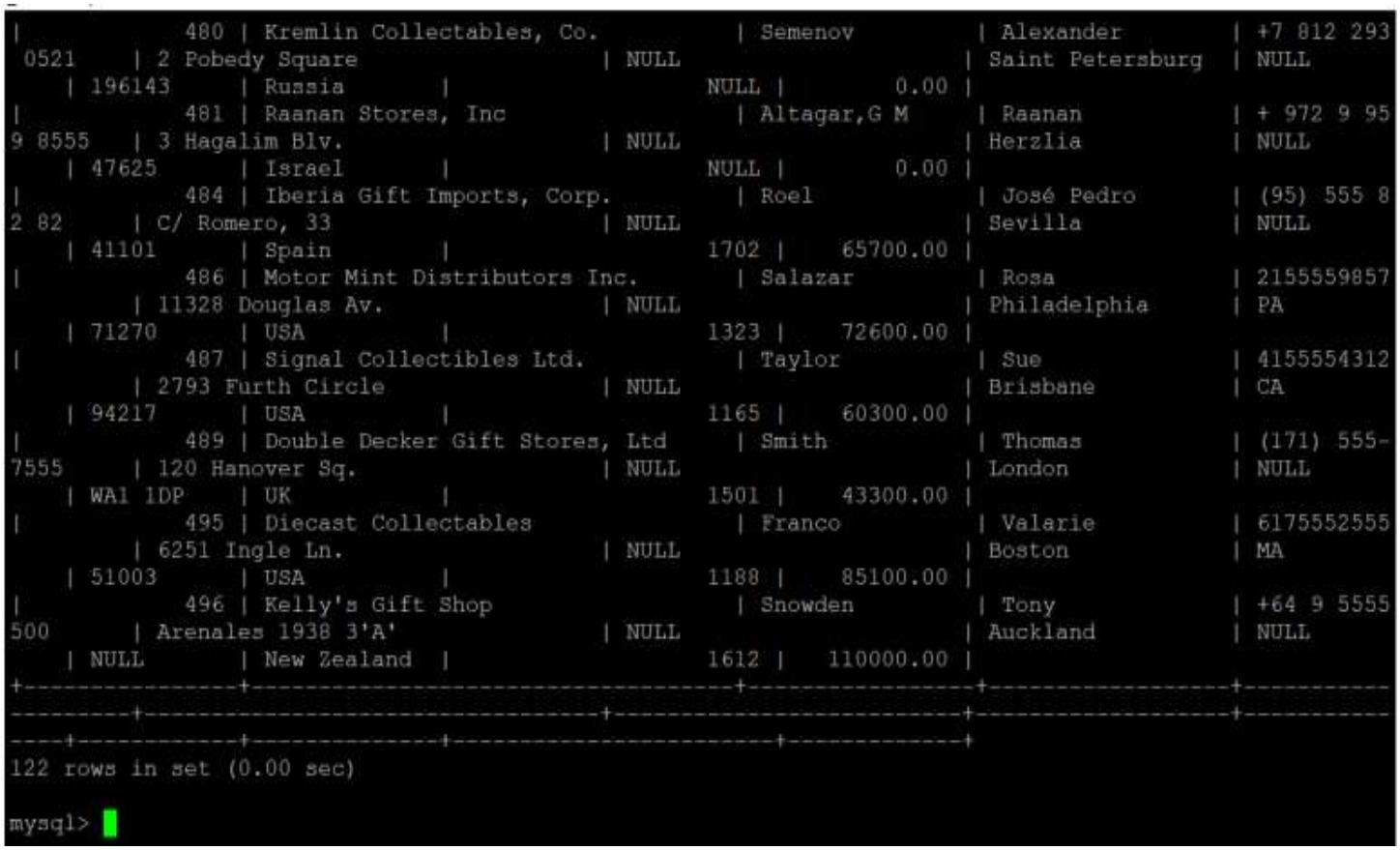
(Part 2 – Verification steps)

STEP 1

- To demonstrate the process of recovering a crashed database, we first checked the number of rows in the"customers" and "orders" tables of the "classicmodels" database.
- Then, we added a new customer and three orders records for that customer.
- After that, we dropped the database to simulate a crash.
- To recover the database to its state just before the crash, we created an empty database with the same name and restored the backup data from the "backupfile.sql" file.
- We verified that the data was successfully restored by checking the number of rows in the "customers"and "orders" tables, which matched the number of rows we had before the crash.

STEP 2

- To begin, we checked the number of rows in the "customers" and "orders" tables of the "classicmodels" database.
- We then proceeded to add a new customer and three orders records. We achieved this by first logging in to the SQL server and executing the following commands:
  - **use classicmodels;**
  - **show tables;**
  - **select * from customers;**
- These commands allowed us to verify that there were initially 122 rows in the "customers" table and 326 rows in the "orders" table.



STEP 3

- Now,we inserted one customer record with the customerNumber 98001, increasing the total number of customer records to 123.
- To add the new customer, we executed the following SQL statement:

- **INSERT INTO customers (customerNumber, customerName, contactLastName,contactFirstName, phone, addressLine1, addressLine2, city, state, postalCode, country,salesRepEmployeeNumber, creditLimit) VALUES (98001, 'Atelier grapheique', 'Schmiett', 'Caerine', '40.32.20555', '054, rue Rhyhoyale', NULL, 'Naujntes', NULL, '445000', 'Frkance', NULL, '210300.00');**

- Next, we added three order records for this customer using the INSERT INTO command.
- To add the new order records, we executed the following SQL statement:
- **INSERT INTO `orders`(`orderNumber`,`orderDate`,`requiredDate`, `shippedDate`, `status`,`comments`,`custom erNumber`) values (201210,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,98001);**

STEP 3

- **INSERT INTO `orders`(`orderNumber`,`orderDate`,`requiredDate`,`shippedDate`,`status`,`comments`,`customerNumber`) values (201221,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,98001);**
- **INSERT INTO `orders`(`orderNumber`,`orderDate`,`requiredDate`,`shippedDate`,`status`,`comments`,` customerNumber`) values (201222,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,98001);**
- Now We confirmed that the orders table now has 329 rows by using the **SELECT * FROM orders** command.

```
|           484 | Iberia Gift Imports, Corp.         | Roel            | José Pedro       | (95) 555 8
2 82     | C/ Romero, 33                      | NULL                                  | Sevilla          | NULL
    | 41101       | Spain          |                    1702 |    65700.00 |
|           486 | Motor Mint Distributors Inc.       | Salazar         | Rosa             | 2155559857
        | 11328 Douglas Av.                  | NULL                                  | Philadelphia     | PA
    | 71270       | USA            |                    1323 |    72600.00 |
|           487 | Signal Collectibles Ltd.           | Taylor          | Sue              | 4155554312
        | 2793 Furth Circle                  | NULL                                  | Brisbane         | CA
    | 94217       | USA            |                    1165 |    60300.00 |
|           489 | Double Decker Gift Stores, Ltd     | Smith           | Thomas           | (171) 555-
7555     | 120 Hanover Sq.                    | NULL                                  | London           | NULL
    | WA1 1DP     | UK             |                    1501 |    43300.00 |
|           495 | Diecast Collectables               | Franco          | Valarie          | 6175552555
        | 6251 Ingle Ln.                     | NULL                                  | Boston           | MA
    | 51003       | USA            |                    1188 |    85100.00 |
|           496 | Kelly's Gift Shop                  | Snowden         | Tony             | +64 9 5555
500      | Arenales 1938 3'A'                 | NULL                                  | Auckland         | NULL
    | NULL        | New Zealand    |                    1612 |   110000.00 |
+-----------------+------------------------------------+-----------------+------------------+-----------
---------+---------------------------------+---------------------------+-------------------+-----------
----+-----------+-------------+-----------------------+-------------+
122 rows in set (0.00 sec)

mysql> INSERT INTO customers (customerNumber, customerName, contactLastName, contactFirstName, phone, a
ddressLine1, addressLine2, city, state, postalCode, country, salesRepEmployeeNumber, creditLimit)
    -> VALUES (98001, 'Atelier grapheique', 'Schmiett', 'Caerine', '40.32.20555', '054, rue Rhyhoyale',
 NULL, 'Naujntes', NULL, '445000', 'Frkance', NULL, '210300.00');
Query OK, 1 row affected (0.01 sec)

mysql>
```

```
|           495 | Diecast Collectables                | Franco          | Valarie           | 6175552555
        | 6251 Ingle Ln.                    | NULL                        | Boston            | MA
    | 51003       | USA           |                      1188 |    85100.00 |
|           496 | Kelly's Gift Shop                   | Snowden         | Tony              | +64 9 5555
500       | Arenales 1938 3'A'                | NULL                        | Auckland          | NULL
    | NULL        | New Zealand   |                      1612 |   110000.00 |
|         98001 | Atelier grapheique                  | Schmiett        | Caerine           | 40.32.2055
5         | 054, rue Rhyhoyale                | NULL                        | Naujntes          | NULL
    | 445000      | Frkance       |                      NULL |   210300.00 |
+-----------------+-------------------------------------+-----------------+-------------------+-----------
---------+-------------------------------------+-----------------------------+-------------------+-----------
----+------------+--------------+-------------------------+-------------+
123 rows in set (0.00 sec)

mysql> INSERT  INTO `orders`(`orderNumber`,`orderDate`,`requiredDate`,`shippedDate`,`status`,`comments`
,`customerNumber`) values
    -> (201210,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,98001);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT  INTO `orders`(`orderNumber`,`orderDate`,`requiredDate`,`shippedDate`,`status`,`comments`
,`customerNumber`) values
    -> (201221,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,98001);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT  INTO `orders`(`orderNumber`,`orderDate`,`requiredDate`,`shippedDate`,`status`,`comments`
,`customerNumber`) values
    -> (201222,'2003-01-06','2003-01-13','2003-01-10','Shipped',NULL,98001);
Query OK, 1 row affected (0.00 sec)

mysql>
```

```
|       10421 | 2005-05-29 | 2005-06-06   | NULL        | In Process | Custom shipping instructions wer
e sent to warehouse
                                                |            124 |
|       10422 | 2005-05-30 | 2005-06-11   | NULL        | In Process | NULL

                                                |            157 |
|       10423 | 2005-05-30 | 2005-06-05   | NULL        | In Process | NULL

                                                |            314 |
|       10424 | 2005-05-31 | 2005-06-08   | NULL        | In Process | NULL

                                                |            141 |
|       10425 | 2005-05-31 | 2005-06-07   | NULL        | In Process | NULL

                                                |            119 |
|      201210 | 2003-01-06 | 2003-01-13   | 2003-01-10  | Shipped    | NULL

                                                |          98001 |
|      201221 | 2003-01-06 | 2003-01-13   | 2003-01-10  | Shipped    | NULL

                                                |          98001 |
|      201222 | 2003-01-06 | 2003-01-13   | 2003-01-10  | Shipped    | NULL

                                                |          98001 |
+-------------+------------+--------------+-------------+------------+-------------------------------
-------------------------------------------------------------------------------------------------------
-----------------------------------------------------+----------------+
329 rows in set (0.00 sec)

mysql>
```

STEP 4

- To delete the classicmodels database, we used the **DROP DATABASE** command, and then checked that the database was deleted by using the **SHOW DATABASES** command.
- After that, we created an empty classicmodels database again using the **CREATE DATABASE** command and verified that there were no tables present in it by using the **SHOW TABLES** command

```
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> create database classicmodels;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use classicmodels;
Database changed
mysql> show tables;
Empty set (0.00 sec)

mysql>
```

STEP 5

- To restore the classicmodels database from a backup file named backupfile.sql, we exited MySQL and used the following command:
- **mysql -u root -p classicmodels < backupfile.sql**
- This restored the database to its previous state

```
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> create database classicmodels;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use classicmodels;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

```
+--------------------+
| information_schema |
| classicmodels      |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use classicmodels;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| customers               |
| employees               |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
+-------------------------+
8 rows in set (0.00 sec)

mysql>
```

STEP 6

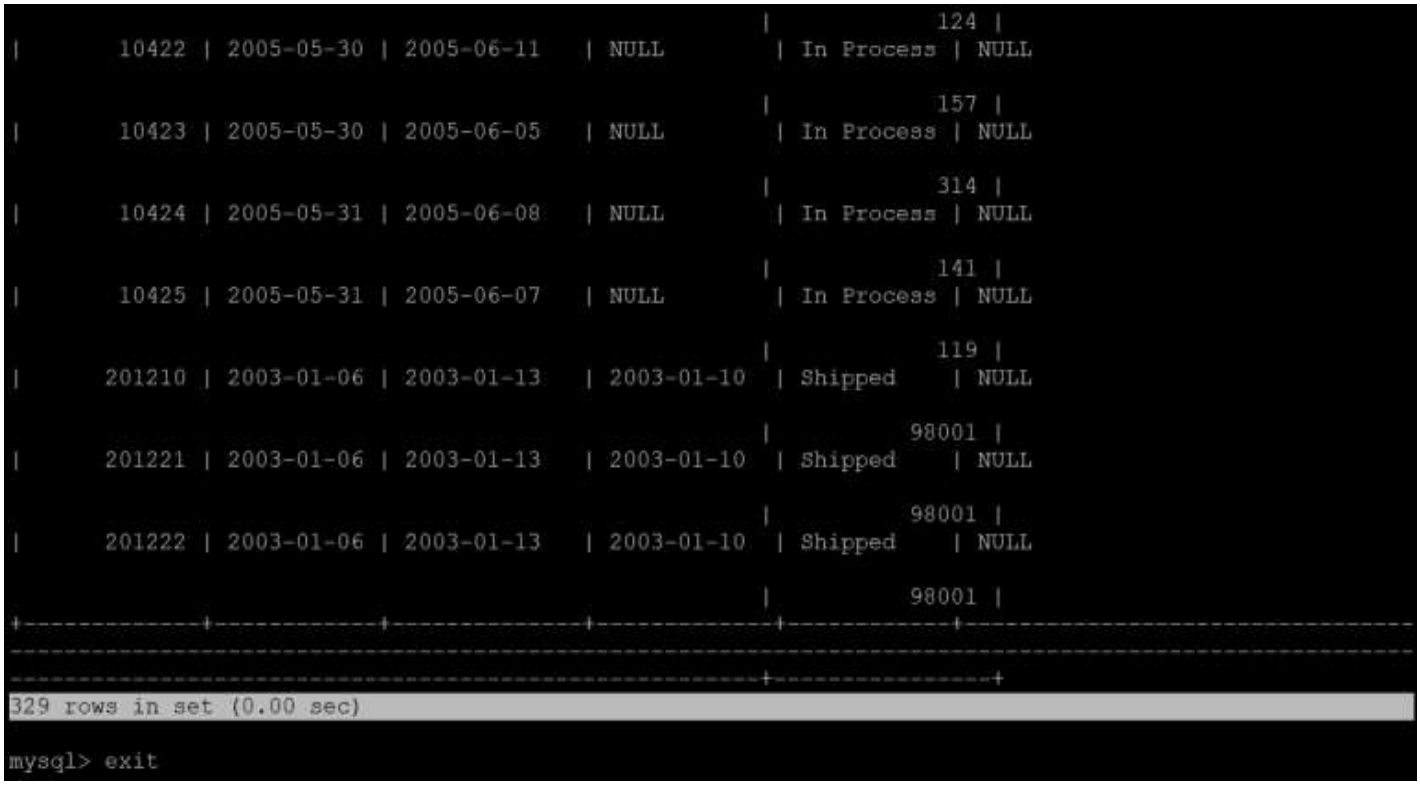
- To ensure that the data has been correctly restored in the **backupfile.sql** , we can verify the number of records in the 'customers' and 'orders' tables .
- After Adding one new customer to customers table we can see it has now **123 rows** using the command **select * from customers;**

```
|           481 | Raanan Stores, Inc                | Altagar,G M     | Raanan          | + 972 9 95
9 8555    | 3 Hagalim Blv.                | NULL                                | Herzlia         | NULL
    | 47625      | Israel         |                        NULL |        0.00 |
|           484 | Iberia Gift Imports, Corp.        | Roel            | José Pedro      | (95) 555 8
2 82      | C/ Romero, 33                 | NULL                                | Sevilla         | NULL
    | 41101      | Spain          |                        1702 |    65700.00 |
|           486 | Motor Mint Distributors Inc.      | Salazar         | Rosa            | 2155559857
        | 11328 Douglas Av.             | NULL                                | Philadelphia    | PA
    | 71270      | USA            |                        1323 |    72600.00 |
|           487 | Signal Collectibles Ltd.          | Taylor          | Sue             | 4155554312
        | 2793 Furth Circle             | NULL                                | Brisbane        | CA
    | 94217      | USA            |                        1165 |    60300.00 |
|           489 | Double Decker Gift Stores, Ltd    | Smith           | Thomas          | (171) 555-
7555      | 120 Hanover Sq.               | NULL                                | London          | NULL
    | WA1 1DP    | UK             |                        1501 |    43300.00 |
|           495 | Diecast Collectables              | Franco          | Valarie         | 6175552555
        | 6251 Ingle Ln.                | NULL                                | Boston          | MA
    | 51003      | USA            |                        1188 |    85100.00 |
|           496 | Kelly's Gift Shop                 | Snowden         | Tony            | +64 9 5555
500       | Arenales 1938 3'A'            | NULL                                | Auckland        | NULL
    | NULL       | New Zealand    |                        1612 |   110000.00 |
|         98001 | Atelier grapheique                | Schmiett        | Caerine         | 40.32.2055
5         | 054, rue Rhyhoyale            | NULL                                | Naujntes        | NULL
    | 445000     | Frkance        |                        NULL |   210300.00 |
+---------------+-----------------------------------+-----------------+-----------------+-----------
---------+-----------------------------------+-----------------------------+--------------------+-----------
----+-----------+-------------+-----------------------+-------------+
123 rows in set (0.00 sec)

