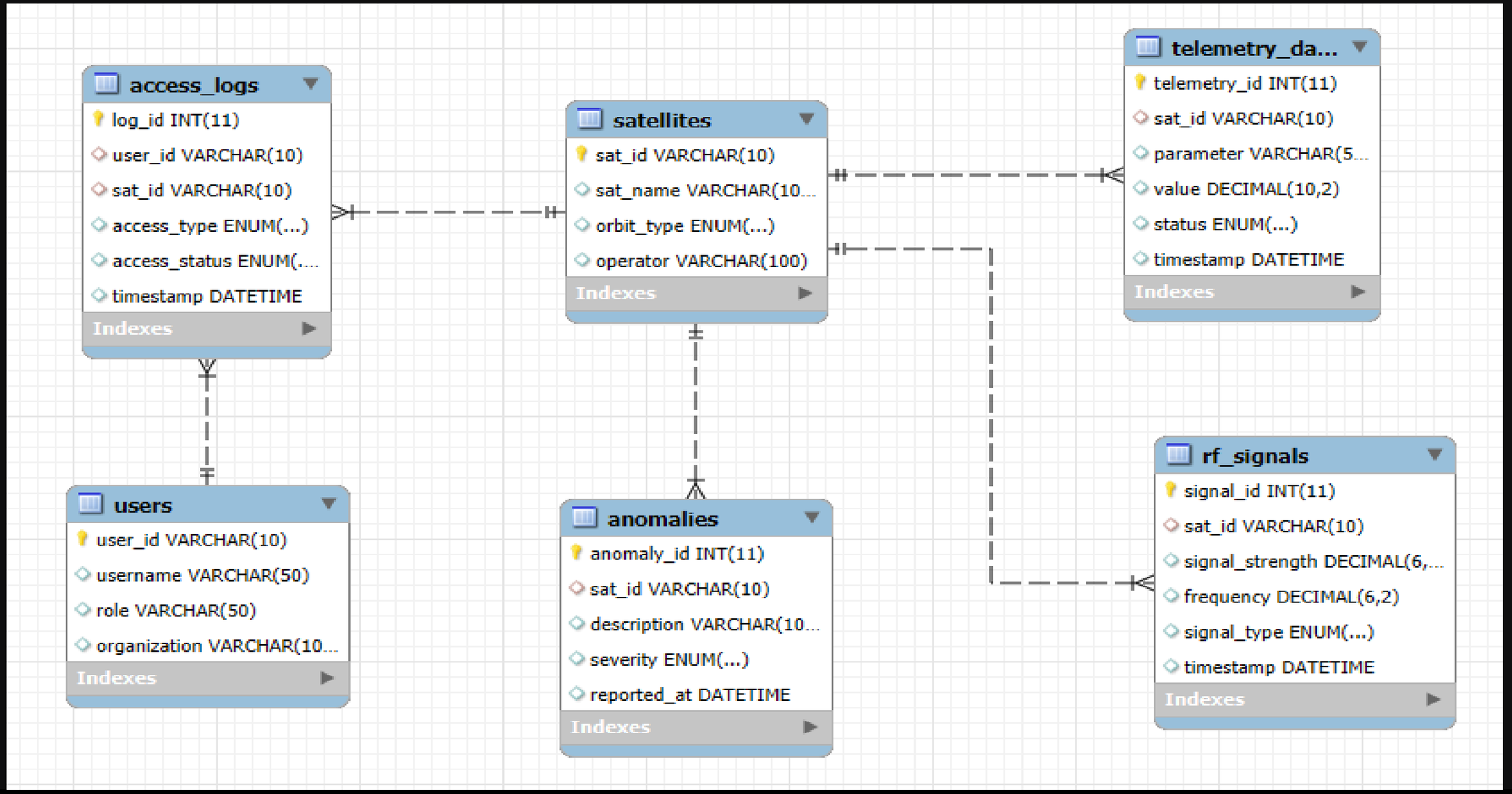
SmartTrack: Satellite Security Analysis with SQL

ABSTRACT

SmartTrack is a SQL-based Satellite Security analysis project focused on monitoring and securing satellite systems. Using structured datasets imported into MySQL Workbench, the project simulates a space infrastructure where telemetry data, RF signals, access logs, user profiles, satellite metadata and anomaly reports are analyzed to detect threats. Key cybersecurity issues such as signal jamming, telemetry spoofing, and unauthorized access are explored through SQL queries. The database schema is designed around six interconnected tables, enabling real-time analysis of suspicious patterns across multiple data sources. This project demonstrates the use of relational databases in a cybersecurity context, applying SQL to detect anomalies, audit user behavior, and investigate potential attacks on satellite networks. SmartTrack highlights practical skills in data modeling, threat analysis, and SQL-driven reporting all essential for a data analyst working in modern security-focused environments.

ER DIAGRAM


access_logs
log_id INT(11)
user_id VARCHAR(10)
sat_id VARCHAR(10)
access_type ENUM(...)
access_status ENUM(...
timestamp DATETIME
Indexes
satellites
sat_id VARCHAR(10)
sat_name VARCHAR(10...
orbit_type ENUM(...)
operator VARCHAR(100)
Indexes
telemetry_da...
telemetry_id INT(11)
sat_id VARCHAR(10)
parameter VARCHAR(5...
value DECIMAL(10,2)
status ENUM(...)
timestamp DATETIME
Indexes
users
user_id VARCHAR(10)
username VARCHAR(50)
role VARCHAR(50)
organization VARCHAR(10...
Indexes
anomalies
anomaly_id INT(11)
sat_id VARCHAR(10)
description VARCHAR(10...
severity ENUM(...)
reported_at DATETIME
Indexes
rf_signals
signal_id INT(11)
sat_id VARCHAR(10)
signal_strength DECIMAL(6,...
frequency DECIMAL(6,2)
signal_type ENUM(...)
timestamp DATETIME
Indexes

STRUCTURE OF TABLE

Database Table Structure for Telemetry Data in Satellite System

The image shows the structure of the telemetry_data table in the satellite database. It has six fields: telemetry_id (primary key), sat_id, parameter, value, status, and timestamp.
The value field stores decimal readings, and status can be normal, critical, or spoofed. The timestamp field logs when the data was recorded. This structure helps store and monitor satellite telemetry efficiently.

```
1 •   use satellite;
2
3 •   describe telemetry_data;
```

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

Field	Type	Null	Key	Default	Extra
telemetry_id	int(11)	NO	PRI	NULL	
sat_id	varchar(10)	YES		NULL	
parameter	varchar(50)	YES		NULL	
value	decimal(10,2)	YES		NULL	
status	enum('normal','critical','spoofed')	YES		NULL	
timestamp	datetime	YES		NULL	

STRUCTURE OF TABLE

Database Table Structure for Radio Frequency Signals Data in Satellite System

The image shows the structure of the rf_signals table in the satellite database. It has six fields: signal_id (primary key), sat_id, signal_strength, frequency, signal_type, and timestamp. The signal_strength and frequency fields store decimal values, while signal_type can be command, telemetry, or beacon.

```
1 •   use satellite;
2
3 •   describe rf_signals;
```

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

Field	Type	Null	Key	Default	Extra
signal_id	int(11)	NO	PRI	NULL	
sat_id	varchar(10)	YES		NULL	
signal_strength	decimal(6,2)	YES		NULL	
frequency	decimal(6,2)	YES		NULL	
signal_type	enum('command','telemetry','beacon')	YES		NULL	
timestamp	datetime	YES		NULL	

STRUCTURE OF TABLE

Database Table Structure for Access Logs in Satellite System

The access_logs table records user interactions with satellites, including user ID, satellite ID, access type, status, and timestamp. It helps monitor authorized and unauthorized access, enabling quick detection of suspicious activity.

```
1 •    use satellite;
2
3 •    describe access_logs;
```

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

Field	Type	Null	Key	Default	Extra
log_id	int(11)	NO	PRI	NULL	
user_id	varchar(10)	YES		NULL	
sat_id	varchar(10)	YES		NULL	
access_type	enum('telemetry','command')	YES		NULL	
access_status	enum('granted','denied')	YES		NULL	
timestamp	datetime	YES		NULL	

STRUCTURE OF TABLE

Database Table Structure for Users in Satellite System

The users table stores information about system users, including user ID, username, role, and organization. It supports access control and links user activities to specific individuals for security auditing.

```
1 •   use satellite;
2
3 •   describe users;
```

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

Field	Type	Null	Key	Default	Extra
user_id	varchar(10)	NO	PRI	NULL	
username	varchar(50)	YES		NULL	
role	varchar(50)	YES		NULL	
organization	varchar(100)	YES		NULL	

STRUCTURE OF TABLE

Database Table Structure for Satellites in Satellite System

The satellites table contains details of each satellite, including satellite ID, name, orbit type, and operator. It links operational data, signals, and anomalies to specific spacecraft for monitoring and analysis.

```
1 •   use satellite;
2
3 •   describe satellites;
```

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

Field	Type	Null	Key	Default	Extra
sat_id	varchar(10)	NO	PRI	NULL	
sat_name	varchar(100)	YES		NULL	
orbit_type	enum('LEO','MEO','GEO')	YES		NULL	
operator	varchar(100)	YES		NULL	

STRUCTURE OF TABLE

Database Table Structure for Anomaliesin Satellite System

The anomalies table records unusual satellite events, including anomaly ID, satellite ID, description, severity, and report time. It helps identify and analyze critical issues affecting satellite performance and security.

```
1 •   use satellite;
2
3 •   describe anomalies;
```

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

Field	Type	Null	Key	Default	Extra
anomaly_id	int(11)	NO	PRI	NULL	
sat_id	varchar(10)	YES		NULL	
description	varchar(100)	YES		NULL	
severity	enum('low','medium','high')	YES		NULL	
reported_at	datetime	YES		NULL	

CONTENT OF TABLE

Table : Telemetry Data

```
1 •    select * from telemetry_data;
```

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

telemetry_id	sat_id	parameter	value	status	timestamp
1	SAT-002	temperature	30.96	normal	2025-06-21 15:01:23
2	SAT-003	gps	60.67	normal	2025-06-06 17:42:29
3	SAT-001	altitude	57.10	normal	2025-06-16 10:38:49
4	SAT-001	temperature	46.36	normal	2025-06-25 20:48:48
5	SAT-001	gps	72.25	normal	2025-06-06 12:17:00
6	SAT-003	altitude	45.07	critical	2025-06-28 00:26:03
7	SAT-003	gps	98.74	critical	2025-06-21 14:43:26
8	SAT-001	altitude	40.16	spoofed	2025-06-20 21:41:20
9	SAT-002	voltage	61.32	normal	2025-06-19 02:35:51
10	SAT-003	voltage	52.69	normal	2025-06-18 03:32:09

CONTENT OF TABLE

Table : Radio Frequency Signals

```
1 •    select * from rf_signals;
```

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

signal_id	sat_id	signal_strength	frequency	signal_type	timestamp
1	SAT-002	-67.87	8.52	telemetry	2025-06-24 02:23:14
2	SAT-002	-62.91	8.29	command	2025-06-16 02:10:47
3	SAT-001	-74.41	7.72	telemetry	2025-07-04 06:51:29
4	SAT-002	-80.15	8.59	command	2025-06-04 16:47:04
5	SAT-001	-69.87	8.49	command	2025-06-13 15:15:42
6	SAT-002	-56.49	7.48	command	2025-06-05 11:06:56
7	SAT-003	-54.18	8.56	beacon	2025-06-20 08:22:26
8	SAT-002	-60.81	8.04	command	2025-06-08 16:39:05

CONTENT OF TABLE

Table : Access Logs

```
1 •    select * from access_logs;
```

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

log_id	user_id	sat_id	access_type	access_status	timestamp
1	U005	SAT-002	telemetry	granted	2025-06-24 16:11:23
2	U001	SAT-003	telemetry	granted	2025-06-17 07:38:00
3	U004	SAT-002	command	granted	2025-06-16 19:58:38
4	U001	SAT-002	command	granted	2025-06-29 16:20:28
5	U001	SAT-001	command	granted	2025-06-04 16:29:08
6	U001	SAT-003	telemetry	granted	2025-06-18 16:55:07
7	U001	SAT-003	telemetry	denied	2025-06-13 20:05:38
8	U004	SAT-001	command	granted	2025-06-15 23:47:37

CONTENT OF TABLE

Table : Users

```
1 •     select * from users;
2
3
4
```

Result Grid | Filter Rows: | Edit:

user_id	username	role	organization
U001	Sky	engineer	SpaceTech
U002	Grace	analyst	OrbitalX
U003	Walter	controller	AstroLabs
U004	Heidi	technician	CosmoComm
U005	Mallory	admin	SkyOps
NULL	NULL	NULL	NULL

CONTENT OF TABLE

Table : Satellites

```
1 •    select * from satellites;
2
3
4
```

Result Grid | Filter Rows: | Edit:

sat_id	sat_name	orbit_type	operator
SAT-001	Horizon-1	LEO	SpaceTech
SAT-002	Nebula-X	GEO	OrbitalX
SAT-003	StellarProbe	MEO	SkyOps
NULL	NULL	NULL	NULL

CONTENT OF TABLE

Table : Anomalies

```
1 •    select * from anomalies;
2
3
4
```

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

anomaly_id	sat_id	description	severity	reported_at
1	SAT-001	spoofed command	medium	2025-06-05 12:33:16
2	SAT-001	unauthorized access	low	2025-06-25 06:09:22
3	SAT-001	spoofed command	high	2025-06-29 02:32:07
4	SAT-003	unauthorized access	medium	2025-06-16 00:20:37
5	SAT-002	data corruption	low	2025-07-01 04:26:20
6	SAT-001	unauthorized access	low	2025-06-25 19:36:50
7	SAT-001	data corruption	medium	2025-06-10 02:16:52
8	SAT-002	unauthorized access	medium	2025-06-21 10:11:39
9	SAT-002	telemetry glitch	medium	2025-06-07 23:28:14
10	SAT-003	data corruption	low	2025-06-08 17:54:51
11	SAT-002	signal loss	medium	2025-06-08 21:20:27

JOINS

A JOIN is used to combine rows from two or more tables based on a related column. In the SmartTrack project, joins help connect satellite data, telemetry, access logs, users, and anomalies to get meaningful insights.
The main types of joins used here are:
•INNER JOIN → returns only matching records between tables.
•LEFT JOIN → returns all records from the left table and matching ones from the right (null if no match).
•RIGHT JOIN → similar to LEFT JOIN but returns all from the right table.

QUERY 1 — ANOMALIES WITH SATELLITE INFO, SORTED BY SEVERITY.

1. This query uses an INNER JOIN to connect ‘anomalies’ with ‘satellites’ based on ‘sat_id’.
2. It returns each anomaly along with the satellite’s name and operator. Results are sorted so the most severe and latest issues appear first.

```
1
2 •  SELECT a.anomaly_id, s.sat_name, s.operator, a.description, a.severity, a.reported_at
3    FROM anomalies a
4    JOIN satellites s ON a.sat_id = s.sat_id
5    ORDER BY a.severity DESC, a.reported_at DESC;
6
```

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

anomaly_id	sat_name	operator	description	severity	reported_at
84	Nebula-X	OrbitalX	unauthorized access	high	2025-07-04 12:40:12
28	Nebula-X	OrbitalX	unauthorized access	high	2025-07-03 01:31:41
89	Horizon-1	SpaceTech	data corruption	high	2025-07-02 01:04:20
59	Horizon-1	SpaceTech	data corruption	high	2025-07-01 18:25:48
3	Horizon-1	SpaceTech	spoofed command	high	2025-06-29 02:32:07
44	StellarProbe	SkyOps	unauthorized access	high	2025-06-27 08:03:44
62	StellarProbe	SkyOps	data corruption	high	2025-06-26 23:15:34
97	Nebula-X	OrbitalX	spoofed command	high	2025-06-26 20:54:07

QUERY 2 — ATTEMPTS ACCESS WITH SATELLITE AND USER INFO.

1. This query joins three tables: access_logs, users and satellites.

2. It shows who accessed which satellite, the type of access, whether it was successful, and when it occurred — ordered with the latest activity first.

```
1
2 •  SELECT al.log_id, u.username, u.role, s.sat_name, al.access_type, al.access_status, al.timestamp
3    FROM access_logs al
4    JOIN users u ON al.user_id = u.user_id
5    JOIN satellites s ON al.sat_id = s.sat_id
6    ORDER BY al.timestamp DESC;
7
```

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

log_id	username	role	sat_name	access_type	access_status	timestamp
23	Heidi	technician	StellarProbe	command	denied	2025-07-04 20:14:06
114	Mallory	admin	Horizon-1	telemetry	denied	2025-07-04 18:18:07
247	Mallory	admin	StellarProbe	telemetry	granted	2025-07-04 17:35:59
136	Sky	engineer	Nebula-X	telemetry	granted	2025-07-04 16:34:13
59	Sky	engineer	StellarProbe	command	granted	2025-07-04 15:19:18
201	Grace	analyst	Nebula-X	telemetry	granted	2025-07-04 02:34:39
61	Walter	controller	Horizon-1	command	granted	2025-07-04 02:14:44
158	Grace	analyst	Nebula-X	telemetry	granted	2025-07-04 01:52:58

QUERY 3 — TELEMETRY WITH ANOMALIES REPORTED THE SAME DAY.

1. Here we use a LEFT JOIN so all telemetry records are shown, even if there is no anomaly on that day. If there is a match in anomalies for the same satellite and date, its description is shown.

```
1
2 •  SELECT t.telemetry_id, s.sat_name, t.parameter, t.value, t.status, a.description AS anomaly
3    FROM telemetry_data t
4    JOIN satellites s ON t.sat_id = s.sat_id
5    LEFT JOIN anomalies a
6        ON t.sat_id = a.sat_id
7       AND DATE(t.timestamp) = DATE(a.reported_at);
```

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

telemetry_id	sat_name	parameter	value	status	anomaly
150	StellarPr...	altitude	51.60	normal	unauthorized access
187	StellarPr...	gps	57.95	normal	unauthorized access
199	StellarPr...	altitude	24.88	normal	unauthorized access
235	StellarPr...	altitude	70.19	normal	unauthorized access
271	StellarPr...	altitude	22.45	normal	unauthorized access
124	Nebula-X	voltage	58.86	critical	data corruption
170	Nebula-X	voltage	56.65	normal	data corruption
196	Nebula-X	altitude	73.85	normal	data corruption

WINDOW FUNCTIONS

1. Window functions perform calculations across a set of rows that are related to the current row, without collapsing them into a single result like GROUP BY would.
2. They are written with an OVER() clause, which defines the "window" of rows for calculation. Common types used here:
3. RANK() → Assigns rank to rows based on a sort order.
4. AVG() OVER(PARTITION BY) → Calculates average within a group without reducing rows.
5. ROW_NUMBER() → Assigns a unique sequential number to rows within a group.

QUERY 1 — RANKING SATELLITES BY ANOMALY COUNT.

1. This uses the RANK() window function to rank satellites based on the number of anomalies. Satellites with the same count get the same rank.

```
1
2 •   SELECT sat_id, COUNT(*) AS anomaly_count,
3             RANK() OVER (ORDER BY COUNT(*) DESC) AS rank_by_anomalies
4     FROM anomalies
5     GROUP BY sat_id;
6
7
```

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

sat_id	anomaly_count	rank_by_anomalies
SAT-001	39	1
SAT-002	37	2
SAT-003	24	3

QUERY 2 — COMPARE EACH SIGNAL TO SATELLITE'S AVERAGE.

