

## Task 1

The dimensional modelling data warehouse design (star schema) created for the dataset is to help keep the data warehouse normalised and simple. This will ensure that the business users easily understand the relationship between tables (dimensions and facts).

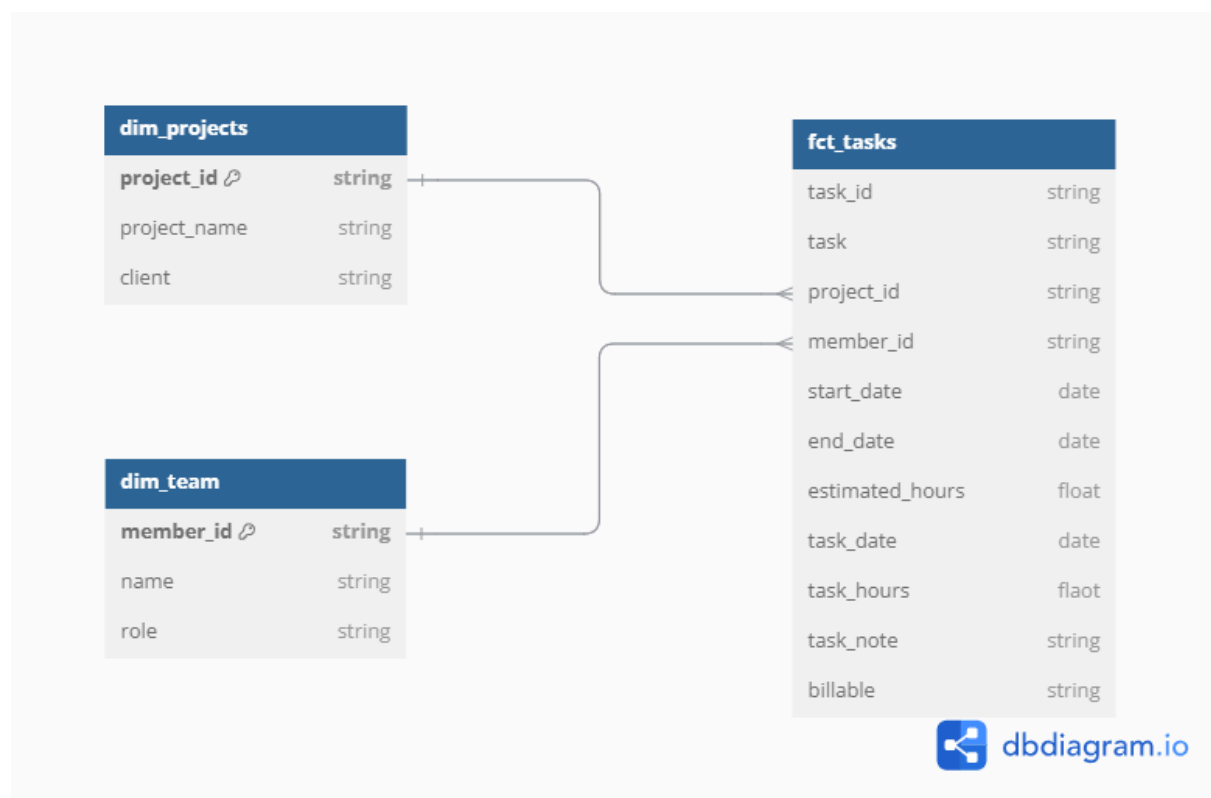
I developed a star schema design from the dataset by creating two dimension tables and a fact table. A brief explanation of the design of this table is provided below:

**Dim\_projects:** This table ensures to keep track of projects assigned by each client. Hence, the business can easily track what projects are assigned by clients without having to get the information from the task table.

**Dim\_team:** This table helps to track members and their roles within the organisation. With this table, we can query information about a member and his/her role.

**Fct\_tasks:** This table is the fact table that logs the hours spent by a team member on a particular task on a particular date. It also has estimated hours that the member is expected to finish the task.

The Snapshot of the design is as below:



The design separates dimension attributes from the facts and helps to avoid any data redundancy. To keep the integrity of the tables, I implemented foreign key referential integrity to ensure that any projects or members logged in the fct\_tasks should have existed in the

dim\_project and dim\_team table respectively. The snapshot of the implementation is provided below:

```
---- Dimensional modelling design
-- create dim_projects table sql
CREATE TABLE IF NOT EXISTS dim.dim_projects (
    project_id VARCHAR(200) PRIMARY KEY,
    project VARCHAR(200) NOT NULL,
    client VARCHAR(200)
);

-- create dim_team table sql
CREATE TABLE IF NOT EXISTS dim.dim_team (
    member_id VARCHAR(200) PRIMARY KEY,
    name VARCHAR(200) NOT NULL,
    role VARCHAR(200)
);

# create fct_tasks table sql
CREATE TABLE IF NOT EXISTS fact.fct_tasks (
    task_id VARCHAR(200),
    task VARCHAR(200) NOT NULL,
    project_id VARCHAR(200) REFERENCES dim.dim_projects(project_id),
    member_id VARCHAR(200) REFERENCES dim.dim_team(member_id),
    start_date DATE,
    end_date DATE,
    estimated_hours FLOAT,
    task_date DATE NOT NULL,
    task_hours FLOAT NOT NULL,
    task_note VARCHAR(250),
    billable VARCHAR(5)
);
```

## Task 2

In order to optimise the performance of the given sql query, I created an index on the name column on both the clickup and float table. This is to help improve the join operation as it does not need to do a sequential scan. Another improvement was to create a multi-column index/clustered index on both name and role columns in the float table, doing this will help the database engine improve shuffling when grouping by those two columns. In order to make use of this clustered index, I called the name column from the float table instead of the click up table.

