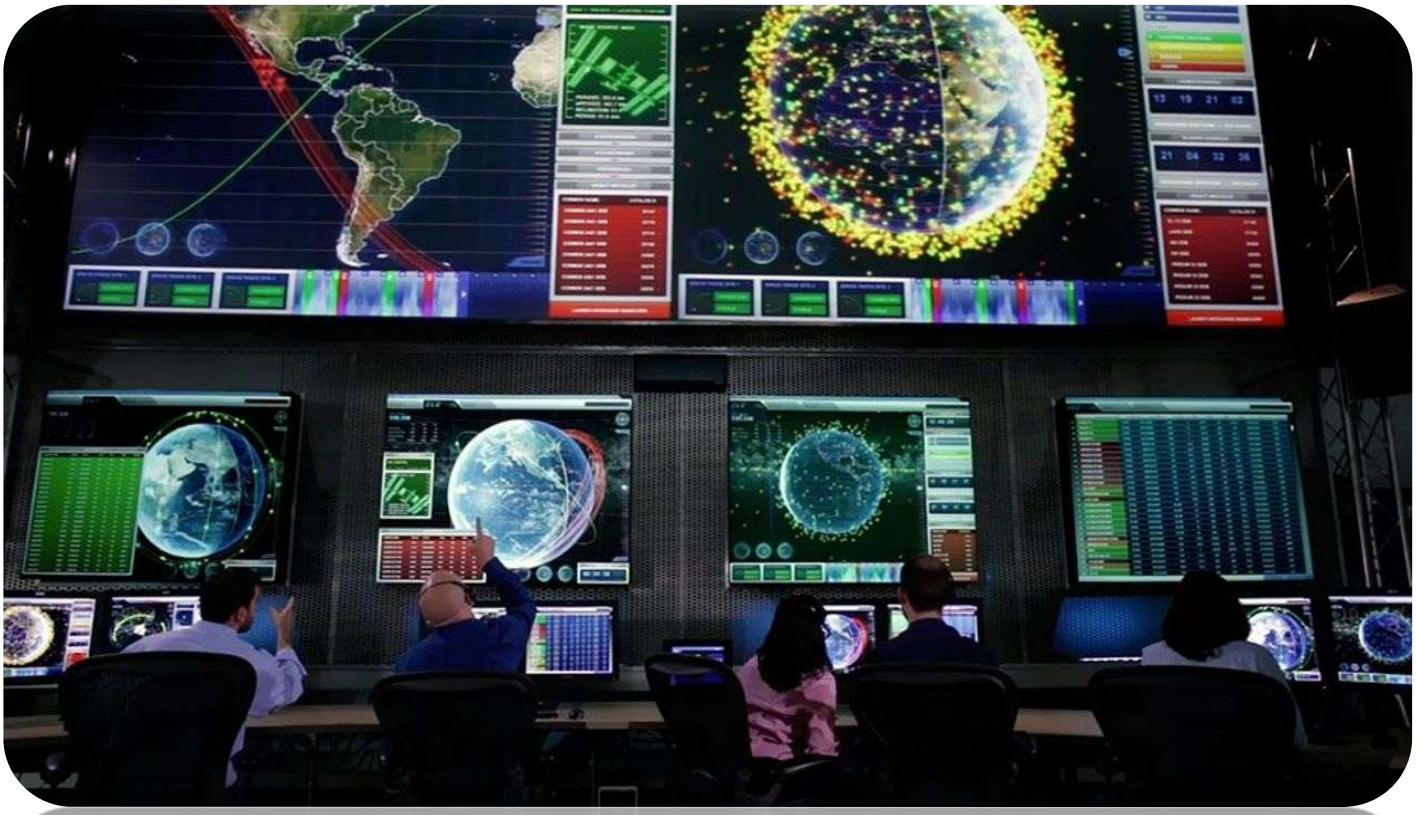
OrbitIQ

A Relational Data Engine for
Advanced Space Mission Analytics
Using SQL



- Disha Tarlekar | BATCH- T341 / DSDA10

PROJECT AIM

To design and implement a structured, SQL-based analytical system called *OrbitIQ* that captures, stores, and analyses space mission data — including mission details, crew assignments, targets, and launch parameters — enabling advanced query-based insights through relational modelling, joins, subqueries, and window functions.