1. AVG() OVER (PARTITION BY) calculates the average signal strength for each satellite without grouping the rows, allowing us to compare each reading to the average.

```
1
2 •   SELECT sat_id, signal_id, signal_strength,
3             AVG(signal_strength) OVER (PARTITION BY sat_id) AS avg_signal
4     FROM rf_signals;
5
6
```

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

sat_id	signal_id	signal_strength	avg_signal
SAT-001	184	-70.87	-65.365234
SAT-001	220	-76.07	-65.365234
SAT-001	232	-67.08	-65.365234
SAT-001	236	-72.74	-65.365234
SAT-001	244	-71.17	-65.365234
SAT-001	260	-60.63	-65.365234
SAT-001	264	-83.98	-65.365234
SAT-001	272	-52.40	-65.365234

QUERY 3 — NUMBER EACH USER'S ACCESS ATTEMPTS IN ORDER.

1. The ROW_NUMBER() function gives a sequence number to each access attempt per user, ordered by time.

```
1
2 •  SELECT user_id, log_id, access_type, access_status,
3           ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY timestamp) AS attempt_number
4    FROM access_logs;
5
6
7
```

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

user_id	log_id	access_type	access_status	attempt_number
U001	121	telemetry	granted	1
U001	19	telemetry	granted	2
U001	137	telemetry	granted	3
U001	5	command	granted	4
U001	33	command	granted	5
U001	142	telemetry	granted	6
U001	146	telemetry	granted	7
U001	97	telemetry	granted	8

VIEWS

1. A VIEW is a stored query that acts like a virtual table.
2. They don’t store the data physically (unless materialized), but store the SQL query definition.
3. Simplifies repeated queries, improves readability, and adds a layer of security (hide complex joins from end users).

QUERY 1 — TELEMETRY DUPLICATES.

1. Creates a saved view showing only telemetry records marked as “critical”. This makes it easy to query only urgent cases.

```
2
3 •    CREATE TABLE critical_telemetry_table AS
4      SELECT *
5      FROM telemetry_data
6      WHERE status = 'critical';
7
8 •    select * from critical_telemetry_table;
```

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

telemetry_id	sat_id	parameter	value	status	timestamp
6	SAT-003	altitude	45.07	critical	2025-06-28 00:26:03
7	SAT-003	gps	98.74	critical	2025-06-21 14:43:26
20	SAT-001	gps	60.13	critical	2025-07-01 18:11:09
64	SAT-001	gps	38.71	critical	2025-07-02 03:49:36
65	SAT-002	altitude	54.22	critical	2025-06-06 19:46:31
69	SAT-001	altitude	39.78	critical	2025-06-25 06:36:19
75	SAT-003	temperature	66.00	critical	2025-06-06 21:18:40
97	SAT-002	altitude	43.64	critical	2025-06-13 18:01:58

QUERY 2 — DENIED ACCESS ATTEMPTS WITH USERNAMES.

1. Stores a view of all denied access attempts, including which user made them.

```
1 •  CREATE VIEW denied_access AS
2    SELECT al.*, u.username
3    FROM access_logs al
4    JOIN users u ON al.user_id = u.user_id
5    WHERE access_status = 'denied';
6
7 •  select * from denied_access;
```

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

log_id	user_id	sat_id	access_type	access_status	timestamp	username
7	U001	SAT-003	telemetry	denied	2025-06-13 20:05:38	Sky
13	U005	SAT-002	telemetry	denied	2025-06-14 17:50:40	Mallory
21	U002	SAT-001	telemetry	denied	2025-06-12 04:04:21	Grace
23	U004	SAT-003	command	denied	2025-07-04 20:14:06	Heidi
25	U002	SAT-001	command	denied	2025-06-17 11:34:28	Grace
36	U003	SAT-001	command	denied	2025-06-14 09:48:18	Walter
48	U003	SAT-003	command	denied	2025-06-24 17:35:40	Walter
65	U003	SAT-001	telemetry	denied	2025-06-03 06:24:05	Walter

QUERY 3 — HIGH SEVERITY ANOMALIES WITH SATELLITE NAMES.

