CASE STUDY1: JOB DATA ANALYSIS

Project Description:

The project stresses on analysis of job data for insights into the job review patterns, throughput, and language share analysis.

Approach:

1. Created the table job_data with columns job_id, actor_id, event, language, time_spent, org, and ds using the database case_study1.
2. Entries are added randomly utilizing the SOL functions.
3. Executed SQL queries for each task.
4. Reviewed obtained outputs to identify patterns.
5. Drew insights and interpretations from the results.

Tech-Stack Used:

1. MySQL Workbench: For executing the SQL queries.
2. SQL: For data analysis.

Queries:

```sql
-- job data analysis
show databases;
create database case_study1;
use case_study1;

CREATE TABLE job_data (
    job_id INT PRIMARY KEY,
    actor_id INT,
    event VARCHAR(20),
    language VARCHAR(20),
    time_spent INT,
    org VARCHAR(50),
    ds DATE
);
```

```sql
-- Insert 10,000 random rows into job_data
INSERT INTO job_data (job_id, actor_id, event, language, time_spent, org, ds)
SELECT
    t.n,
    FLOOR(1 + (RAND() * 1000)),
    ELT(FLOOR(1 + (RAND() * 3)), 'decision', 'skip', 'transfer'),
    ELT(FLOOR(1 + (RAND() * 5)), 'English', 'Spanish', 'French', 'German', 'Chinese'),
    FLOOR(1 + (RAND() * 300)),
    ELT(FLOOR(1 + (RAND() * 5)), 'OrgA', 'OrgB', 'OrgC', 'OrgD', 'OrgE'),
    DATE_ADD('2020-11-01', INTERVAL FLOOR(RAND() * 30) DAY)
FROM (
    SELECT a.N + b.N * 10 + c.N * 100 + d.N * 1000 AS n
    FROM
        (SELECT 0 N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNI
        (SELECT 0 N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNI
        (SELECT 0 N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNI
        (SELECT 0 N UNION SELECT 1 UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5 UNION SELECT 6 UNION SELECT 7 UNION SELECT 8 UNI
    LIMIT 100000
) t;
```

```sql
select * from job_data limit 0,10000;

-- jobs reviewed over time
-- throughput analysis
```

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

job_id	actor_id	event	language	time_spent	org	ds
9983	888	transfer	Spanish	89	OrgD	2020-11-24
9984	197	decision	English	252	OrgA	2020-11-25
9985	285	transfer	Spanish	221	OrgE	2020-11-19
9986	669	decision	English	245	OrgD	2020-11-17
9987	832	decision	German	230	OrgD	2020-11-12
9988	201	decision	Chinese	286	OrgB	2020-11-10
9989	340	decision	German	234	OrgE	2020-11-20
9990	488	decision	German	245	OrgD	2020-11-30
9991	625	decision	French	92	OrgB	2020-11-14
9992	478	skip	English	164	OrgD	2020-11-29
9993	491	decision	Chinese	52	OrgB	2020-11-19
9994	68	transfer	French	279	OrgB	2020-11-04
9995	302	skip	Spanish	61	OrgA	2020-11-27
9996	587	decision	French	188	OrgD	2020-11-21
9997	960	skip	English	81	OrgA	2020-11-17
9998	200	skip	Spanish	39	OrgE	2020-11-05
9999	781	decision	Spanish	54	OrgE	2020-11-07
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```sql
-- throughput analysis
-- language share analysis
-- duplicate rows detection

-- jobs reviewed/day for nov 2020
SELECT
    ds,
    COUNT(job_id) AS jobs_reviewed
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds
ORDER BY ds;

-- 7-day rolling avg of throughput or daily metric
```

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

ds	jobs_reviewed
2020-11-01	331
2020-11-02	347
2020-11-03	358
2020-11-04	339
2020-11-05	343
2020-11-06	332
2020-11-07	354
2020-11-08	336
2020-11-09	304