OBJECTIVES

1. **To design and structure a relational database model** for managing key aspects of space missions including mission details, targets, crew information, and launch data.
2. **To implement optimized data handling using SQL**, with a focus on normalization, primary and foreign key relationships, and data integrity.
3. **To develop a wide range of analytical SQL queries** using JOINs, subqueries, CASE statements, and window functions to extract actionable insights from the mission data.
4. **To analyse mission performance, cost efficiency, success rates, and crew productivity** using custom SQL logic and derived metrics.
5. **To create a reusable, scalable foundation** for future integration with BI tools, dashboards, or real-time mission tracking systems.

.

ER DIAGRAM

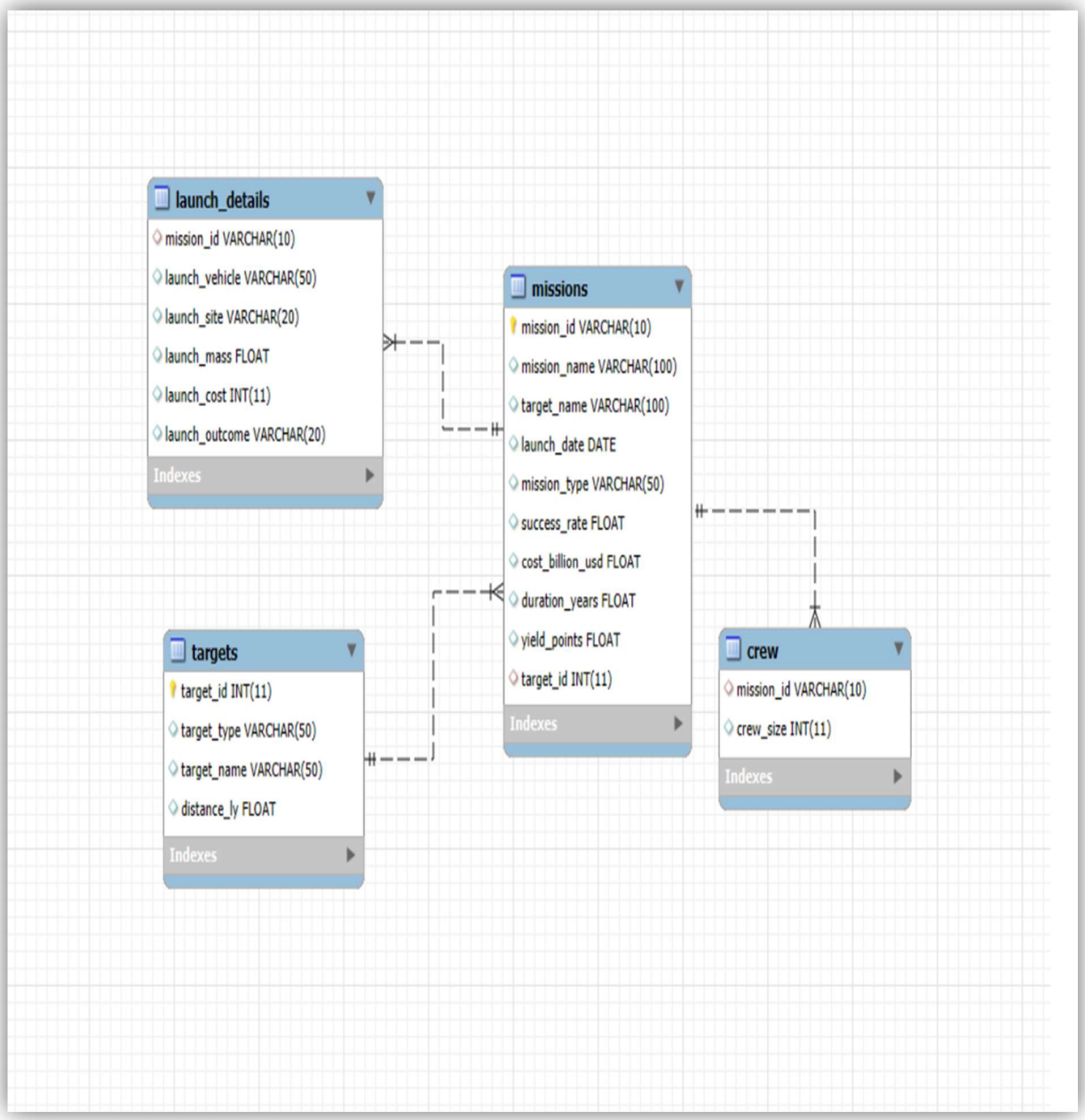

launch_details
mission_id VARCHAR(10)
launch_vehicle VARCHAR(50)
launch_site VARCHAR(20)
launch_mass FLOAT
launch_cost INT(11)
launch_outcome VARCHAR(20)
Indexes
missions
mission_id VARCHAR(10)
mission_name VARCHAR(100)
target_name VARCHAR(100)
launch_date DATE
mission_type VARCHAR(50)
success_rate FLOAT
cost_billion_usd FLOAT
duration_years FLOAT
yield_points FLOAT
target_id INT(11)
Indexes
targets
target_id INT(11)
target_type VARCHAR(50)
target_name VARCHAR(50)
distance_ly FLOAT
Indexes
crew