The snapshot of the index creation is as below:


```
-- Create Indexes on the tables
-- Create index on 'Name' in ClickUp table
CREATE INDEX idx_clickup_name ON raw.clickup_opt (name);

-- Create index on 'Name' in Float table
CREATE INDEX idx_float_name ON raw.float_opt (name);

-- Create a multi-column index on 'Name' and 'Role' in the Clickup table
CREATE INDEX idx_float_name_role ON raw.float_opt (name, role);
```

The performance improvement after taking the above steps is as shown below:

### Explain plan of the initial query:

	QUERY PLAN text	
1	Sort (cost=39.59..39.79 rows=80 width=444) (actual time=1.094..1.096 rows=6 loops=1)	
2	Sort Key: (sum(f.estimated_hours)) DESC	
3	Sort Method: quicksort Memory: 25kB	
4	-> HashAggregate (cost=34.07..37.07 rows=80 width=444) (actual time=1.065..1.070 rows=6 loops=1)	
5	Group Key: c.name, f.role	
6	Filter: (sum(c.hours) > '100'::double precision)	
7	Batches: 1 Memory Usage: 37kB	
8	Rows Removed by Filter: 2	
9	-> Hash Join (cost=10.68..29.51 rows=456 width=444) (actual time=0.085..0.547 rows=998 loops=1)	
10	Hash Cond: ((c.name)::text = (f.name)::text)	
11	-> Seq Scan on clickup c (cost=0.00..12.56 rows=456 width=18) (actual time=0.010..0.118 rows=456 loops=1)	
12	-> Hash (cost=10.30..10.30 rows=30 width=844) (actual time=0.065..0.065 rows=13 loops=1)	
13	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
14	-> Seq Scan on "float" f (cost=0.00..10.30 rows=30 width=844) (actual time=0.053..0.057 rows=13 loops=1)	
15	Planning Time: 0.526 ms	
16	Execution Time: 1.153 ms	

## Explain plan of the optimised table and query:

	QUERY PLAN text	
1	Sort (cost=31.39..31.42 rows=10 width=852) (actual time=0.660..0.661 rows=6 loops=1)	
2	Sort Key: (sum(f.estimated_hours)) DESC	
3	Sort Method: quicksort Memory: 25kB	
4	-> HashAggregate (cost=30.85..31.23 rows=10 width=852) (actual time=0.649..0.651 rows=6 loops=1)	
5	Group Key: f.name, f.role	
6	Filter: (sum(c.hours) > '100'::double precision)	
7	Batches: 1 Memory Usage: 24kB	
8	Rows Removed by Filter: 2	
9	-> Nested Loop (cost=0.15..26.29 rows=456 width=852) (actual time=0.103..0.425 rows=998 loops=1)	
10	-> Seq Scan on clickup_opt c (cost=0.00..12.56 rows=456 width=18) (actual time=0.009..0.044 rows=456 loops=1)	
11	-> Memoize (cost=0.15..0.27 rows=1 width=844) (actual time=0.000..0.000 rows=2 loops=456)	
12	Cache Key: c.name	
13	Cache Mode: logical	
14	Hits: 448 Misses: 8 Evictions: 0 Overflows: 0 Memory Usage: 2kB	
15	-> Index Scan using idx_float_name_role on float_opt f (cost=0.14..0.26 rows=1 width=844) (actual time=0.012..0.012 rows=2 loops=8)	
16	Index Cond: ((name)::text = (c.name)::text)	
17	Planning Time: 0.243 ms	
18	Execution Time: 0.705 ms	

From the snapshots provided below, the initial query has execution time of 1.153 ms while the improved table & query has an execution plan of 0.705 ms. The difference in execution time has reduced by almost 40% which is very significant if it is to be a large dataset. Partitioning was not used in the optimization because where clause wasn't applied.

## Task 3

To improve the performance of the pyspark job for the processing of the logs data for scalability & efficiency, there were different considerations as highlighted below:

- Increased the number of workers
- Increased the number of cores and executors
- Increased memory used per worker nodes
- Ensure any worker node and executor is not idle
- Increase parallelism
- Made use of memory cache where applicable to avoid reading the same records from disk multiple times

Initial worker configuration:

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241018155911-172.22.0.3-7000	172.22.0.3:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20241018155912-172.22.0.4-7000	172.22.0.4:7000	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Upgraded worker configuration

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-20241018170352-172.22.0.3-7000	172.22.0.3:7000	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20241018170353-172.22.0.4-7000	172.22.0.4:7000	ALIVE	2 (0 Used)	1024.0 MiB (0.0 B Used)	

After the configuration for the spark cluster was updated and the spark script optimised for efficient processing, below is the change in runtime of the spark jobs:

app-20241018163042-0000	Spark Assessment	2	1024.0 MiB		2024/10/18 16:30:42	root	FINISHED	9.1 min
-------------------------	------------------	---	------------	--	---------------------	------	----------	---------

The initial configuration completed the job in about 9.1 minutes.

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20241018175900-0004	Spark Assessment	4	1024.0 MiB		2024/10/18 17:59:00	root	FINISHED	4.9 min

The optimized script and upgraded spark cluster configuration was completed in 4.9 minutes. This optimization made the script run for approximately 46% less of the initial time.

Task 4

I have given a detailed explanation of the dimensional model design implemented for this case study in task 1.

For the operational use by the business, it is important to keep data as compact as possible to avoid data redundancy. In order to ensure this, the model was designed to break the data into several dimensions and even sub-dimensions. The snapshot of the design is as shown below:

