

Technical Report – IoT Operations Platform (Lab 8)

Transactional Integrity, Concurrency, and Access Control in SQL Server
AdventureWorks 2019 – IoT Schema

1. Executive Summary

This lab focused on building several key components of a reliable IoT backend system using SQL Server.

The main goals were to design a proper database schema for IoT devices, implement a multi-step transactional workflow for telemetry ingestion, analyze concurrency behavior through different isolation levels, and set up role-based access control that follows the principles of least privilege.

By completing all six tasks, I was able to create a working IoT schema, build a transactional stored procedure that handles data validation and error logging, test how concurrent sessions interact, and configure multiple user roles that match the business requirements. Overall, the lab helped me understand how SQL Server handles transactions, locking, and permission management in a realistic IoT scenario where thousands of devices may send telemetry at the same time.

2. IoT Business Case Summary

AdventureWorks is expanding into IoT products, including SmartBike devices and factory-floor monitoring equipment.

Each device reports telemetry data every 30 seconds, and the backend must handle more than 250,000 telemetry messages per hour.

Because of that scale, the system must deal with several issues:

- Incomplete or corrupted telemetry cycles
- Conflicts caused by concurrent updates
- The need for atomic transactions (either insert all required records or none)
- Strict separation of permissions due to privacy rules

The purpose of this lab was to build the database foundation required for an enterprise IoT platform.

Tasks included designing the schema, building reliable ingestion logic, handling concurrency, and enforcing security based on user responsibilities.

3. Transaction Workflow Description (Stored Procedure)

The telemetry ingestion stored procedure (`IoT.IngestTelemetry`) was written to simulate how a real IoT backend processes one complete telemetry cycle.

The procedure performs the following steps inside a single transaction:

Step 1 — Device validation

The procedure first checks if the DeviceID exists. If not, it throws an error.

Step 2 — Sensor data validation

Several checks ensure that data is within acceptable ranges:

- Temperature cannot be lower than -50 or higher than 150
- Battery level must be between 0 and 100
- GPS coordinates cannot be NULL

If any validation fails, the procedure throws an error.

Step 3 — Insert telemetry record

The raw telemetry packet is inserted into `IoT.Telemetry`.

Step 4 — Update LastHeartbeat

The `IoT.Device` table stores the time of the most recent telemetry update.

Step 5 — Insert DeviceHealth record

The system calculates a temperature status, battery status, and vibration status and inserts a summary row into `IoT.DeviceHealth`.

Step 6 — Generate alerts

Depending on the telemetry values, several alerts may be created:

- Low battery
- High temperature
- Invalid GPS coordinates

Step 7 — Write success audit

A summary of the processed telemetry cycle is written into `IoT.TelemetryAudit`.

Error handling

If anything fails:

- The entire transaction is rolled back
- A new error log entry is inserted into `IoT.TelemetryErrorLog`
- The error is re-thrown to the user

This ensures atomicity: **either all steps succeed, or none do.**

(Reference: Stored Procedure.sql)

4. Concurrency Test Results

To observe the behavior of concurrent transactions, I used two SSMS sessions (Session A and Session B) to update the same rows and run SELECT statements under different isolation levels.

Blocking Scenario

When both sessions attempted to update the same DeviceID, one session became blocked until the other finished its transaction. This demonstrated how SQL Server enforces locking to prevent inconsistent updates.

(Reference: Session A.sql and Session B.sql)

Dirty Read (READ UNCOMMITTED)

Under this isolation level, Session B was able to read uncommitted data from Session A. This proved that dirty reads are possible when using READ UNCOMMITTED.

Read Committed

When using READ COMMITTED, Session B waited until Session A committed before it could read the updated row. This prevented dirty reads.

Snapshot Isolation

After enabling snapshot isolation, Session B read the old committed version of the row while Session A was still updating it. No blocking occurred, and no dirty reads happened.

(Reference: Snapshot Isolation Setup.sql)

Serializable

Under SERIALIZABLE, SQL Server prevented both sessions from inserting rows that could lead to phantom reads.

This was the strictest isolation level and caused the most blocking.

Summary of Observed Behaviors

Isolation Level	Dirty Reads	Blocking	Notes
READ UNCOMMITTED	Yes	No	Fast but unsafe
READ COMMITTED	No	Yes	SQL Server default
SNAPSHOT	No	No	Reads older version
SERIALIZABLE	No	Yes (most)	Strictest isolation

Overall, the tests matched what is expected from SQL Server's concurrency model.