mission_id VARCHAR(10)
crew_size INT(11)
Indexes

TABLE DESCRIPTION

1) TARGETS:

Field	Type	Null	Key	Default	Extra
target_id	int(11)	NO	PRI	NULL	auto_increment
target_type	varchar(50)	YES		NULL	
target_name	varchar(50)	YES		NULL	
distance_ly	float	YES		NULL	

2) MISSIONS:

Field	Type	Null	Key	Default	Extra
mission_id	varchar(10)	NO	PRI	NULL	
mission_name	varchar(100)	YES		NULL	
target_name	varchar(100)	YES		NULL	
launch_date	date	YES		NULL	
mission_type	varchar(50)	YES		NULL	
success_rate	float	YES		NULL	
cost_billion_usd	float	YES		NULL	
duration_years	float	YES		NULL	
yield_points	float	YES		NULL	
target_id	int(11)	YES	MUL	NULL	

3) CREW:

Field	Type	Null	Key	Default	Extra
mission_id	varchar(10)	YES	MUL	NULL	
crew_size	int(11)	YES		NULL	

4) LAUNCH_DETAILS:

Field	Type	Null	Key	Default	Extra
mission_id	varchar(10)	YES	MUL	NULL	
launch_vehicle	varchar(50)	YES		NULL	
launch_site	varchar(20)	YES		NULL	
launch_mass	float	YES		NULL	
launch_cost	int(11)	YES		NULL	
launch_outcome	varchar(20)	YES		NULL	

CREATING DATABASE -

```
CREATE DATABASE SpaceMissionAnalytics;
USE SpaceMissionAnalytics;
```

TABLE CREATION & INSERTION COMMANDS -

1) Create Table Targets:

```
CREATE TABLE targets (
   target_id INT AUTO_INCREMENT PRIMARY KEY,
   target_type VARCHAR (50),
   target_name VARCHAR (50),
   distance_ly FLOAT
);
```