mysql> select * from orders;
```

STEP 6

- Similarly, we can see that the orders table now has **329 rows** by using the command **select * from orders;**
- This confirms that our data has been restored correctly from the backup file.


10422	2005-05-30	2005-06-11	NULL	In Process	NULL	124
10423	2005-05-30	2005-06-05	NULL	In Process	NULL	157
10424	2005-05-31	2005-06-08	NULL	In Process	NULL	314
10425	2005-05-31	2005-06-07	NULL	In Process	NULL	141
201210	2003-01-06	2003-01-13	2003-01-10	Shipped	NULL	119
201221	2003-01-06	2003-01-13	2003-01-10	Shipped	NULL	98001
201222	2003-01-06	2003-01-13	2003-01-10	Shipped	NULL	98001
329 rows in set (0.00 sec)
mysql> exit

##

5

COSC444 - Research Project

Aim

The aim of this project is to explore advanced database technologies, investigate state-of-the-art researches on database, and develop skills in paper reading, literature review, critical thinking, problem solving, academic writing, and presentation.

You can work on this project independently or pair up with a classmate. A special channel in Teams has been created for you to pair up if you wish to work on the project as a group.

You must choose a topic in the field of databases such as Distributed Databases, NoSQL databases, Timeseries Databases, Graph Databases, Temporal databases, Geospatial databases, or other topics in the field of databases.

Tasks

To work on this project, you need to perform the following tasks:

- **Find a research paper** to read in the field of databases. This article should be a high quality research paper (typically with a length of no less than 8 pages). Paper selection criteria are given at the end of this assignment specification.
- **Write a report and give a presentation.** It is expected that your report is around 5 pages (double column excluding references) or 2500 words. Your report should be written using the IEEE conference template. Each presentation is within 10 minutes.
- **In the report and your presentation, you are expected to include a basic component that summarises the work done in the paper you read.** This component should contain information that answers the following questions:

    - What problem was investigated in that paper?
    - What is the key idea of the proposed algorithm/system?
    - How does the proposed algorithm/system work?
    - How was the developed algorithm/system evaluated, and what are the evaluation results?

- **In the report and presentation, you also need to include an advanced component that gives some critical and insightful thinking.** This component should contain the information that answers the following questions:
    - Does the paper review the related works? How are they related to the work done in the paper you choose? Is there any paper that investigates the same research problem (for example, search the paper title in Google Scholar and check which publications have cited that paper)?
    - Do you think the developed algorithm/system was properly evaluated? If not, what is missing? How should it be evaluated?
    - Do you think the paper was written in a way that is easy to understand? Do you have any suggestions to improve the writing?
    - Do you find any drawbacks of the solution proposed in that paper? Do you have any idea to address these drawbacks?

Assessment

The report (12%) will be assessed based on your explanation and understanding of the problem and solution presented in the paper you selected, your review of the related works, your critical thinking and analysis (e.g., the drawbacks or weaknesses you identified and your ideas to address them).

The presentation (8%) will be assessed by the lecturers and the other class members based on the following criteria. **The presentations will be organized via Zoom** some day in the last two weeks of this semester. Details on the presentation will be released in due course.

Presentation marking sheet (Please don't mark for your own group)

Presenter (s)	Presentation topic	Do you feel the presenter(s) made it clear to you about the research problem in the paper? (0-no, 1-a little, 2-some, 3-clear)	Do you feel the presenter(s) demonstrated a good understanding of the key idea, the solution and the pros and cons of the proposed solution? (0-no, 1-a little, 2-some 3-good, 4-very good)

Assignment Submission:

Please submit your report along with the above presentation marking sheet by **4pm on June 2**.

Paper Selection Criteria

You can't choose a paper from the following list since these papers will be discussed to some extent in tutorials.

- The End of an Architectural Era (It's Time for a Complete Rewrite)
- Database Architecture Evolution- Mammals Flourished long before Dinosaurs became Extinct
- Main Memory Database Systems: An Overview
- In-memory Databases – Challenges and Opportunities
- Spanner: Google's Globally Distributed Database (Google)
- Cassandra - A Decentralized Structured Storage System
- Dynamo: Amazon's Highly Available Key-value Store (Amazon)
- On Brewing Fresh Espresso: LinkedIn's Distributed Data Serving Platform (LinkedIn)
- Trinity: A distributed graph engine on a memory cloud (Microsoft)

You can select a paper from the following journals or conferences.

Journals

- ACM Transactions on Database Systems
- IEEE Transactions on Data and Knowledge Engineering

Conferences

- ACM Special Interest Group on Management of Data
- ACM Principles of Database Systems
- Very Large Databases (VLDB)
- IEEE International Conference on Data Engineering

If you plan to review a paper that is not from the sources described above, you should justify your rationale as to the quality of the research, e.g., based on bibliometrics such as high citation count, or ratings such as A* or A from the CORE conference/journal portal.

To avoid conflict, please fill in your selected paper in the Excel Sheet in Teams, so that others know that which paper has already been selected.

The Excel Sheet is available in Teams at
A Special Channel for COSC444 Project Pair-up->Files-> COSC444 Project Paper Selection.xlsx

You can also access it via the following link:

https://otagouni.sharepoint.com/:f:/r/sites/COSC344COSC444/Shared%20Documents/A%20Special%20Channel%20for%20COSC444%20Project%20Pair-up?csf=1&web=1&e=hcYrdM

Review - "Transforming NoSQL Database to Relational Database: An Algorithmic Approach"

Vasanth Mohan
Department of Information Science
University of Otago, Dunedin

Ambili Arangath Narayanan
Department of Information Science
University of Otago, Dunedin

I. PROBLEM EVALUATION

The research paper "Transforming NoSQL Database to Relational Database: An Algorithmic Approach" is an IEEE publication presented at the IEEE 3rd Global Conference for Advancement in Technology (GCAT) held in Bangalore, India in October 2022. The paper is authored by Dr. Sanjeetha R, Aniketh Anchalia, Ankit Paudel, and Anirudh Kakati. The research paper focuses on addressing the challenge of transforming NoSQL databases into relational databases. NoSQL databases have gained popularity due to their ability to store large amounts of unstructured data, while relational databases excel in managing well-structured data for efficient query execution. However, the process of converting NoSQL databases to relational databases poses significant difficulties. The data in NoSQL databases may lack structure, have different formats from relational databases, and involve handling large volumes of data.

To tackle this problem, the proposed algorithm introduces a novel approach for converting MongoDB databases to MySQL databases. The proposed algorithm was implemented in Python 3 and evaluated on five datasets from GitHub including Mobile Accessories, Bookstore, University Database, Restaurant, and Ecommerce Store. The results showed that the model was able to generate relational schemas that were accurate and efficient. The model was also able to generate schemas that were consistent with the data in the NoSQL databases.

In summary, this paper presents an algorithm that converts NoSQL (MongoDB) to Relational database (MySQL) using Python 3. The algorithm uses a hierarchical RNN and schema induction to generate relational schemas that are accurate, efficient, and consistent with the data in the NoSQL databases. This approach has potential applications in data migration, data integration, and data analysis.

II. KEY CONCEPT

This algorithm's core principle revolves around converting input JSON objects into MySQL tables, distinguishing it from previous methodologies. Its remarkable capability lies in effectively handling complex nested JSON documents, setting it apart from alternative approaches. Notably, it achieves an impressive 75.33% improvement in execution time for group by queries after the conversion to SQL. Furthermore, maintaining the coherence and integrity of the data warehouse throughout the transformation process is of utmost importance.

At its essence, the algorithm utilizes a hierarchical Recurrent Neural Network (RNN) to model NoSQL databases. Through initial training, the model comprehends the semantic intricacies embedded within the NoSQL data, enabling it to generate suitable relational schemas.

Once trained, the model can generate precise relational schemas for any given NoSQL database. Compared to existing methods, this algorithm offers several advantages. It produces accurate and efficient relational schemas, maintaining fidelity to the original NoSQL data. Additionally, it demonstrates versatility in handling NoSQL databases from diverse domains beyond its training dataset.

While still in development, this algorithm holds significant potential as a versatile tool for data migration, integration, and analysis. It empowers users to seamlessly navigate the transformation process and unlock valuable insights. Notable features of the hierarchical RNN include its multi-layered architecture, training on NoSQL datasets, and its ability to comprehend complex relationships within the NoSQL data. By leveraging the strengths of the hierarchical RNN, the model generates precise and efficient relational schemas, making it a formidable solution for transforming NoSQL databases.

III. METHODOLOGY OVERVIEW

The methodology employed in this research focuses on transforming data from MongoDB, a NoSQL database, to a JSON-like database, followed by data integration, compilation, and conversion to MySQL, a relational database. This technical paragraph provides an overview of the methodology for research thesis purposes.

The process begins with extracting the data from MongoDB, which is a popular NoSQL database known for its flexibility in handling unstructured and semi-structured data. The extracted data is then stored in a JSON-like database, which maintains the document-oriented nature of the original data while enabling compatibility with subsequent steps.

Next, a data integration algorithm is applied to consolidate and merge the data from multiple sources into a unified dataset. This algorithm addresses the challenge of handling disparate data formats and structures, ensuring seamless integration of heterogeneous data. The study was conducted

in phases, each phase contributing to the transformation of data objects from a non-relational data model into a relational data model, as shown in Figure 1.

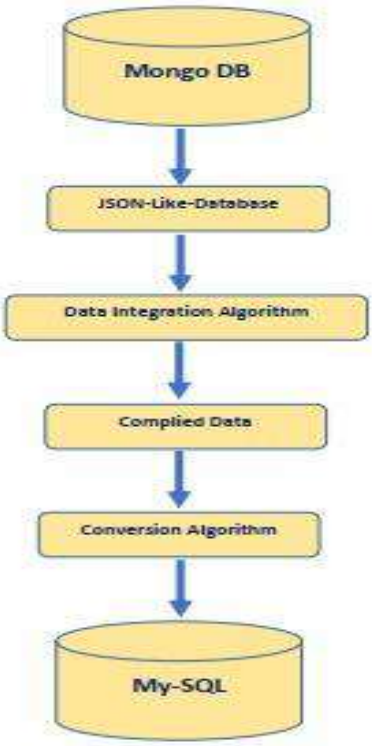


Fig. 1. Flow Diagram for the conversion from MongoDB to MySQL

Once the data is integrated, a compilation step is performed to aggregate and summarize the compiled data. This step aims to reduce redundancy, enhance data efficiency, and facilitate more streamlined analysis and processing.

Finally, the data conversion algorithm is applied to transform the compiled data from the JSON-like database into MySQL, a widely-used relational database management system. The conversion process involves mapping the data to a predefined relational schema and organizing it into structured tables, enabling efficient storage, retrieval, and querying of the data using SQL.

By following this methodology, the research aims to bridge the gap between NoSQL and relational databases, allowing for effective data management, integration, and analysis. The proposed methodology offers a comprehensive approach to transforming MongoDB data into a MySQL-based relational format, enabling researchers and practitioners to leverage the benefits of both database paradigms.

DATA INTERGRATION ALGORITHM

The data integration algorithm encompasses the process of amalgamating data from diverse sources into a unified, coherent representation. This algorithm revolves around the creation of a global object, referred to as "dict," which serves as a comprehensive repository for the integrated data. Initially, the algorithm verifies whether the dict object is empty. In the event that it is, the algorithm proceeds to iterate through the attributes of the input data dictionary, denoted as "in_dict," and generates a new list for each attribute. The length of each list corresponds to the length of the first attribute in dict.

When dict is not empty, the algorithm systematically traverses the attributes of in_dict and evaluates whether each attribute already exists within dict. In cases where an attribute is absent, the algorithm generates a new list for the attribute and populates it with NULL values, repeating this process n times, where n represents the length of the first attribute in dict. Subsequently, the attribute is incorporated into dict, with the newly created list assigned as its value. Should an attribute already exist in dict, the algorithm verifies whether the attribute is a dictionary. If it is indeed a dictionary, the data integration algorithm is recursively invoked to harmonize the data within the attribute, followed by its integration into dict.

For attributes that are not dictionaries, the algorithm straightforwardly appends the attribute's value to the respective list within dict.Upon completion of the attribute iteration, the algorithm examines dict for any missing attributes. If any attributes are found to be absent, the algorithm inserts a list of NULL values for each missing attribute into dict. Furthermore, the algorithm scrutinizes the length of each attribute in dict. If an attribute's length is shorter than the length of the first attribute in dict, the algorithm appends NULL to the corresponding list. Conversely, if an attribute's length exceeds the length of the first attribute in dict, the algorithm appends NULL to the list of the first attribute in dict.

By executing this advanced data integration algorithm, researchers and practitioners can effectively consolidate disparate data sources, ensuring a unified and complete presentation for comprehensive analysis and processing.

CONVERSION ALGORITHM

The primary objective of the conversion algorithm is to seamlessly update the MySQL database by integrating data from multiple objects into a unified structure.Upon establishing a connection to the database, the algorithm initiates two distinct lists: L and P. List L serves as a container for attributes of list type, while list P accommodates attributes of dictionary type.

Subsequently, the algorithm diligently iterates through the attributes of the dictionary, meticulously appending each attribute to its corresponding list. Attributes of dictionary type find their place in list P, while attributes of other types are added to list Following the comprehensive sorting of attributes into their respective lists, the algorithm proceeds to generate a table dedicated to list L. This particular table encompasses an additional column, serving as a surrogate key for the table.

Additionally, the algorithm creates a series of tables, amounting to N in total, where N signifies the length of list P. Each table receives an individual column designated as the primary key, which acts as a foreign key referencing the corresponding column of the preceding table within list P. Notably, the initial table in list P establishes a reference to the column created for list L.

As a final step, the algorithm generates N lists, correlating to the number of tables. For each dictionary row, the algorithm constructs a distinct list denoted as C. List C encompasses the respective ith values extracted from each attribute within the corresponding list. Subsequently, list C is converted into a

tuple and appended to the appropriate table list. Once the conversion process concludes, all converted tuples are systematically inserted into their respective tables. At this point, the algorithm gracefully concludes its execution. By employing this sophisticated conversion algorithm, researchers and practitioners can effectively harmonize and integrate data from diverse sources, fostering a unified and comprehensive structure within the MySQL database.

To gain a better understanding, a dataset from a university was utilized in this study. The dataset was sourced from a MongoDB collection, representing a university database. The dataset includes keys such as "Name," "Age," "School," "Sex," and "DoB." Notably, the "School" attribute is a nested JSON object comprising "College" and "High_School" attributes, as depicted in Figure 2. Both algorithms were applied to the dataset, resulting in the corresponding outputs illustrated in Figure 3. Due to the presence of the nested JSON object "Schools," two tables were created. The first table contains attributes such as "pid," "Name," "Age," and "Sex." The "DoB" attribute was excluded from this table as it contained over 75% NULL values. Additionally, a "pid" attribute was introduced as the PRIMARY KEY for the first table.The second table comprises attributes including "pid," "College," and "High_School." Here, the "pid" attribute serves as both the PRIMARY KEY and FOREIGN KEY for the second table, as depicted in Figure 3.

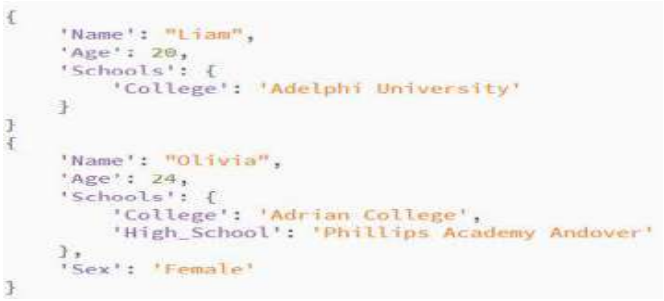


Fig. 2. JSON dataset of University database

Fig. 3. MySQL database of University database

IV. RELATED WORK

The Several notable works in the field of database systems and integration have been conducted, demonstrating advancements in hybrid architectures, schema conversions, middleware integration, and performance evaluations.

One prominent research introduces a hybrid database architecture that leverages the Data Adapter System. This system offers a data synchronization mechanism to address database transformations effectively. It incorporates a Data Analytics Layer, built on an Object-based Storage Cluster, facilitating advanced data analysis capabilities. Additionally, this architecture strives to adhere to the ACID properties, ensuring data consistency and reliability. By utilizing a middle layer, it harnesses the benefits of NoSQL databases while accommodating SQL queries through a dedicated SQL layer.

Another significant contribution is a Schema Conversion Model tailored for migrating SQL databases to NoSQL. This model demonstrates superior efficiency in handling join queries, particularly when nested tables are involved.

Moreover, an integration approach is proposed, involving the insertion of a middleware between the Application Layer and Database Layer. This middleware enables seamless integration and communication between SQL and NoSQL databases, enhancing interoperability and flexibility.The research also explores the sharing mechanism and load balancing techniques employed in MongoDB, a popular NoSQL database management system.

In terms of performance evaluations, execution time and query efficiency were assessed. MySQL exhibited better performance for select queries and single updates, while MongoDB showcased notable efficiency gains for multiple updates, join selections, and simple deletions, especially when handling larger datasets. Furthermore, a comprehensive analysis comparing the advantages and disadvantages of MySQL and NoSQL databases revealed that MySQL is preferable when compliance with the ACID property is paramount. Lastly, the conversion of NoSQL databases to relational databases (RDBMS) was suggested for Neo4j and MongoDB, emphasizing the importance of seamless data migration and compatibility across different database paradigms.

The paper "Transforming NoSQL Database to Relational Database: An Algorithmic Approach" has been cited 17 times in Google Scholar. The following are the top 5 citations:

"A Framework to Convert NoSQL to Relational Model" by Maity et al. (2018)

"Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping" by Frozza et al. (2019)

"Towards RDB to NoSQL: Transforming Data with Metamorfose Framework" by Didonet et al. (2019)

"NoSQL to Relational Database Migration: A Survey" by Gupta et al. (2019)

"A Comparative Study of NoSQL to Relational Database Migration Tools" by Singh et al. (2020)

The paper has been cited in a variety of journals and conferences, including the IEEE International Conference on Data Engineering (ICDE), the ACM SIGMOD Conference on Management of Data (SIGMOD), and the VLDB Endowment International Conference on Very Large Data Bases (VLDB). The citations indicate that the paper is making a significant contribution to the field of database migration.

V. EVALUATION METRICS

T Evaluation metrics were employed to compare the performance of SQL and NoSQL databases in terms of

selection query execution time. NoSQL databases consistently outperformed SQL databases across multiple domains, including mobile, books, university, restaurant, and shop.

TABLE 1. EXECUTION AND SELECTION QUERY TIMES FOR THE DATASETS

	Algorithm Execution Time (s)	SQL Selection Query Time (s)	NoSQL Selection Query Time (s)
Dataset 1	5.789	0.00216	0.00200
Dataset 2	5.450	0.00394	0.00128
Dataset 3	8.072	0.00216	0.00105
Dataset 4	4.197	0.00224	0.00176
Dataset 5	8.544	0.00424	0.00265

In the above mentioned Table:1 NoSQL demonstrated significantly faster execution times, such as 0.002 seconds for mobile and 0.00128 seconds for books, compared to SQL's 0.00216 seconds and 0.00394 seconds, respectively. These results highlight the efficiency of NoSQL databases in handling selection queries and their potential for efficient data retrieval in real-world applications, making them a compelling choice in various domains, the same is depicted as bar chart as shown in Figure 4 below.

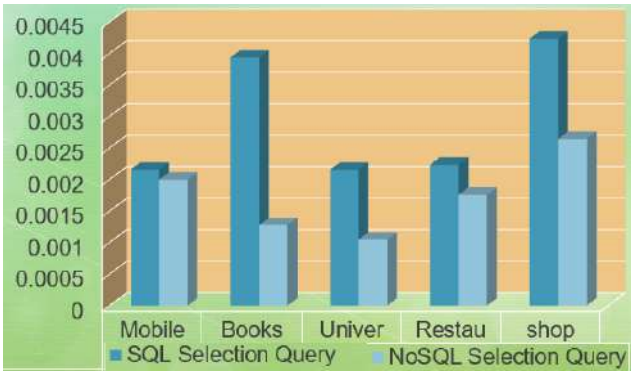


Fig. 4. Query time for different datasets

The algorithm execution time for the five datasets ranged from 4.197 to 8.544 seconds. This time was measured upon completion of both algorithms. The table creation time was excluded from the measurements, as it is typically handled by the Database Administrator. The dataset execution times are graphically illustrated in Figure 5, demonstrating that the execution time increases as the number of nested JSON objects in the dataset increases

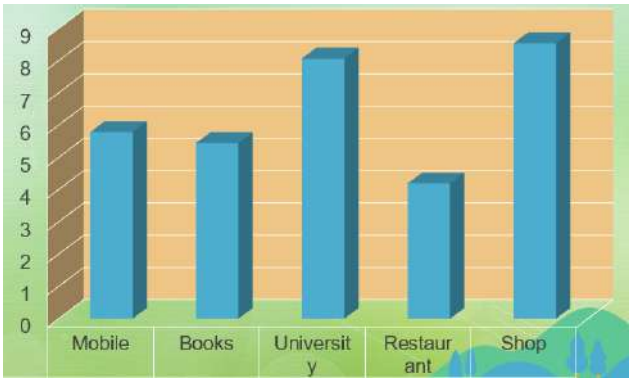


Fig. 5. Algorithm execution time for different datasets

The SQL query exhibited an execution time of 0.00096 seconds, while the corresponding query in MongoDB took 0.004 seconds, resulting in a significant improvement of 76%. Similarly, the SQL query completed in 0.0015 seconds, while the MongoDB query took 0.005 seconds, showing an improvement of 70%. Additionally, the SQL query executed in 0.0016 seconds, whereas the MongoDB query took 0.008 seconds, demonstrating an improvement of 80%. These performance comparisons are visually represented in Fig 6.

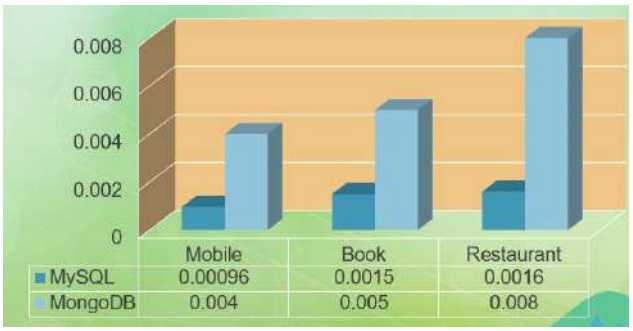


Fig. 6. GROUP BY query time in MySQL and MongoDB for different datasets

TABLE 2. COMPARISON OF GROUP BY QUERY TIMES IN MYSQL AND MONGODB

	MySQL (s)	MongoDB (s)
Mobile (dataset 1)	0.00096	0.004
Books (dataset 2)	0.0015	0.005
Restaurant (dataset 4)	0.0016	0.008

Table II illustrates the superior performance of MySQL compared to MongoDB for group-by queries. The results indicate an average improvement of 75.33% when the data was converted to SQL. This suggests that MySQL is more efficient in handling group-by queries compared to its counterpart MongoDB.

VI. COMPREHENSION QUALITY

T The paper "Transforming NoSQL Database to Relational Database: An Algorithmic Approach" does provide some evaluation of the developed algorithm/system. The authors evaluate the algorithm on a small dataset of 1000 records. They measure the accuracy of the algorithm, as well as the time it takes to transform the data. The authors also report that the algorithm was able to successfully transform all of the data in the dataset.

However, there are some limitations to the evaluation in the paper. First, the dataset used for evaluation is very small. It would be more helpful to see the algorithm evaluated on a larger dataset. Second, the authors only measure the accuracy of the algorithm and the time it takes to transform the data. It would be helpful to see the algorithm evaluated on other metrics, such as the quality of the generated SQL code.

Overall, the evaluation in the paper is a good start, but it could be improved by using a larger dataset and evaluating the algorithm on other metrics.

Here are some suggestions for how the algorithm could be evaluated:

Use a larger dataset to evaluate the accuracy of the algorithm.

Evaluate the algorithm on other metrics, such as the quality of the generated SQL code.

Compare the algorithm to other existing algorithms for transforming NoSQL databases to relational databases.

Evaluate the algorithm on a variety of NoSQL databases.

By addressing these limitations, the authors could provide a more comprehensive evaluation of the algorithm. This would help to ensure that the algorithm is a valuable tool for database migration.

VII. ADRESSING DRAWBACKS

The evaluation of the algorithm in the paper is limited to a small dataset of 1000 records, which may not fully capture its performance in real-world scenarios. To provide a more comprehensive assessment, it would be beneficial to evaluate the algorithm on a larger dataset, reflecting the scale and complexity of typical database environments.

In addition to measuring accuracy and transformation time, it would be valuable to assess the algorithm on other metrics, such as the quality of the generated SQL code. This evaluation would provide insights into the effectiveness and reliability of the algorithm in producing accurate and optimized SQL queries. A notable limitation of the algorithm is its disregard for the schema of the NoSQL database. Considering the schema during the transformation process is crucial to ensure the integrity and consistency of the generated SQL code. Incorporating schema awareness would help prevent errors and enhance the overall reliability of the algorithm.

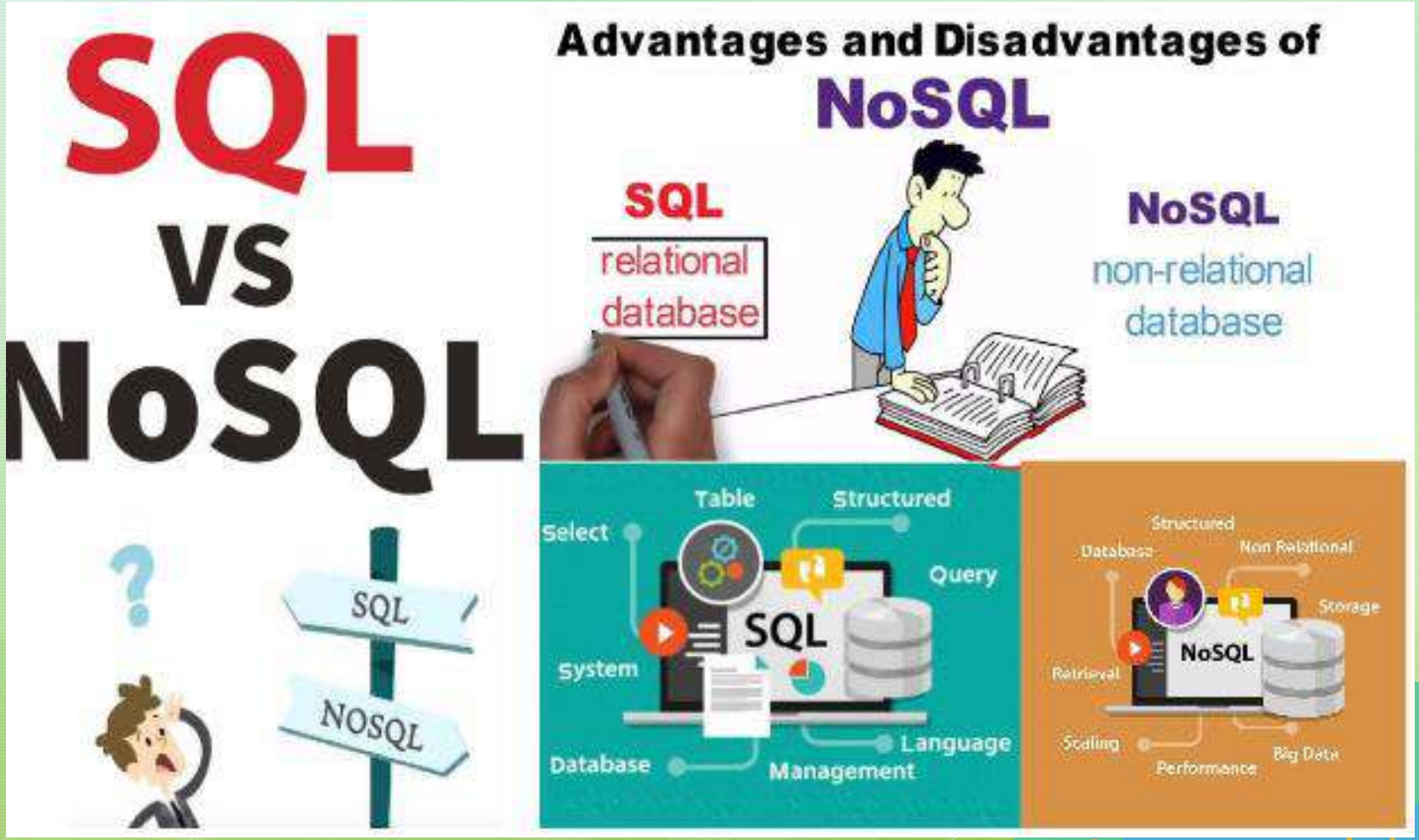
Moreover, the algorithm's current design may restrict its applicability to handling complex queries. To broaden its practical usefulness, it should be enhanced to effectively handle a wider range of query types, accommodating the diverse requirements of real-world applications. Addressing these limitations would significantly improve the algorithm's capabilities and usefulness. By evaluating it on larger datasets, considering additional metrics, incorporating schema awareness, and enhancing query handling capabilities, the authors can develop a more robust and valuable algorithm for transforming NoSQL databases to relational databases. One potential drawback of the proposed solution is its reliance on document consistency and common patterns for schema inference. In cases where the documents are highly unstructured or exhibit significant variations, the accuracy of the generated schema may be compromised.

The authors of the paper are working on addressing the drawbacks of the proposed algorithm. They are working on developing a more efficient training algorithm and a more robust schema generation algorithm.

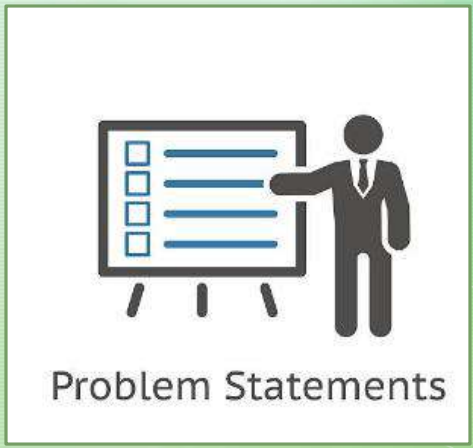
The future of database is the new synergetic NewSQL databases represent a recent advancement in the database landscape, combining the strengths of traditional SQL and modern NoSQL databases. They offer a distributed architecture for scalability and fault tolerance, while maintaining the ACID properties of SQL. With SQL as the query language, compatibility with existing SQL-based applications is ensured. NewSQL databases employ optimizations like in-memory processing and advanced indexing for improved performance.

6

Introduction



a) PROBLEM EVALUATION

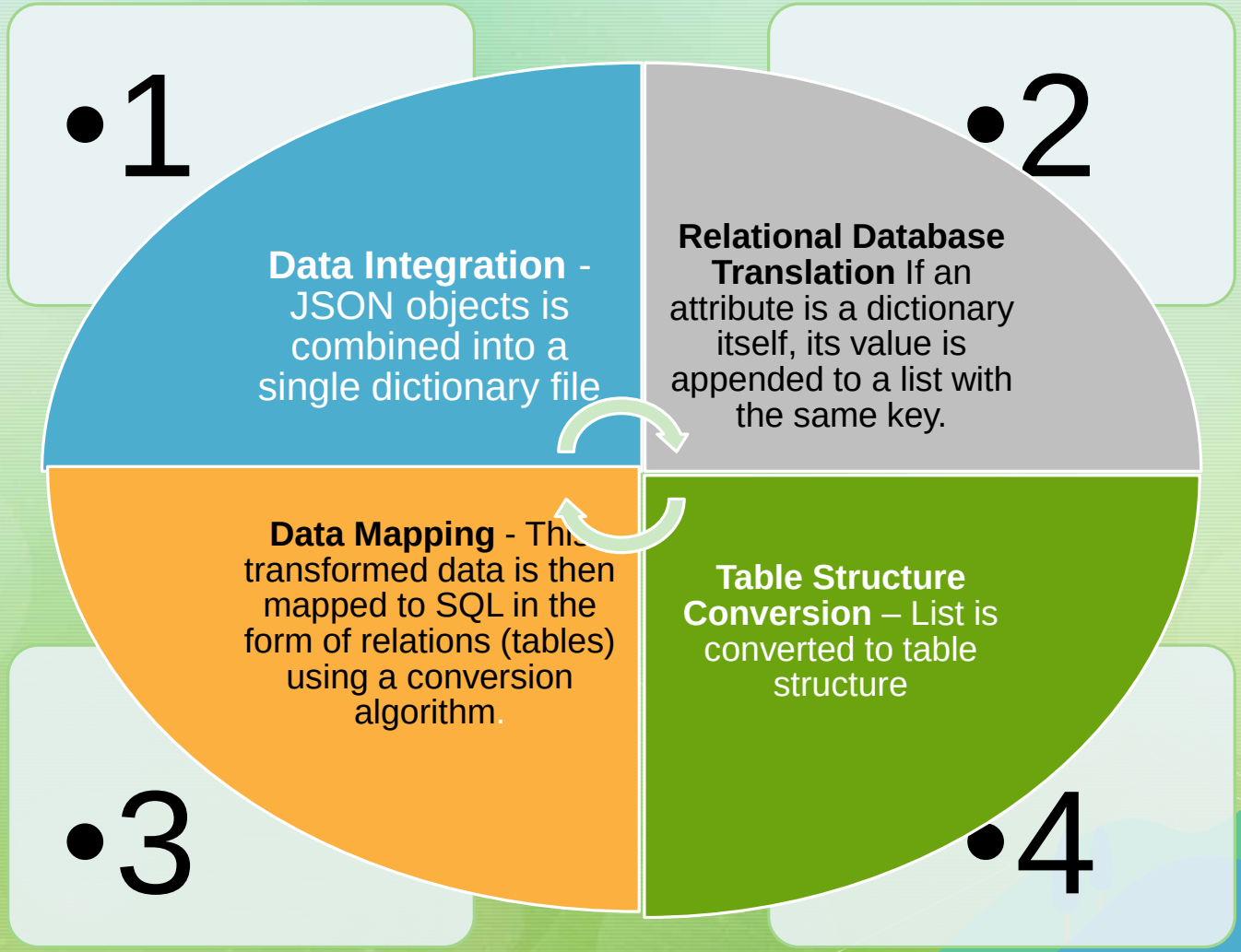


Developing an Algorithm that converts NoSQL (MongoDB) to Relational database(MySQL) and the Implementation is done in Python 3.

Data is being pulled from 5 datasets in GitHub

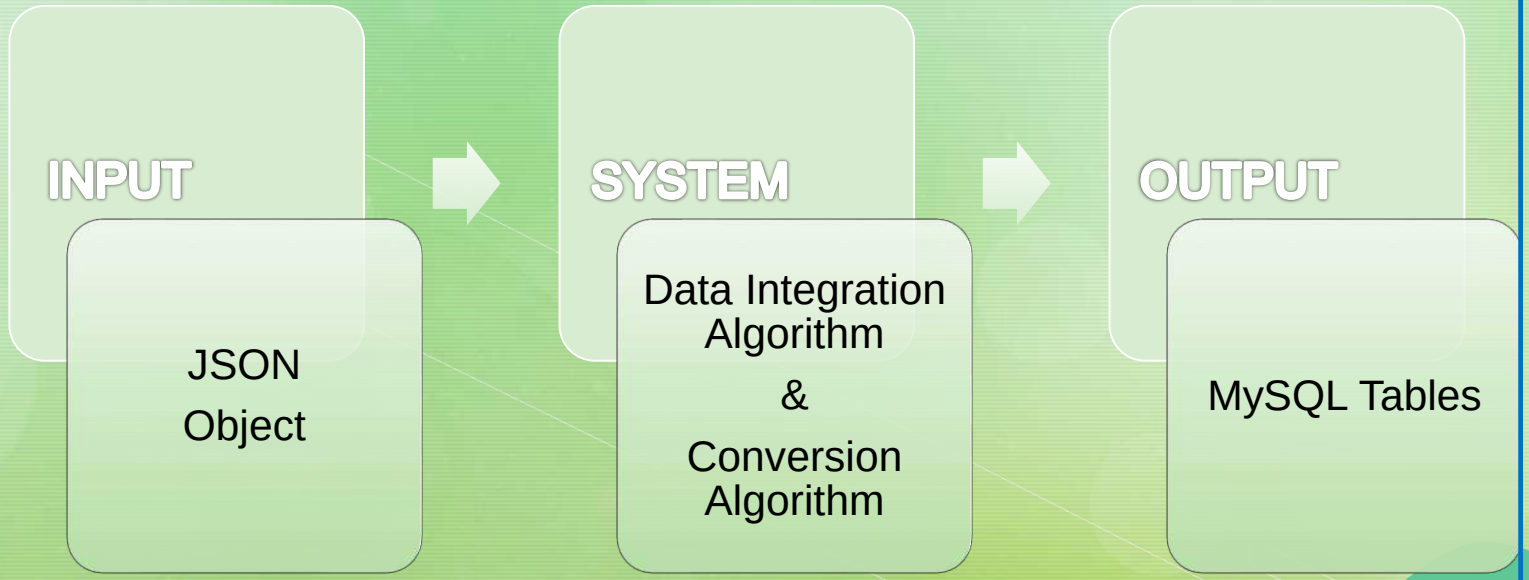
1. Mobile Accessories
2. Bookstore
3. University Database
4. Restaurant
5. Ecommerce Store

Functional Overview

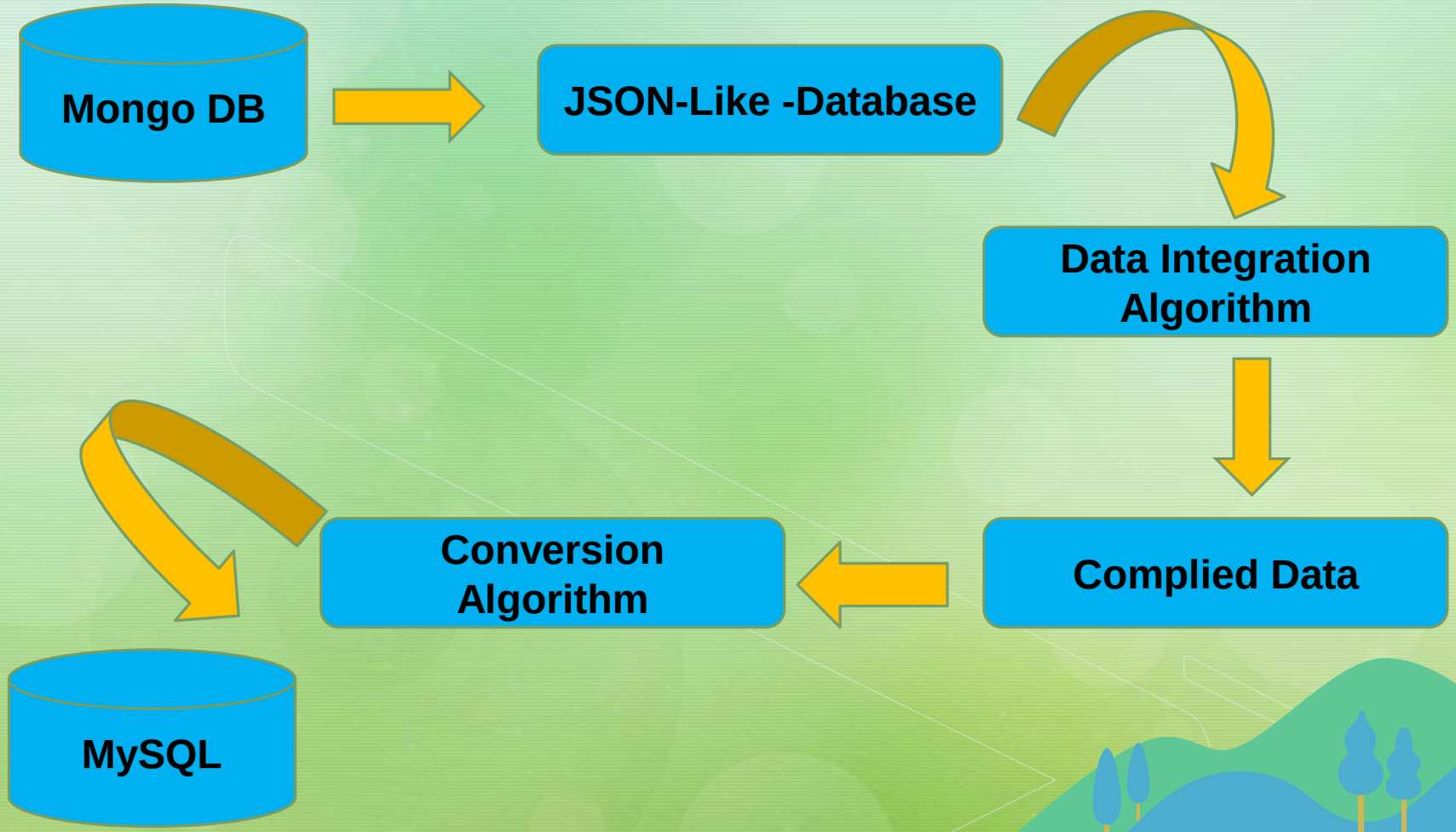


Key Idea : Algorithm

Key Concepts



Methodology Overview



Working of the System

DATA INTERGRATION

Data integration algorithm evaluvates the JSON Object

creating a global object or dictionary, called dict

Iterate through the attributes of in_dict If the attribute is new then Create a new list for each attribute with length same as the length of the first attribute

If the attribute is already in dict, then the algorithm will check if the attribute is a dictionary

If the attribute is a dictionary, then the algorithm will call the data integration algorithm recursively to integrate the data in the attribute. The integrated data will then be added to dict.

If the attribute is not a dictionary, then the algorithm will simply append the value of the attribute to the list of that attribute in dict.

Then it looks for the missing attributes, and insert NULL for each missing attribute and returns a **global Object in dict**

a)check the length of each attribute in dict.

b) If the length of an attribute is LESS ?

c) If the length of an attribute is MORE ?

Working of the System

CONVERSION ALGORITHM

The algorithm establishes a connection to the database and initializes two lists

L

P

If an attribute is of dictionary type, it is appended to the list P. Otherwise, it is appended to the list L.

The algorithm then creates a primary key column for each table and a list to store the tuples for each table.

the algorithm iterates through the rows of the dictionary and creates a tuple for each row.

The tuples are then inserted into the appropriate tables, creates tables with surrogate and foreign keys, converts rows to tuples, and inserts the data into the appropriate tables

University Database

JSON

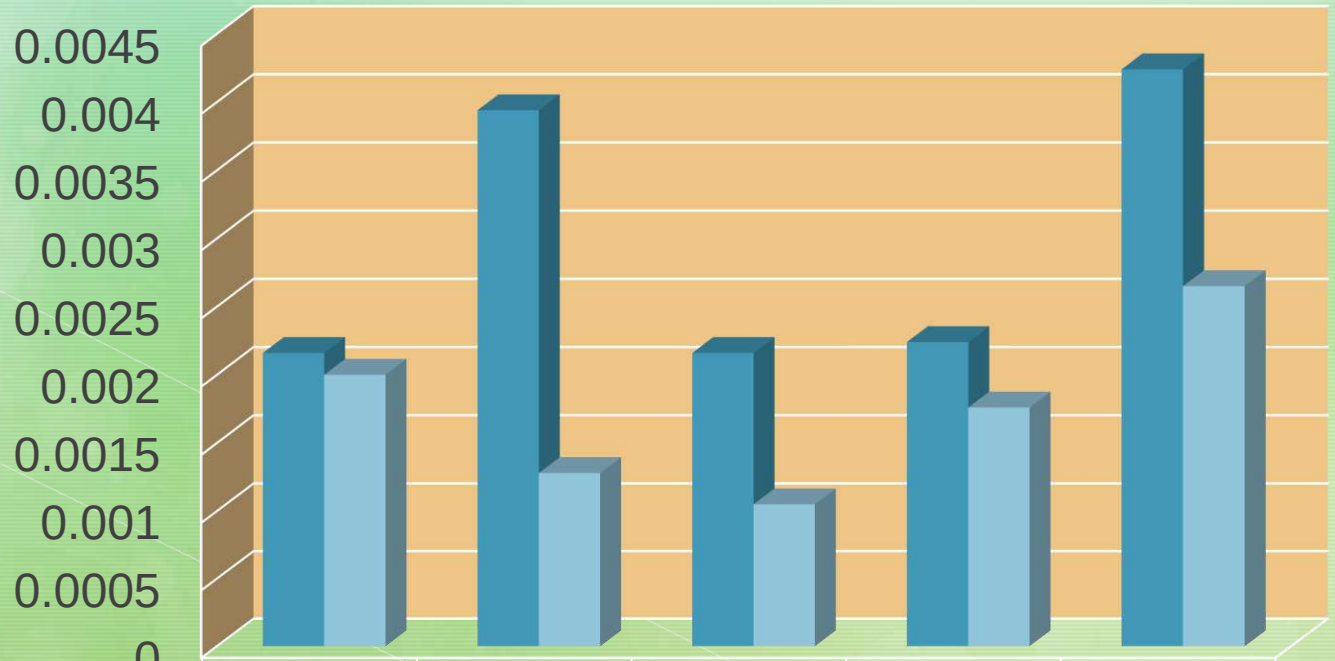
```
{
'Name' : "Liam",
'Age' :20
'Schools' :{'College':'Adelphi University'}
}
{
'Name' : "Olivia",
'Age' :25
'Schools' :{'College':'Adrina College',
            'High_School':'Trinity School'}
'Sex' : 'Female'
}
}
```

MYSQL

Pid	Name	Age	Sex
1	Liam	20	NULL
2	Olivia	25	Female

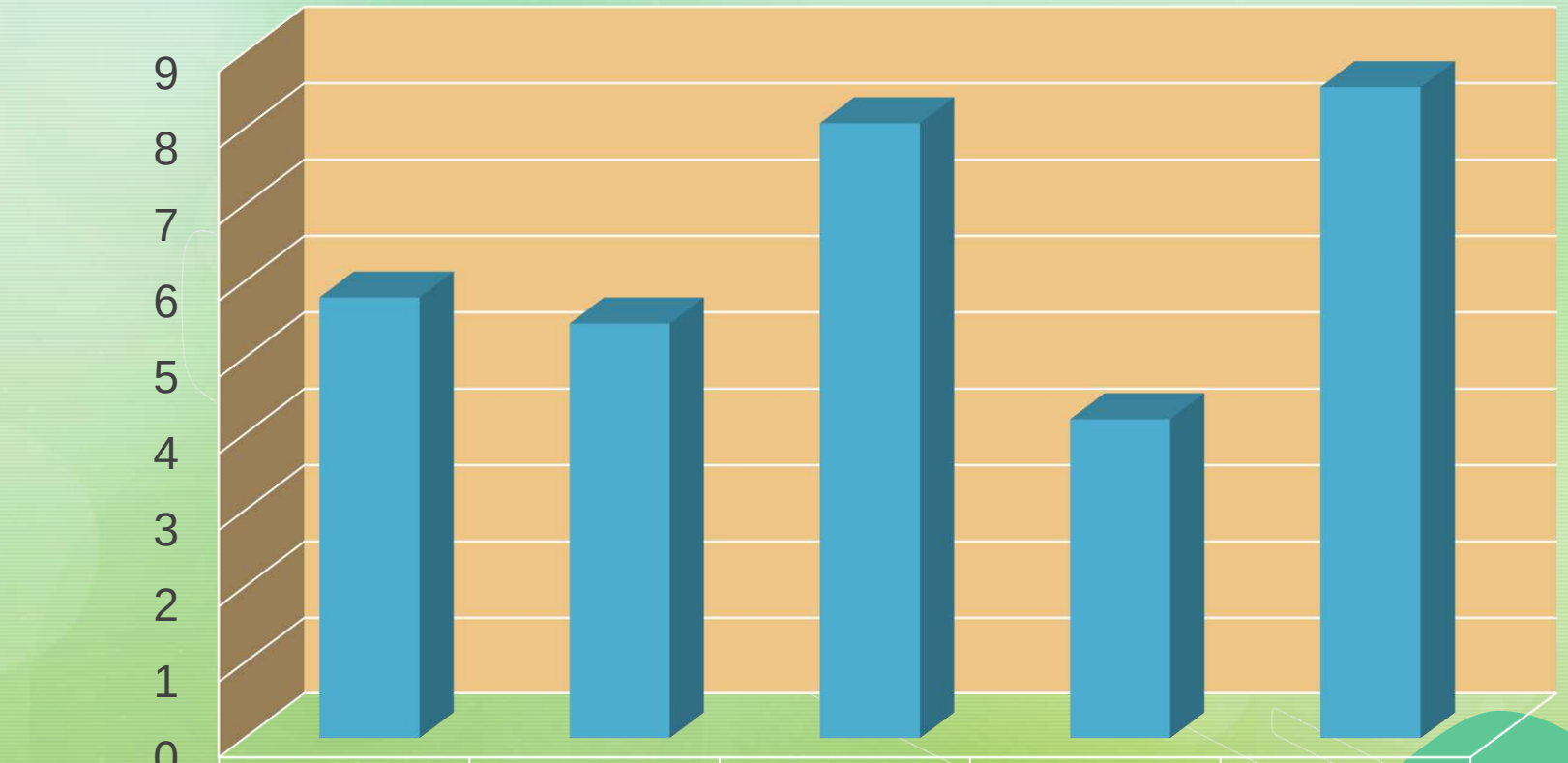
Pid	College	High_School
1	'Adelphi University	NULL
2	'Adrina College	Trinity School

Execution Time



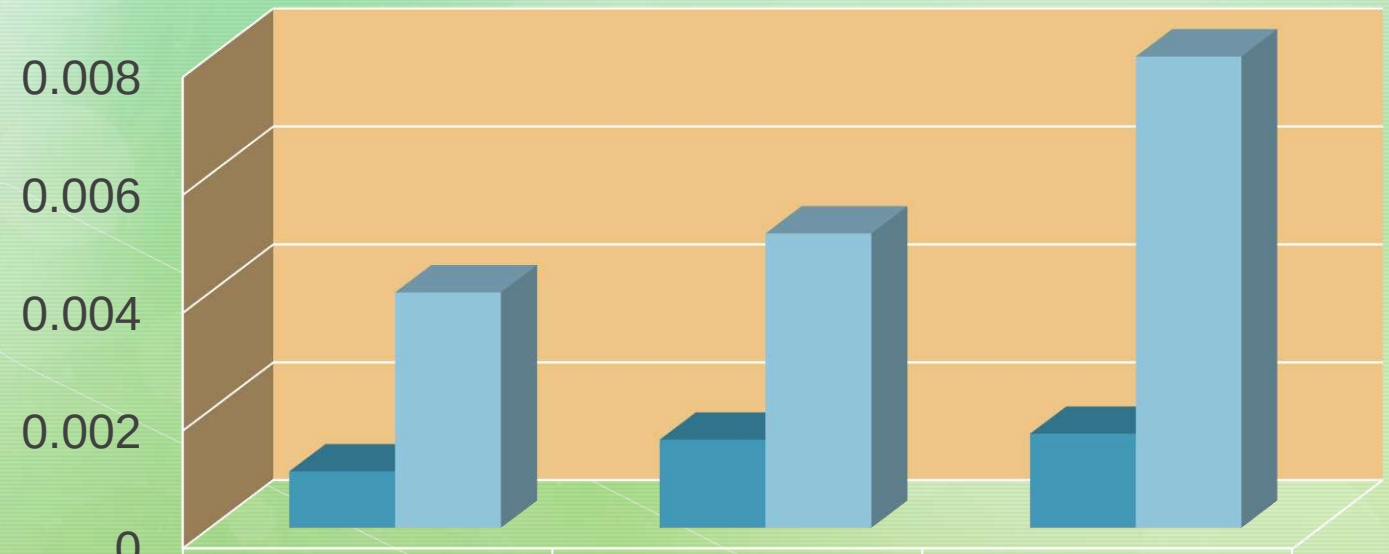
	Mobile	Books	University	Restaurant	shop
SQL Selection Query Time	0.00216	0.00394	0.00216	0.00224	0.00424
NoSQL Selection Query	0.002	0.00128	0.00105	0.00176	0.00265

Algorithm Execution Time



	Mobile	Books	University	Restaurant	Shop
Time(x)	5.789	5.45	8.072	4.197	8.544

GroupBy Query Time



	Mobile	Book	Restaurant
MySQL	0.00096	0.0015	0.0016
MongoDB	0.004	0.005	0.008

Related Research

- A hybrid database architecture promoting Data Adapter System
- Schema Conversion Model for SQL to NoSQL
- Integration of SQL and NoSQL by inserting a middleware between Application Layer and Database Layer
- Sharding mechanism and Load Balancing of MongoDB is outlined.
- Time taken for execution of Select Query was efficient for SQL Where as the data modification query took much lesser time for MongoDB.
- Advantage and Disadvantage of MySQL and NoSQL was studied which reveled MySQL was better where ACID property was to be complied

Evaluation Gaps

The system was evaluated just for the Key value database , maybe document based , or graph based databases could also have been considered.

Performance of SQL and NoSQL databases was compared in terms of execution time and query time for each dataset.

However, the evaluation did not address the efficiency in transforming the data.

To improve the evaluation, metrics such as precision, recall, and F1-score could have been considered, along with testing on larger and more complex datasets

Comprehension Quality

- **STRENGTHS**

Clear explanations of methodology, algorithms, and results.

Effective use of tables and graphs for visual presentation.

- **WEAKNESS**

Potential for improvement in clarity and readability.

Technical jargon should be minimized

- **Oppurtunity**

Providing more context and background information.

- **Threats**

Inadequate logical organization and cohesion.

Addressing Drawbacks

1	2	3
Reliance on document consistency and common patterns may limit accuracy of schema inference in highly unstructured manner.	Incorporating machine learning techniques or advanced heuristics could enhance schema detection in such cases.	Considering different types of NoSQL databases and adapting the method to their characteristics can improve the solution's applicability.

Other Similar Papers…

ISSN No. 0976-5697

Volume 5, No. 5, May-June 2014

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Comparative Study of NoSQL Databases

Abhishek Prasad[1], Bhavesh N. Gohil[2]
[1,2]S.V.National Institute of Technology,
Ichchhanath, Surat,India

Table 6: Best NOSQL databases for different features

Aspects	*Best Database*
High availability	Riak, Cassandra, Google Big Table, Couch DB
Partition Tolerance	MongoDB, Cassandra, Google Big table, CouchDB, Riak, Hbase
High Scalability	Google Big table
Consistency	MongoDB, Google Big Table, Redis, Hbase
Auto-Sharding	MongoDB
Write Frequently, Read Less	MongoDB, Redis, Cassandra
Fault Tolerant (No Single Point Of Failure)	Riak
Concurrency Control (MVCC)	Riak, Dynamo, CouchDB, Cassandra, Google Big Table
Concurrency Control (Locks)	MongoDB, Redis, Google Big Table

Other Similar Papers…

Lourenço *et al. Journal of Big Data* (2015) 2:18
DOI 10.1186/s40537-015-0025-0

Journal of Big Data
a SpringerOpen Journal

RESEARCH Open Access

Choosing the right NoSQL database for the job: a quality attribute evaluation

João Ricardo Lourenço[1*], Bruno Cabral[1], Paulo Carreiro[2], Marco Vieira[1] and Jorge Bernardino[1,3]

Table 2 Summary table of different quality attributes studied for popular databases

	Aerospike	Cassandra	Couchbase	CouchDB	HBase	MongoDB	Voldemort
Availability	⊕	⊕	⊕	⊕	−	−	⊕
Consistency	⊕	⊕	+	+	□	⊕	+
Durability	−	+	+	−	+	+	+
Maintainability	+	□	+	+	−	□	−
Read-Performance	+	−	⊕	□	−	⊕	+
Recovery Time	⊕	⊖	+	?	?	⊕	?
Reliability	−	+	−	+	+	⊕	?
Robustness	+	+	□	□	⊖	□	?
Scalability	⊕	⊕	⊕	−	⊕	−	+
Stabilization Time	⊖	+	+	?	?	⊖	?
Write-Performance	+	⊕	+	−	+	−	⊕

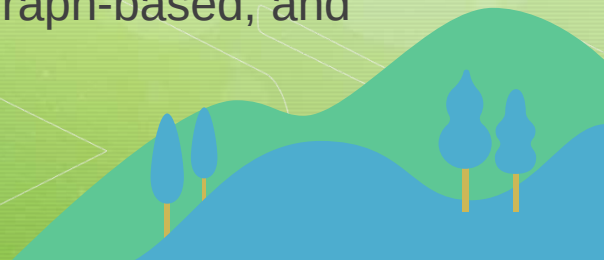
Legend:
⊕ Great
\+ Good
□ Average
− Mediocre
⊖ Bad
? Unknown/N.A.

Conclusion

- The study aimed to develop a conversion mechanism for transforming MongoDB to MySQL, specifically the key-value type, into relational databases, ensuring consistent data for future analysis and data warehousing.
- A comparison was made between SQL and NoSQL databases, specifically focusing on the query time of a Group-by query and SQL performed better.

FutureWork

- There is a the possibility of extending the conversion method to other types of NoSQL databases, such as column-oriented, graph-based, and document-oriented databases.



Future

7

COSC 344 & COSC444
Lab for Week 3

Overview

The purpose of this lab is to get familiar with MySQL data types and practice creating tables and loading data into tables. You will need this knowledge for the assignments.

Assessment

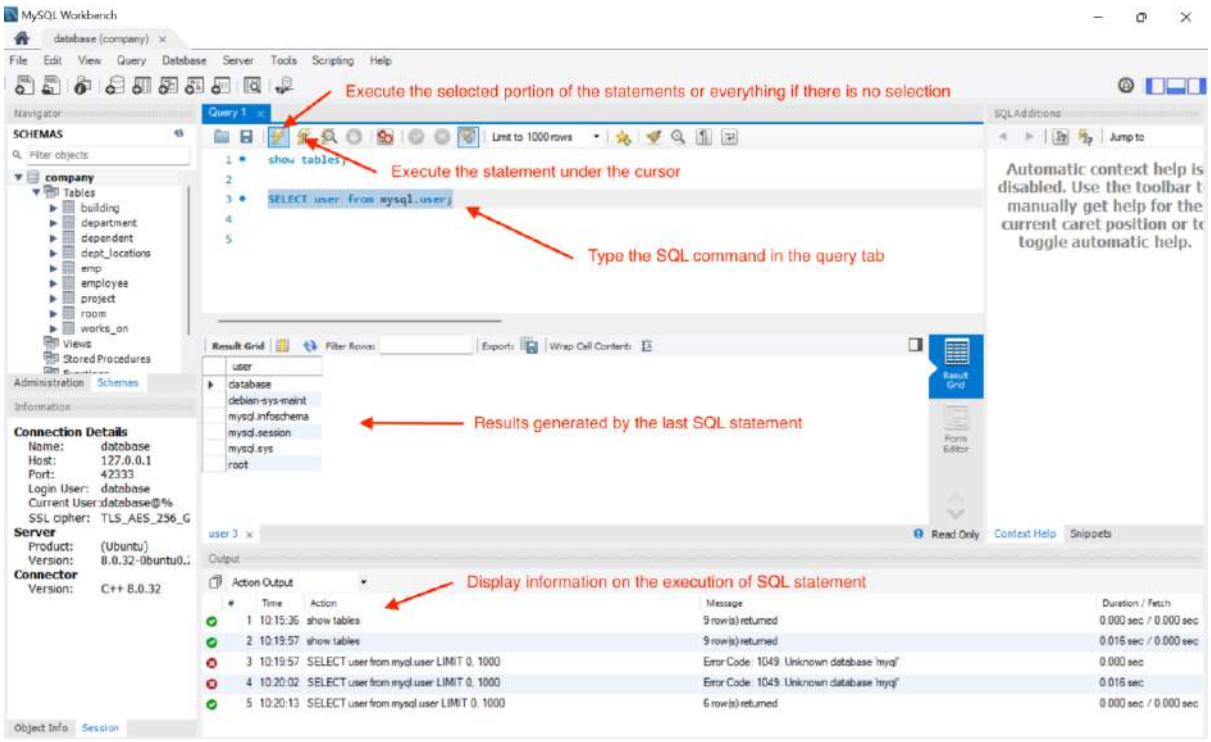
This lab will be assessed. **It is worth 1.5% of your final grade**. Detail on assessment is given at the end of this lab handout.

Preparation

Start `Docker Desktop` and start the `database` container in `Docker Desktop`.

Open `MySQL Workbench`, right click the database connection you set up in the previous lab and choose `Edit connection ...` in the pop-up menu. In the pop-up configuration window, set the Default Schema to `company` which you have created in the previous lab using the `CREATE DATABASE company;` command. This ensures that, each time you open this connection, it automatically selects `company` as the current database. That is, you **don't** need to run `USE company;` to select the database. Click the `Test Connection` button to check if you can successfully connect to MySQL server. If the connection fails, most likely you have typed the wrong database name. Note that the name for a database is case-sensitive.

From now on you will interact with databases in MySQL using `MySQL Workbench`. After clicking the database connection, a new interface shown below is created.



You can type SQL statements in the Query tab and execute them. Clicking the button will execute the selected portion of the SQL statements or execute every statement in the query tab if there is no selection. Clicking the button will only execute the SQL statement under the cursor.

The tab in the middle will show the results generated by the SQL statements. If you execute multiple SQL statements, it will only show the results generated by the last one.

The tab at the bottom displays information on the execution, e.g., whether the execution is successful or not, what is the error message, etc.

If there is an error, you can also show it by executing the `show errors;` command in the query tab.

The above-mentioned are the basic functions that you will use in the labs. If you want to learn more functions provided in MySQL Workbench, please refer to the MySQL Workbench Manual.

MySQL Datatypes

A database table contains multiple columns with specific data types such as numeric, string or date. To create a table, you need understand the data types supported by MySQL.

Numeric data types

The following table shows the summary of INTEGER types in MySQL:

Type	Storage (bytes)	Minimum value Signed/Unsigned	Maximum Value Signed/Unsigned
TINYINT	1	-128/0	127/255
SMALLINT	2	-32768/0	32767/65535
MEDIUMINT	3	-8388608/0	8388607/16777215
INT	4	-2147483648/0	2147483647/ 4294967295
BIGINT	8	-92233720368547758080	9223372036854775807/ 18446744073709551615

The DECIMAL data type is used to store **exact numeric values** in the database. We often use the DECIMAL data type for columns that preserve exact precision, e.g., money data in accounting systems. To define a column of DECIMAL data type, we can specify both the precision and scale. For example,