5. Role-Based Permission Design

The roles were designed according to the business rules in the case description.

IoTDeviceAgent

- Can insert/update only IoT.Device
- Cannot access telemetry or customer data

TelemetryIngestionService

- Can insert into Telemetry and DeviceHealth
- Can update only LastHeartbeat
- Cannot read customer tables

IoTAnalyst

- Read-only access to Telemetry and DeviceHealth
- No write permissions
- No access to customer data

IoTFieldTechnician

- Can insert into MaintenanceLog
- Cannot query DeviceOwner
- UPDATE on MaintenanceLog fails because this role lacks SELECT

(This matches the “least privilege” policy.)

SecurityComplianceOfficer

- Read-only access to Device, DeviceOwner, and Telemetry
- No modification privileges

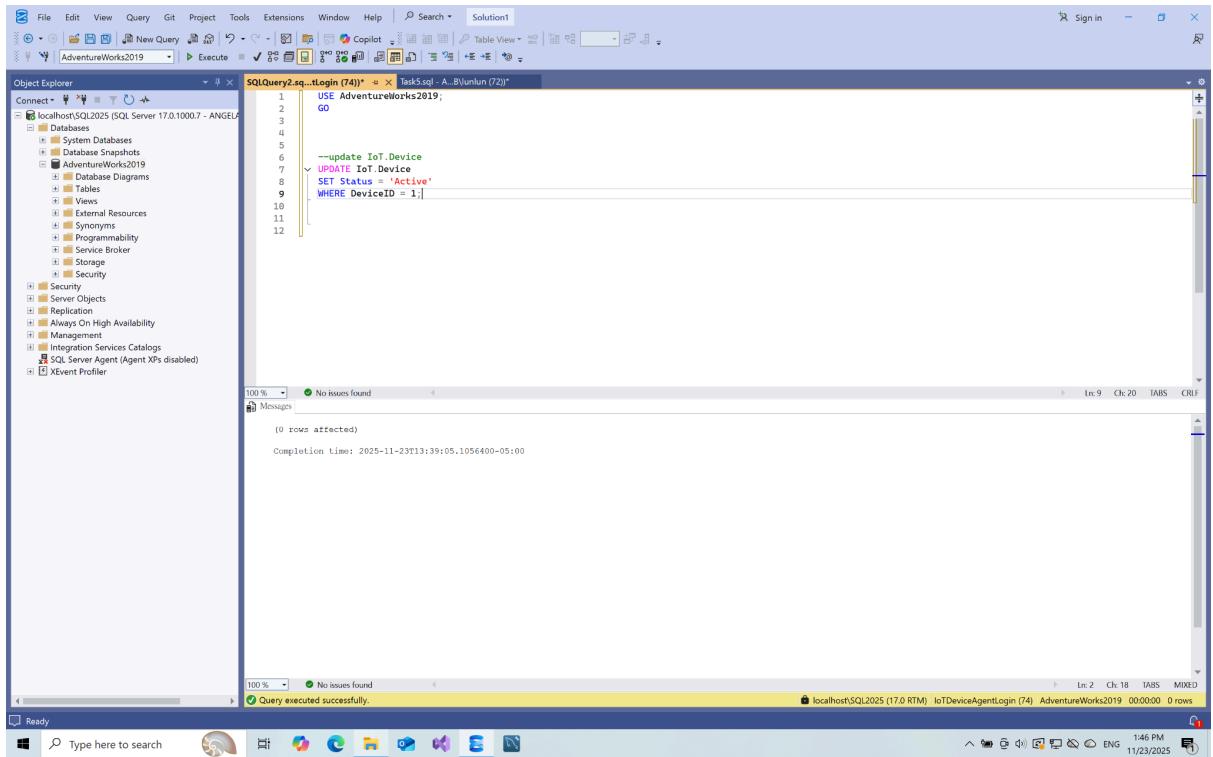
(Reference: Task5.sql)

6. Permission Validation (Task 6)

Each role was tested by logging in under a separate SQL Server login and executing statements allowed or denied by the role.