Inserting Values into Targets:

```
INSERT INTO targets (target_id, target_type, target_name, distance_ly) VALUES
(1, 'Star', 'Titan', 7.05),
(2, 'Exoplanet', 'Betelgeuse', 41.76),
(3, 'Asteroid', 'Mars', 49.22),
(4, 'Exoplanet', 'Titan', 26.33),
(5, 'Exoplanet', 'Proxima b', 8.67),
(6, 'Moon', 'Ceres', 13.69),
(7, 'Asteroid', 'Ceres', 1.02),
(8, 'Asteroid', 'Mars', 45.72),
(9, 'Asteroid', 'Betelgeuse', 5.98),
(10, 'Exoplanet', 'Betelgeuse', 28.87),…;
```

OUTPUT:

target_id	target_type	target_name	distance_ly
1	Star	Titan	7.05
2	Exoplanet	Betelgeuse	41.76
3	Asteroid	Mars	49.22
4	Exoplanet	Titan	26.33
5	Exoplanet	Proxima b	8.67
6	Moon	Ceres	13.69
7	Asteroid	Ceres	1.02
8	Asteroid	Mars	45.72
9	Asteroid	Betelgeuse	5.98
10	Exoplanet	Betelgeuse	28.87

2) Create Table Missions:

```
CREATE TABLE missions (
    mission_id VARCHAR(10) PRIMARY KEY,
    mission_name VARCHAR(100),
    target_name varchar(100),
    launch_date DATE,
    mission_type VARCHAR(50),
    success_rate FLOAT,
    cost_billion_usd FLOAT,
    duration_years FLOAT,
    yield_points FLOAT,
    target_id INT,
    FOREIGN KEY (target_id) REFERENCES targets(target_id)
);
```

Inserting Values into Missions:

INSERT INTO missions (mission_id, mission_name, target_name, launch_date, mission_type, success_rate, cost_billion_usd, duration_years, yield_points) **VALUES**
('MSN-0001', 'Mission-1', 'Titan', '2025-01-01 00:00:00', 'Colonization', 100.0, 526.68, 5.2, 64.3),
('MSN-0002', 'Mission-2', 'Betelgeuse', '2025-01-08 00:00:00', 'Colonization', 89.6, 234.08, 23.0, 84.4),
('MSN-0003', 'Mission-3', 'Mars', '2025-01-15 00:00:00', 'Exploration', 98.6, 218.68, 28.8, 98.6),
('MSN-0004', 'Mission-4', 'Titan', '2025-01-22 00:00:00', 'Colonization', 90.0, 232.89, 17.8, 36.0),
('MSN-0005', 'Mission-5', 'Proxima b', '2025-01-29 00:00:00', 'Mining', 73.2, 72.14, 9.2, 96.5),
('MSN-0006', 'Mission-6', 'Ceres', '2025-02-05 00:00:00', 'Colonization', 100.0, 452.42, 8.8, 45.1),…;

OUTPUT:

mission_id	mission_name	target_name	launch_date	mission_type	success_rate	cost_billion_usd	duration_years	yield_points	target_id
MSN-0001	Mission-1	Titan	2025-01-01	Colonization	100	526.68	5.2	64.3	NULL
MSN-0002	Mission-2	Betelgeuse	2025-01-08	Colonization	89.6	234.08	23	84.4	NULL
MSN-0003	Mission-3	Mars	2025-01-15	Exploration	98.6	218.68	28.8	98.6	NULL
MSN-0004	Mission-4	Titan	2025-01-22	Colonization	90	232.89	17.8	36	NULL
MSN-0005	Mission-5	Proxima b	2025-01-29	Mining	73.2	72.14	9.2	96.5	NULL
MSN-0006	Mission-6	Ceres	2025-02-05	Colonization	100	452.42	8.8	45.1	NULL

3) Create Table Crew:

CREATE TABLE crew (
  mission_id VARCHAR(10),
  crew_size INT,
  FOREIGN KEY (mission_id) REFERENCES missions(mission_id)
);