```sql
-- 7-day rolling avg of throughput or daily metric
WITH daily_events AS (
    SELECT
        ds,
        COUNT(*) / 86400 AS events_per_second
    FROM job_data
    GROUP BY ds
)
SELECT
    ds,
    AVG(events_per_second) OVER (
        ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) AS rolling_avg_throughput
FROM daily_events;
```

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

ds	rolling_avg_throughput
2020-11-01	0.00380000
2020-11-02	0.00390000
2020-11-03	0.00396667
2020-11-04	0.00395000
2020-11-05	0.00396000
2020-11-06	0.00393333
2020-11-07	0.00395714
2020-11-08	0.00397143
2020-11-09	0.00390000
2020-11-10	0.00392857
2020-11-11	0.00391429

Result 26

```sql
-- percentage share/language over last 30 days
WITH last_30_days AS (
    SELECT language, COUNT(*) AS language_count
    FROM job_data
    WHERE ds >= DATE_SUB('2020-11-30', INTERVAL 30 DAY)
    GROUP BY language
)
SELECT
    language,
    language_count,
    (language_count * 100.0) / SUM(language_count) OVER () AS l
FROM last_30_days
ORDER BY language_share_percentage DESC;
```

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

language	language_count	language_share_percentage
French	2036	20.33966
Chinese	2020	20.17982
German	1993	19.91009
English	1992	19.90010
Spanish	1969	19.67033

```sql
-- display duplicate rows
SELECT
    job_id, actor_id, event, language, time_spent, org, ds,
    COUNT(*) AS duplicate_count
FROM job_data
GROUP BY job_id, actor_id, event, language, time_spent, org, ds
HAVING COUNT(*) > 1;
-- no duplicates in the dataset
-- adding the duplicates
-- Insert 10 duplicate rows into job_data
-- Insert EXACT duplicates 10 times
INSERT INTO job_data (job_id, actor_id, event, language, time_spent, org, ds)
VALUES
(11001, 201, 'decision', 'English', 120, 'OrgA', '2020-11-10'),
(11002, 202, 'skip', 'Spanish', 80, 'OrgB', '2020-11-11'),
```

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

	job_id	actor_id	event	language	time_spent	org	ds	duplicate_count

```
(11001, 201, 'decision', 'English', 120, 'OrgA', '2020-11-10'),
(11002, 202, 'skip', 'Spanish', 80, 'OrgB', '2020-11-11'),
(11003, 203, 'transfer', 'French', 150, 'OrgC', '2020-11-12'),
(11004, 204, 'decision', 'German', 200, 'OrgD', '2020-11-13'),
(11005, 205, 'skip', 'Chinese', 90, 'OrgE', '2020-11-14');

-- checking for duplicates again
SELECT
    actor_id, event, language, org, ds,
    COUNT(*) AS duplicate_count
FROM job_data
GROUP BY actor_id, event, language, org, ds
HAVING duplicate_count > 1;
```

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

actor_id	event	language	org	ds	duplicate_count
61	decision	Spanish	OrgE	2020-11-08	2
958	transfer	German	OrgE	2020-11-12	2
784	transfer	Spanish	OrgE	2020-11-09	2
902	decision	French	OrgC	2020-11-01	2
987	transfer	French	OrgE	2020-11-23	2
761	skip	English	OrgC	2020-11-11	2
451	decision	Chinese	OrgC	2020-11-14	2
98	transfer	Spanish	OrgC	2020-11-23	2
686	transfer	French	OrgE	2020-11-06	2
286	skip	Chinese	OrgA	2020-11-08	2
498	transfer	Spanish	OrgE	2020-11-20	2
758	decision	English	OrgC	2020-11-09	2
532	skip	Spanish	OrgC	2020-11-20	2
701	skip	Chinese	OrgA	2020-11-06	2
314	decision	French	OrgA	2020-11-18	2

Result 29

Insights:

1. Jobs reviewed/day: the daily review count shows job activity patterns. Spikes indicate high-priority days/ system anomalies while dips highlight holidays/outages.
2. Throughput analysis: the rolling average overcome daily fluctuations, hence providing long-term patterns. It is preferred over daily metrics for identifying consistent patterns whereas in case of spotting sudden spikes or dips, daily metric should be preferred.
3. Language share analysis: highest for French, least for Spanish.

4. Duplicate rows detection: for data inconsistency identification. There were no duplicate rows initially, so after inserting exact duplicates, checking whether the query works correctly. It did.

Results:

1. Developed and executed SQL queries for all tasks.
2. Provided insights into job review patterns, throughput trends, and language shares.
3. Validated duplicate detection with sample data.

CASE STUDY2: INVESTIGATING METRIC SPIKE

Project description:

This project focuses on analysing user engagement, growth, retention, and email activity to identify and investigate metric spikes. The goal is to uncover meaningful insights that can guide business decisions and product improvements. SQL queries were used to extract and analyse data from three key datasets: users, events, and email_events.

Approach:

1. Data Exploration: Reviewed the structure of the three datasets:
    - users: Contains user information, including signup dates.
    - events: Logs user actions, including timestamps and device information.
    - email_events: Tracks email interactions with columns for user actions and timestamps.
2. SQL Query Development: Constructed queries to address key metrics such as user engagement, growth, and email interaction.
3. Results Analysis: Evaluated outputs for trends, spikes, and anomalies.

4. Report Compilation: Documented queries, outputs, and insights.

Tech-stack used:

1. MySQL Workbench: For writing and executing SQL queries.
2. SQL: Structured Query Language for data analysis

Queries:

```
123     -- weekly user engagement
124 •   SELECT
125         YEAR(e.occured_at) AS year,
126         WEEK(e.occured_at) AS week,
127         COUNT(DISTINCT e.user_id) AS active_users
128     FROM events e
129     GROUP BY year, week
130     ORDER BY year, week;
```

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

year	week	active_users
2014	17	663
2014	18	1068
2014	19	1113
2014	20	1154
2014	21	1121
2014	22	1186
2014	23	1232
2014	24	1275
2014	25	1264
2014	26	1302
2014	27	1372
2014	28	1365
2014	29	1376
2014	30	1467
2014	31	1299
2014	32	1225
2014	33	1225
2014	34	1204
2014	35	104

```sql
-- user growth analysis
SELECT
    YEAR(created_at) AS year,
    MONTH(created_at) AS month,
    COUNT(user_id) AS new_users
FROM users
GROUP BY year, month
ORDER BY year, month;
```

Result Grid | Filter Rows: | Export: | Wr

year	month	new_users
2013	1	160
2013	2	160
2013	3	150
2013	4	181
2013	5	214
2013	6	213
2013	7	284
2013	8	316
2013	9	330
2013	10	390
2013	11	399
2013	12	486
2014	1	552
2014	2	525
2014	3	615
2014	4	726
2014	5	779
2014	6	873
2014	7	997
2014	8	1031

Result 2

```sql
-- weekly retention analysis
WITH signup_cohort AS (
    SELECT
        user_id,
        DATE(created_at) AS signup_date
    FROM users
),
weekly_engagement AS (
    SELECT
        u.user_id,
        s.signup_date,
        YEAR(e.occured_at) AS year,
        WEEK(e.occured_at) AS week
    FROM events e
    JOIN users u ON e.user_id = u.user_id
    JOIN signup_cohort s ON u.user_id = s.user_id
)
SELECT
    signup_date,
    year,
    week,
    COUNT(DISTINCT user_id) AS retained_users
FROM weekly_engagement
GROUP BY signup_date, year, week
ORDER BY signup_date, year, week;
```

signup_date	year	week	retained_users
2013-01-01	2014	17	1
2013-01-01	2014	18	1
2013-01-01	2014	19	2
2013-01-01	2014	20	2
2013-01-01	2014	21	1
2013-01-01	2014	22	1
2013-01-01	2014	23	1
2013-01-01	2014	24	2
2013-01-01	2014	25	2
2013-01-01	2014	26	1
2013-01-01	2014	27	1
2013-01-01	2014	30	2
2013-01-01	2014	31	1
2013-01-02	2014	17	1
2013-01-02	2014	18	2
2013-01-02	2014	19	1
2013-01-02	2014	20	1
2013-01-02	2014	21	1
2013-01-02	2014	22	2
2013-01-02	2014	23	2

```sql
-- weekly engagement per device
SELECT
    YEAR(e.occured_at) AS year,
    WEEK(e.occured_at) AS week,
    e.device,
    COUNT(DISTINCT e.user_id) AS active_users
FROM events e
GROUP BY year, week, e.device
ORDER BY year, week, active_users DESC;
```

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

year	week	device	active_users
2014	17	macbook pro	143
2014	17	lenovo thinkpad	86
2014	17	iphone 5	65
2014	17	macbook air	54
2014	17	samsung galaxy s4	52
2014	17	dell inspiron notebook	46
2014	17	iphone 5s	42
2014	17	nexus 5	40
2014	17	ipad air	27
2014	17	asus chromebook	21
2014	17	iphone 4s	21
2014	17	acer aspire notebook	20
2014	17	ipad mini	19
2014	17	dell inspiron desktop	18
2014	17	nexus 7	18
2014	17	nokia lumia 635	17
2014	17	htc one	16
2014	17	nexus 10	16

```sql
-- email engagement analysis
SELECT DISTINCT action
FROM email_events;

SELECT
    DATE(occured_at) AS email_date,
    COUNT(*) AS total_emails_sent,
    SUM(CASE WHEN action = 'email_open' THEN 1 ELSE 0 END) AS emails_opened,
    SUM(CASE WHEN action = 'email_clickthrough' THEN 1 ELSE 0 END) AS emails_clicked,
    ROUND((SUM(CASE WHEN action = 'email_open' THEN 1 ELSE 0 END) * 100.0) / COUNT(*), 2) AS open_rate,
    ROUND((SUM(CASE WHEN action = 'email_clickthrough' THEN 1 ELSE 0 END) * 100.0) / COUNT(*), 2) AS click_rate
FROM email_events
```

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

email_date	total_emails_sent	emails_opened	emails_clicked	open_rate	click_rate
2014-05-01	680	145	61	21.32	8.97
2014-05-02	704	142	82	20.17	11.65
2014-05-03	73	23	23	31.51	31.51
2014-05-04	68	22	21	32.35	30.88
2014-05-05	1164	255	115	21.91	9.88
2014-05-06	757	168	82	22.19	10.83
2014-05-07	647	141	63	21.79	9.74
2014-05-08	709	156	65	22.00	9.17
2014-05-09	687	148	64	21.54	9.32
2014-05-10	69	22	20	31.88	28.99
2014-05-11	86	29	25	33.72	29.07
2014-05-12	1174	258	117	21.98	9.97
2014-05-13	807	193	92	23.92	11.40
2014-05-14	667	151	62	22.64	9.30

Result 7

Insights:

1. Weekly user engagement trends highlight active user patterns. Spikes may indicate successful feature launches or marketing campaigns.
2. Growth trends show the pace at which users join the platform. Sharp increases may point to effective onboarding strategies or viral events.
3. Retention metrics help identify how well the platform retains users after sign-up. Strong weekly retention suggests user satisfaction and engagement.
4. Device-level engagement provides insights into user preferences, helping prioritize mobile or desktop experiences.
5. Tracking email action rates evaluates the effectiveness of email campaigns. Low engagement may signal the need for improved content or targeting.

Results:

1. Successfully executed SQL queries to investigate user engagement, growth, retention, and email activity.

2. Identified potential metric spikes linked to user actions and email interactions.
3. Delivered actionable insights for product and marketing teams.