IoTDeviceAgent

- UPDATE Device → Success



The screenshot shows the SQL Server Management Studio (SSMS) interface. In the Object Explorer on the left, the database 'AdventureWorks2019' is selected. In the center, a query window titled 'SQLQuery2.sq... (Login (74))' is open, displaying the following T-SQL code:

```
1 USE Adventureworks2019;
2 GO
3
4
5
6 --update IoT.Device
7 UPDATE IoT.Device
8 SET Status = 'Active'
9 WHERE DeviceID = 1;
10
11
12
```

Below the code, the 'Messages' pane shows the results of the execution:

- No issues found
- (0 rows affected)
- Completion Time: 2025-11-23T13:39:05.1056400-05:00

The status bar at the bottom right indicates the session details: localhost\SQL2025 (17.0 RTM) | IoTDeviceAgent>Login (74) | AdventureWorks2019 | 00:00:00 | 0 rows.

- INSERT Telemetry → Permission denied (Correct)

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the database 'AdventureWorks2019' is selected. In the center pane, a query window titled 'SQLQuery2.sql...Login (74)*' contains the following T-SQL code:

```

USE Adventureworks2019;
GO
--IoTDeviceAgent
--update IoT_Device
UPDATE IoT_Device
SET Status = 'Active'
WHERE DeviceID = 1;
--insert
INSERT INTO IoT_Telemetry (DeviceID, Timestamp, Speed, Cadence, Temperature, BatteryLevel, GPSLatitude, GPSLongitude)
VALUES (1, SYSDATETIME(), 10, 80, 25, 90, 40.7128, -74.0860);

```

In the Messages pane at the bottom, an error message is displayed:

Msg 229, Level 14, State 5, Line 13
The INSERT permission was denied on the object 'Telemetry', database 'AdventureWorks2019', schema 'IoT'.
Completion time: 2025-11-23T13:47:52.3037351-05:00

TelemetryIngestionService

- INSERT Telemetry → Success

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the database 'AdventureWorks2019' is selected. In the center pane, a query window titled 'SQLQuery1.sql...onLogin (90)*' contains the following T-SQL code:

```

--main account insert
USE Adventureworks2019;
GO
INSERT INTO IoT_Device (SerialNumber, DeviceType, FirmwareVersion, Status)
VALUES ('SNO001', 'SmartBike', '1.0', 'Active');

SELECT * FROM IoT_Device;

--TelemetryIngestionService
--INSERT Telemetry
USE Adventureworks2019;
GO
--INSERT INTO IoT_Telemetry (DeviceID, Timestamp, Speed, Cadence, Temperature, BatteryLevel, GPSLatitude, GPSLongitude)
VALUES (1, SYSDATETIME(), 10, 80, 25, 90, 40.7128, -74.0860);

```

In the Messages pane at the bottom, a success message is displayed:

(1 row affected)
Completion time: 2025-11-23T14:20:48.0457745-05:00

At the bottom right, a status bar indicates 'Query executed successfully.'

- SELECT DeviceOwner → Permission denied

- UPDATE Device → Permission denied (except LastHeartbeat)

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The top menu bar includes File, Edit, View, Query, Git, Project, Tools, Extensions, Window, Help, and Solution1. The toolbar contains various icons for database management tasks. The Object Explorer on the left lists several databases and their objects, such as AdventureWorks2019, AdventureworksLT, and AdventureworksDW. The central area features a query editor with three tabs open:

- Task6.sql - ...onLogin (90) - Contains a script for creating a login and granting permissions.
- Task5.sql - ...unlun (72) - Contains a script for inserting data into IoT.Device and IoT.Telemetry tables.
- SQLQuery1.sql - ...unlun (74) - Contains a script for inserting data into IoT.Device and IoT.Telemetry tables, including a test update statement.

The status bar at the bottom displays the completion time, number of rows affected, and connection details (localhost(SQL2025 (17.0 RTM) TelemetryingestionLogins... AdventureWorks2019 00:00:00 0 rows)).

IoTAnalyst

- SELECT Telemetry → Success

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists various database objects like Tables, Views, and Procedures. The main pane displays a script window with the following SQL code:

```

--test:SELECT DeviceOwner
SELECT * FROM IoT.DeviceOwner;

--test: UPDATE IoT.Device
UPDATE IoT_Device
SET Status = 'Inactive'
WHERE DeviceID = 1;

--IoTAnalyst
--SELECT IoT.Telemetry
USE AdventureWorks2019;
GO
SELECT TOP 10 * FROM IoT_Telemetry;

```

The results pane shows a single row of data from the IoT_Telemetry table:

TelemetryID	DeviceID	Timestamp	Speed	Cadence	Temperature	BatteryLevel	GPSLatitude	GPSLongitude
1	1	2025-11-23 14:28:48	12.00	85	23.00	88.00	40.712800	-74.060600

A status bar at the bottom indicates "Query executed successfully." and "localhost(SQL2025 (17.0 RTM)) IoTAnalystLogin (52) AdventureWorks2019 00:00:00 1 rows".