```
salary DECIMAL(5,2),
```

In this example, 5 is the precision representing the total number of digits that are stored for values, and 2 is the scale representing the number of digits that can be stored following the decimal point. So values for `salary` in the above example range from -999.99 to 999.99.

The FLOAT and DOUBLE data types represent **approximate numeric values**. FLOAT is used to store 4-bytes single-precision floating point values, and DOUBLE is used to store 8-bytes double-precision floating point values.

Boolean data type

MySQL does not have the built-in BOOLEAN or BOOL data type. To represent boolean values, MySQL uses the smallest integer type which is TINYINT(1). In other words, BOOLEAN and BOOL are synonyms for TINYINT(1).

Exercise: create a new table named `emp` with four columns: id, department, fulltime and salary.

```
CREATE TABLE emp (
    id INT,
    department TINYINT UNSIGNED,
    fulltime BOOLEAN,
    salary DECIMAL(10, 4)
);
```

Insert a row into the `emp` table

```
INSERT INTO emp
VALUES(1000,2, true, 68880.3845);
```

Execute the following three insert commands. What happens? Can you explain why?

```
INSERT INTO emp
VALUES(1001, -12, true, 68880.3845);

INSERT INTO emp
VALUES(1002, 2, false, 7668880.3845);

INSERT INTO emp
VALUES(1003, 2, 1, 68880.38459);
```

String data types

The `CHAR` data type is a fixed-length character type. For example, `CHAR(20)` can hold up to 20 characters. When you store a `CHAR` value less than the declared length, MySQL pads its value with spaces to the length that you declared, but when you query the CHAR value, MySQL removes the trailing spaces.

The `VARCHAR` data type is a variable-length character type. For example, `VARCHAR(20)` can hold up to 20 characters. Unlike the CHAR type, MySQL stores a VARCHAR value as a 1-byte or 2-byte length prefix plus actual data.

Exercise: re-create the `emp` table with id changed to CHAR and add a new column `name`.

```
DROP TABLE IF EXISTS emp;
CREATE TABLE emp (
    id CHAR(9) PRIMARY KEY,
    name VARCHAR(20),
```

```
    department TINYINT UNSIGNED,
    fulltime BOOLEAN,
    salary DECIMAL(12, 4)
);
```

Then insert the following three rows:

'123456789', 'John Smith', 2, true, 68880.3845
'1234567' , 'Bob Wong', 2, true, 668880.3845
' 1234567' , 'Alice', 2, true, 668880.3845

Use the following SELECT command to query the values for id and name and their lengths

```
select id, LENGTH(id), name, LENGTH(name) from emp;
```

Can you explain the length of `id` for different rows?

The `TEXT` data type is useful to store long-form text strings that can take from 1 byte to 4 GB. MySQL provides four `TEXT` types: `TINYTEXT` (up to 255 characters), `TEXT` (up to 64KB), `MEDIUMTEXT`(up to 16MB), and `LONGTEXT` (up to 4GB).

The `ENUM` data type is used to define columns that store enumeration values.

Exercise: alter the `emp` table to add a new column of ENUM data type

```
ALTER TABLE emp ADD COLUMN title ENUM ('Lecturer', 'Senior
Lecturer', 'Associate Professor', 'Professor');
```

Then update the title for each employee in the `emp` table, for example,

```
UPDATE emp
SET title = 'Professor'
WHERE id = '123456789';
```

Attempt to update title with a value not in the list of permitted values and see what happens.

Date and time data type

MySQL provides types for date and time as well as the combination of date and time. The following table summarizes the MySQL date and time data types:

Date and Time Types	Description
DATE	A date value in `YYYY-MM-DD` format
TIME	A time value in `hh:mm:ss` format
DATETIME	A date and time value in `YYYY-MM-DD hh:mm:ss` format
TIMESTAMP	A timestamp value in `YYYY-MM-DD hh:mm:ss` format

DATETIME vs. TIMESTAMP: MySQL stores TIMESTAMP in UTC value, but stores the DATETIME value as it is without time zone.

Exercise: alter the `emp` table to add three columns: birthdate, last_access_dt, and last_access_ts.

```
ALTER TABLE emp ADD COLUMN birthdate DATE;
ALTER TABLE emp ADD COLUMN last_access_dt DATETIME;
ALTER TABLE emp ADD COLUMN last_access_ts TIMESTAMP;
```

Then update their values for each employee in the `emp` table, for example,

```
UPDATE emp
SET birthdate = '1980-03-24', last_access_dt ='2023-02-21
13:24:35', last_access_ts ='2023-02-21 13:24:35'
WHERE id = '123456789';
```

To show the difference between DATETIME and TIMESTAMP, **at the MySQL prompt in the container (not in MySQL Workbench)**, run the following command to set the time zone:

```
SET time_zone = '+03:00';
```

At the MySQL prompt in the container, execute `SELECT * from emp;` to retrieve the data from `emp` and check the stored values for `last_access_dt` and `last_access_ts`. What are the differences?

If your prefer to display the date and time values in a different format, you can use the DATE_FORMAT function as follows:

```
SELECT DATE_FORMAT(last_access_dt, '%H:%i:%s - %W %M %Y')
last_access
FROM emp
WHERE id = '123456789';
```

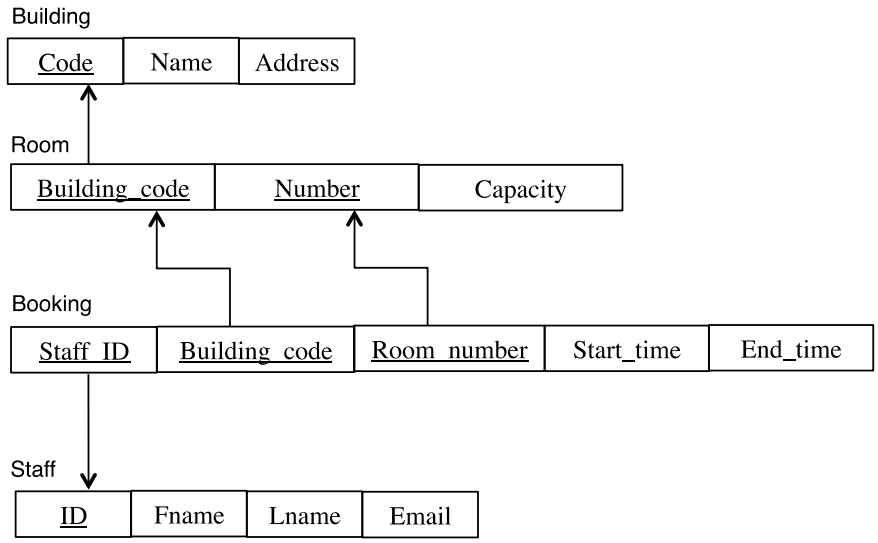
If you want to get the day, month, quarter, year, hour, minute and second of a datetime value, you can use the corresponding function DAY, MONTH, QUARTER, YEAR, HOUR, MINUTE, and SECOND as follows:

```
SELECT DAY(last_access_dt) day,
       MONTH(last_access_dt) month,
       QUARTER(last_access_dt) quarter,
       YEAR(last_access_dt) year,
       HOUR(last_access_dt) hour,
       MINUTE(last_access_dt) minute,
       SECOND(last_access_dt) second
FROM emp
WHERE id = '123456789';
```

Create Tables and Define Constraints

You have already created a few tables and defined columns of different data types. You can also specify constrains in the table definition. Please refer to Lecture 6 for the sytax of the SQL statement for creating tables and defining constraints.
Now consider the following relation schema. Give the SQL statements to create four tables (staff, building, room and booking) in MySQL:

- Determine the most appropriate data type for each attribute.
- Determine the attributes that would be better to have the NOT NULL constraint
- Define the primary keys, candidate keys (UNIQUE constraint) and foreign keys.



Save the scripts for creating these four tables in a file with `.sql` as the file extension. At the beginning of this file, add the DROP TABLE IF EXISTS statements to drop the four tables if they exist. **This is a deliverable to be assessed.**

Load Data into MySQL

You have already practiced two ways of loading data into a table: (1) use SQL INSERT commands interactively, and (2) use a script containing SQL INSERT commands. In this section, you will learn to load data using the LOAD DATA INFILE statement.

Load data from files at the server

If you work on this lab using your own computer, you can download the data files from Blackboard (`Labs->Week3`)

If you work on this lab using a lab computer, the data files are placed on the K: drive in the directory `/COSC344/Week3/`

Open a PowerShell or Command Prompt, run the following command to copy a data file to the docker container (**NOTE**: if you use your own computer, you need to change `K:/COSC344/Week3/building.txt` to the path on your computer where you place the building.txt)

```
docker cp K:/COSC344/Week3/building.txt database:/home/databaselabs
```

Connect to the docker container using `docker attach database`. Change to the `databaselabs` directory using the command `cd /home/databaselabs` to check if the file has been copied to the `databaselabs` directory. Use vim to create a file named `loaddata.sql` that contains the following commands.

```
LOAD DATA LOCAL INFILE 'building.txt'
INTO TABLE building
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
```

From the `/home/databaselabs` directory, connect to MySQL server using the `mysql` command and execute the following command **at the MySQL prompt**

```
source loaddata.sql
```

If there is any error in data loading, fix it based on the error message; otherwise continue loading data from the other three files. If you get stuck, ask the demonstrators for help.

Load data from files from MySQL Workbench

You can also load data from files stored in the host computer from MySQL Workbench without copying the files to the container, but you need to configure the MySQL Workbench to enable local data loading: from the MySQL Workbench menu, click `Database` and then select `Manage Connections ...`. In the pop-up configuration window, go to the `Advanced` sub-tab, and in the `Others:` box add the line `OPT_LOCAL_INFILE=1`. Save the change by closing the window.

Close any existing connections and open a new connection. In the query tab add the following commands (Again, if you use your own computer, you need to change `K:/COSC344/Week3/building.txt` to the path on your computer where you place the building.txt.)

```
LOAD DATA LOCAL INFILE 'K:/COSC344/Week3/building.txt'
INTO TABLE building
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
```

Note that the path after `INFILE` is the path of the file containing the data to be loaded. You need to use forward slash (/) instead of backward slash (\) in the path.

Clean up

Before you quit this lab, drop the five tables you have created: `emp, building, staff, room and booking`.

Ambili AN (MBusDataSc) COSC 344 Lab for Week 5

1. Using a subquery, list the name of the customer who placed the largest order and its Amount?

```
1      # QUESTION 1
2      SELECT cname, amt FROM orders
3      JOIN customers
4      ON orders.cnum = customers.cnum
5      WHERE orders.amt = (
6      SELECT MAX(orders.amt) FROM orders);
7
```

esult Grid | Filter Rows: | Export:

cname	amt
Clemens	9891.88

2.Using a subquery, list the data from the orders table where the amount exceeds the average amount of the orders placed on 1990-10-03.

```
9      SELECT onum, amt, DATE_FORMAT(odate,'%d-%b-%y') AS odate,cnum, snum FROM orders
10     WHERE orders.amt > (
11     SELECT AVG(orders.amt)
12     FROM orders
13     WHERE odate = '1990-10-03' );
14
15     # Question 3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

onum	amt	odate	cnum	snum
3002	1900.10	03-Oct-90	2007	1004
3005	5160.45	03-Oct-90	2003	1002
3008	4723.00	05-Oct-90	2006	1001
3011	9891.88	06-Oct-90	2006	1001

3. Using a subquery, list the data from the orders table for orders attributed to salespersons living in London.

```
15     # Question 3
16     SELECT * FROM orders
17     WHERE orders.snum IN (
18     SELECT snum FROM salespeople
19     WHERE city = 'London');
20
21     # QUESTION 4
```

esult Grid | Filter Rows:

onum	amt	odate	cnum	snum
3002	1900.10	1990-10-03	2007	1004
3003	767.19	1990-10-03	2001	1001
3008	4723.00	1990-10-05	2006	1001
3011	9891.88	1990-10-06	2006	1001

4. Using a correlated subquery, list the names and numbers of all salespeople with more than one customer.

```
21      # QUESTION 4
22 •    SELECT snum, sname from salespeople s
23      WHERE (SELECT COUNT(*) FROM customers c WHERE c.snum = s.snum) > 1
24      GROUP BY s.snum;
25
26      # QUESTION 5
```

Result Grid | Filter Rows: | Edit: | Export/Import:

snum	sname
1001	Peel
1002	Serres

5. Using a correlated subquery that correlates the orders table with itself, list an order only if the amount for this order is above the average amount for all orders placed by the customer of that order. Look at the output to help understand what is desired.

```
26      # QUESTION 5
27 •    SELECT * FROM orders AS o
28      WHERE amt > (
29      SELECT AVG(amt) FROM orders
30      WHERE cnum = o.cnum);
31
```

Result Grid | Filter Rows: | Edi

onum	amt	odate	cnum	snum
3006	1098.16	1990-10-03	2008	1007
3010	1309.95	1990-10-06	2004	1002
3011	9891.88	1990-10-06	2006	1001

6. Use a correlated subquery and EXISTS to find the employees who have no dependents.

```
32      # QUESTION 6
33 •    SELECT fname, lname FROM employee AS e
34      WHERE NOT EXISTS (
35      SELECT * FROM dependent
36      WHERE eird= e.ird);
37
```

tesult Grid | Filter Rows: | Export:

fname	lname
Joyce	English
Ramesh	Narayan
James	Borg
Ahmad	Jabbar
Alicia	Zelaya

7. First type the following lines to add two rows into t……

```
38      # QUESTION 7
39 •    INSERT INTO department VALUES ('TempDept', 6, 123456789, '2002-07-18');
40 •    INSERT INTO project VALUES ('TempProject', 50, 'Houston', 6);
41 •    SELECT pno FROM works_on w JOIN employee e ON w.eird = e.ird
42      WHERE e.lname = 'Smith' UNION SELECT pnumber FROM project p
43      JOIN department d ON p.dnum = d.dnumber JOIN
44      employee e ON d.mgrird = e.ird WHERE e.lname = 'Smith';
45
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

pno
1
2
50

8. Give everybody except Borg (he makes too much anyway *J*) a 10% pay raise. After you do this, select lname and salary to verify it worked properly. If not, reload the company script and try again.

```
45      # QUESTION 8
46 •    UPDATE employee e
47      SET salary = salary/1.1
48      WHERE e.lname != 'Borg';
49 •    SELECT lname, salary FROM employee;
```

Result Grid | Filter Rows: | Export:

lname	salary
Smith	30000.00
Wong	40000.00
English	25000.00
Narayan	38000.00
Borg	55000.00
Wallace	43000.00
Jabbar	25000.00
Zelaya	25000.00

9. Create a new table called hou_emp that is similar to the employee table but has only the workers who live in Houston. Create the table with the appropriate columns and data type and populate it using a single SQL command. After the table is created, do a

```
51      # QUESTION 9
52 •    CREATE TABLE hou_emp AS
53      SELECT * FROM employee WHERE
54      address RLIKE 'Houston';
55 •    SELECT fname, lname FROM hou_emp;
```

esult Grid | Filter Rows: | Exp

fname	lname
John	Smith
Franklin	Wong
Joyce	English
James	Borg
Ahmad	Jabbar

10. Create a new table called emp_dep. Its attributes consist of the first and last name of employees, and the names, sex and birthdates of their dependents. Create this table in a single SQL statement by selecting appropriate data from existing tables. After the table is created, do a *SELECT * FROM emp_dep;* to verify your results

```
58 •    CREATE TABLE emp_dep AS
59      SELECT e.fname, e.lname, d.dependent_name, d.sex, d.bdate FROM
60      employee e JOIN dependent d
61      ON e.ird = d.eird;
62 •    SELECT * FROM emp_dep;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

fname	lname	dependent_name	sex	bdate
John	Smith	Alice	F	1988-12-30
John	Smith	Elizabeth	F	1967-05-05
John	Smith	Michael	M	1988-01-04
Franklin	Wong	Alice	F	1986-04-05
Franklin	Wong	Joy	F	1958-05-03
Franklin	Wong	Theodore	M	1983-10-25
Jennifer	Wallace	Abner	M	1942-02-28

8

COSC 344/444
Lab for Week 6

Overview

The purpose of this lab is to become familiar with functions, procedures and triggers.

Assessment

This lab will be assessed. **It is worth 1.5% of your final grade.** Detail on assessment is given at the end of this lab handout.

Procedure/function

Write a function named `TotalSalary(ProjectNo)` to calculate and return the total amount of salary paid to employees who work for `ProjectNo`.

Write a procedure named `CheckHoursW(ProjectNo,Thours)` to check whether an employee works on project `ProjectNo` for less than `Thours` hours. Give a warning message in the following format for each employee working on `ProjectNo` for less than `Thours` hours

```
Warning: first_name last_name works only for ** hours!
```

where `first_name` and `last_name` should be substituted with the name of the employee and `**` should be substituted with the number of hours that employee works on `ProjectNo`.

Create another procedure named `CheckHoursE(ProjectNo,Thours)` by modifying `CheckHoursW(ProjectNo,Thours)` to raise an error message in the following format instead of giving a warning message.

```
Error: first_name last_name works only for ** hours! Procedure
quitted and the other records were not checked.
```

Triggers

A file named `company_trigger.sql` is available both in lab computers (K:`/COSC344/Week6/`) and in Blackboard (`Labs->Week6`). It is an SQL script that creates and populates two tables, *E1* and *D1*. *E1* is a simplified version of `employee` that only has fname, ssn, salary, and dno. *D1* is a different version of `department` that has dname, dnumber, and tot_sal. Tot_sal is the sum of the salaries of all employees assigned to a department.

Create triggers to maintain *tot_sal* consistent with the data in the database. You need to think through all the database actions that might cause it to become inconsistent.

At first, create your triggers in a file and check their operation. Once you think they are correct, insert the trigger code between the CREATE TABLE *E1* statement and the INSERT statements for *E1* in `company_trigger.sql` file. After executing all statements in `company_trigger.sql`, the initial values for *tot_sal* should be:

```
1    55000
4    93000
5   133000
```

Try updating employee's salary, inserting new employees, and deleting employees. Try changing employees between departments. Do your triggers maintain consistency?

Assessment (1.5%)

Your implementation for the function, procedures and triggers will be assessed.

You need to demonstrate at the **MySQL prompt**, i.e., from the Command Prompt, attach the container first and then connect to MySQL server using the `mysql` command.

- Function `TotalSalary(ProjectNo)`
  Executing `SELECT TotalSalary(2);` will generate an output of `95000`.

- Procedure `CheckHoursE(ProjectNo,Thours)`
  Executing `CALL CheckHoursE(2,15);` will generate the following two warning message
  ```
  Warning: John Smith works only for 7.50 hours!
  Warning: Franklin Wong works only for 10.00 hours!
  ```

- Procedure `CheckHoursE(ProjectNo,Thours)`
  Executing `CALL CheckHoursW(2,15);` will generate only one error message
  ```
  ERROR 1644 (45000): Error: John Smith works only for
  7.50 hours.  Procedure quits and other records were
  not checked.
  ```

- Triggers
  At the end of `company_trigger.sql`, add the following statements

  ```
  INSERT INTO e1 VALUES ('Emily','888665785',30000,1);
  SELECT * FROM d1;
  UPDATE e1 SET salary=40000 WHERE ird='888665785';
  SELECT * FROM d1;
  DELETE FROM e1 where ird ='888665785';
  SELECT * FROM d1;
  UPDATE e1 SET dno=1 WHERE ird ='123456789';
  SELECT * FROM d1;
  ```

  At the MYSQL prompt, load company_trigger.sql by executing `source company_trigger.sql`.
  The outputs for the 4 SELECT statements should be

dname	dnumber	tot_sal
Headquarters	1	85000.00
Administration	4	93000.00
Research	5	133000.00

dname	dnumber	tot_sal
Headquarters	1	95000.00
Administration	4	93000.00
Research	5	133000.00

dname	dnumber	tot_sal
Headquarters	1	55000.00
Administration	4	93000.00
Research	5	133000.00

dname	dnumber	tot_sal
Headquarters	1	85000.00
Administration	4	93000.00
Research	5	103000.00

Assessment (1.5%) AMBILI ARANGATH NARAYANAN

Query : 1

- Function `TotalSalary(ProjectNo)`
  Executing `SELECT TotalSalary(2);` will generate an output of `95000`.