Inserting Values into Crew:

```
INSERT INTO crew (mission_id, crew_size) VALUES
('MSN-0001', 21),
('MSN-0002', 72),
('MSN-0003', 16),
('MSN-0004', 59),
('MSN-0005', 31),
('MSN-0006', 42),…;
```

OUTPUT:

	mission_id	crew_size
▶	MSN-0001	21
	MSN-0002	72
	MSN-0003	16
	MSN-0004	59
	MSN-0005	31
	MSN-0006	42

5) Create Table Launch_details:

```
CREATE TABLE launch_details (
    mission_id VARCHAR(10),
    launch_vehicle VARCHAR(50),
    launch_site varchar(20),
    launch_mass FLOAT,
    launch_cost int,
    launch_outcome varchar(20),
    FOREIGN KEY (mission_id) REFERENCES missions(mission_id)
);
```

Inserting Values into Launch_details:

INSERT INTO launch_details (mission_id, launch_vehicle, launch_site, launch_mass, launch_cost, launch_outcome) **VALUES**
('MSN-0001', 'SLS', 'Satish Dhawan Space Centre', 100511.88, 526680, 'Successful'),
('MSN-0002', 'Starship', 'Cape Canaveral', 49917.41, 234080, 'Partial Failure'),
('MSN-0003', 'Starship', 'Cape Canaveral', 41028, 218680, 'Successful'),
('MSN-0004', 'Starship', 'Baikonur Cosmodrome', 43239.05, 232890, 'Successful'),
('MSN-0005', 'Starship', 'Baikonur Cosmodrome', 13292.76, 72140, 'Partial Failure'),
('MSN-0006', 'Ariane 6', 'Xichang Satellite Launch Center', 89767.29, 452420, 'Successful'),…;

OUTPUT:

mission_id	launch_vehicle	launch_site	launch_mass	launch_cost	launch_outcome
MSN-0001	SLS	Satish Dhawan Space	100512	526680	Successful
MSN-0002	Starship	Cape Canaveral	49917.4	234080	Partial Failure
MSN-0003	Starship	Cape Canaveral	41028	218680	Successful
MSN-0004	Starship	Baikonur Cosmodrome	43239.1	232890	Successful
MSN-0005	Starship	Baikonur Cosmodrome	13292.8	72140	Partial Failure
MSN-0006	Ariane 6	Xichang Satellite La	89767.3	452420	Successful

BASIC SQL QUERIES

1. Total Missions per Mission Type.

```
SELECT mission_type, COUNT(*) AS total_missions
FROM missions
GROUP BY mission_type
ORDER BY total_missions DESC;
```

OUTPUT:

mission_type	total_missions
Research	132
Exploration	127
Colonization	125
Mining	116

2. Top 5 Most Expensive Missions.

```
SELECT mission_name, cost_billion_usd
FROM missions
ORDER BY cost_billion_usd DESC LIMIT 5;
```

OUTPUT:

mission_name	cost_billion_usd
Mission-207	538.32
Mission-488	532.62
Mission-422	527.77
Mission-1	526.68
Mission-149	526.51

3. Top 3 Most Frequently Used Launch Vehicles

SELECT launch_vehicle, COUNT(*) AS usage_count
FROM launch_details
GROUP BY launch_vehicle
ORDER BY usage_count **DESC LIMIT 3**;

OUTPUT:

	launch_vehicle	usage_count
▶	Ariane 6	124
	Falcon Heavy	121
	Starship	140

4. Classify Mission Outcome Using CASE

SELECT mission_name, success_rate,
CASE WHEN success_rate >= 90 **THEN** 'Successful' **WHEN** success_rate >= 60 **THEN** 'Partial Failure' ELSE 'Failed' END AS mission_outcome
FROM missions;

OUTPUT:

	mission_name	success_rate	mission_outcome
▶	Mission-1	100	Successful
	Mission-2	89.6	Partial Failure
	Mission-3	98.6	Successful
	Mission-4	90	Successful
	Mission-5	73.2	Partial Failure
	Mission-6	100	Successful