- SELECT DeviceHealth → Success

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists various database objects like Tables, Views, and Procedures. The main pane displays a script window with the following SQL code:

```

--test:SELECT DeviceOwner
SELECT * FROM IoT.DeviceOwner;

--test: UPDATE IoT.Device
UPDATE IoT_Device
SET Status = 'Inactive'
WHERE DeviceID = 1;

--IoTAnalyst
--test: SELECT IoT.Telemetry
USE AdventureWorks2019;
GO
SELECT TOP 10 * FROM IoT_Telemetry;

--test: SELECT IoT.DeviceHealth
SELECT TOP 10 * FROM IoT.DeviceHealth;

```

The results pane shows a single row of data from the IoT_DeviceHealth table:

HealthID	DeviceID	Timestamp	IsHealthy	TemperatureStatus	BatteryStatus	VibrationStatus
1	1	2025-11-23 14:28:48	True	Normal	High	Low

A status bar at the bottom indicates "Query executed successfully." and "localhost(SQL2025 (17.0 RTM)) IoTAnalystLogin (52) AdventureWorks2019 00:00:00 0 rows".

- INSERT Telemetry → Denied

The screenshot shows the SQL Server Management Studio interface with three tabs open: Task6.sql, Task5.sql, and Task5.sql. The Object Explorer on the left shows the database structure, including tables like IoT.DeviceOwner, IoT.Device, IoT.Telemetry, and IoT.DeviceHealth. The central pane displays a T-SQL script attempting to insert data into the IoT.Telemetry table:

```

--test: SELECT * FROM IoT.DeviceOwner;
SELECT * FROM IoT.DeviceOwner;

--test: UPDATE IoT.Device
UPDATE IoT.Device
SET Status = 'Inactive'
WHERE DeviceID = 1;

--IoTAnalyst
--test: SELECT IoT.Telemetry
USE AdventureWorks2019;
GO
SELECT TOP 10 * FROM IoT.Telemetry

--test: SELECT IoT.DeviceHealth
SELECT TOP 10 * FROM IoT.DeviceHealth;

--test: INSERT Telemetry
INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed, Cadence, Temperature, BatteryLevel, GPSLatitude, GPSLongitude)
VALUES (1, SYSDATETIME(), 10, 80, 25, 90, 40.7128, -74.0669);

```

The execution results in an error message in the Messages pane:

Msg 229, Level 14, State 5, Line 57
The INSERT permission was denied on the object 'Telemetry', database 'AdventureWorks2019', schema 'iot'.

Completion time: 2025-11-23T14:45:00.6330987-05:00

- SELECT DeviceOwner → Denied

The screenshot shows the SQL Server Management Studio interface with three tabs open: Task6.sql, Task5.sql, and Task5.sql. The Object Explorer on the left shows the database structure, including tables like IoT.DeviceOwner, IoT.Device, IoT.Telemetry, and IoT.DeviceHealth. The central pane displays a T-SQL script attempting to select from the IoT.DeviceOwner table:

```

--test: SELECT IoT.DeviceOwner
SELECT * FROM IoT.DeviceOwner;

--IoTAnalyst
--test: SELECT IoT.Telemetry
USE AdventureWorks2019;
GO
SELECT TOP 10 * FROM IoT.Telemetry

--test: SELECT IoT.DeviceHealth
SELECT TOP 10 * FROM IoT.DeviceHealth;

--test: INSERT Telemetry
INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed, Cadence, Temperature, BatteryLevel, GPSLatitude, GPSLongitude)
VALUES (1, SYSDATETIME(), 10, 80, 25, 90, 40.7128, -74.0669);

--test: SELECT DeviceOwner
SELECT * FROM IoT.DeviceOwner;

```

The execution results in an error message in the Messages pane:

Msg 229, Level 14, State 5, Line 61
The SELECT permission was denied on the object 'DeviceOwner', database 'AdventureWorks2019', schema 'iot'.