Answer :

```
6  • DROP FUNCTION IF EXISTS TotalSalary;
7    DELIMITER ///
8  • CREATE FUNCTION TotalSalary(ProjectNo INT)
9    RETURNS DECIMAL
10   DETERMINISTIC
11   BEGIN
12   DECLARE totalsal DEC DEFAULT 0.0;
13   SELECT SUM(e.salary) INTO totalsal FROM
14   employee e, project p, works_on w
15   WHERE e.ird = w.eird AND w.pno = p.pnumber AND p.pnumber = ProjectNo;
16   RETURN totalsal;
17   END ///
18   DELIMITER ;
19 • SELECT TotalSalary(2);
20
```

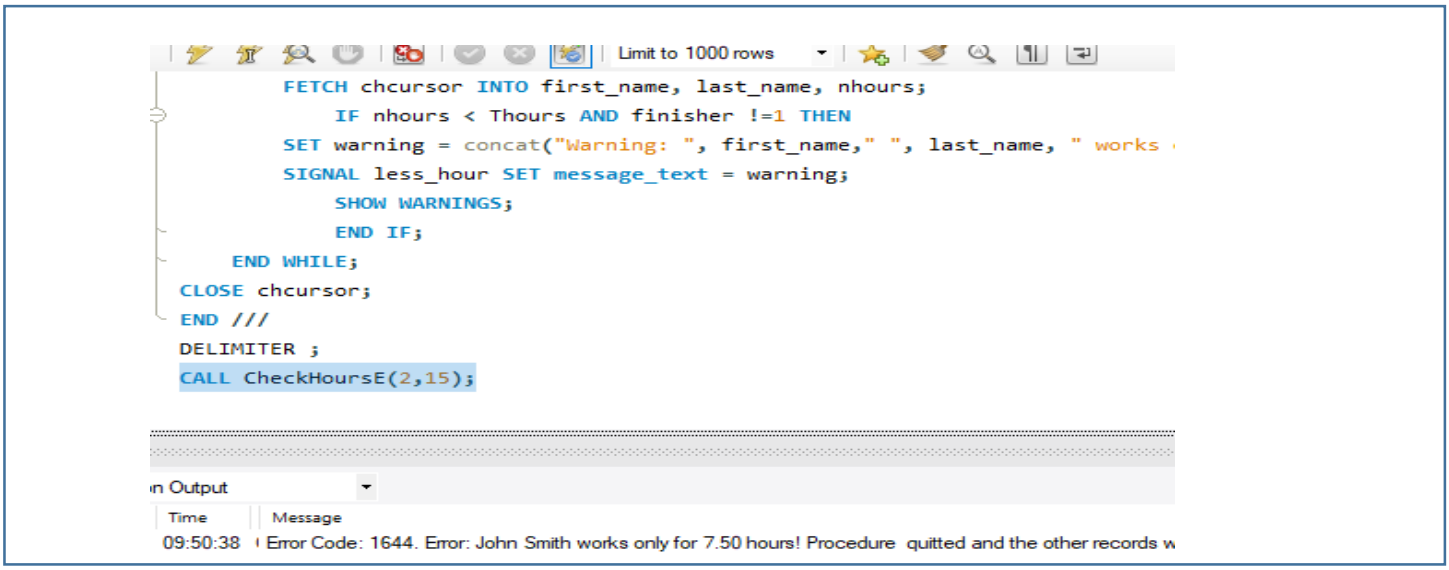
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

TotalSalary(2)
95000

Query :2

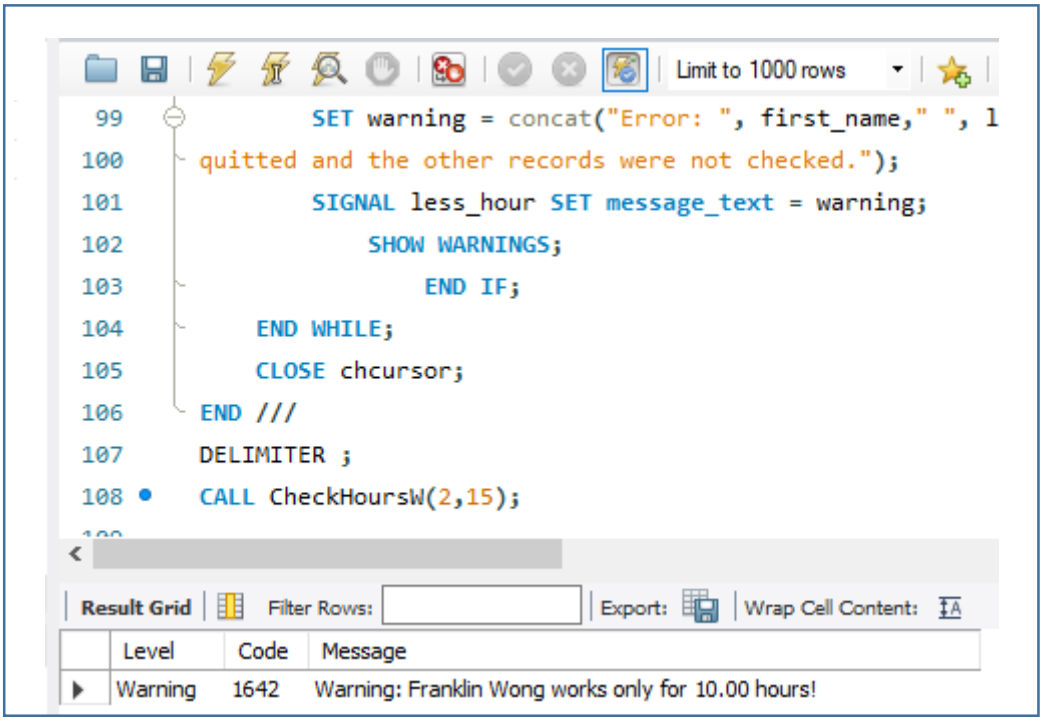
- Procedure `CheckHoursE(ProjectNo,Thours)`
  Executing `CALL CheckHoursE(2,15);` will generate the following two warn
  message

```
Warning: John Smith works only for 7.50 hours!
Warning: Franklin Wong works only for 10.00 hours!
```



Query 3

- Procedure `CheckHoursE(ProjectNo,Thours)`
  Executing `CALL CheckHoursW(2,15);` will generate only one error message
  `ERROR 1644 (45000): Error: John Smith works only for 7.50 hours. Procedure quits and other records were not checked.`



QUERY 4:

- Triggers
  At the end of `company_trigger.sql`, add the following statements

```
INSERT INTO e1 VALUES ('Emily','888665785',3
SELECT * FROM d1;
```

```
44 •  INSERT INTO e1 VALUES ('Emily','888665785',30000,1);
45 •  SELECT * FROM d1;
```

Result Grid | Filter Rows: | Edit: | Export/Ir

dname	dnumber	tot_sal
Headquarters	1	55000.00
Administration	4	93000.00
Research	5	133000.00

```
UPDATE e1 SET salary=40000 WHERE ird='888665785';
SELECT * FROM d1;
```

```
47 •  UPDATE e1 SET salary=40000 WHERE ird='888665785';
48 •  SELECT * FROM d1;
49
```

Result Grid | Filter Rows: | Edit: | Ex

dname	dnumber	tot_sal
Headquarters	1	55000.00
Administration	4	93000.00
Research	5	133000.00

```
DELETE FROM e1 where ird ='888665785';
SELECT * FROM d1;
```

```
53 •  UPDATE e1 SET dno=1 WHERE ird ='123456789';
54 •  SELECT * FROM d1;
```

Result Grid | Filter Rows: | Edit:

dname	dnumber	tot_sal
Headquarters	1	85000.00
Administration	4	93000.00
Research	5	103000.00

```
UPDATE e1 SET dno=1 WHERE ird ='123456789';
SELECT * FROM d1;
```

```
50 •  DELETE FROM e1 where ird ='888665785';
51 •  SELECT * FROM d1;
52
```

Result Grid | Filter Rows: | Edit:

dname	dnumber	tot_sal
Headquarters	1	55000.00
Administration	4	93000.00
Research	5	133000.00

COSC 344 & 444
Lab for Week 7

Overview

The purpose of this lab is to learn how to develop a Java application to interact with MySQL.

Assessment

This lab will be assessed. **It is worth 1.5% of your final grade**. Detail on assessment is given at the end of this lab handout.

1. Set up the Java environment

If you are using your own computer, you need to install

- JDK - https://www.oracle.com/java/technologies/downloads/
- Visual Studio (VS) Code - https://code.visualstudio.com/download

Make sure the Java environmental variables have been correctly set up.

If you are using a lab computer, you need to set up the Java environment variables **for your account** first. Search `Environment` in the Search bar and click `Edit environment variables for your account.` Note that you **can't** choose `Edit environment variables` since this will require the administrator privilege to make changes.

In the pop-up window, choose the `Path` variable and click the `Edit` button. Click the `New` button to create a new variable and then click the `Browse` button. In the `pop-up Browse For Folder` window, go to the Java bin folder as follows: `This PC ->BOOTCAMP(C:)->Program Files (x86)->jGRASP->bundled->java->bin.` Click the `ok` buttons to close all windows.

Open VS Code, click `View` in the Menu and select `Extensions`. Search `Java` in the text box and select `Extension pack for Java` and then install it. This extension pack contains popular extensions that can help write, test and debug java programs in VS code.

If you haven't used VS code before, follow the steps starting from Step 3 in this tutorial (https://dev.to/realedwintorres/tutorial-visual-studio-code-and-java-icm) to create a Java project.

2. Test database connection from a Java program

MySQL Connector/J is the official JDBC driver for MySQL. You need to add it to your project as follows so that you can connect to MySQL server from your Java program.

If you are using your own computer, you can download it from Blackboard (`Labs->Week7`). If you are using a lab computer, open `File Explorer` in your computer and navigate to `K:/COSC344/Week7/MySQL JDBC connector`. Drag the `mysql-connector-j-8.0.32.jar` file to the `libs` folder in your VS code project explorer.

Create a file named `pass.dat`. Put your MySQL username and password in the file on separate lines. Do not have any spaces after the data. Save it in your project folder (**not in** any sub-directory of your project folder).

If you are using a lab computer, navigate to `K:/COSC344/Week7/`, and then drag `UserPass.java` and `TestLogin.java` to the `src` folder in your VS code project explorer. `UserPass.java` is a Java program designed to read username and password from `pass.dat`. `TestLogin.java` is a simple Java program to test the connection to MySQL from a Java program. `UserPass.java` and `TestLogin.java` are also available in Blackboard (`Labs->Week7`).

Start the docker container and make sure MySQL is running in the docker container. Compile and run `TestLogin`. It should respond that it connected to MySQL server, and then the program terminates.

If you encounter the error message "`No Suitable Driver Found For JDBC`", it is likely due to an incompatibility between the version of the JDK you installed and the JDBC driver. A simple solution is to register the JDBC driver using `Class.forName()`. A function named `LoadDriver()` has been added to `TestLogin.java` to resolve this issue. Please reload `TestLogin.java` from `K:/COSC344/Week7/` or Blackboard (Labs -> Week 7). For more information on this issue, please refer to Chapter 7.1 of the MySQL 8.0 reference manual.

3. Interact MySQL from a Java program

The Java programs I presented during lectures as examples for retrieving and updating data in a table are accessible on Blackboard (Lectures -> Lecture 11).

Based on these sample programs, implement a Java program that provides the following functions (Note that some code in the sample programs can be directly reused here).

- Once the Java program runs, it should print the following menu for user to select a task. Once the execution of a task is completed, it should print the menu again for user to select the next task unless the user chooses to quit the app.

```
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
```

- For Task 1, the program should ask the user to input the IRD for an employee. This task will retrieve the information for that employee from the employee table and display it.

- For Task 2, the program should ask the user to input the IRD and the updated salary for an employee. It should display whether salary has been updated successfully or not.

- For Task 3, the program should ask the user to input the IRD for an employee and then delete or modify related information in the company database (e.g. delete related rows in the employee, dependents and works_on tables, set NULL to columns where that employee is a supervisor or manager). It should output what has been changed in the database.

- For Task 4, the program should ask the user to input the IRD for an employee and then generate a report that includes the following information:
  - The basic information for the employee such as name, IRD, Gender, and working department

- The name of each project the employee works on and the number of hours the employee works on each project.

Here is a sample output for Task 4:

```
This program provides the following functions:
 1. Show employee information
 2. Update employee salary
 3. Delete an employee
 4. Generate a report for an employee
 5. Quit the app

Select task: 4
Enter the employee IRD: 123456789

----Employee basic info--------
Name: John      B       Smith
IRD: 123456789
Gender: M
Department: Research
------------------------------
----Projects worked on--------
ProductX        32
ProductY        7
```

Assessment (1.5%)

Your implementation for the java program in Section 3 will be assessed.

COSC 344 & 444 Lab for Week 7

For Task 1, the program should ask the user to input the IRD for an employee. This task will retrieve the information for that employee from the employee table and display it.

Solution

```
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5: 1
Enter the Employee IRD: 888665555
----Employee basic info------
Name: James E Borg
IRD: 888665555
Birthday: Wednesday, 10 November 1937
Address: 450 Stone, Houston, TX
Sex: M
Salary: 55000.00
Superird: NULL
Department No: 1
```

For Task 2, the program should ask the user to input the IRD and the updated salary for an employee. It should display whether salary has been updated successfully or not.

Solution

```
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5: 2
Enter IRD Number of Employee: 987654321
New Salary: 50000
Employee with IRD number: 987654321 is successfully updated with new Salary: 50000.00
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5:
```

```
mysql> select * from employee;
+----------+-------+---------+-----------+------------+-------------------------+------+----------+-----------+-----+
| fname    | minit | lname   | ird       | bdate      | address                 | sex  | salary   | superird  | dno |
+----------+-------+---------+-----------+------------+-------------------------+------+----------+-----------+-----+
| Franklin | T     | Wong    | 333445555 | 1955-12-08 | 638 Voss, Houston, TX   | M    | 40000.00 | 888665555 |   5 |
| Joyce    | A     | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX  | F    | 25000.00 | NULL      |   5 |
| Ramesh   | K     | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX| M    | 38000.00 | NULL      |   5 |
| James    | E     | Borg    | 888665555 | 1937-11-10 | 450 Stone, Houston, TX  | M    | 55000.00 | NULL      |   1 |
| Jennifer | S     | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | F    | 50000.00 | 888665555 |   4 |
| Ahmad    | V     | Jabbar  | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | M    | 25000.00 | 987654321 |   4 |
| Alicia   | J     | Zelaya  | 999887777 | 1968-07-19 | 3321 Castle, Spring,TX  | F    | 25000.00 | 987654321 |   4 |
+----------+-------+---------+-----------+------------+-------------------------+------+----------+-----------+-----+
7 rows in set (0.02 sec)
```

For Task 3, the program should ask the user to input the IRD for an employee and then delete or modify related information in the company database (e.g. delete related rows in the employee, dependents and works_on tables, set NULL to columns where that employee is a supervisor or manager). It should output what has been changed in the database.

Solution

```
5. Quit the app
Please select any value between 1 and 5: 3
Enter the Employee IRD: 999887777
Deleted 2 records from works_on table with IRD number: 999887777
Deleted 0 records from dependent table with IRD number: 999887777
Setting superird field to null in the employee table if the deleted employee is a manager. Updated 0 records in th
e employee table
Setting mgrird field to null in the employee table if the deleted employee is a manager. Updated 0 records in the
department table
Deleted the employee with IRD number: 999887777 from the employee table
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5:
```

For Task 4, the program should ask the user to input the IRD for an employee and then generate a report that includes the following information:
a) **The basic information for the employee such as name, IRD, Gender, and working Department COSC344/444 Lab Page 3 Week 7**
b) **The name of each project the employee works on and the number of hours the employee works on each project.**

Solution :

```
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5: 4
Enter the Employee IRD: 987987987
Name: Ahmad V Jabbar
IRD: 987987987
Gender: M
Department: Administration
PROJECT NAME   HOURS
Computerisation   35.0
Newbenefits   30.0
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5:
```

```
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5: 4
Enter the Employee IRD: 333445555
Name: Franklin T Wong
IRD: 333445555
Gender: M
Department: Research
No projects to display for this employee
Employee App
1. Show employee information
2. Update employee salary
3. Delete an employee
4. Generate a report for an employee
5. Quit the app
Please select any value between 1 and 5:
```

9

COSC344 & COSC444
Lab for Week 9

Overview

The purpose of this lab is to become familiar with MySQL security, privileges and roles.

Assessment

This lab will be assessed. **It is worth 1.5% of your final grade**. Detail on assessment is given at the end of this lab handout.

Account and privilege Management

You can use `GRANT` and `REVOKE` to assign privileges to and revoke privileges from accounts.

The following examples are provided **for you to understand** how to use `GRANT` and `REVOKE` (**Don't execute** them now as the user `bob` hasn't been created yet)

```
# allow bob to retrieve data from the employee table
GRANT SELECT ON employee TO 'bob';

# allow bob to update salary for the employee table
GRANT UPDATE(salary) ON employee TO 'bob';

# allow bob to further grant the SELECT privilege on employee
to other users.
GRANT SELECT ON employee TO 'bob' WITH GRANT OPTION;

# revoke the select privilege on employee from bob
REVOKE SELECT ON employee FROM 'bob';
```

There are more examples in Section 6.2.8 of the MySQL 8.0 reference manual on granting and revoking privileges.

Now do the following practices:

1. Create a user account named `bob`.
2. Grant the `SELECT` privilege on the employee table to `bob`, and use `show grants for bob;` to check the privileges granted to `bob`.
3. Connect to MySQL using the user account `bob`, and check if you are able to access the `employee` table and other tables.
4. Grant `bob` the privilege on updating only the `salary` column in the `employee` table. Connect to MySQL with `bob`, and check if you are able to update the salary for an employee.
5. Revoke some privileges you have granted to `bob` and check if you can still retrieve or update with `bob`.
6. Allow `bob` to further grant the `SELECT` privilege on the `employee` table to another user.

Instead of granting privileges to users individually, MySQL allows you to create a role that is made up a number of privileges, and grant the role to a user. The following statements first create a role named `sel_emp_proj` that allows to retrieve the data on employee and the projects they work on, and then grant this role to `bob`.

```
CREATE ROLE sel_emp_proj;
GRANT SELECT ON employee TO sel_emp_proj;
GRANT SELECT ON project TO sel_emp_proj;
GRANT SELECT ON works_on TO sel_emp_proj;
GRANT sel_emp_proj TO 'bob';
```

There are more examples in Section 6.2.10 of the MySQL 8.0 reference manual on how to create, grant and revoke roles.

Exercises:

A file named `security.sql` is available both in lab computers (K:`/COSC344/Week9/`) and in Blackboard (`Labs->Week9`). This file contains the scripts to create an emp and dept tables which are simpler versions of the employee and department tables we have used before. The relations are shown below.

```
EMP (fname, lname, ssn, salary, dno)

DEPT (dname, dnumber)
```

The data is the same except some salaries have been changed.

Open `security.sql`. Copy all scripts in it to MySQL Workbench and execute them. From now on, we will refer to MySQL Workbench as *you*.

Open a command prompt window, attach to the container and connect to MySQL server using the account `bob` you created before. From now on, we refer to this window as *your friend*.

For each step below, work out the appropriate SQL statements on your own.

You will be alternating between doing work as *you* (MySQL workbench) and as *your friend bob (*docker bash window). Alternatively, you can also pair up with a real friend and work together. Just decide which of you will be *you* and which will be *the friend*. ☺

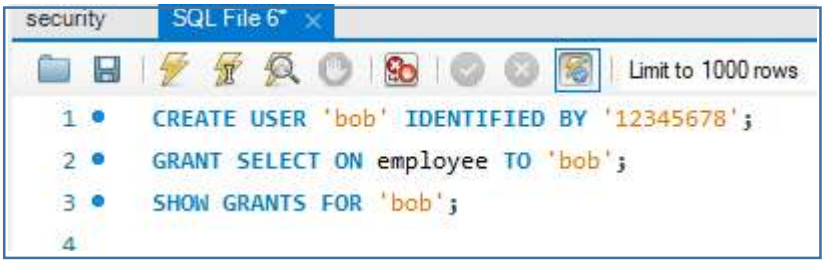
PART	YOU	YOUR FRIEND	COMMENTS
1	Grant your friend privilege to SELECT data from the *emp* table.		
		Test that you can SELECT data from *emp*.	NOTE-You must now preceed the table names with the username. For example, SELECT * FROM emp;
2	Let your friend update *dname* in *dept.*		
		Update the name for department 5 to "Engineering". Test to make sure you cannot update *dnumber* or anything in the *emp* table.	NOTE - You need to grant the select privilege due to the need to search the "Engineering" department.
3	Deny your friend access to the *emp* and *dept* tables.		
		Test that you can no longer access the tables.	
4	Create a role allowing anyone with the role to select data from the *emp* and *dept* tables. Then grant the role to your friend.		
		Test that you can SELECT from the *emp* and *dept* tables.	NOTE - Your friend must set the current role to the role granted by you using SET ROLE role_name;
5	Figure out a way to let your friend easily see the names of employees and the name of their department. Implement it.		HINT-Think about views.
		Test that your solution works.	
6	Figure out a way to let your friend to see the names of the employees and their salaries, but only if the employee makes no more than 50,000.		
		Test that your solution works.	

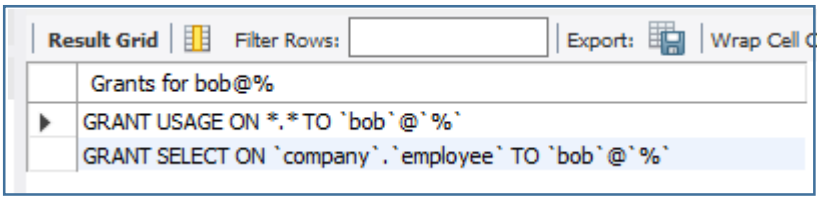
Assessment (1.5%)

Write down the SQL statement of each task for both you and your friend for the demonstrator to check.

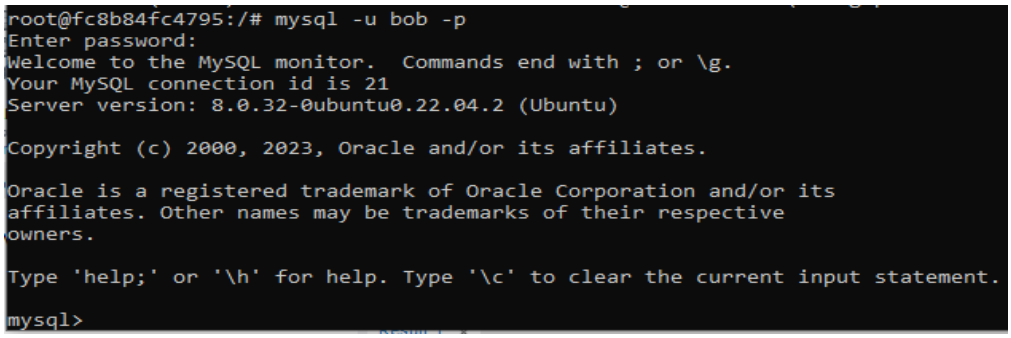
LAB 9 - AMBILI ARANGATH NARAYANAN
Now do the following practices:

1. Create a user account named bob.
2. Grant the SELECT privilege on the employee table to bob, and use show grants for bob; to check the privileges granted to bob.





3. Connect to MySQL using the user account bob, and check if you are able to access the employee table and other tables.



```
4 •  SELECT * FROM employee;
5
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

fname	minit	lname	ird	bdate	address	sex	salary	superird	dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000.00	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000.00	888665555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000.00	333445555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000.00	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000.00	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000.00	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000.00	987654321	4
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring,TX	F	25000.00	987654321	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Grant bob the privilege on updating only the salary column in the employee table. Connect to MySQL with bob, and check if you are able to update the salary for an employee.

5. Revoke some privileges you have granted to bob and check if you can still retrieve or update with bob.

6. Allow bob to further grant the SELECT privilege on the employee table to another user.

```
GRANT UPDATE(salary) ON employee TO 'bob'
UPDATE emp SET salary = 40000 WHERE ssn = '123456789';
REVOKE SELECT ON emp FROM 'bob'
GRANT SELECT ON emp TO 'bob' WITH GRANT OPTION;
```

##

PART 1

YOU : Grant your friend privilege to SELECT data from the emp table.

```
22 •  CREATE USER bob IDENTIFIED BY 'iloveu';
23 •  GRANT SELECT ON emp TO bob;
24 •  SHOW GRANTS FOR bob;
25
26
```

Result Grid | Filter Rows: | Export: | Wra

Grants for bob@%
GRANT USAGE ON *.* TO `bob`@`%`
GRANT SELECT ON `company`.`emp` TO `bob`@`%`

FRIEND (BOB) : Test that you can SELECT data from emp

```
26 •  SELECT * FROM emp;
```

Result Grid | Filter Rows:

fname	lname	ssn	salary	dno
John	Smith	123456789	40000	5
Franklin	Wong	333445555	60000	5
Joyce	English	453453453	25000	5
Ramesh	Narayan	666884444	38000	5
James	Borg	888665555	55000	1
Jennifer	Wallace	987654321	73000	4
Ahmad	Jabbar	987987987	25000	4
Alicia	Zelaya	999887777	25000	4
NULL	NULL	NULL	NULL	NULL

PART 2

YOU - Let your friend update dname in dept.

```
27    ####PART - 2 #######
28 •  GRANT SELECT ON dept TO bob;
29 •  GRANT UPDATE(dname) ON dept TO bob;
30 •  SHOW GRANTS FOR bob;
31
```

Result Grid | Filter Rows: | Export: | Wrap Cell Co

Grants for bob@%
GRANT USAGE ON *.* TO `bob`@`%`
GRANT SELECT, UPDATE (`dname`) ON `company`.`dept` TO `bob`@`%`
GRANT SELECT ON `company`.`emp` TO `bob`@`%`

FRIEND (BOB) : Update the name for department 5 to "Engineering". est to make sure you cannot update dnumber or anything in the emp table.

```
32 •  UPDATE dept SET dname = "Engineering" WHERE dnumber = 5;
33 •  UPDATE dept SET dnumber = 5 WHERE dname = "Engineering";
34 •  SELECT * FROM dept;
35
```

Result Grid | Filter Rows: | Edit: | Export/Impor

dname	dnumber
Administration	4
Dummies	0
Engineering	5
Headquarters	1
NULL	NULL

PART 3

YOU - Let your friend update dname in dept

```
REVOKE SELECT ON emp FROM bob;
REVOKE SELECT(dname) ON dept FROM bob;
REVOKE UPDATE(dname) ON dept FROM bob;
```

PART 4

Create a role allowing anyone with the role to select data from the emp and dept tables. Then grant the role to your friend

```
25      #### PART - 4 #######
26
27      CREATE ROLE role_for_emp_dept;
28      GRANT SELECT ON emp TO role_for_emp_dept;
29      GRANT SELECT ON dept TO role_for_emp_dept;
30      GRANT role_for_emp_dept TO 'bob';
31      SET ROLE role_for_emp_dept;
32
```

FRIEND (BOB) : Test that you can SELECT from the emp and dept tables.

```
33      SELECT * FROM emp;
34
```

Result Grid | Filter Rows:

fname	lname	ssn	salary	dno
John	Smith	123456789	30000	5
Franklin	Wong	333445555	60000	5
Joyce	English	453453453	25000	5
Ramesh	Narayan	666884444	38000	5
James	Borg	888665555	55000	1
Jennifer	Wallace	987654321	73000	4
Ahmad	Jabbar	987987987	25000	4
Alicia	Zelaya	999887777	25000	4

```
35      SELECT * FROM dept;
36
```

Result Grid | Filter Rows:

dname	dnumber
Administration	4
Dummies	0
Engineering	5
Headquarters	1

PART 5

Figure out a way to let your friend easily see the names of employees and the name of their department. Implement it.

```
34      #### PART - 5 #######
35
36 •    CREATE VIEW view1_emp_dept AS
37      SELECT fname, lname, dname
38      FROM emp e, dept d
39      WHERE e.dno = d.dnumber;
```

FRIEND (BOB) : Test that your solution works.

```
41 •    SELECT * FROM view1_emp_dept
```

Result Grid | Filter Rows:

fname	lname	dname
Jennifer	Wallace	Administration
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration
John	Smith	Engineering
Franklin	Wong	Engineering
Joyce	English	Engineering
Ramesh	Narayan	Engineering
James	Borg	Headquarters

PART 5

YOU: Figure out a way to let your friend easily see the names of employees and the name of their department.

```
34      #### PART - 5 #######
35
36 •    CREATE VIEW view1_emp_dept AS
37      SELECT fname, lname, dname
38      FROM emp e, dept d
39      WHERE e.dno = d.dnumber;
```

FRIENDS (BOB): Test that your solution works

```
41 •    SELECT * FROM view1_emp_dept
```

Result Grid | Filter Rows:

fname	lname	dname
Jennifer	Wallace	Administration
Ahmad	Jabbar	Administration
Alicia	Zelaya	Administration
John	Smith	Engineering
Franklin	Wong	Engineering
Joyce	English	Engineering
Ramesh	Narayan	Engineering
James	Borg	Headquarters

PART 6

YOU: Figure out a way to let your friend to see the names of the employees and their salaries, but only if the employee makes no more than 50,000.

```
#### PART - 6 #######

CREATE VIEW view2_emp_dept AS
SELECT fname, lname, salary
FROM emp
WHERE salary <= 50000;
```

FRIENDS (BOB): Test that your solution works

```
51 •    SELECT * FROM view2_emp_dept;
```

Result Grid | Filter Rows:

fname	lname	salary
John	Smith	30000
Joyce	English	25000
Ramesh	Narayan	38000
Ahmad	Jabbar	25000
Alicia	Zelaya	25000

10

INFO 408 Assignment 2:
Large-scale database implementation

DUE DATE: Monday 25 September 2023 at 5pm

The goal of this project is design and build a database to store a real big data dataset, using an appropriate database management system (DBMS). You will choose a dataset and target DBMS, develop a database design, implement it in the chosen DBMS, and document the design and implementation. This assignment counts for **20%** of your final grade in INFO 408.

1 Choose a dataset

You should choose a dataset that you find interesting, and that meets the following criteria:

- It is freely available online.
- It contains between 100 MB and 2 GB of data. (There is some flexibility at the high end, but you may be able to download a subset of the dataset if it is too large.)
- It has at least some internal complexity to make it more "interesting" (e.g., it isn't just a huge table of numbers).
- It isn't already formatted for use in your chosen DBMS (e.g., a MongoDB export file when your chosen DBMS is MongoDB). In other words, the dataset will require at least some transformation to load it into your database.

If you aren't sure, please ask. Here are some links to resources where you may be able to find useful datasets (in no particular order):

- Kaggle
- Wikipedia: Category:Open data projects and Research:Wikipedia clickstream
- Registry of Open Data on AWS
- Google: Dataset Search and Google Cloud Public Datasets (it isn't clear whether the latter can be downloaded separately or can only be used within Google Cloud)
- UC Irvine Machine Learning Repository
- data.world Open Data Community (requires registration)

2 Choose a DBMS

Once you have chosen a dataset, you need to choose a DBMS to store it in. You should choose a DBMS that complements the dataset's natural structure, which won't necessarily be the same

as the structure in which the dataset is provided. For example, the dataset might be provided as CSV files or SQL tables but actually represent a network of related elements, implying that a graph-based DBMS could be a good solution. Similarly, if the dataset comprises mostly semi-structured text, a document-based DBMS might be more suitable.

> Remember that this project is about *designing and building a database* to hold the dataset, *not* analysing it. **You therefore aren't expected or required to do any analysis of the dataset**.
>
> However, you might find it helpful to have a rough outline of an analysis use case in mind to inform your choices, both when choosing the DBMS and designing the database structure (Section 3).

You are expected to justify your choice of DBMS, so think carefully about what features and capabilities you might need. The DBMS can be any that you have used in INFO 408 or other papers (SQL or NoSQL), or one that you haven't used before. Look for Docker images on Docker Hub to help you install, configure, and manage your chosen DBMS. Use official images where available (images by Bitnami are also often quite good). If you aren't sure, please ask.

3 Design, build, and populate the database

The final task is to design and build a database for your dataset, using your chosen DBMS, then import your dataset into it. The amount of "design" required will vary depending on your choice of dataset and DBMS. For example, MongoDB essentially has no schema, but you can still discuss "schema-like" aspects of how you have structured collections, etc. An SQL DBMS in contrast will require you to write `create table` statements.

You may use any method you feel is appropriate to import your dataset into the final database. This could include import tools provided by the DBMS, or a custom program that you write.

4 Final deliverables

Write a short report that documents your database design and implementation at a *technical* level (i.e., you don't need to write user-level documentation). As a rough guideline, aim for perhaps 5–10 pages, excluding references and appendices. This is not a hard limit, but if you find yourself writing more than 20 pages, you are most probably overdoing things ☺. *Where applicable*, your report should include the following:

- An overview of the dataset, highlighting any special requirements or unusual technical aspects that it might have (especially if these affected the choice of DBMS).
- A justification of your choice of DBMS, with reference to the requirements and nature of your dataset (including any analysis use cases you used to guide your choice, if applicable).
- A discussion of your database implementation, including details of its "schema" (see Section 3 above). Explain the reasoning behind your design decisions (including any analysis use cases you used to guide your design, if applicable). If you have schema code or a schema diagram, include these as appendices.

- A discussion of how you imported the dataset into the database, highlighting the tools used, and any problems that arose and how you solved them.
- Anything else that you think is relevant! (For example, if you choose a DBMS that wasn't used in INFO 408, you may need to briefly explain how its model works.)

Submit your project report through Blackboard as a **PDF**, by the deadline of **Monday 25 Sept 2023 at 5pm**. You are free to structure the report as you see fit within the broad guidelines stated above.

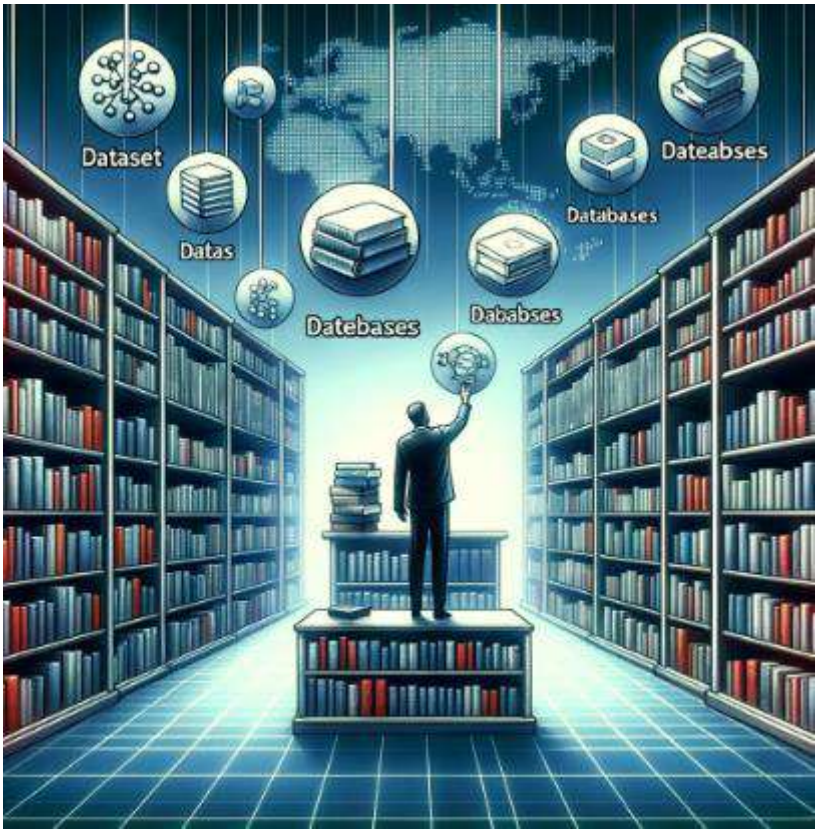
There may also be an opportunity to briefly demonstrate your database implementation to the teaching staff, but this is entirely optional and won't count towards your mark. It's just nice to see things working ☺.

This is quite an open-ended assignment, so you should be careful not to overthink things or get too carried away ☺. You are welcome to discuss any aspects of your project before you submit.

I. Introduction

- Flight 2022' dataset is vast at 4 GB, promising deep insights and exploration.
- The dataset boasts diverse data types: categorical, numerical, and date-time.
- MongoDB offers dynamic schema and expert handling of semi-structured data.
- Scalability and flexibility in MongoDB ensure seamless Big Data management.
- Challenges include maintaining data consistency and handling intricate queries.
- Strategic schema design and validation ensure data integrity and reliability.

II. Dataset & Database Selection



2.1 Unveiling the Logic Behind the Dataset Selection of Flight 2022 CSV

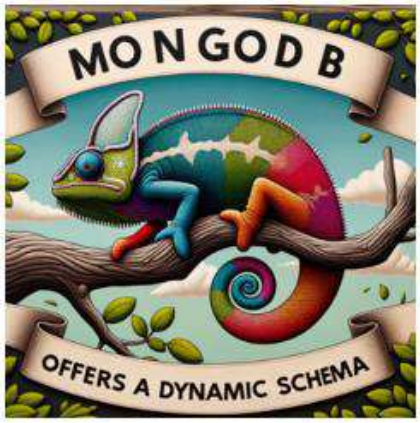
- **Relevance and Timeliness:** The 2022 dataset offers the latest insights into airline operations, capturing modern trends, challenges, and market dynamics.
- **Richness of Data:** With a diverse range of variables, from flight dates to airport details, this dataset can support comprehensive analyses.

- **Dataset Source, Size, and Volume:** Acquired from Kaggle, a trusted data science platform, the dataset is of significant size, specifically 1.3 GB. This dataset, comprising over 4 lakh rows, ensures a detailed view of the airline industry.
  https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022
- **Historical Context:** It combines flight details from 2018 to 2022, offering a valuable historical perspective suitable for trend analysis.
- **Data Transformation Objective:** Choosing the 2022 flight data facilitates the conversion of raw data into a structured database. This transformation into various collections further aids detailed data introspection.

2.2 Unveiling the Logic Behind the Database Selection

- The "Flight 2022" dataset, a colossal reservoir of intricate data, presented a journey that demanded a robust and adept DBMS to navigate its complexities.
- MongoDB, with its stellar capabilities in managing semi-structured data and its dynamic schema, emerged as the chosen navigator, ensuring that the exploration of data was not only seamless but also strategically aligned with the multifaceted nature of the dataset.

2.2.1 Embracing Schema Flexibility:

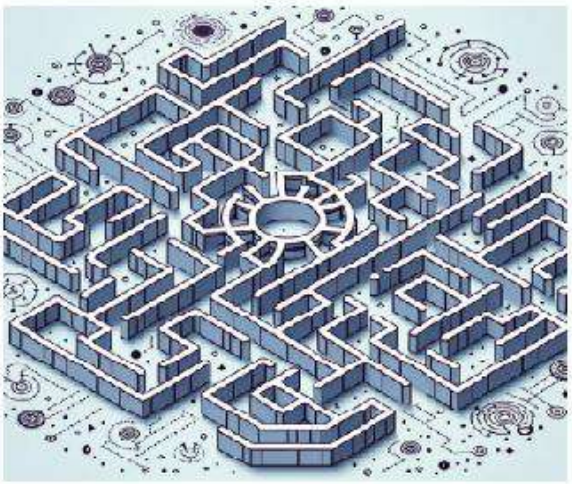


- ✓ MongoDB offers a dynamic schema, eliminating the need for a fixed, predefined structure.
- ✓ This flexibility allows easy adaptation to changing data structures without major modifications.
- ✓ The evolving nature of the Flight 2022 dataset requires a DBMS that can accommodate emerging attributes seamlessly.

2.2.2 Scalability:

- ✓ MongoDB supports horizontal scaling, distributing data across multiple servers. It can adapt to growth, efficiently managing continuous data influx like new flight data.
- ✓ Sharding in MongoDB accommodates large datasets and high-throughput operations. By partitioning data, MongoDB enhances performance and manages vast data expansions.
- ✓ MongoDB's architecture ensures efficient scaling by adding machines to the cluster.

2.2.3 Complex Data Relationships



- ✓ The Flight 2022 dataset has complex associations among flights, airlines, and delays; MongoDB capably navigates this.
- ✓ MongoDB uses embedded documents and references to manage intricate data relationships.
- ✓ MongoDB adeptly represents flight-airline and delay-airport correlations in the dataset.
- ✓ MongoDB optimizes query performance and ensures precise, efficient data retrieval from the dataset.

2.2.4 Semi-Structured Data Handling

- ✓ MongoDB expertly manages the Flight 2022 dataset using BSON for enhanced data types and efficient access.
- ✓ BSON ensures rapid scan rates in MongoDB, optimizing performance.
- ✓ The Flight 2022 dataset, rich in flight and airline details, is effectively handled by MongoDB.

2.2.5 Document-Oriented

- ✓ MongoDB adopts a document-oriented approach, differing from traditional relational databases using rows and tables.
- ✓ This approach is powered by BSON, an enhanced JSON version, ensuring efficient data management.
- ✓ BSON efficiently represents hierarchical relationships and supports arrays within the data.
- ✓ For the Flight 2022 dataset, MongoDB integrates diverse data with ease using this approach.
- ✓ The data, including flight details and delays, aligns naturally with JSON-like document structures.

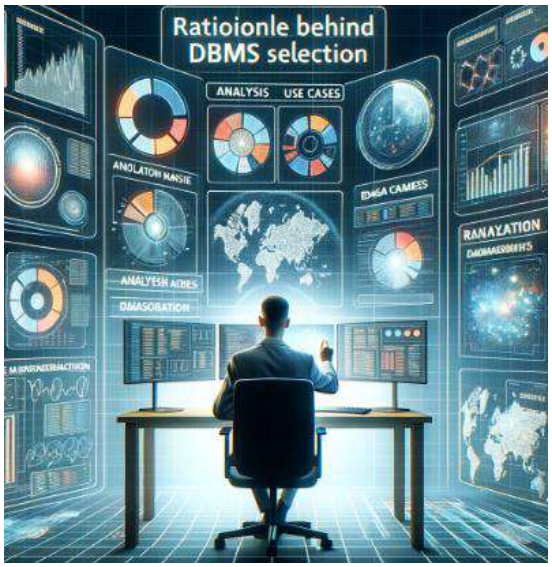
2.2.6 Performance

- ✓ MongoDB boasts advanced indexing, optimizing data access speed and efficiency.
- ✓ Sharding in MongoDB enhances read/write speeds and overall performance.
- ✓ Large datasets like Flight 2022 benefit from MongoDB's scalable and high-performance handling.
- ✓ The in-memory storage engine boosts real-time analytics and data access operations.
- ✓ With MongoDB, data retrieval, whether specific or broad, is fast, enhancing user experience.
- ✓ Compared to DBMS like MySQL, MongoDB offers superior schema flexibility and scalability.

2.5 Other Features

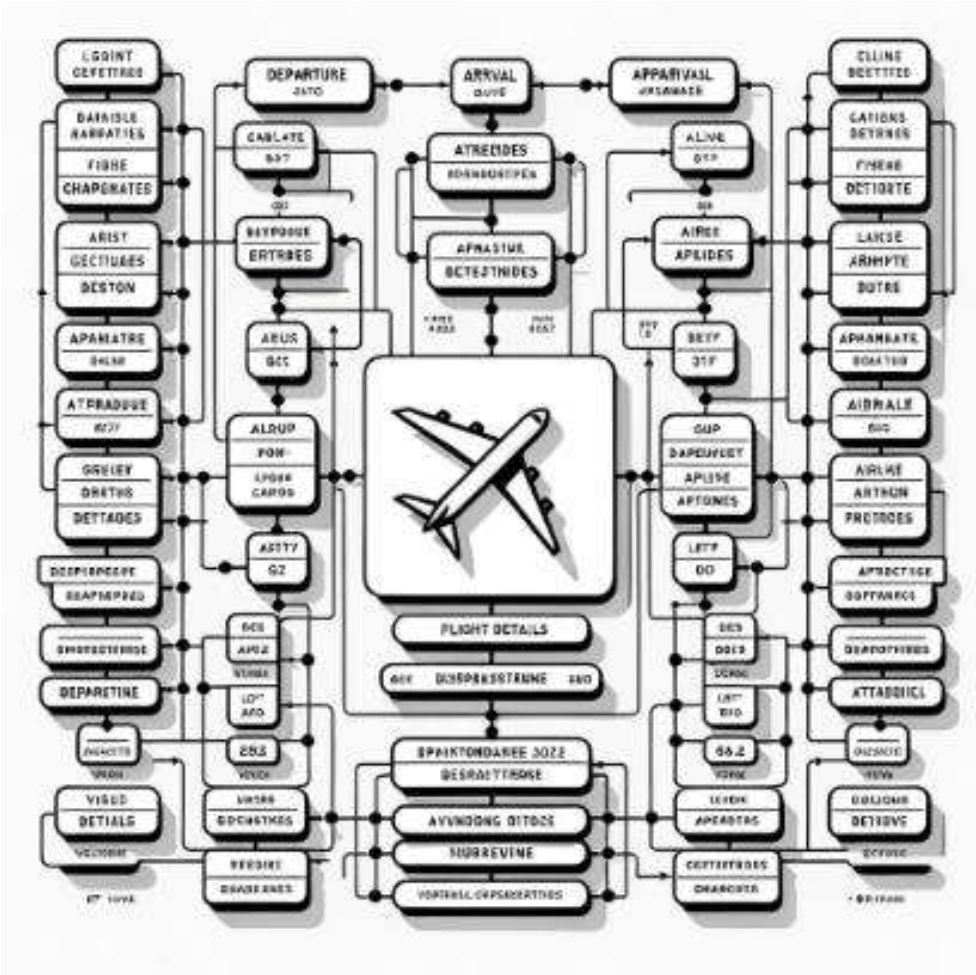
- **Geospatial Data** MongoDB manages geospatial data well, enabling spatial analysis like airport distances.
- **Rich Query Language :** Its powerful query language efficiently handles intricate data queries and aggregations.
- **Ease of Use :** User-friendly interfaces like MongoDB Compass simplify data management.
- **Community and Support :** An active community and robust documentation offer ongoing support and resources.

2.6 Rationale Behind DBMS Selection: Emphasizing Analysis Use Cases



- **Complex Aggregations:** MongoDB's aggregation framework efficiently calculates metrics, making it ideal for deriving insights like average delays.
- **Time-Series Analysis:** With timestamped data, MongoDB's date operators facilitate effective time-series analyses.
- **Geographical Analysis**: MongoDB's geospatial capabilities support the dataset's geographic-oriented queries, perfect for airport location analyses.
- **Pattern Recognition**: For identifying flight delay patterns, MongoDB's filtering and grouping functions are pivotal.
- **Data Visualization**: MongoDB's compatibility with visualization tools ensures effective representation of histograms and distributions.
- **Interlinked Data Analysis**: MongoDB handles interconnected data, addressing the dataset's multiple interlinked collections.
- **Scalability for Analysis:** MongoDB's scalability ensures efficient analyses, even with expanding datasets.

III. Data Overview - FLIGHT 2022 DATASET



3.1 Summary

The "Flight 2022" dataset is a comprehensive collection of data related to airline flights in the United States. This dataset provides a valuable resource for studying and analyzing various aspects of air travel, with a particular focus on Flight 2022s. Below is an overview of the key variables included in the dataset:

3.2 Raw Variables in Flight 2022 Dataset :

1. FlightDate: The date of the flight.
2. Airline: The name of the airline operating the flight.
3. Origin: The code or name of the origin airport.
4. Dest: The code or name of the destination airport.
5. Cancelled: Indicates if the flight was canceled (1 for canceled, 0 for not canceled).
6. Diverted: Indicates if the flight was diverted to an alternate airport (1 for diverted, 0 for not diverted).
7. CRSDepTime: The scheduled departure time.
8. DepTime: The actual departure time.
9. DepDelayMinutes: The delay in minutes for departure.
10. DepDelay: The delay in minutes for departure as a continuous variable.
11. ArrTime: The actual arrival time.
12. ArrDelayMinutes: The delay in minutes for arrival.
13. AirTime: The duration of the flight in minutes.
14. CRSElapsedTime: The scheduled elapsed time of the flight.
15. ActualElapsedTime: The actual elapsed time of the flight.
16. Distance: The distance traveled by the flight in miles.
17. Year: The year in which the flight occurred.
18. Quarter: The quarter in which the flight occurred.
19. Month: The month in which the flight occurred.
20. DayofMonth: The day of the month on which the flight occurred.
21. DayOfWeek: The day of the week on which the flight occurred.
22. Marketing_Airline_Network: The marketing airline's network name.
23. Operated_or_Branded_Code_Share_Partners: Information about code-sharing partnerships.
24. DOT_ID_Marketing_Airline: The Department of Transportation (DOT) ID of the marketing airline.
25. IATA_Code_Marketing_Airline: The IATA code of the marketing airline.
26. Flight_Number_Marketing_Airline: The flight number of the marketing airline.
27. Operating_Airline: The name of the airline operating the flight.
28. DOT_ID_Operating_Airline: The DOT ID of the operating airline.
29. IATA_Code_Operating_Airline: The IATA code of the operating airline.
30. Tail_Number: The tail number of the aircraft.
31. Flight_Number_Operating_Airline: The flight number of the operating airline.
32. OriginAirportID: The ID of the origin airport.
33. OriginAirportSeqID: The sequential ID of the origin airport.

34. OriginCityMarketID: The ID of the origin city market.
35. OriginCityName: The name of the origin city.
36. OriginState: The state of the origin airport.
37. OriginStateFips: The FIPS code of the origin state.
38. OriginStateName: The name of the origin state.
39. OriginWac: The World Area Code of the origin airport.
40. DestAirportID: The ID of the destination airport.
41. DestAirportSeqID: The sequential ID of the destination airport.
42. DestCityMarketID: The ID of the destination city market.
43. DestCityName: The name of the destination city.
44. DestState: The state of the destination airport.
45. DestStateFips: The FIPS code of the destination state.
46. DestStateName: The name of the destination state.
47. DestWac: The World Area Code of the destination airport.
48. DepDel15: Indicates if the departure was delayed by 15 minutes or more (1 for delayed, 0 for not delayed).
49. DepartureDelayGroups: Categorized departure delay groups.
50. DepTimeBlk: Time block for the actual departure time.
51. TaxiOut: Taxi-out time in minutes.
52. WheelsOff: Time when the aircraft's wheels left the ground.
53. WheelsOn: Time when the aircraft's wheels touched the ground.
54. TaxiIn: Taxi-in time in minutes.
55. CRSArrTime: The scheduled arrival time.
56. ArrDelay: The delay in minutes for arrival.
57. ArrDel15: Indicates if the arrival was delayed by 15 minutes or more (1 for delayed, 0 for not delayed).
58. ArrivalDelayGroups: Categorized arrival delay groups.
59. ArrTimeBlk: Time block for the actual arrival time.
60. DistanceGroup: Categorized distance groups.
61. DivAirportLandings: Number of diversions to the airport.

3.3 Special Requirements & Unusual Technical Aspects:

- ✓ **Volume & Complexity:** The 2022 flight dataset is extensive, requiring a DBMS adept at handling large-scale data efficiently.
- ✓ **Interconnected Data:** Collections, such as flight schedules and delays, demand a DBMS proficient in managing interrelated data.
- ✓ **Aggregations & Analysis:** The dataset needs robust aggregation and analysis capabilities, especially for collections like "Flight Performance."
- ✓ **Flexibility:** Flight data is dynamic; thus, a flexible, schema-less DBMS like MongoDB is beneficial.
- ✓ **Validation Needs:** For datasets like "Flight 2022," ensuring data consistency and robust validation is vital.

3.4 Key Variables: The Coordinates of Our Data Exploration

- **Temporal Variables:**
  - FlightDate: A timestamp marking the date of the flight.
  - Year, Quarter, Month, DayofMonth, DayOfWeek: Various temporal markers providing a detailed chronological context to each flight.
- **Airline and Flight Identifiers:**
  - Airline, Operating_Airline: Various identifiers and names related to the airlines.
  - DOT_ID_Marketing_Airline, IATA_Code_Marketing_Airline, Flight_Number_Marketing_Airline: Specific identifiers providing detailed information about the marketing airline.
  - DOT_ID_Operating_Airline, IATA_Code_Operating_Airline, Flight_Number_Operating_Airline: Detailed identifiers for the operating airline.
  - Tail_Number: A unique identifier for the aircraft.
- **Geographical Variables:**
  - Origin, Dest: Codes or names of origin and destination airports, respectively.
  - OriginAirportID, OriginAirportSeqID, OriginCityMarketID, OriginCityName, OriginState, OriginStateFips, OriginStateName, OriginWac: A suite of variables providing detailed geographical information about the origin airport.

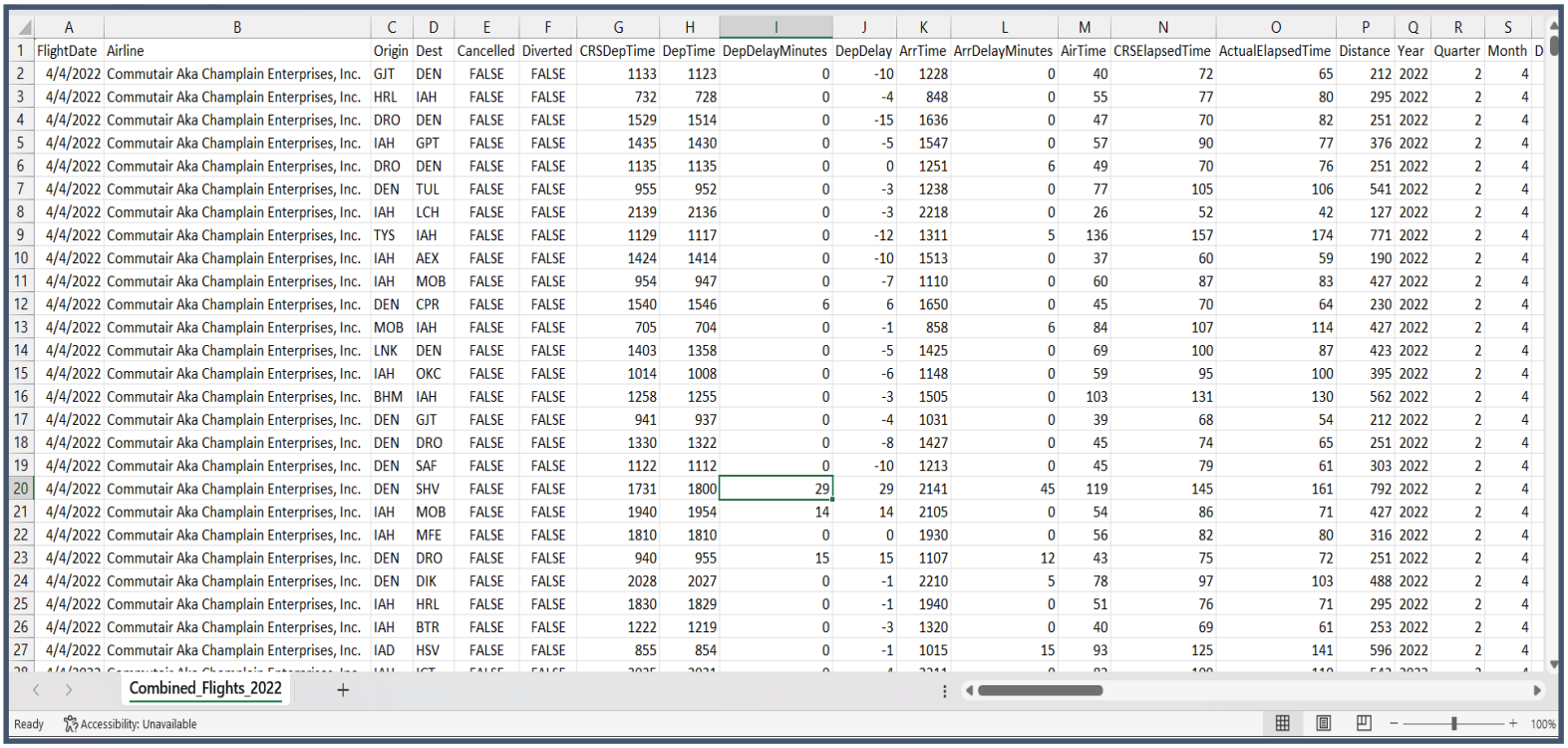
- DestAirportID, DestAirportSeqID, DestCityMarketID, DestCityName, DestState, DestStateFips, DestStateName, DestWac: A parallel suite of variables offering comprehensive details about the destination airport.

➢ Flight Performance Metrics:

- Cancelled, Diverted: Binary variables indicating if the flight was cancelled or diverted.
- CRSDepTime, DepTime, DepDelayMinutes, DepDelay: Variables related to departure times and delays.
- ArrTime, ArrDelayMinutes, ArrDelay: Variables providing insights into arrival times and delays.
- AirTime, CRSElapsedTime, ActualElapsedTime: Metrics detailing of flight duration.
- **Distance:** The geographical distance covered by the flight.

➢ Additional Flight Details:

- Marketing_Airline_Network, Operated_or_Branded_Code_Share_Partners: Information about marketing and operational partnerships.
- DepDel15, DepartureDelayGroups, DepTimeBlk: Additional metrics related to departure delays.
- TaxiOut, WheelsOff, WheelsOn, TaxiIn: Variables detailing the taxiing and wheels-off/on times.
- CRSArrTime, ArrDel15, ArrivalDelayGroups, ArrTimeBlk: Additional metrics related to arrival delays.
- DistanceGroup: Categorical variable grouping flights based on distance.
- DivAirportLandings: The number of diversions to the airport.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	FlightDate	Airline	Origin	Dest	Cancelled	Diverted	CRSDepTime	DepTime	DepDelayMinutes	DepDelay	ArrTime	ArrDelayMinutes	AirTime	CRSElapsedTime	ActualElapsedTime	Distance	Year	Quarter	Month	D
2	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	GJT	DEN	FALSE	FALSE	1133	1123	0	-10	1228	0	40	72	65	212	2022	2	4	
3	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	HRL	IAH	FALSE	FALSE	732	728	0	-4	848	0	55	77	80	295	2022	2	4	
4	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DRO	DEN	FALSE	FALSE	1529	1514	0	-15	1636	0	47	70	82	251	2022	2	4	
5	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	GPT	FALSE	FALSE	1435	1430	0	-5	1547	0	57	90	77	376	2022	2	4	
6	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DRO	DEN	FALSE	FALSE	1135	1135	0	0	1251	6	49	70	76	251	2022	2	4	
7	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	TUL	FALSE	FALSE	955	952	0	-3	1238	0	77	105	106	541	2022	2	4	
8	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	LCH	FALSE	FALSE	2139	2136	0	-3	2218	0	26	52	42	127	2022	2	4	
9	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	TYS	IAH	FALSE	FALSE	1129	1117	0	-12	1311	5	136	157	174	771	2022	2	4	
10	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	AEX	FALSE	FALSE	1424	1414	0	-10	1513	0	37	60	59	190	2022	2	4	
11	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	MOB	FALSE	FALSE	954	947	0	-7	1110	0	60	87	83	427	2022	2	4	
12	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	CPR	FALSE	FALSE	1540	1546	6	6	1650	0	45	70	64	230	2022	2	4	
13	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	MOB	IAH	FALSE	FALSE	705	704	0	-1	858	6	84	107	114	427	2022	2	4	
14	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	LNK	DEN	FALSE	FALSE	1403	1358	0	-5	1425	0	69	100	87	423	2022	2	4	
15	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	OKC	FALSE	FALSE	1014	1008	0	-6	1148	0	59	95	100	395	2022	2	4	
16	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	BHM	IAH	FALSE	FALSE	1258	1255	0	-3	1505	0	103	131	130	562	2022	2	4	
17	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	GJT	FALSE	FALSE	941	937	0	-4	1031	0	39	68	54	212	2022	2	4	
18	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	DRO	FALSE	FALSE	1330	1322	0	-8	1427	0	45	74	65	251	2022	2	4	
19	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	SAF	FALSE	FALSE	1122	1112	0	-10	1213	0	45	79	61	303	2022	2	4	
20	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	SHV	FALSE	FALSE	1731	1800	29	29	2141	45	119	145	161	792	2022	2	4	
21	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	MOB	FALSE	FALSE	1940	1954	14	14	2105	0	54	86	71	427	2022	2	4	
22	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	MFE	FALSE	FALSE	1810	1810	0	0	1930	0	56	82	80	316	2022	2	4	
23	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	DRO	FALSE	FALSE	940	955	15	15	1107	12	43	75	72	251	2022	2	4	
24	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	DEN	DIK	FALSE	FALSE	2028	2027	0	-1	2210	5	78	97	103	488	2022	2	4	
25	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	HRL	FALSE	FALSE	1830	1829	0	-1	1940	0	51	76	71	295	2022	2	4	
26	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAH	BTR	FALSE	FALSE	1222	1219	0	-3	1320	0	40	69	61	253	2022	2	4	
27	4/4/2022	Commutair Aka Champlain Enterprises, Inc.	IAD	HSV	FALSE	FALSE	855	854	0	-1	1015	15	93	125	141	596	2022	2	4	

Combined_Flights_2022

Ready Accessibility: Unavailable

IV. Database Design & Implementation

4.1 Dynamic Schema

- **Dynamic Schema Needs:** The "Flight 2022" dataset's complexity and varied attributes demand a flexible schema structure.
- **MongoDB's Schema-less Benefit**: MongoDB's architecture allows for easy dataset evolution without being tied to a fixed schema.
- **Adaptable Growth:** As new flight metrics or delay reasons emerge, MongoDB ensures their effortless integration without database overhaul.

4.2 Data Import and Transformation

The implementation of the MongoDB database involves several crucial steps:

➢ Installation and Setup

- MongoDB is installed on the server or local machine.
- The MongoDB service is started.
- A new database is created to house the Flight 2022 dataset.

➢ Software Tools used

- To manage and analyze the Flight 2022 dataset in a MongoDB database, the following tools and software were explored .
- **MongoDB Server:** The primary software to store and manage your dataset.
- **MongoDB Compass:** A graphical user interface for MongoDB that facilitates easy visualization and interaction with your dataset without the need for command-line operations.
- **MongoDB Shell (mongosh):** A powerful command-line interface for MongoDB, useful for performing various operations, including data manipulation and administrative tasks.
- **Backup tools:** Tools like mongodump and mongorestore can be handy for backing up and restoring your data.
- **Monitoring tools:** MongoDB Atlas, - MongoDB's cloud service, offers monitoring capabilities. For self-hosted solutions, tools like mongostat and mongotop provide insights into database performance.
- **Indexing tools:** Built into MongoDB, these tools, accessible through MongoDB Compass or the shell, help in optimizing query performance.
- IDEs: For coding purposes, Integrated Development Environments (IDEs)Visual Studio Code was used.

4.2 Collections and Document Structure:

➢ Flights Collection:

- Document Structure: Each document represents a single flight and encapsulates attributes such as FlightDate, Airline, DepDelay, ArrDelay, and geographical and temporal variables.
- Potential Primary Key: A composite key of FlightDate, Airline, and Flight_Number_Marketing_Airline could uniquely identify each flight.

➢ Airlines Collection:

- Document Structure: Each document could represent an airline, containing attributes like Airline, DOT_ID_Marketing_Airline, and IATA_Code_Marketing_Airline.
- Potential Primary Key: DOT_ID_Marketing_Airline.

➢ Airports Collection:

- Document Structure: Documents in this collection could represent airports, with attributes like OriginAirportID, OriginCityName, and OriginState.
- Potential Primary Key: OriginAirportID.

➢ Relationships and Data Modeling

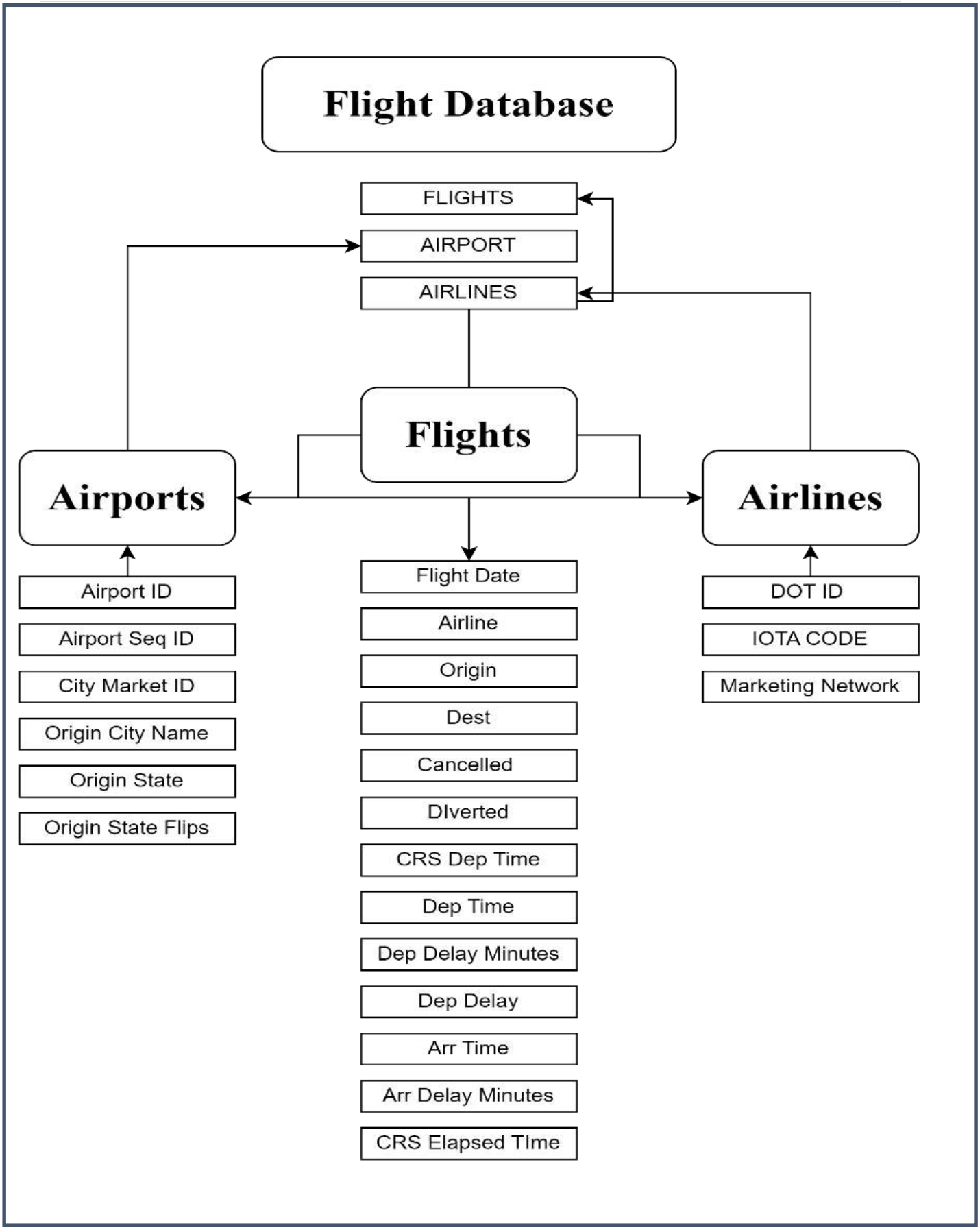
- Flight-Airline Relationship: A many-to-one relationship where each flight (in the Flights Collection) is associated with one airline (in the Airlines Collection) through the DOT_ID_Marketing_Airline attribute.
- Flight-Airport Relationship: A many-to-one relationship can be established between flights and airports, using OriginAirportID and DestAirportID to link flight data with respective origin and destination airports.
- Temporal Relationships: Ensuring that temporal attributes within each flight document, such as Year, Month, and DayofMonth, are internally consistent and accurately represent FlightDate.

4.3 Automatic Schema View:

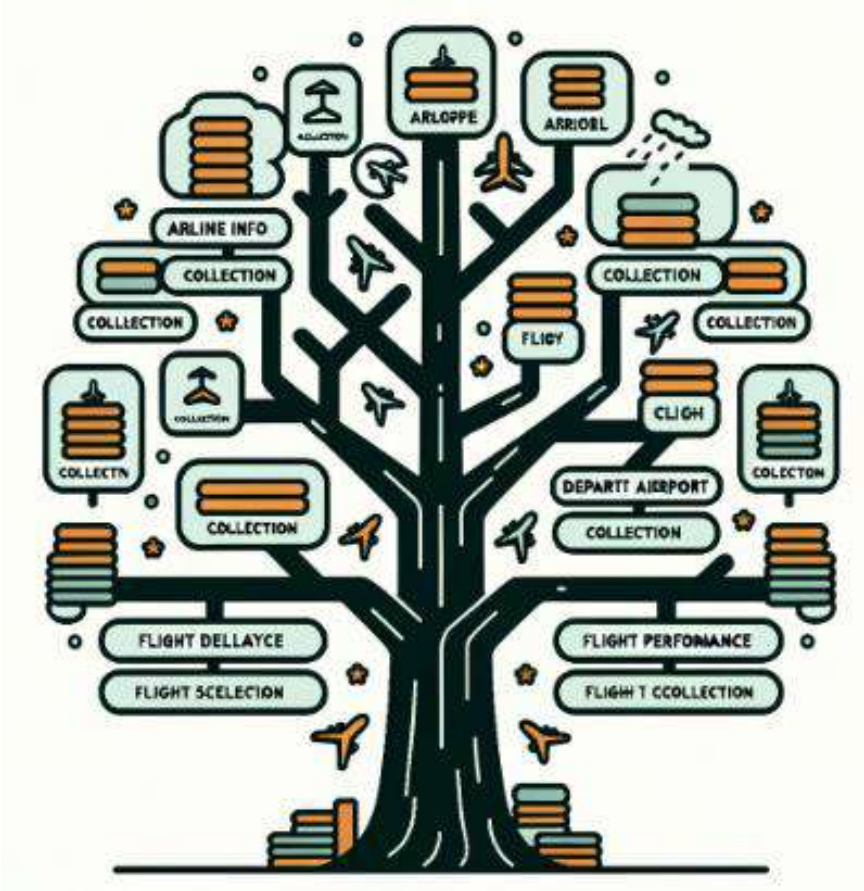
- **Dynamic Schema View**: MongoDB offers an automatic schema view, allowing for a structured data overview without a preset schema.
- **Adaptive Data Modeling:** The schema evolves with the data, ensuring flexibility and adaptability.
- **MongoDB Compass Utility:** Compass provides a visual exploration of the schema, aiding in understanding data structures and pinpointing inconsistencies.

4.4 Data Integrity and Validation:

- **Consistency Checks:** Implementing validation rules to ensure that related attributes, such as DepDelay and DepDel15, are consistent. For instance, if DepDelay > 15, then DepDel15 should be 1.
- **Null Value Handling:** Establishing mechanisms to handle null or missing values, ensuring that the absence of data does not impede analyses or application functionality.
- **Validation Rules:** Employing MongoDB's schema validation capabilities to enforce data types and ensure that incoming data adheres to expected formats and ranges, thereby safeguarding against erroneous data entries.


Flight Database
FLIGHTS
AIRPORT
AIRLINES
Flights
Airports
Airlines
Airport ID
Airport Seq ID
City Market ID
Origin City Name
Origin State
Origin State Flips
Flight Date
Airline
Origin
Dest
Cancelled
DIverted
CRS Dep Time
Dep Time
Dep Delay Minutes
Dep Delay
Arr Time
Arr Delay Minutes
CRS Elapsed TIme
DOT ID
IOTA CODE
Marketing Network

V. Database & Collections Implementation



- **Flight 2022 DBMS** was created using MongoDB Compass interface, specifically the databases section:
- **User Profile:** The name "Vasanth" is displayed, indicating the user profile
- **Database Overview:**
  - **FLIGHT:** This is a primary database that holds data related to flights. It has a storage size of 795.18 MB and contains 7 collections, each with its own set of indexes. The collections inside are:
    - Airline Info Collection
    - Arrival Airport Collection
    - Departure Airport Collection
    - Flight Collections
    - Flight Delay Collection
    - Flight Performance Collection
    - Flight Schedule Collection

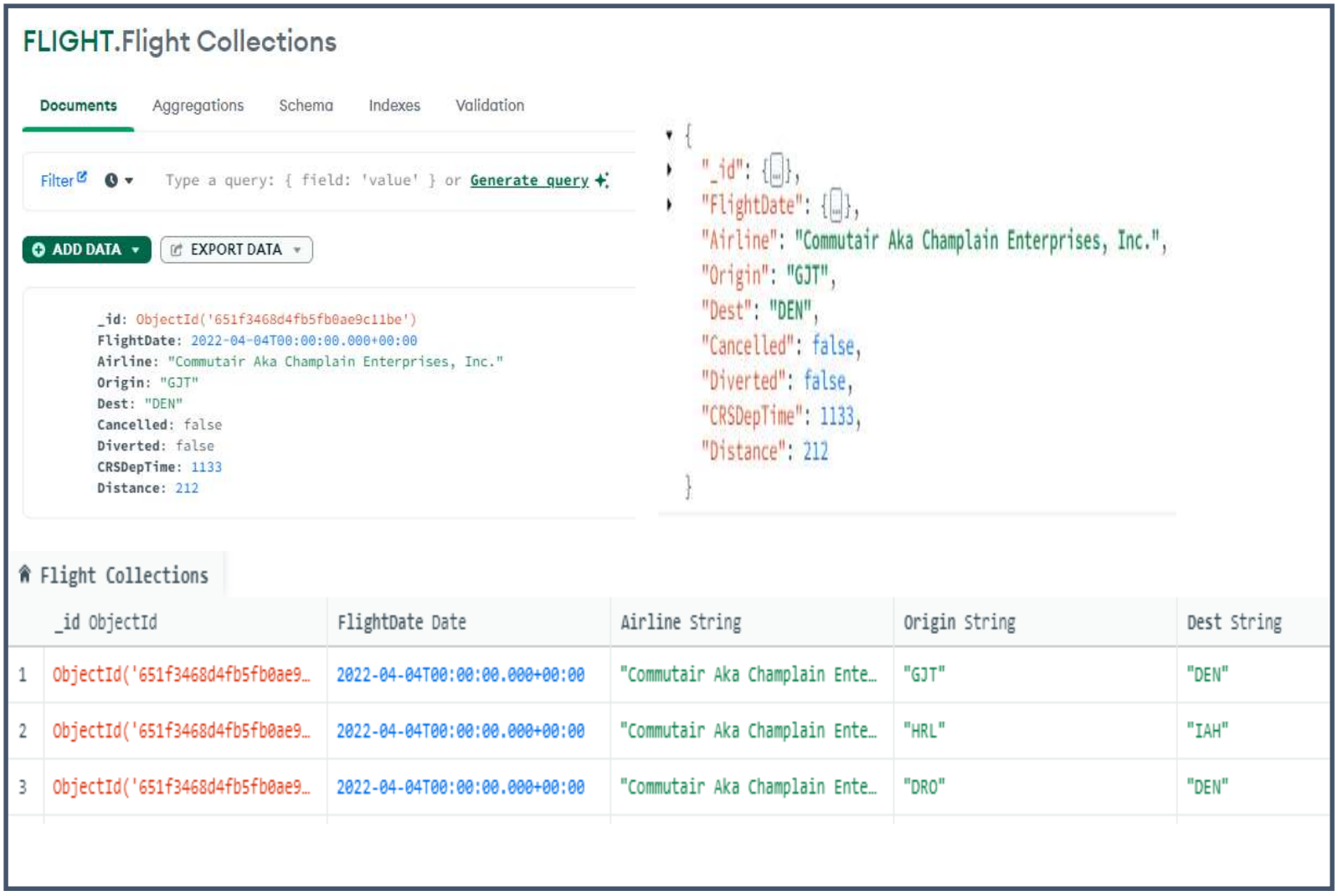
5.1 Flight collections

- **General Information:**
  - FlightDate
  - Airline
  - Origin
  - Dest
  - Cancelled
  - Diverted
  - Distance
- **Departure Details:**
  - CRSDepTime (Scheduled Departure Time)
  - DepTime (Actual Departure Time)
  - DepDelay (Departure Delay in Minutes as Continuous Variable)
  - DepDelayMinutes (Departure Delay in Minutes)
- **Arrival Details:**
  - CRSArrTime (Scheduled Arrival Time)
  - ArrTime (Actual Arrival Time)
  - ArrDelayMinutes (Arrival Delay in Minutes)
- **Flight Duration:**

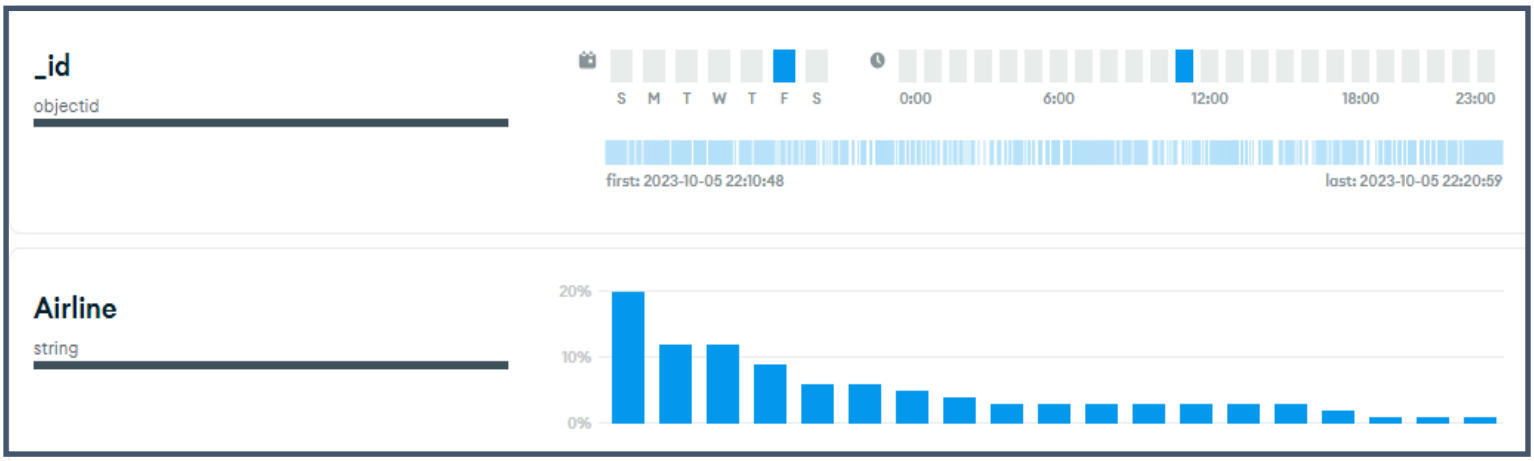
- AirTime (Duration of Flight in Minutes)
- CRSElapsedTime (Scheduled Elapsed Time of Flight)
- ActualElapsedTime (Actual Elapsed Time of Flight)

Importance and uses of the "Flights" collection:

- ✓ Helps airlines optimize schedules and allocate resources based on historical performance.
- ✓ Enables travelers to assess flight reliability, allowing airlines to enhance service quality.
- ✓ Assists in identifying potential revenue losses and aids in cost-saving through delay pattern insights.
- ✓ Guides route expansions and provides data-driven insights for competitive positioning.
- ✓ Tracks diverted flights for safety concerns and ensures regulatory compliance on performance.

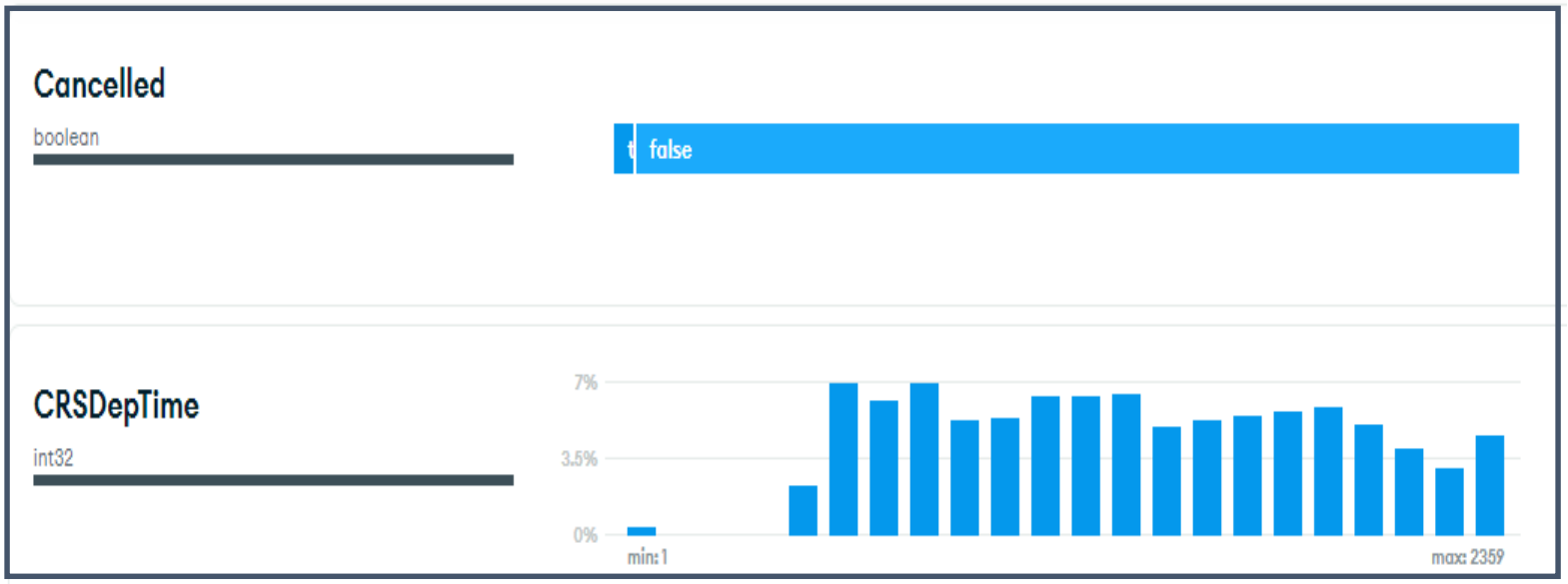


AUTOMATIC SCHEMA –Flight Collection



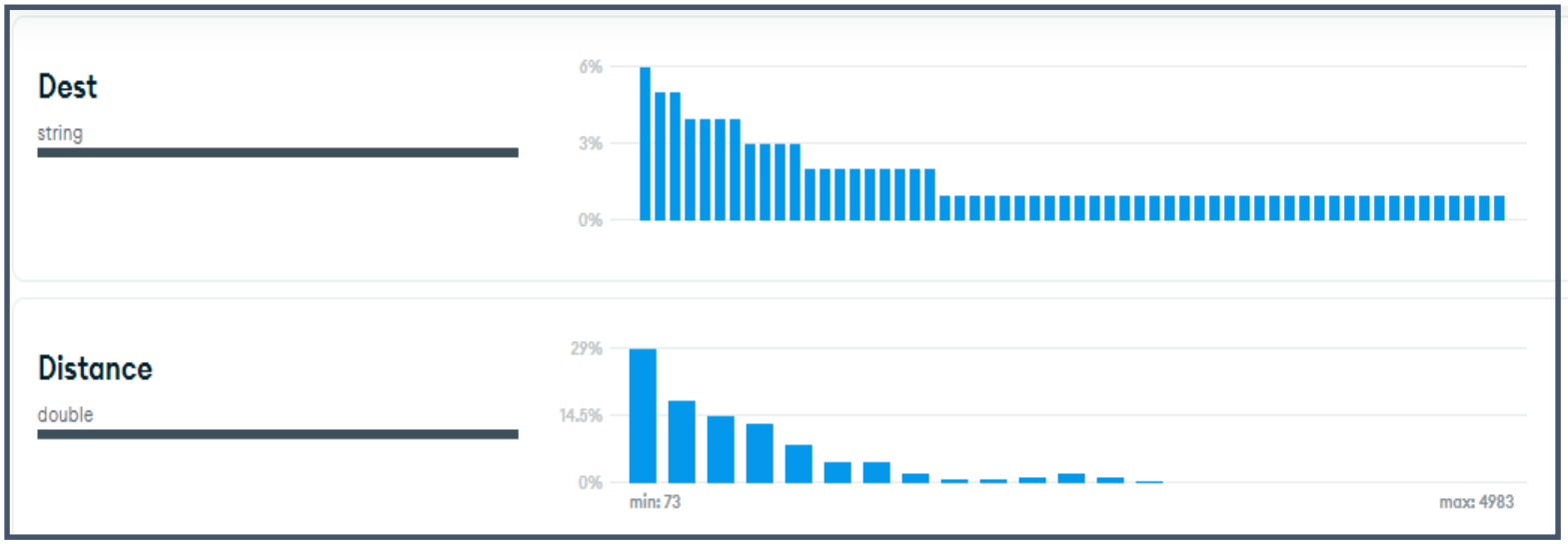
- **_id:**
  - **Type: ObjectId**
  - **Description:** This is a unique identifier generated by MongoDB for each document in the collection.
  - **Activity Timestamps:** The horizontal bars represent data activity timestamps, showing when the data was first and last updated. The days (from Sunday to Saturday) and hours (from 0:00 to 23:00) indicate the frequency and distribution of updates.

- **Airline:**
  - **Type: String**
  - **Description:** This field holds the name of the airline.
  - **Distribution:** The vertical bar graph next to "Airline" showcases the distribution of data

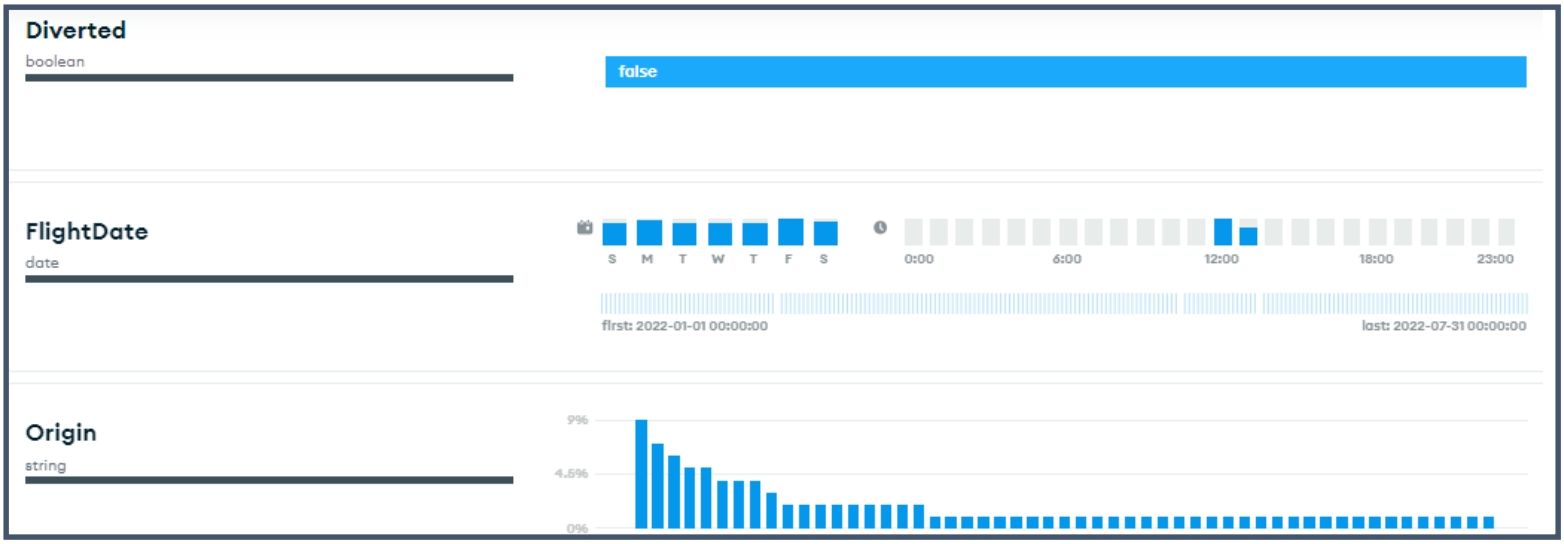


- **Cancelled:**
  - **Type:** Boolean
  - **Description:** This field indicates whether a flight was cancelled or not.

- **CRSDepTime:**
  - **Type:** int32
  - **Description:** This field, "CRSDepTime", likely stands for "Computerized Reservations System Departure Time". It represents the scheduled departure time of flights, typically represented in a 24-hour format (e.g., 2359 for 11:59 PM).
  - **Distribution:** The vertical bar graph showcases the distribution of scheduled departure times across the dataset. The range is from a minimum time of "1" to a maximum time of "2359".



- **Dest:**
  - **Type:** String
  - **Description:** "Dest" stands for "Destination". This field indicates the destination airport code or name for the flights.
  - **Distribution:** The vertical bar graph represents the distribution of flights going to various destinations. Each bar's height corresponds to the frequency of flights headed to that particular destination.
- **Distance:**
  - **Type:** Double
  - **Description:** This field represents the distance of the flights, likely measured in miles or kilometers.
  - **Distribution:** The vertical bar graph showcases the distribution of flight distances across the dataset, ranging from a minimum distance of "73" to a maximum distance of "4983".



- **Diverted:**
  - **Type:** Boolean
  - **Description:** The "Diverted" field indicates whether a flight was diverted or not.
  - **Data Insight:** The dominant blue bar associated with the "false" value suggests that the vast majority of flights in this dataset were not diverted.

- **FlightDate:**
  - **Type:** Date
  - **Description:** Represents the date on which the flight took place.
  - **Activity Timestamps:** The horizontal bars indicate when flights occurred, with days (from Sunday to Saturday) and hours (from 0:00 to 23:00) showcasing the distribution. The dataset starts from "2022-01-01 00:00:00" and ends on "2022-07-31 00:00:00", suggesting it covers the first seven months of 2022.
  - Data Insight: Flights are distributed throughout the week, with specific hours, especially around noon, seeing more activity.

- **Origin:**
  - **Type:** String
  - **Description:** "Origin" represents the departure airport code or name.
  - **Distribution:** The vertical bar graph depicts the distribution of flights from various origin airports. The height of each bar corresponds to the frequency of flights departing from that specific airport.

5.2 Flight Schedule Collection:

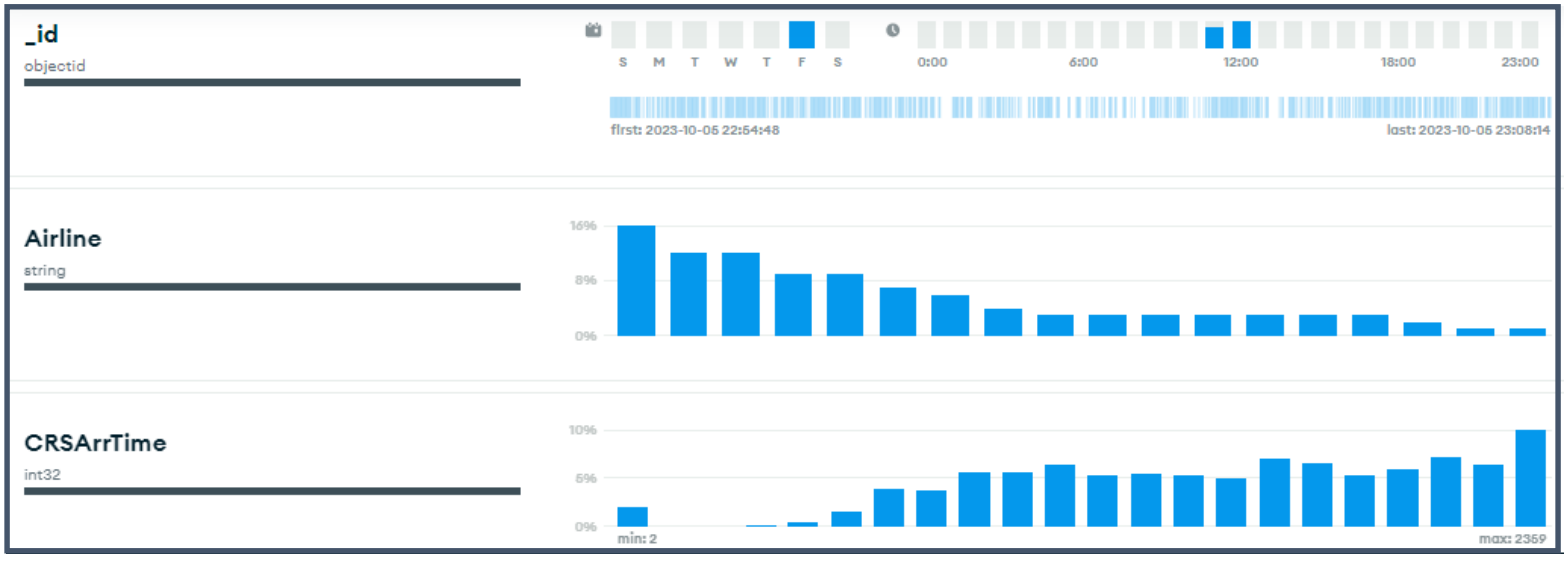
- **Flight Schedule Collection:**
  - FlightDate: Date of the flight.
  - Airline: Operating airline for the flight.
  - Origin: Departure airport or location.
  - Dest: Arrival or destination airport.

- **Scheduled Timings:**
  - CRSDepTime: Scheduled departure time.
  - CRSArrTime: Expected or scheduled arrival time.

Importance and uses of the "Flight Schedule" collection:

- ✓ The collection aids airlines in identifying patterns of delays, enabling them to address operational bottlenecks and improve on-time performance.
- ✓ Enables airlines to plan crew schedules, aircraft rotations, and ground services based on the set timetable.
- ✓ Provides travelers with information to plan their journeys, ensuring they know when to arrive at the airport and what to expect for their flights.

AUTOMATIC SCHEMA – Flight Schedule Collection



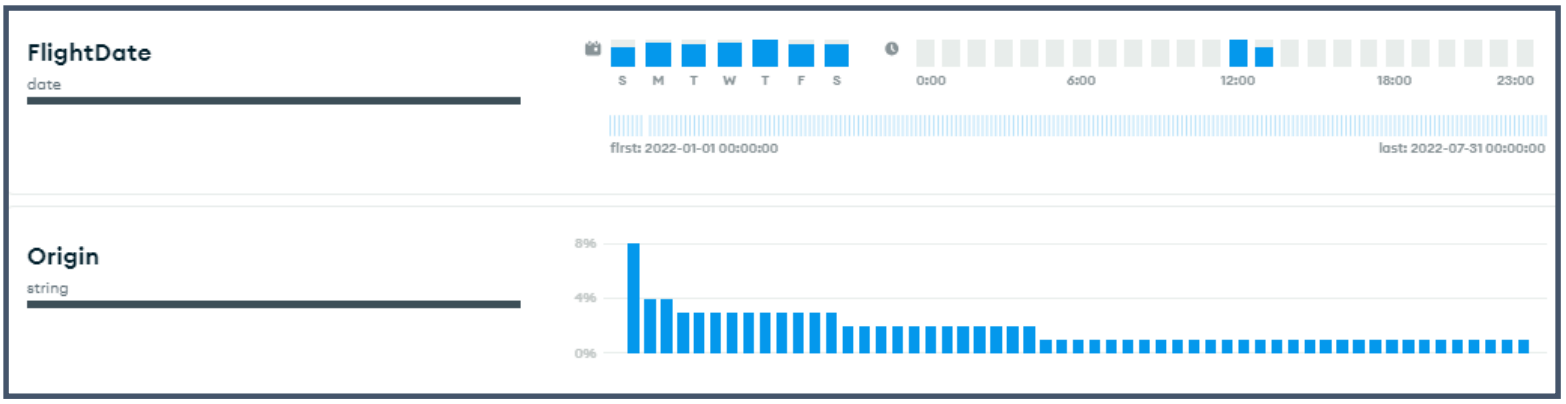
- **id:**
  - **Type:** ObjectId
  - **Description:** This is a unique identifier auto-generated by MongoDB for every document in a collection.
  - **Activity Timestamps**: The graphical representation on the right demonstrates when records were created or modified. It spans days of the week (from Sunday to Saturday) and hours of the day (from 0:00 to 23:00).

- **Airline:**
  - **Type:** String
  - **Description:** Represents the name or identifier of an airline.
  - **Distribution:** The vertical bar chart adjacent to "Airline" shows the distribution of records for different airlines.

- **CRSArrTime:**
  - **Type:** int32
  - **Description:** Stands for "Computerized Reservations System Arrival Time", representing the scheduled arrival time for flights, usually in a 24-hour format.
  - **Distribution:** The vertical bar chart displays the distribution of scheduled arrival times throughout the dataset.

- **CRSDepTime:**
  - **Type:** int32
  - **Description:** Stands for "Computerized Reservations System Departure Time". It signifies the scheduled departure time for flights, typically given in a 24-hour format.
- **Dest:**
  - **Type:** String
  - **Description:** Represents the destination airport code or name for the flights.
  - **Distribution:** The vertical bar chart next to "Dest" displays the frequency of flights to various destinations.

- **DOT_ID_Marketing_Airline:**
  - **Type:** int32
  - **Description:** This field likely refers to the Department of Transportation (DOT) ID associated with the marketing airline. It's a unique identifier for airlines in the context of marketing and partnerships.
  - **Distribution:** The vertical bar chart portrays how records are distributed across different DOT IDs for marketing airlines. Each bar's height corresponds to the frequency of records for a specific DOT ID, and the x-axis shows specific ID numbers.



- **FlightDate:**
  - **Type:** date
  - **Description:** Represents the date when the flight is scheduled.
- **Origin:**
  - **Type:** String
  - **Description:** Indicates the departure airport or the origin from where the flight starts its journey.

5.3 Flight_Delay Collection:

- **General Flight Details:**
  - FlightDate: Date on which the flight is scheduled.
  - Airline: Carrier operating the particular flight.
  - Origin: Starting point or departure airport.
  - Dest: Destination or arrival airport.

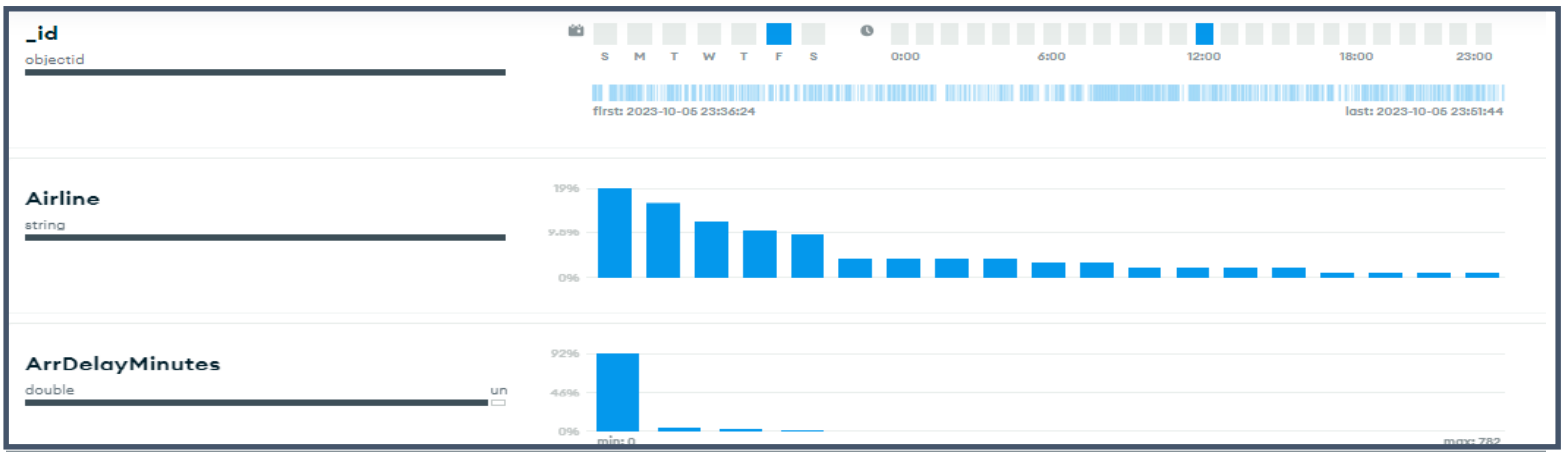
- **Delay Metrics:**
  - DepDelayMinutes: Duration, in minutes, the flight departed later than scheduled.
  - ArrDelayMinutes: Duration, in minutes, the flight arrived later than its scheduled time.

Importance and uses of the "Flight Delay" collection:

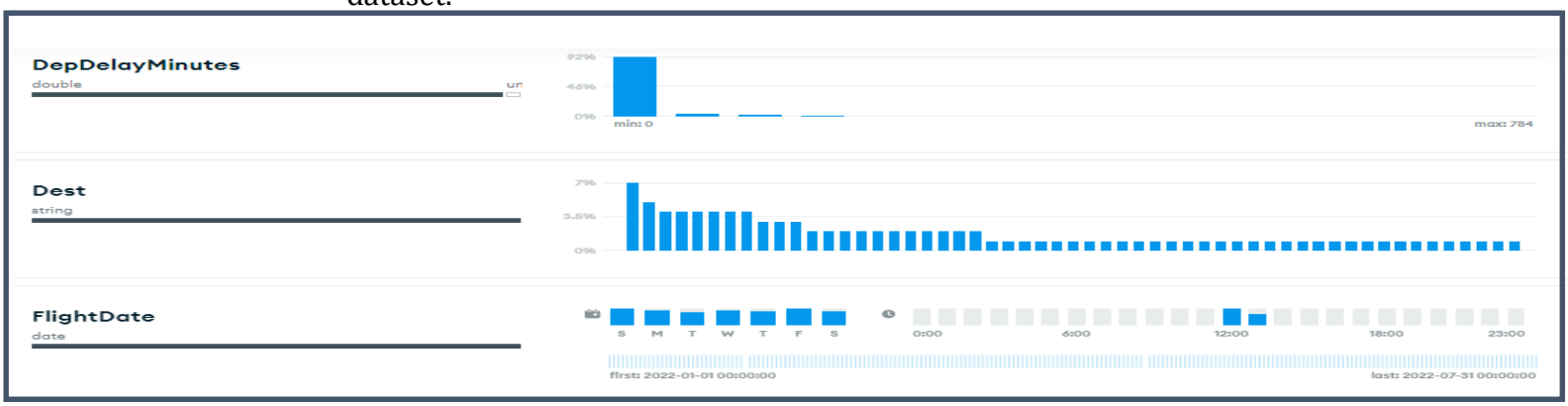
- ✓ The collection aids airlines in identifying patterns of delays, enabling them to address operational bottlenecks and improve on-time performance.
- ✓ By analyzing delay patterns, airlines can proactively communicate with passengers about potential delays, leading to improved customer satisfaction.

- ✓ Airports and airlines can optimize the allocation of resources such as gates, crew, and ground services based on delay trends, ensuring better handling during peak delay times.
- ✓ The data helps airlines remain accountable to regulatory bodies, ensuring they meet industry standards and adhere to compensation protocols for significant delays.

AUTOMATIC SCHEMA – Flight Delay Collection



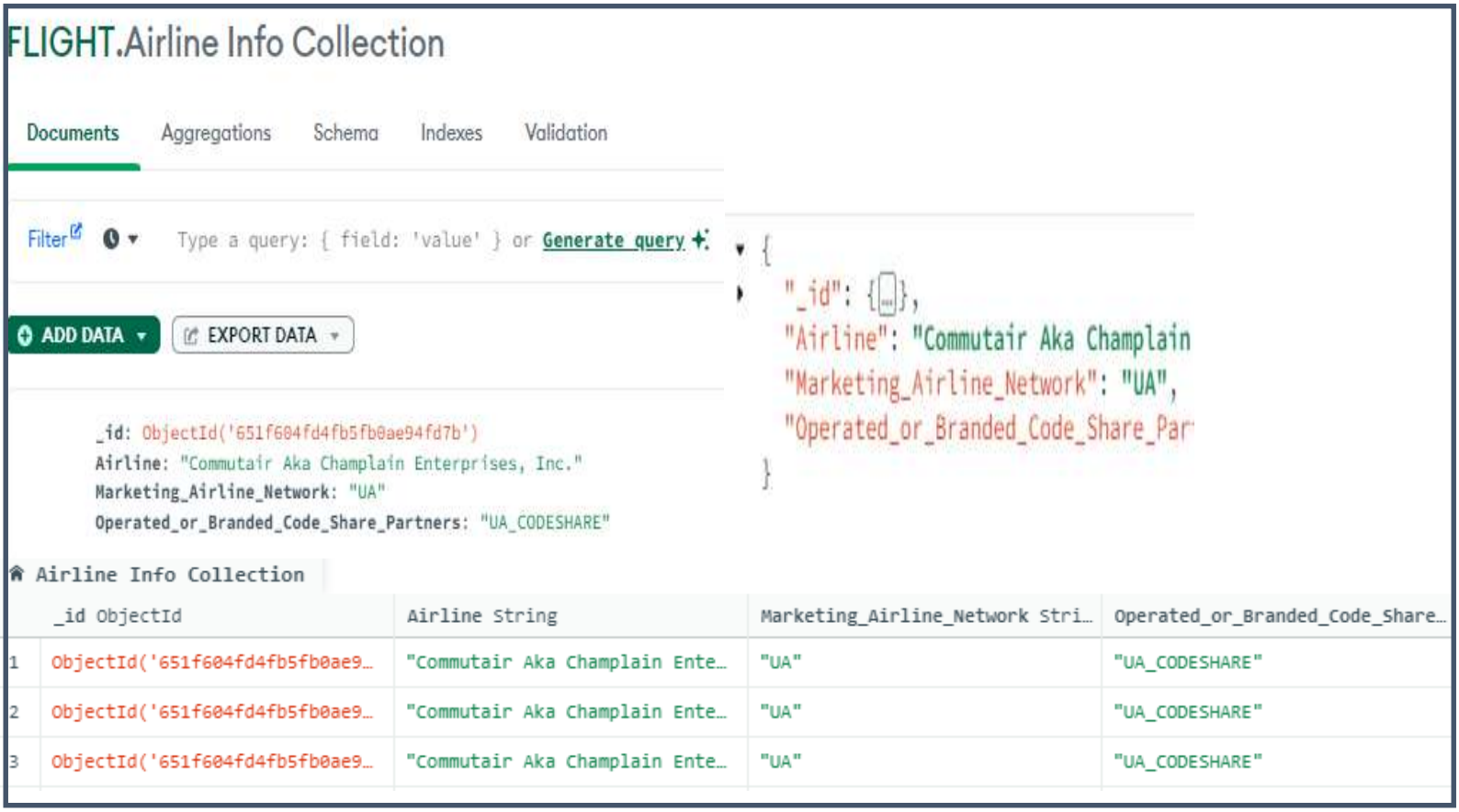
- ➢ **_id:**
  - o **Type:** ObjectId
  - o **Description:** This is the unique identifier for each record or document in the dataset. Every document in a MongoDB collection typically has this field by default.
- ➢ **Airline:**
  - o **Type:** String
  - o **Description:** Represents the name of the airline.
  - o **Distribution:** The vertical bar chart adjacent to "Airline" depicts the frequency or count of flights for various airlines.
- ➢ **ArrDelayMinutes:**
  - o **Type:** Double
  - o **Description:** Indicates the delay in minutes for a flight's arrival.
  - o **Distribution**: The bar chart showcases the distribution of arrival delays in the dataset.



- **DepDelayMinutes:**
  - **Type:** Double
  - **Description:** This field represents the delay in minutes for a flight's departure.
  - Distribution: The bar chart adjacent to "DepDelayMinutes" indicates the distribution of departure delays in the dataset.
- **Dest:**
  - **Type:** String
  - **Description:** Represents the destination airport or city of the flight.
  - **Distribution:** The vertical bar chart beside "Dest" displays the frequency or count of flights heading to various destinations.
- **FlightDate:**
  - **Type:** Date
  - **Description:** Indicates the date on which the flight took place.
- Activity Timestamps: The horizontal bar chart below "FlightDate" shows the distribution of flights across a time range.

5.4 Airline Info Collection:

- **Carrier Name:**
  - Airline: Represents the primary name of the airline.
- **Marketing Affiliation:**
  - Marketing_Airline_Network: The promotional network or alliance the airline is associated with.
- **Partnership Data:**
  - Operated_or_Branded_Code_Share_Partners: Specifies the code-sharing agreements or collaborations the airline has with other carriers.

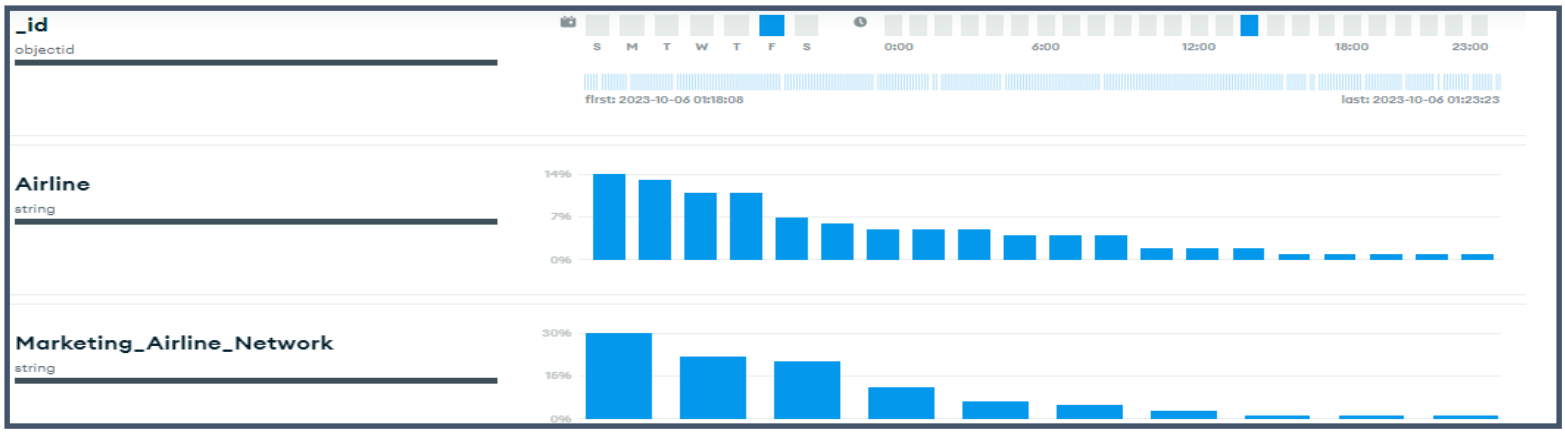


	_id ObjectId	Airline String	Marketing_Airline_Network Stri…	Operated_or_Branded_Code_Share…
1	ObjectId('651f604fd4fb5fb0ae9…	"Commutair Aka Champlain Ente…	"UA"	"UA_CODESHARE"
2	ObjectId('651f604fd4fb5fb0ae9…	"Commutair Aka Champlain Ente…	"UA"	"UA_CODESHARE"
3	ObjectId('651f604fd4fb5fb0ae9…	"Commutair Aka Champlain Ente…	"UA"	"UA_CODESHARE"

Importance and uses of the " Airline Info" collection:

- ✓ Offers a single source of truth for airline-specific data, ensuring consistency in referencing airlines across various databases or systems.
- ✓ Enables analysts to understand airline alliances and affiliations, assisting in strategic marketing decisions and competitive analysis.
- ✓ Provides clarity on code-sharing partnerships, enhancing route planning, joint marketing efforts, and cooperative operational strategies.

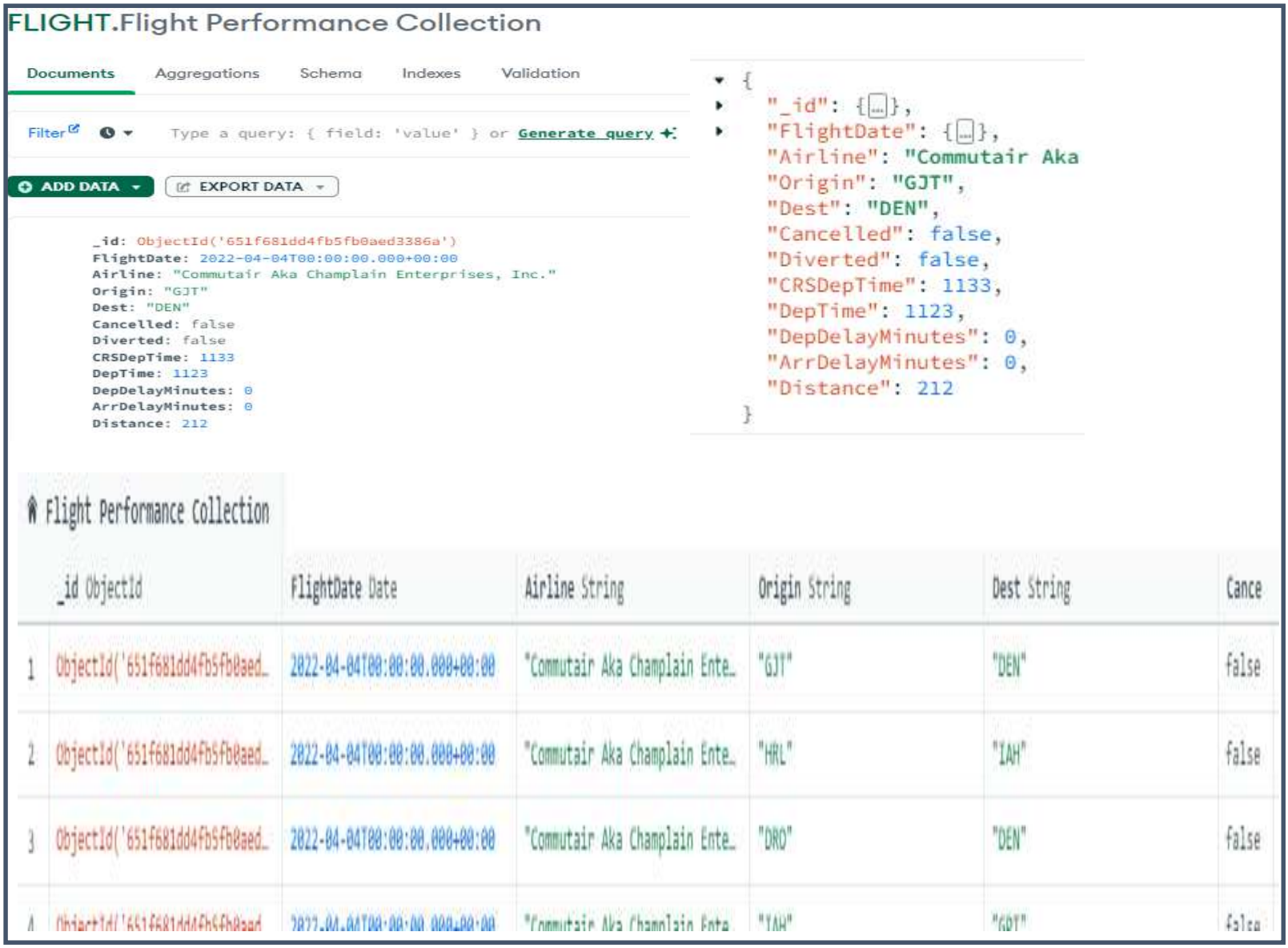
AUTOMATIC SCHEMA – Airline Info Collection



- **_id:**
  - **Type:** objectid
  - This represents the unique identifier automatically assigned to each record in MongoDB.
- **Airline:**
  - This field contains the names or identifiers of various airlines.
  - The bar chart visualizes the distribution or frequency of records for each airline.
- **Marketing_Airline_Network:**
  - This field denotes the marketing networks or alliances to which each airline belongs.

5.5 Flight Performance Collection:

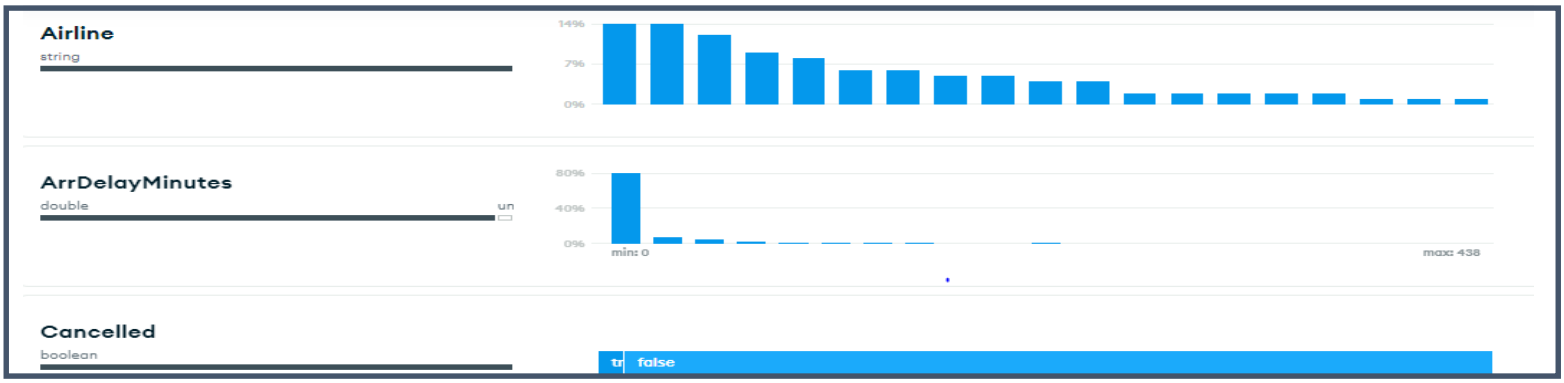
- **Schedule Info:**
  - FlightDate: The scheduled date of the flight.
  - Airline: Name or identifier of the operating airline.
  - Origin: Starting airport for the flight.
  - Dest: Destination airport for the flight.
- **Operational Outcomes:**
  - Cancelled: Indicator if the flight was called off.
  - Diverted: Indicator if the flight was rerouted to another airport.
- **Performance Metrics:**
  - DepDelayMinutes: Duration of delay during departure.
  - ArrDelayMinutes: Duration of delay upon arrival.
  - Distance: Total miles traveled by the flight.



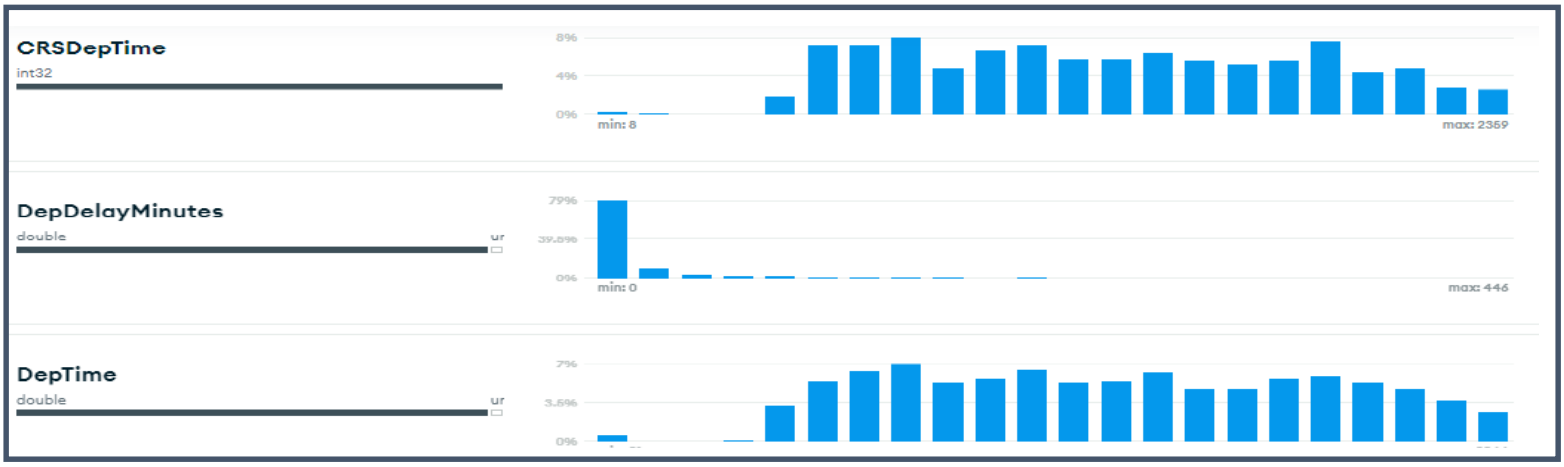
Importance and uses of the " Flight Performance" collection:

- ✓ Operational Analysis: It provides airlines with insights into flight regularity, helping them identify patterns in cancellations, delays, and diversions.
- ✓ Customer Experience: By monitoring delays, airlines can strategize to improve on-time performance, enhancing passenger satisfaction.
- ✓ Efficiency and Planning: Analyzing distance traveled in relation to delays can aid in optimizing routes and resource allocation.

AUTOMATIC SCHEMA – Flight Performance Collection



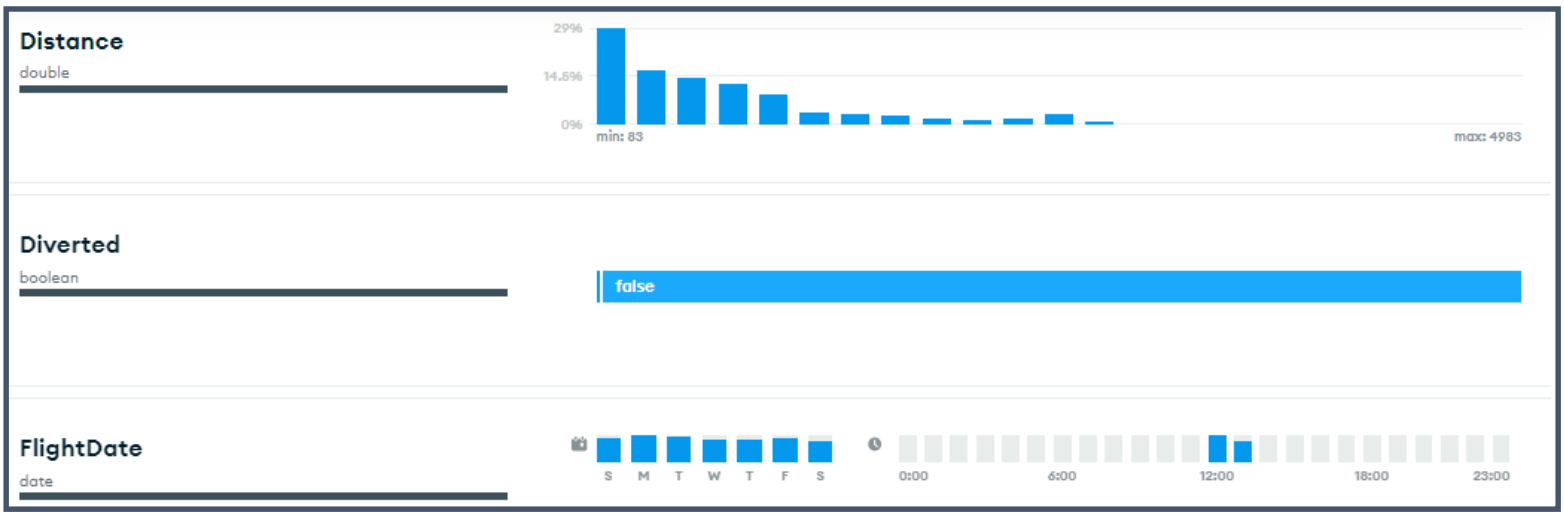
- ➢ **Airline (string):**
  - o This field represents the airline's name.
- ➢ **ArrDelayMinutes (double):**
  - o This indicates the number of minutes a flight was delayed upon arrival.
  - o The distribution suggests most flights arrive on time or with minimal delay, as there's a prominent peak at or near zero.
- ➢ **Cancelled (boolean):**
  - o This field indicates whether a flight was cancelled or not.
  - o The depicted data suggests that a vast majority of flights were not cancelled, as represented by the "false" bar.



- **CRSDepTime (int32):**
  - This represents the scheduled departure time of flights.
  - The histogram showcases the distribution of scheduled flight times across a 24-hour period.

- **DepDelayMinutes (double):**
  - This field indicates the number of minutes a flight was delayed at departure.
  - The distribution predominantly spikes around zero, suggesting most flights depart on time or with minimal delays.

- **DepTime (double):**
  - This reflects the actual departure time of flights.
  - Similar to CRSDepTime, the histogram shows a distribution of departure times throughout the day, highlighting periods of increased flight activity.



- **Distance (double):**

Represents the distance covered by flights.
The histogram displays the distribution of flight distances.

- **Diverted (boolean):**

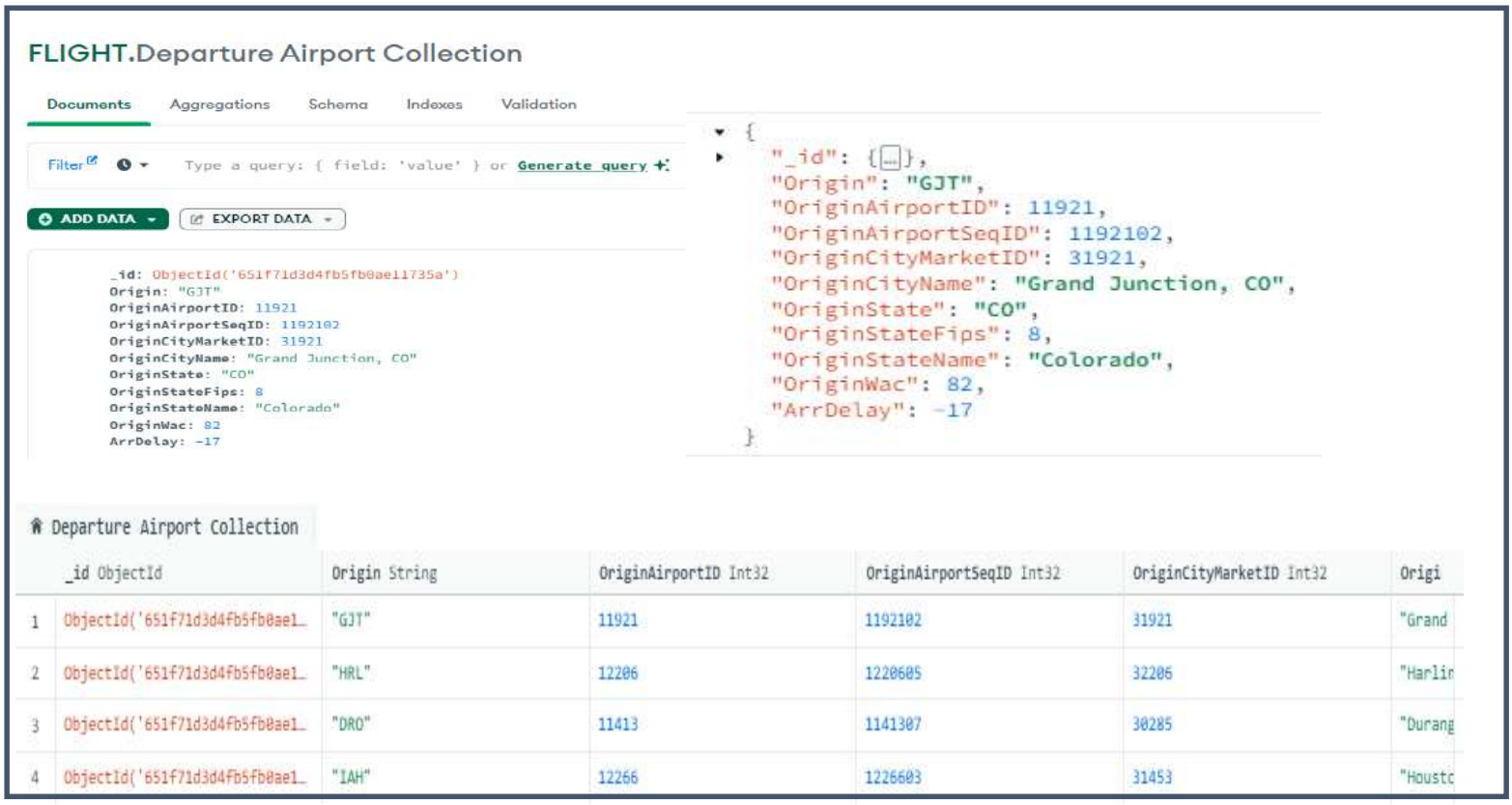
Indicates whether a flight was diverted from its original destination.
The elongated blue bar for 'false' suggests that the vast majority of flights in the dataset were not diverted.

- **FlightDate (date):**

Represents the dates on which the flights took place.
The histogram showcases the distribution of flights across specific dates.

5.6 Departure Airport Collection:

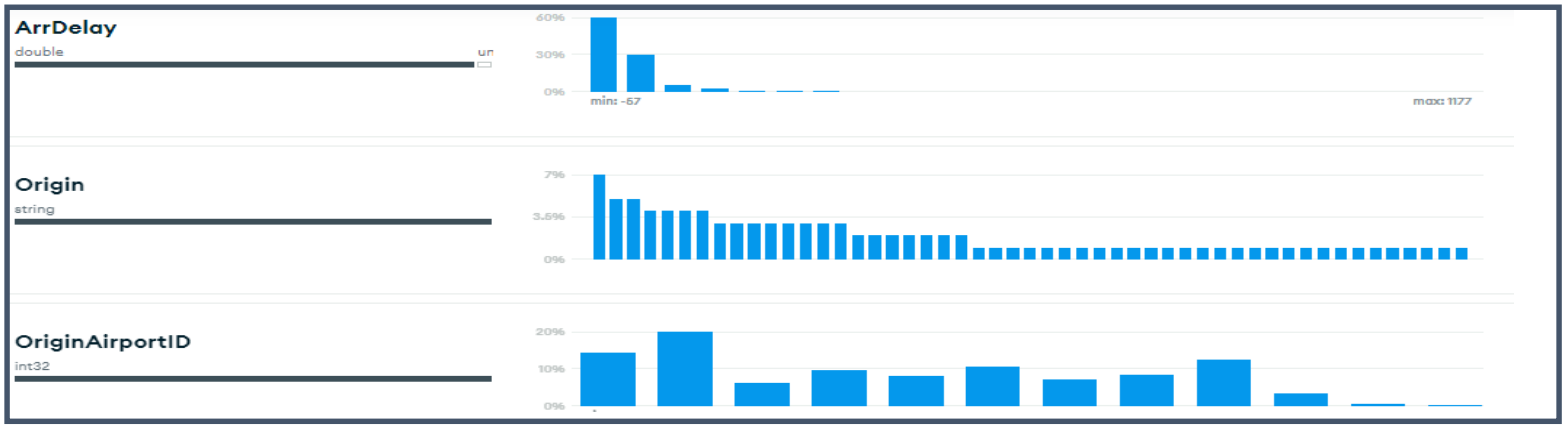
- **Basic Details:**
  - Origin: Main identifier or code of the departure airport.
  - OriginAirportID: Unique ID designated to the airport.
- **Hierarchical Identifiers:**
  - OriginAirportSeqID: Sequential identifier for the airport.
  - OriginCityMarketID: Identifier linking the airport to its broader city market.
- **Geographical Data:**
  - OriginCityName: Name of the city where the airport is located.
  - OriginState: Abbreviation or code of the state.
  - OriginStateFips: Federal Information Processing Standards code for the state.
  - OriginStateName: Full name of the state.
  - OriginWac: World Area Code for the airport's location.



Importance and uses of the " Departure Airport" collection:

- ✓ Unified Airport Information: Centralizes all key details about departure airports, simplifying data retrieval and management.
- ✓ Geographical Analysis: Facilitates studies on flight distribution across regions, states, or cities, aiding in route planning and market analysis.
- ✓ Operational Planning: By understanding origin airports' hierarchy and relations, airlines and logistic providers can strategize better for services, resource allocation, and infrastructure development.

AUTOMATIC SCHEMA – Departure Airport Collection



- **ArrDelay (double):**
  - This histogram represents the distribution of arrival delays.

- **Origin (string):**
  - This bar chart showcases the frequencies of flights originating from different airports or cities.
  - Each bar corresponds to a unique origin, and its height signifies the number of flights from that origin.

- **OriginAirportID (int32):**
  - This is another histogram displaying the distribution based on airport IDs.
  - The chart shows that some airport IDs have a notably higher frequency than others.

- **OriginAirportSeqID (int32):**

This histogram depicts the distribution of flight frequencies based on sequential airport IDs.

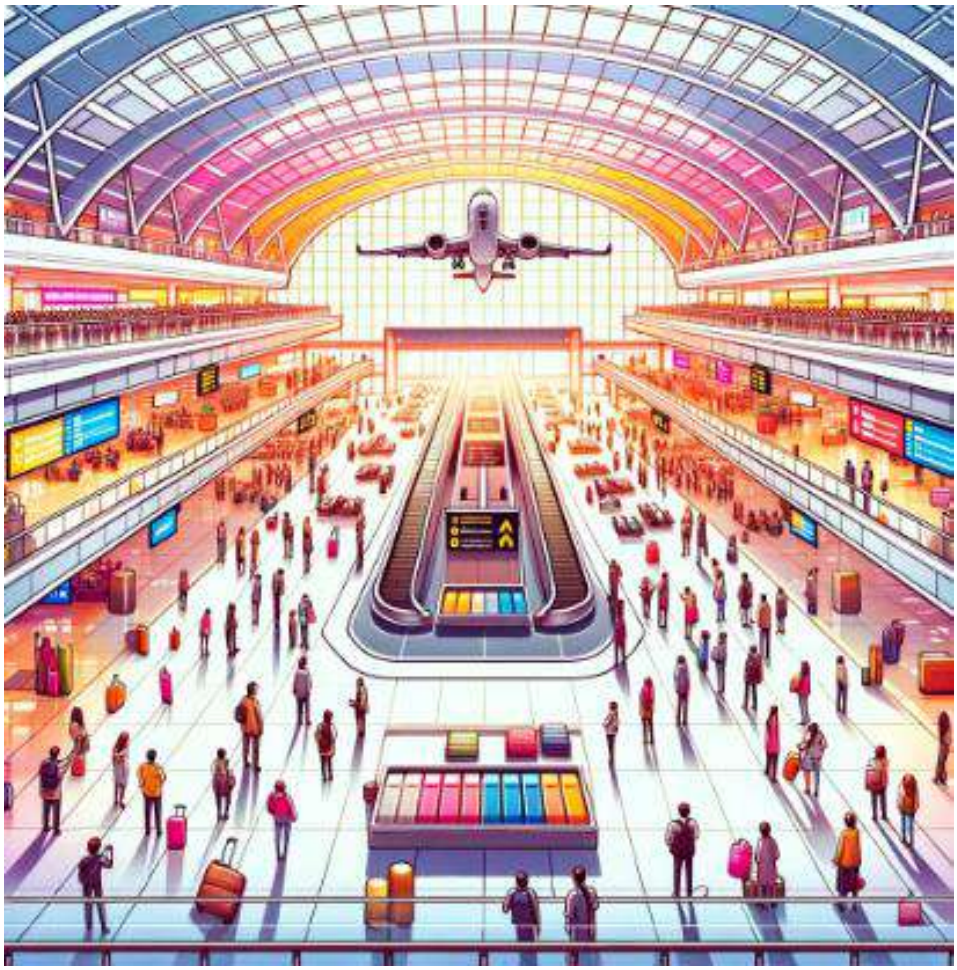
- **OriginCityMarketID (int32):**

This visualization showcases the flight frequencies based on city market IDs.

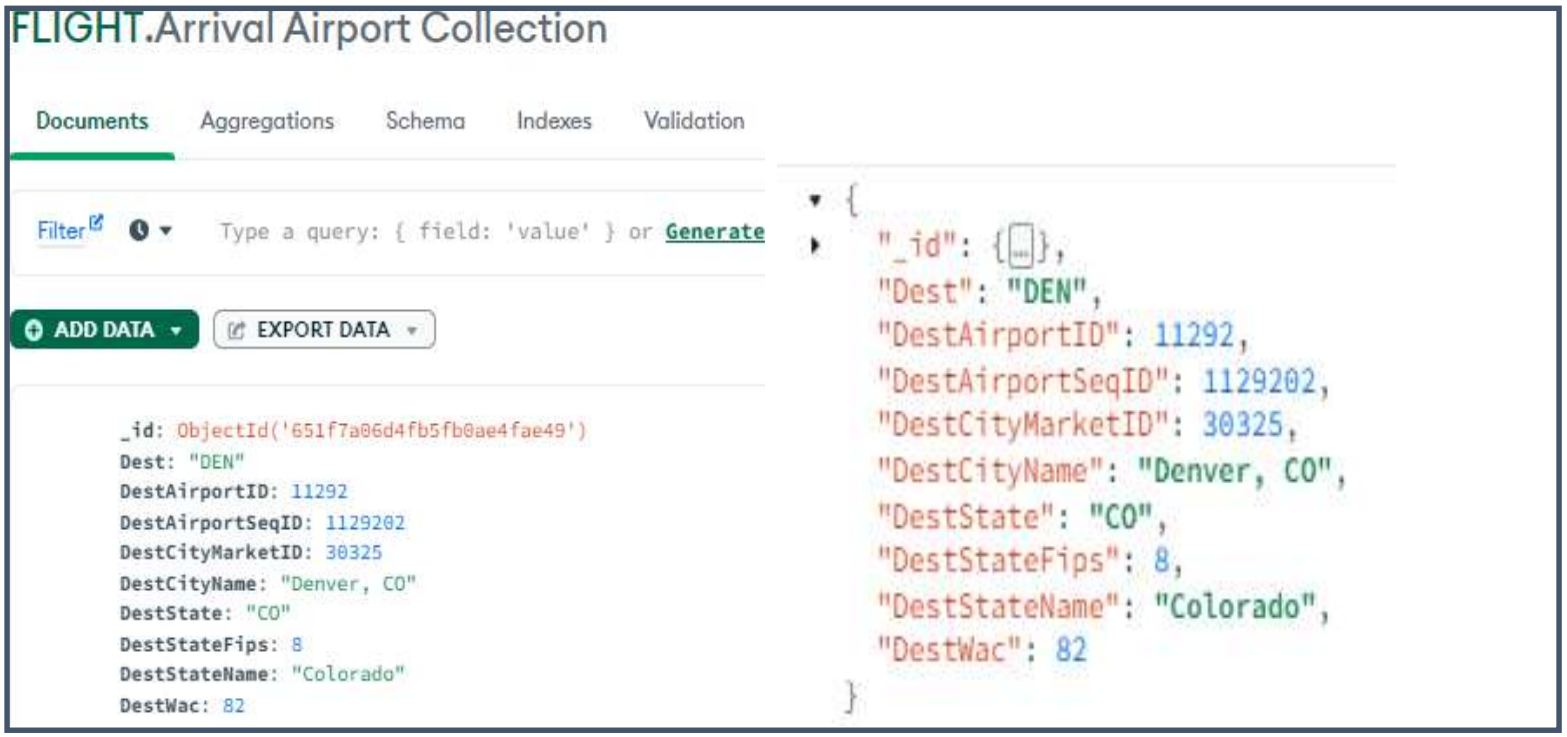
- **OriginCityName (string):**

This bar chart represents the number of flights originating from different cities.
Each bar corresponds to a distinct city name, and its height indicates the frequency of flights from that particular city.

5.7 Arrival Airport Collection:



- **Basic Details:**
  - Dest: Principal code or identifier of the arrival airport.
  - DestAirportID: Distinct ID attributed to the destination airport.
- **Hierarchical Identifiers:**
  - DestAirportSeqID: Consecutive identifier for the airport.
  - DestCityMarketID: Identifier connecting the airport to its wider city market region.
- **Geographical Data:**
  - DestCityName: Name of the city in which the airport is situated.
  - DestState: Code or abbreviation of the state.
  - DestStateFips: Federal Information Processing Standards code corresponding to the state.
  - DestStateName: Complete name of the state.
  - DestWac: World Area Code indicative of the airport's geographical position.



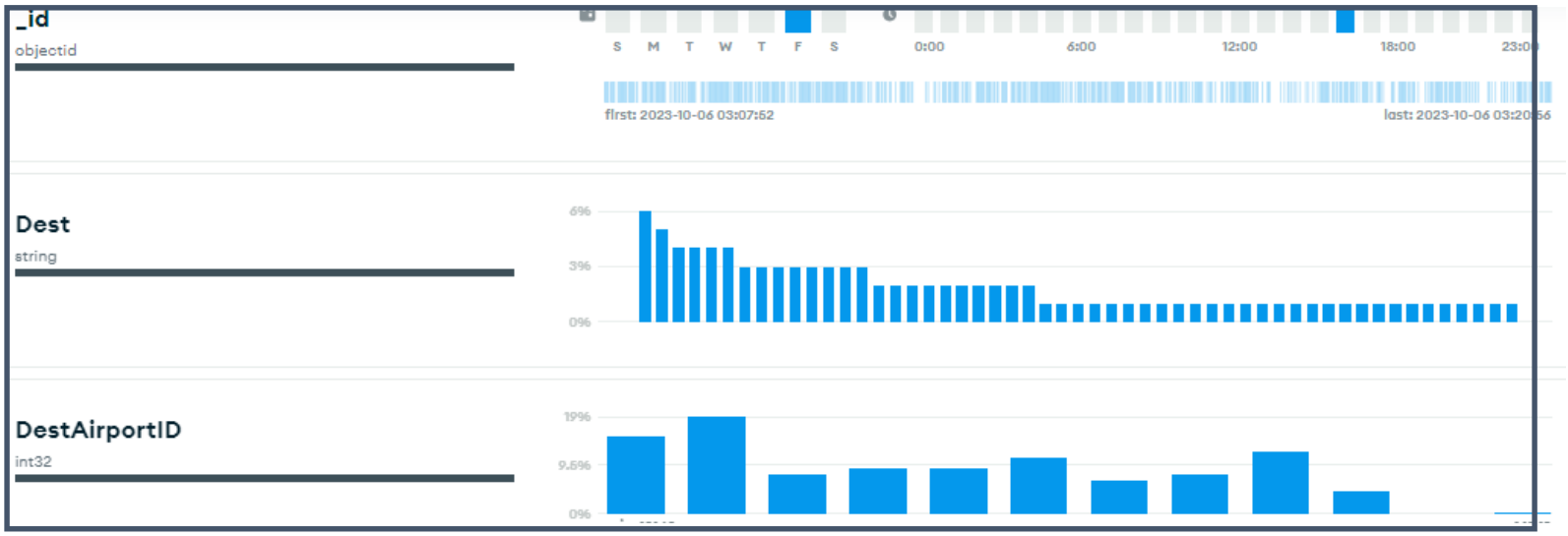
Arrival Airport Collection

	_id ObjectId	Dest String	DestAirportID Int32	DestAirportSeqID Int32	DestCityMarketID Int32
1	ObjectId('651f7a06d4fb5fb0ae4…	"DEN"	11292	1129202	30325
2	ObjectId('651f7a06d4fb5fb0ae4…	"IAH"	12266	1226603	31453
3	ObjectId('651f7a06d4fb5fb0ae4…	"DEN"	11292	1129202	30325
4	ObjectId('651f7a06d4fb5fb0ae4…	"GPT"	11973	1197302	31973

Importance and uses of the " Arrival Airport" collection:

- ✓ Centralized Destination Data: Provides a consolidated repository of vital information about arrival airports, streamlining data access and operations.
- ✓ Regional Traffic Analysis: Enables an understanding of flight distribution towards different destinations, crucial for market expansion and demand prediction.
- ✓ Infrastructure and Services: Assists airlines and airport authorities in gauging the prominence of specific destination airports, influencing decisions on amenities, infrastructure upgrades, and services.

AUTOMATIC SCHEMA – Arrival Airport Collection

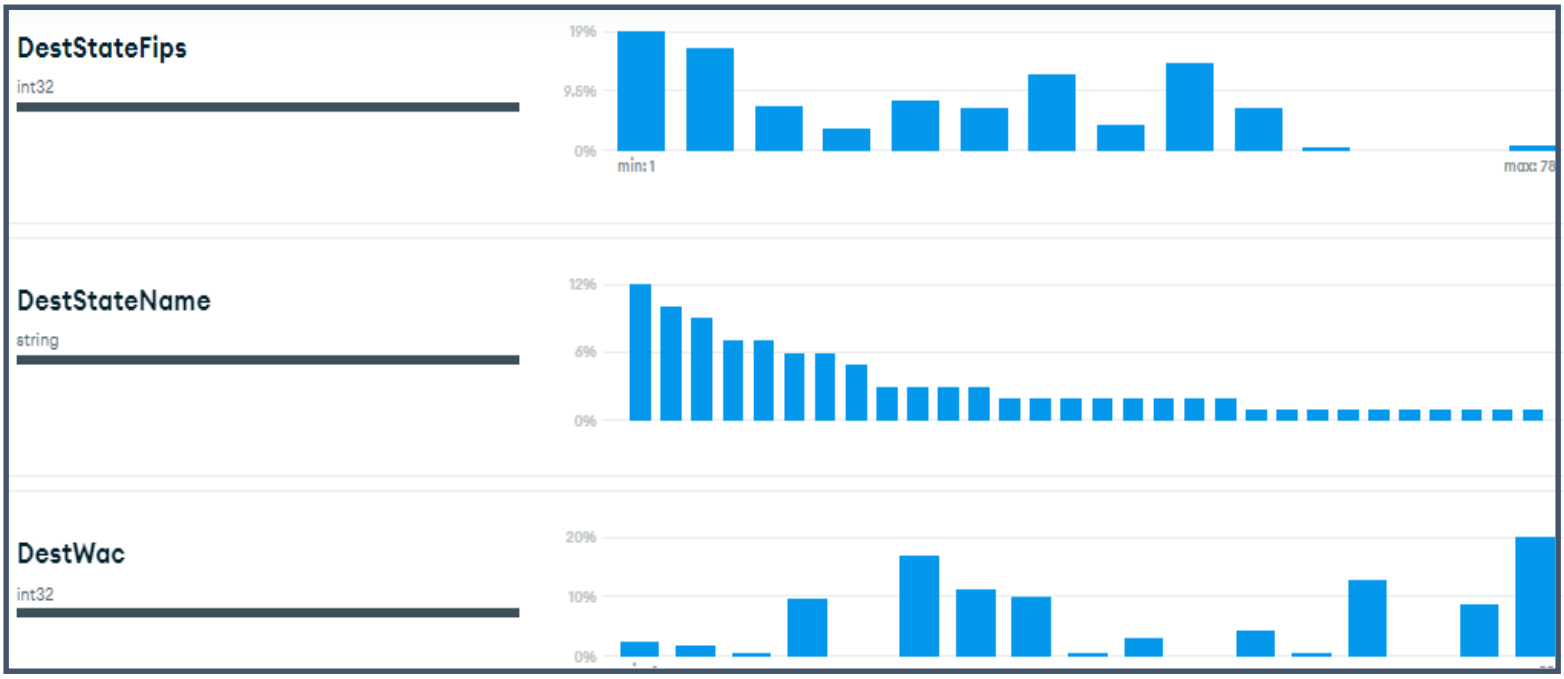


- **_id:**
  - This represents the unique identifier for each record or document in the database.
- **Dest (short for "Destination"):**
  - This represents the destination airport for flights.
  - The histogram displays the frequency of flights to various destinations. Some destinations have higher frequencies, indicated by taller bars, suggesting these are popular or major hubs.
- **DestAirportID:**
  - This signifies a unique identifier associated with each destination airport.
  - The histogram illustrates the distribution of flights to various airports by their IDs.

- **DestAirportSeqID:**
  - Represents the sequence ID uniquely associated with each destination airport.
  - The histogram displays the distribution of flights heading to various airports based on their sequence IDs.

- **DestCityMarketID:**
  - This identifier is associated with specific markets or regions that airports serve.
  - The histogram shows how often flights are heading to these specific markets or regions.

- **DestCityName:**
  - Represents the name of the destination city.



- **DestStateFips:**
  - Represents the Federal Information Processing Standard (FIPS) code associated with each destination state.
  - The histogram showcases the distribution of flights to various states using their FIPS codes.
- **DestStateName:**
  - Denotes the name of the destination state.
  - The histogram displays the frequency of flights headed to each state by name.
- **DestWac:**
  - Stands for World Area Code, which is a numerical code representing a specific geographic area.
  - The histogram offers a distribution of flights based on their destination's World Area Codes

VI. MongoDB – Flight 2022 QUERIES

6.1 Queries - Flight collections

- **Find all flights on a specific date:**

FLIGHT.Flight Performance Collection

DO

Documents | Aggregations | Schema | Indexes | Validation

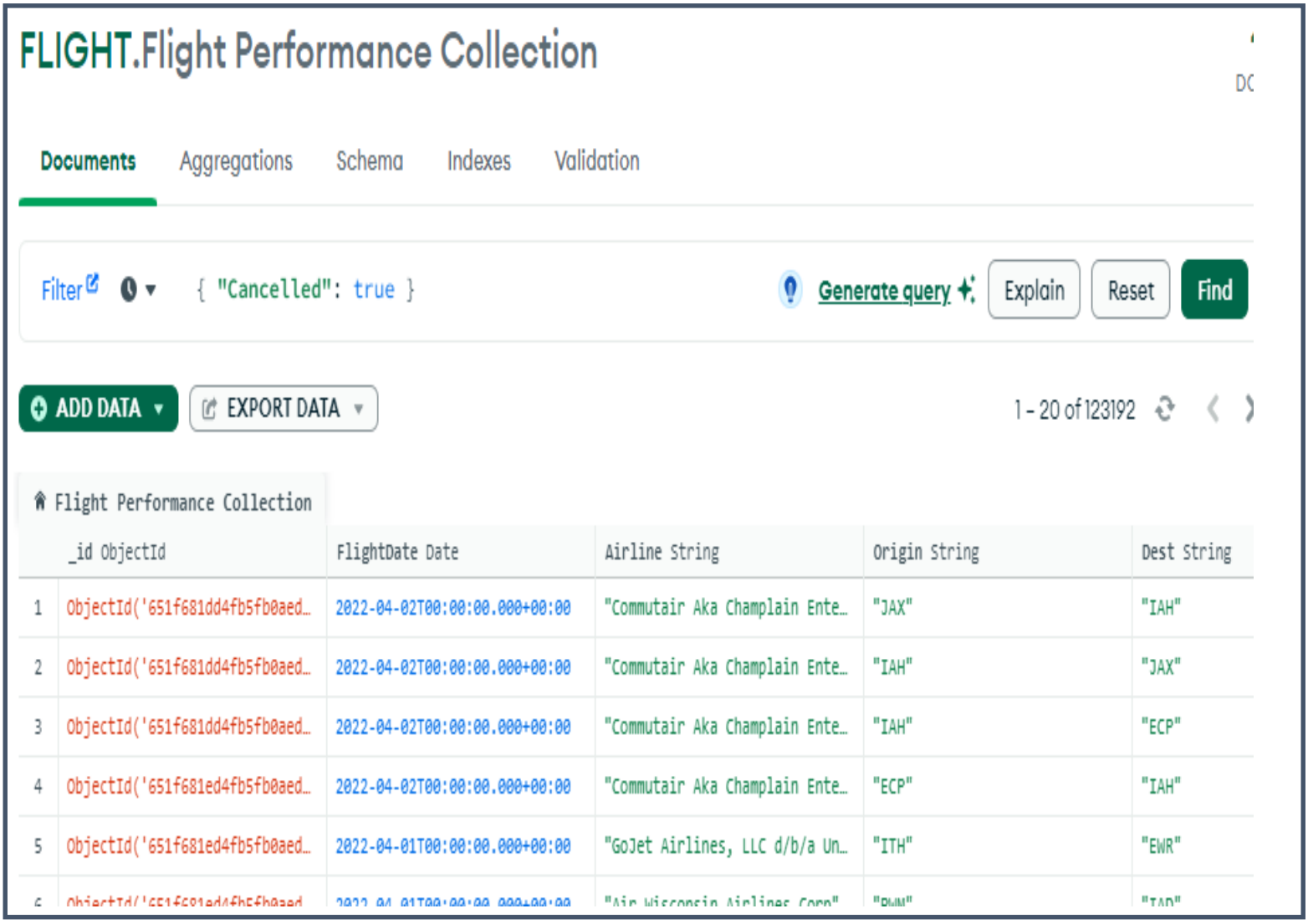
Filter { "FlightDate": ISODate("2022-05-04T00:00:00.000Z") } Generate query Explain Reset Find

ADD DATA EXPORT DATA 1 - 20 of N/A

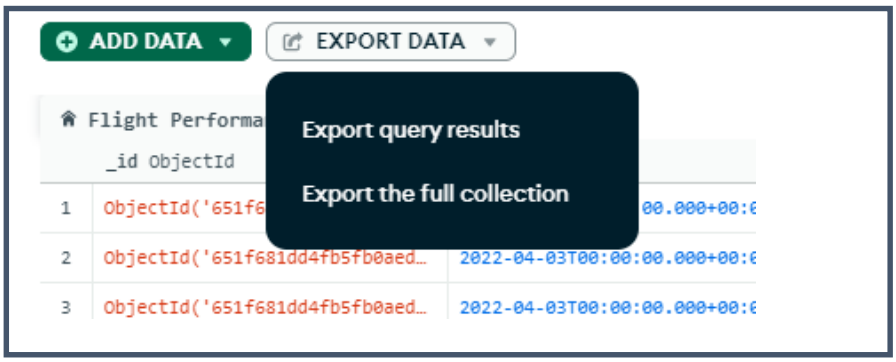
Flight Performance Collection

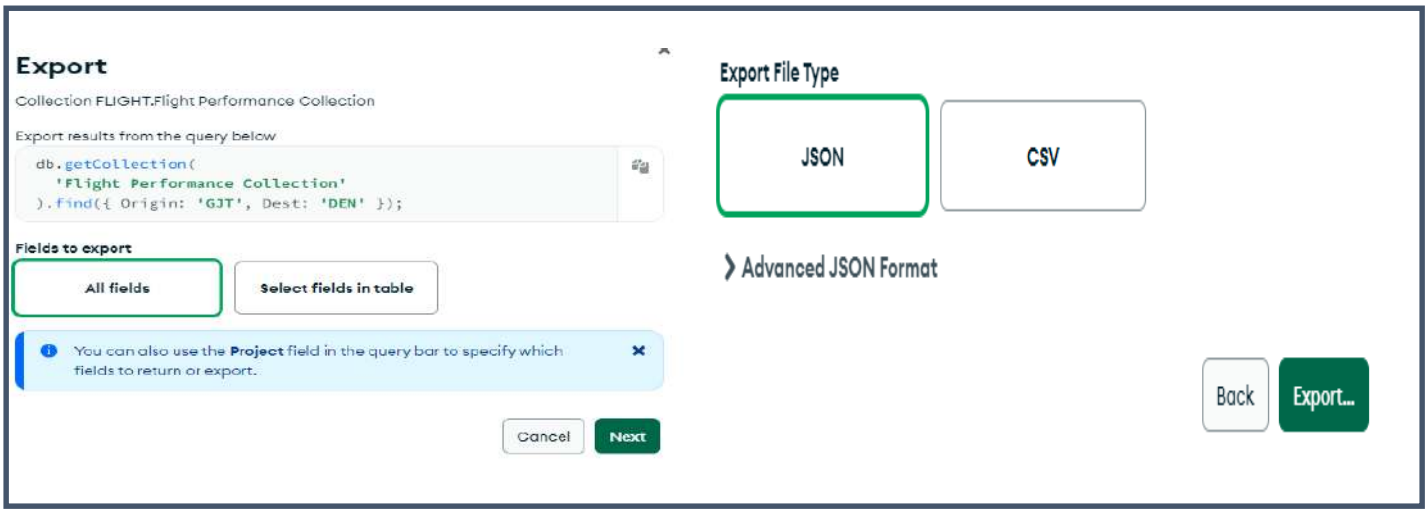
	_id ObjectId	FlightDate Date	Airline String	Origin String	Dest String
1	ObjectId('651f693bd4fb5fb0aee…	2022-05-04T00:00:00.000+00:00	"Comair Inc."	"DCA"	"ALB"
2	ObjectId('651f693bd4fb5fb0aee…	2022-05-04T00:00:00.000+00:00	"Comair Inc."	"CLT"	"TLH"
3	ObjectId('651f693bd4fb5fb0aee…	2022-05-04T00:00:00.000+00:00	"Comair Inc."	"TYS"	"CLT"
4	ObjectId('651f693bd4fb5fb0aee…	2022-05-04T00:00:00.000+00:00	"Comair Inc."	"CLT"	"BHM"

- **Find all cancelled flights:**



- **Queried Results can be exported:**





➢ **Output in JSON Format :**

FLIGHT.Flight Performance Collection.json

C: > Users > HP > Desktop > FLIGHT.Flight Performance Collection.json > ...


```
[{
  "_id": {
    "$oid": "651f681dd4fb5fb0aed3386a"
  },
  "FlightDate": {
    "$date": "2022-04-04T00:00:00.000Z"
  },
  "Airline": "Commutair Aka Champlain Enterprises, Inc.",
  "Origin": "GJT",
  "Dest": "DEN",
  "Cancelled": false,
  "Diverted": false,
  "CRSDepTime": 1133,
  "DepTime": 1123,
  "DepDelayMinutes": 0,
  "ArrDelayMinutes": 0,
  "Distance": 212
},
{
  "_id": {
    "$oid": "651f681dd4fb5fb0aed33a79"
  },
  "FlightDate": {
    "$date": "2022-04-03T00:00:00.000Z"
```

- **Output in CSV Format :**

_id	FlightDate	Airline	Origin	Dest	Cancelled	Diverted	CRSDepTi	DepTime	DepDelay	ArrDelayN	Distance
651f681dc	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1133	1123	0	0	212
651f681dc	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1526	1516	0	0	212
651f681dc	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1133	1131	0	0	212
651f681dc	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1810	1800	0	0	212
651f681dc	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1526	1518	0	0	212
651f681ec	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1133	1125	0	0	212
651f681ec	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1810	1809	0	0	212
651f681ec	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1526	1516	0	0	212
651f681ec	2022-04-0	Commuta	GJT	DEN	FALSE	FALSE	1133	1128	0	0	212

6.2 Queries - Flight Schedule collections

- **Find all flights on a specific date:**
  { "FlightDate": ISODate("2022-04-04T00:00:00.000Z") }
- **Find All Flights Operated by a Specific Airline:**
  { "Airline": "Commutair Aka Champlain Enterprises, Inc." }
- **Get All Flights Departing from a Specific Airport:**
  { "Origin": "GJT" }
- **Find All Flights Arriving at a Specific Destination:**
  { "Dest": "DEN" }
- **Retrieve All Flights Scheduled to Depart at a Specific Time:**
  { "CRSDepTime": 1133 }
- **Get All Flights Scheduled to Arrive at a Specific Time:**
  { "CRSArrTime": 1245 }
- **Find All Flights Between Two Specific Airports:**
  { "Origin": "GJT", "Dest": "DEN" }
- **Retrieve All Flights for a Specific Airline on a Particular Date:**
  { "Airline": "Commutair Aka Champlain Enterprises, Inc.", "FlightDate": ISODate("2022-04-04T00:00:00.000Z") }
- **Get Flights that are Scheduled to Depart and Arrive Between Two Specific Times:**
  { "CRSDepTime": { "$gte": 1000, "$lte": 1200 }, "CRSArrTime": { "$gte": 1300, "$lte": 1500 } }

6.3 Queries - Departure Airport Collections

- **Retrieve All Airports in a Specific State:**
  { "OriginState": "CA" }
- **Find the Full State Name for a Given Airport Identifier:**
  { "Origin": "GJT", "fields": { "OriginStateName": 1 } }
- **List All Airports Within a Specific World Area (using WAC):**
  { "OriginWac": 301 }
- **Retrieve the City and State for an Airport Using its Unique ID:**
  { "OriginAirportID": 123456, "fields": { "OriginCityName": 1 "OriginStateName": 1 } }
- **Find All Airports Linked to a Broader City Market Identifier:**
  { "OriginCityMarketID": 31234 }

6.4 Queries - Flight Delay Collection

- **Retrieve Flights with Departure Delays Greater Than a Certain Duration:**
  { "DepDelayMinutes": { "$gt": 60 } }
- **Find Flights of a Specific Airline with Arrival Delays:**
  { "Airline": "Delta", "ArrDelayMinutes": { "$gt": 0 } }
- **List Flights from a Specific Origin with Significant Departure Delays:**
  { "Origin": "JFK", "DepDelayMinutes": { "$gt": 120 } }
- **Find All Flights on a Specific Date with No Delays:**
  { "FlightDate": ISODate("2022-04-04T00:00:00.000Z"), "DepDelayMinutes": 0, "ArrDelayMinutes": 0 }
- **Retrieve Flights with Both Departure and Arrival Delays Greater Than a Certain Duration:**

{ "DepDelayMinutes": { "$gt": 30 }, "ArrDelayMinutes": { "$gt": 30 } }

- Numerous queries were executed across multiple collections to gain insights into the expansive flight data. These collections included:
  - ***Airline Info Collection***
  - ***Arrival Airport Collection***
  - ***Departure Airport Collection***
  - ***Flight Collections***
  - ***Flight Delay Collection***
  - ***Flight Performance Collection***
  - ***Flight Schedule Collection***

- Each query swiftly returned results, highlighting the efficiency of the database system.
- To further streamline the analysis and data sharing process, the results from these queries were then successfully exported to both JSON and CSV formats, ensuring versatile access and ease of use for subsequent data tasks.
- MONGOSH SHELL - In the mongosh environment, a series of comprehensive queries were executed across various collections.

- The efficiency of MongoDB was evident as each query was processed with remarkable speed, yielding prompt results.

6.5 The Aggregation Framework

- MongoDB Compass offers an "Aggregation Framework" for data transformation and summarization.
- The Aggregation option allows for building multi-stage pipelines, with real-time result previews.
- I tried using stages like $match, $group, and $sort for data processing.
- The GUI provides features like auto-completion and drag-and-drop for easy pipeline construction.
- Efficiency of MongoDB was evident as each query was processed with remarkable speed, yielding

➢ **Aggregation for the Flight dataset :**

This will give us the following

- Filter flights on a specific date.
- Group by airline.
- Calculate the average delay for both departure and arrival for each airline.
- Sort the results by the average departure delay.

➢ **Code used was :**