1. A ready-to-use view for listing all high-severity anomalies along with the satellite name.

```sql
CREATE VIEW high_severity_anomalies AS
SELECT a.*, s.sat_name
FROM anomalies a
JOIN satellites s ON a.sat_id = s.sat_id
WHERE severity = 'high';

select * from high_severity_anomalies;
```

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

anomaly_id	sat_id	description	severity	reported_at	sat_name
3	SAT-001	spoofed command	high	2025-06-29 02:32:07	Horizon-1
17	SAT-002	data corruption	high	2025-06-25 11:38:31	Nebula-X
18	SAT-003	signal loss	high	2025-06-25 14:18:23	StellarProbe
28	SAT-002	unauthorized access	high	2025-07-03 01:31:41	Nebula-X
42	SAT-002	spoofed command	high	2025-06-03 12:32:28	Nebula-X
43	SAT-002	spoofed command	high	2025-06-09 05:52:31	Nebula-X
44	SAT-003	unauthorized access	high	2025-06-27 08:03:44	StellarProbe
45	SAT-002	unauthorized access	high	2025-06-22 15:14:38	Nebula-X

SUBQUERIES

1. A Subquery is a query inside another query.
2. It is often used to filter results based on another query.
3. It compares against the aggregated values and check for existence of data.

QUERY 1 — SATELLITES WITH ABOVE AVERAGE ANOMALIES.

1. The inner subquery calculates the average anomalies per satellite. The outer query returns only those satellites exceeding this average.

```sql
SELECT sat_id, COUNT(*) AS anomaly_count
FROM anomalies
GROUP BY sat_id
HAVING COUNT(*) > (SELECT AVG(anomaly_count)
    FROM (SELECT COUNT(*) AS anomaly_count
        FROM anomalies
        GROUP BY sat_id
    ) AS sub
);
```

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

sat_id	anomaly_count
SAT-001	39
SAT-002	37

QUERY 2 — USER WITH DENIED ACCESS.

1. Finds all usernames that appear in the access logs with at least one denied attempt.

```sql
SELECT username
FROM users
WHERE user_id IN (
    SELECT DISTINCT user_id
    FROM access_logs
    WHERE access_status = 'denied'
);
```

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

username
Sky
Grace
Walter
Heidi
Mallory

QUERY 3 — SATELLITES WITH LOWEST AVERAGE SIGNAL STRENGTH.

1. Compares average signal strength of satellites and returns those with the minimum average.

```
1  SELECT sat_id
2  FROM rf_signals
3  GROUP BY sat_id
4  HAVING AVG(signal_strength) = (SELECT MIN(avg_signal)
5      FROM (SELECT AVG(signal_strength) AS avg_signal
6          FROM rf_signals
7  GROUP BY sat_id) AS sub);
```

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

sat_id
SAT-001

KEY CYBERSECURITY CHALLENGES

SmartTrack directly confronts prevalent threats to satellite systems, enabling proactive detection and response.

Signal Anomalies

Detection of unusual RF signal strength or frequency patterns that may indicate interference or jamming attempts.

Telemetry Spoofing

Identification of telemetry parameters crossing critical thresholds to detect potential system failures or spoofing.

Unauthorized Access Tracking

Analysis of user access logs to spot repeated failed login attempts, privilege misuse, or unusual access times.

CONCLUSION

The SmartTrack project successfully demonstrates how SQL can be applied to monitor and secure satellite systems by integrating telemetry data, RF signals, access logs, anomaly reports, and user information into a unified relational database. Using MySQL Workbench, the project showcases advanced querying techniques including joins, window functions, subqueries, and views to detect anomalies, analyze suspicious activity, and identify potential security threats such as signal jamming, telemetry spoofing, and unauthorized access. This end-to-end database design and analysis highlights the effectiveness of SQL as a powerful tool for cybersecurity threat detection in critical space infrastructure, while also building practical skills essential for data analysts in security-focused domains.