Completion time: 2025-11-23T14:45:47.9155904-05:00

IoTFieldTechnician

- **INSERT MaintenanceLog → Success**

The screenshot shows the SQL Server Management Studio interface with three tabs open: Task6.sql, Task7.sql, and Task5.sql. The Task6.sql tab contains the following SQL code:

```

--test: INSERT Telemetry
57 INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed, Cadence, Temperature, BatteryLevel, GPSLatitude, GPSLongitude)
58 VALUES (1, SYSDATETIME(), 10, 80, 25, 90, 40.7128, -74.0860);
59
60 --test: SELECT DeviceOwner
61 SELECT * FROM IoT.DeviceOwner;
62
63
64 --IoTFieldTechnician
65 USE AdventureWorks2019;
66 GO
67
68 -- INSERT INTO IoT.MaintenanceLog (DeviceID, TechnicianName, Notes)
69 INSERT INTO IoT.MaintenanceLog (DeviceID, TechnicianName, Notes)
70 VALUES (1, 'Tech John', 'Performed routine inspection.');
    
```

The Messages pane shows the output:

```

(1 row affected)
Completion time: 2025-11-23T14:51:42.6281566-05:00
    
```

The status bar at the bottom indicates "Query executed successfully."

- **UPDATE MaintenanceLog → Fails due to missing SELECT (expected)**

The screenshot shows the SQL Server Management Studio interface with three tabs open: Task6.sql, Task7.sql, and Task5.sql. The Task6.sql tab contains the following SQL code:

```

--IoTFieldTechnician
65 --UPDATE MaintenanceLog
66 USE AdventureWorks2019;
67 GO
68
69
70 -- INSERT INTO IoT.MaintenanceLog (DeviceID, TechnicianName, Notes)
71 INSERT INTO IoT.MaintenanceLog (DeviceID, TechnicianName, Notes)
72 VALUES (1, 'Tech John', 'Performed routine inspection.');
73
74
75 --UPDATE MaintenanceLog
76 UPDATE IoT.MaintenanceLog
77 SET Notes = 'Updated notes by technician.'
WHERE LogID = 1;
    
```

The Messages pane shows the error output:

```

Msg 220, Level 14, State 5, Line 77
The SELECT permission was denied on the object 'MaintenanceLog', database 'AdventureWorks2019', schema 'IoT'.
    
```

The status bar at the bottom indicates "Query completed with errors."

- **SELECT DeviceOwner → Denied**

The screenshot shows the SQL Server Management Studio interface with the following details:

- Object Explorer:** Shows the database structure for AdventureWorks2019, including tables like System Tables, External Tables, Graph Tables, and various department and employee tables.
- SQL Query Editor:** Contains the following T-SQL code:


```
--IoTFieldTechnician
--INSERT MaintenanceLog
USE AdventureWorks2019;
GO

-- INSERT INTO IoT.MaintenanceLog (DeviceID, TechnicianName, Notes)
VALUES (1, 'Tech John', 'Performed routine inspection.');

--UPDATE MaintenanceLog
UPDATE IoT.MaintenanceLog
SET Notes = 'Updated notes by technician.'
WHERE LogID = 1;

--SELECT DeviceOwner
SELECT * FROM IoT.DeviceOwner;
```
- Messages Window:** Displays error message 229, Level 14, State 5, Line 81: "The SELECT permission was denied on the object 'DeviceOwner', database 'AdventureWorks2019', schema 'iot'." Completion time: 2025-11-23T14:58:44.5235650-05:00.
- Status Bar:** Shows the query completed with errors, and the session information: localhost\SQL2025 (17.0 RTM) IoTFieldTechLogin (76) AdventureWorks2019 00:00:00 0 rows.

- **INSERT Telemetry → Denied**

The screenshot shows the SQL Server Management Studio interface with the following details:

- Object Explorer:** Shows the database structure for AdventureWorks2019, including tables like System Tables, External Tables, Graph Tables, and various department and employee tables.
- SQL Query Editor:** Contains the following T-SQL code:


```
--IoTFieldTechnician
--INSERT MaintenanceLog
USE AdventureWorks2019;
GO

-- INSERT INTO IoT.MaintenanceLog (DeviceID, TechnicianName, Notes)
VALUES (1, 'Tech John', 'Performed routine inspection.');

--UPDATE MaintenanceLog
UPDATE IoT.MaintenanceLog
SET Notes = 'Updated notes by technician.'
WHERE LogID = 1;

--SELECT DeviceOwner
SELECT * FROM IoT.DeviceOwner;

-- INSERT Telemetry
INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed)
VALUES (1, SYSDATETIME(), 20);
```
- Messages Window:** Displays error message 229, Level 14, State 5, Line 85: "The INSERT permission was denied on the object 'Telemetry', database 'AdventureWorks2019', schema 'iot'." Completion time: 2025-11-23T14:59:43.3467425-05:00.
- Status Bar:** Shows the query completed with errors, and the session information: localhost\SQL2025 (17.0 RTM) IoTFieldTechLogin (76) AdventureWorks2019 00:00:00 0 rows.