5. Total Launch Cost by Launch Site

```
SELECT launch_site, SUM(launch_cost) AS total_cost_million_usd
FROM launch_details
GROUP BY launch_site
ORDER BY total_cost_million_usd DESC;
```

OUTPUT:

launch_site	total_cost_million_usd
Cape Canaveral	23239520
Vandenberg Air Force	20942590
Xichang Satellite La	20725780
Tanegashima Space Ce	20484870
Guiana Space Centre	20309820
Baikonur Cosmodrome	16993520

6. Return All Missions Sorted by Yield Efficiency

```
SELECT mission_name, yield_points, cost_billion_usd,
ROUND (yield_points / cost_billion_usd, 2) AS yield_per_billion
FROM missions
ORDER BY yield_per_billion DESC;
```

OUTPUT:

mission_name	yield_points	cost_billion_usd	yield_per_billion
Mission-156	66.2	13.32	4.97
Mission-291	90.9	26.83	3.39
Mission-162	82.4	28.26	2.92
Mission-231	52.5	19.43	2.70
Mission-80	90.4	38.07	2.37
Mission-487	76.8	36.82	2.09

JOINS / WINDOW FUNCTION QUERIES

1. Missions with Yield Per Crew Member.

```
SELECT m.mission_name, m.yield_points / c.crew_size AS yield_per_astronaut
FROM missions m
JOIN crew c ON m.mission_id = c.mission_id
ORDER BY yield_per_astronaut DESC LIMIT 10;
```

OUTPUT:

mission_name	yield_per_astronaut
Mission-372	91.69999694824219
Mission-377	78.19999694824219
Mission-393	67.0999984741211
Mission-35	65.9000015258789
Mission-489	48
Mission-258	47.20000076293945

2. Rank Missions by Cost Using Window Function.

```
SELECT mission_name, cost_billion_usd,
RANK() OVER (ORDER BY cost_billion_usd DESC) AS cost_rank
FROM missions;
```

OUTPUT:

mission_name	cost_billion_usd	cost_rank
Mission-207	538.32	1
Mission-488	532.62	2
Mission-422	527.77	3
Mission-1	526.68	4
Mission-149	526.51	5
Mission-166	524.47	6

3. Average Mission Success by Launch Site.

```
SELECT ld.launch_site, AVG(m.success_rate) AS avg_success_rate
FROM launch_details ld
JOIN missions m ON ld.mission_id = m.mission_id
GROUP BY ld.launch_site
ORDER BY avg_success_rate DESC;
```

OUTPUT:

launch_site	avg_success_rate
Tanegashima Space Ce	93.88630132805811
Guiana Space Centre	93.0794116749483
Cape Canaveral	92.84146341463415
Xichang Satellite La	92.77599965413411
Vandenberg Air Force	92.49740283520191
Baikonur Cosmodrome	91.598437666893

4. Window Function – Percentile Crew Size.

```
SELECT mission_id, crew_size,
NTILE(4) OVER (ORDER BY crew_size DESC) AS crew_size_quartile
FROM crew;
```

OUTPUT:

mission_id	crew_size	crew_size_quartile
MSN-0093	99	1
MSN-0212	99	1
MSN-0246	98	1
MSN-0356	98	1
MSN-0487	98	1
MSN-0190	98	1

5. Missions with Highest Launch Mass per Vehicle.

SELECT mission_id, launch_vehicle, launch_mass
FROM (SELECT *, RANK() OVER (**PARTITION BY** launch_vehicle **ORDER BY** launch_mass **DESC**) AS rnk
FROM launch_details) ranked **WHERE** rnk = 1;

OUTPUT:

mission_id	launch_vehicle	launch_mass
MSN-0444	Ariane 6	100785
MSN-0216	Falcon Heavy	103578
MSN-0207	SLS	103922
MSN-0072	Starship	101019