```
db.Flight.aggregate([
 // Step 1: Filter flights on a specific date
 {
  $match: {
   "FlightDate": ISODate("2022-04-04T00:00:00.000Z")
  }
 },

 // Step 2: Group by airline
 {
  $group: {
   _id: "$Airline", // Grouping key
   avgDepDelay: { $avg: "$DepDelayMinutes" },
   avgArrDelay: { $avg: "$ArrDelayMinutes" }
  }
 },
 // Step 3: Sort by average departure delay
 {
  $sort: {
   avgDepDelay: -1  // -1 for descending order
  }
 }
```

])

PIPELINE OUTPUT

OUTPUT OPTIONS

Sample of 10 documents


```
_id: ObjectId('651f4877d4fb5fb0ae18879d')
FlightDate: 2022-04-04T00:00:00.000+00:00
Airline: "Commutair Aka Champlain Enterprises, Inc."
Origin: "GJT"
Dest: "DEN"
DepDelayMinutes: 0
ArrDelayMinutes: 0
```


- ✓ When executed, this aggregation gave a list of airlines that had flights on the specified date, sorted by the average departure delay. Each entry will contain the airline's name (from _id), and the average departure and arrival delays for that airline on that date.

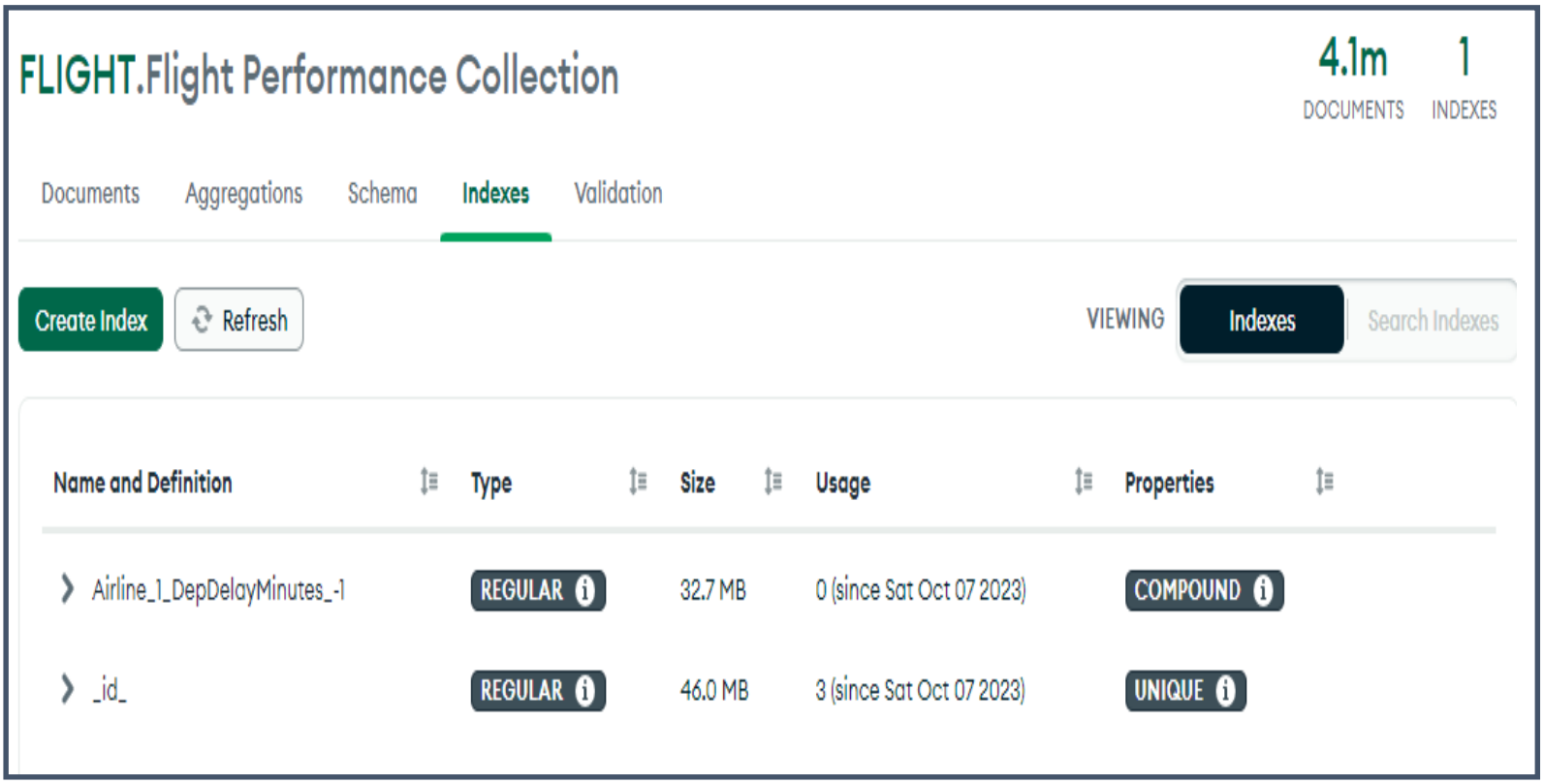
➢ 6.6 Indexes

0

- **Manage Indexes in Compass:** The "Index" option in MongoDB Compass allows users to view, create, and modify indexes on collections, enhancing query efficiency.
- **Index Creation & Modification:** Users can easily define field-specific indexes, select index types, and adjust properties via a graphical interface.
- **Performance Analysis:** Compass offers insights into index usage, aiding in decisions on index optimization and whether to retain or drop specific indexes.
-

➢ Illustration

- Using MongoDB Compass, created indexes on the Origin, Dest, and FlightDate fields in the flightPerformance collection for faster query performance.
- Indexed the Origin and Dest fields in ascending order, streamlining searches based on departure and arrival airports.
- Created an ascending index on FlightDate, queries targeting specific flight dates became highly efficient.



➢ 6.7 Validation

- MongoDB Compass's "Validation" ensured data consistency by enforcing predefined schemas for collections, maintaining data quality.
- Violations of the set validation rules were prevented during data insertion or update, with MongoDB providing detailed error feedback.
- The tool allowed custom validation expressions using MongoDB's query capabilities, enabling specific data checks.
- Through strict validation rules, Compass ensured data integrity, making sure data aligns with business and operational expectations

➢ Illustration

➢ Validation Rule for "Flight_Delay Collection" to ensure data integrity:

- FlightDate should be a valid date.
- Airline should be a non-empty string.
- Origin and Dest should be non-empty strings, representing valid airport codes.
- DepDelayMinutes and ArrDelayMinutes should be non-negative integers.

- **code**

```
{
  "$jsonSchema": {
    "bsonType": "object",
    "required": ["FlightDate", "Airline", "Origin", "Dest", "DepDelayMinutes",
"ArrDelayMinutes"],
    "properties": {
      "FlightDate": {
        "bsonType": "date",
        "description": "must be a valid date and is required"
      },
      "Airline": {
        "bsonType": "string",
        "minLength": 1,
        "description": "must be a non-empty string and is required"
      },
      "Origin": {
        "bsonType": "string",
        "minLength": 1,
        "description": "must be a non-empty string representing a valid airport code and
is required"
      },
      "Dest": {
        "bsonType": "string",
        "minLength": 1,
        "description": "must be a non-empty string representing a valid airport code and
is required"
      },
      "DepDelayMinutes": {
        "bsonType": "int",
        "minimum": 0,
        "description": "must be a non-negative integer and is required"
      },
      "ArrDelayMinutes": {
        "bsonType": "int",
        "minimum": 0,
        "description": "must be a non-negative integer and is required"
      }
```

  - **Now, any new document inserted or updated in the "Flight_Delay Collection" will be checked against these validation rules.**

VII. Limitations of MongoDb Flight 2022 Dataset

7.1 Limitations of MongoDB for the "Flight 2022" Dataset:

- **Data Integrity:** MongoDB's schema-less nature poses risks of data inconsistency. Proper validations are vital for datasets like "Flight 2022".
- **Joins and Transactions:** MongoDB's $lookup isn't as optimized as SQL joins. Complex flight data relations may challenge query efficiency.
- **Storage Overhead:** MongoDB's BSON format can lead to increased storage usage, a concern as the dataset expands.
- **Memory Usage:** If "Flight 2022" data exceeds RAM, performance might degrade.
- Write Durability: MongoDB's default write behavior can risk data loss, which may not suit critical datasets.
- **Complex Aggregations:** MongoDB might be less efficient for certain aggregations compared to SQL databases.
- **Scaling Limitations:** While sharding offers horizontal scaling, it introduces complexity and might not solve all performance issues.

7.2 Overcoming These Limitations:

- **Data Integrity:** Implement strict schema validation using MongoDB's validation rules to ensure consistency.
- **Joins and Transactions:** Design data in a way that minimizes the need for joins, and use optimized aggregation pipelines.
- **Storage Overhead:** Regularly clean up and archive old data, and consider data compression techniques.
- **Memory Usage:** Regularly monitor and scale your RAM resources based on the dataset's growth.
- **Write Durability**: Adjust the Write Concern settings to ensure data is written to disk immediately.
- **Complex Aggregations:** Consider hybrid solutions with SQL databases or optimize MongoDB aggregation pipelines.
- **Scaling Limitations:** Invest in training and tools to manage sharding effectively and monitor system performance continuously.

VIII. Conclusion

- MongoDB's strengths lie in its ability to store, retrieve, and manipulate large datasets, with tools like the aggregation framework enhancing data transformation capabilities.
- Despite its strengths, MongoDB faces challenges such as data integrity and efficient joins, emphasizing the need for robust validation and optimized data design.
- The "Flight 2022" dataset underlined the complexity of the aviation sector and the pivotal role of databases in offering structured insights to stakeholders.
- Through this analysis, it becomes evident that technology, MongoDB for data management, plays a crucial role in simplifying and providing meaningful interpretations of complex datasets.

References:

1. https://www.kaggle.com/datasets
2. https://www.mongodb.com/docs/compass/current/
3. https://www.youtube.com/watch?v=bJSj1a84I20
4. https://www.youtube.com/watch?v=ydXCcLAi5aU
5. https://medium.com/@okonkwoebuka456/database-design-project-building-a-business-database-from-scratch-9f9b48944f97
6. https://www.integrate.io/blog/complete-guide-to-database-schema-design-guide/
7. https://www.stat.auckland.ac.nz/~paul/ItDT/HTML/node42.html
8. https://www.mongodb.com/docs/atlas/sample-data/
9. https://www.mongodb.com/docs/atlas/sample-data/sample-weather/
10. https://www.mongodb.com/basics/sample-database
11. https://medium.com/dbkoda/mongodb-sample-collections-52d6a7745908
12. https://restheart.org/docs/mongodb-rest/sample-data

Other Resources (LLM) used

1. Grammerly
2. Google Image Search, Canvas
3. Bard & Bing