SUBQUERY-BASED QUERIES

1. Above-Average Cost Missions

```
SELECT mission_name, cost_billion_usd
FROM missions
WHERE cost_billion_usd > (SELECT AVG(cost_billion_usd)
FROM missions);
```

OUTPUT:

mission_name	cost_billion_usd
Mission-1	526.68
Mission-6	452.42
Mission-9	361.35
Mission-11	327.76
Mission-13	436.87
Mission-14	511.71

2. Cost Above Avg of Same Mission Type (Correlated Subquery)

```
SELECT mission_name, mission_type, cost_billion_usd
FROM missions m1 WHERE cost_billion_usd > (SELECT AVG(cost_billion_usd)
FROM missions m2
WHERE m1.mission_type = m2.mission_type);
```

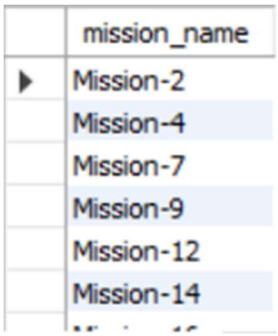
OUTPUT:

mission_name	mission_type	cost_billion_usd
Mission-1	Colonization	526.68
Mission-6	Colonization	452.42
Mission-9	Exploration	361.35
Mission-11	Research	327.76
Mission-13	Mining	436.87
Mission-14	Colonization	511.71

3. Crew Size Greater Than Average

```
SELECT mission_name
FROM missions
WHERE mission_id IN (SELECT mission_id
FROM crew
WHERE crew_size > (SELECT AVG(crew_size) FROM crew));
```

OUTPUT:

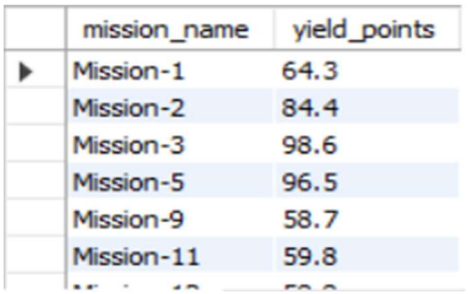


mission_name
Mission-2
Mission-4
Mission-7
Mission-9
Mission-12
Mission-14

4. Yield Greater Than Average

```
SELECT mission_name, yield_points
FROM missions
WHERE yield_points > (SELECT AVG(yield_points)
FROM missions);
```

OUTPUT:



mission_name	yield_points
Mission-1	64.3
Mission-2	84.4
Mission-3	98.6
Mission-5	96.5
Mission-9	58.7
Mission-11	59.8

5. Missions with Launch Cost Greater Than Vehicle-Wise Average

```
SELECT mission_id, launch_vehicle, launch_cost
FROM launch_details l1 WHERE launch_cost > (SELECT AVG(launch_cost)
FROM launch_details l2 WHERE l2.launch_vehicle = l1.launch_vehicle);
```

OUTPUT:

mission_id	launch_vehicle	launch_cost
MSN-0001	SLS	526680
MSN-0006	Ariane 6	452420
MSN-0009	Ariane 6	361350
MSN-0011	Starship	327760
MSN-0013	Ariane 6	436870
MSN-0014	Starship	511710

CONCLUSION

The **OrbitIQ – Space Mission Analytics** project represents a complete end-to-end application of SQL in the real-world context of interstellar mission management. By designing a fully normalized relational database and applying advanced SQL techniques like **joins**, **subqueries**, and **window functions**, this project demonstrates how structured data can be transformed into mission-critical insights.

From analysing mission performance and launch costs to identifying patterns in crew efficiency and launch outcomes, each query was crafted to reflect a real analytical need. The project not only strengthened my command over SQL but also enhanced my ability to think like a data analyst, understand relationships between entities, and write optimized logic.

Through OrbitIQ, I've proven that even futuristic and complex systems like space missions can be broken down, modelled, and understood through relational data and SQL — a foundation that can now be scaled into dashboards, BI tools, or data-driven decision-making systems.