SecurityComplianceOfficer

- All SELECT operations → Success

File Edit View Query Git Project Tools Window Help | Search | Solution

AdventureWorks2019 | Execute | Copilot | Table View |

Object Explorer

```

84     INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed)
85         VALUES (1, SYSDATETIME(), 20);
86
87
88
89 --SecurityComplianceOfficer
90 USE Adventureworks2019;
91 GO
92
93 -- SELECT Device
94 SELECT TOP 10 * FROM IoT_Device;
95
96 -- SELECT DeviceOwner
97 SELECT TOP 10 * FROM IoT_DeviceOwner;
98
99 -- SELECT Telemetry
100 SELECT TOP 10 * FROM IoT_Telemetry;
101
102
103

```

Results | Messages

DeviceID	SerialNumber	DeviceType	FirmwareVersion	RegistrationDate	LastHeartbeat	Status
1	SN0001	SmartBike	1.0	2025-11-23 19:27:19	NULL	Active

Query executed successfully.

localhostSQL2025 (17.0 RTM) SecurityComplianceLog.. AdventureWorks2019 00:00:00 1 rows

Type here to search

File Edit View Query Git Project Tools Window Help | Search | Solution

AdventureWorks2019 | Execute | Copilot | Table View |

Object Explorer

```

84     INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed)
85         VALUES (1, SYSDATETIME(), 20);
86
87
88
89 --SecurityComplianceOfficer
90 USE Adventureworks2019;
91 GO
92
93 -- SELECT Device
94 SELECT TOP 10 * FROM IoT_Device;
95
96 -- SELECT DeviceOwner
97 SELECT TOP 10 * FROM IoT_DeviceOwner;
98
99 -- SELECT Telemetry
100 SELECT TOP 10 * FROM IoT_Telemetry;
101
102
103

```

Results | Messages

OwnerID	DeviceID	DeviceName	SensitivityInfo
---------	----------	------------	-----------------

Query executed successfully.

localhostSQL2025 (17.0 RTM) SecurityComplianceLog.. AdventureWorks2019 00:00:00 0 rows

Type here to search

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the AdventureWorks2019 database is selected. In the center pane, a query window titled 'Task6.sql - A...celogin (81)' is open, displaying a script. The script includes several comments and a single INSERT statement:

```

84    INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed)
85        VALUES (1, SYSDATETIME(), 20);
86
87
88
89    --SecurityComplianceOfficer
90    USE Adventureworks2019;
91    GO
92
93    -- SELECT Device
94    SELECT TOP 10 * FROM IoT.Device;
95
96    -- SELECT DeviceOwner
97    SELECT TOP 10 * FROM IoT.DeviceOwner;
98
99    -- SELECT Telemetry
100   SELECT TOP 10 * FROM IoT.Telemetry
101
102
103

```

Below the script, the results pane shows a table with one row of data:

TelemetryID	DeviceID	Timestamp	Speed	Cadence	Temperature	BatteryLevel	GPSLatitude	GPSLongitude
1	1	2025-11-23 14:28:48	12.00	85	23.00	88.00	40.12800	-74.006000

A status bar at the bottom indicates 'Query executed successfully.'

- All DML operations (INSERT/UPDATE/DELETE) → Denied

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, the AdventureWorks2019 database is selected. In the center pane, a query window titled 'Task6.sql - A...celogin (81)' is open, displaying a script. The script includes several comments and a single INSERT statement:

```

82    INSERT INTO IoT.Telemetry (DeviceID, Timestamp, Speed)
83        VALUES (1, SYSDATETIME(), 20);
84
85
86
87
88
89    --SecurityComplianceOfficer
90    USE Adventureworks2019;
91    GO
92
93    -- SELECT Device
94    SELECT TOP 10 * FROM IoT.Device;
95
96    -- SELECT DeviceOwner
97    SELECT TOP 10 * FROM IoT.DeviceOwner;
98
99    -- SELECT Telemetry
100   SELECT TOP 10 * FROM IoT.Telemetry;
101
102
103
104
105
106
107

```

Below the script, the messages pane shows an error message:

```

Msg 229, Level 14, State 5, Line 103
The INSERT permission was denied on the object 'Device', database 'Adventureworks2019', schema 'iot'.
Completion time: 2025-11-23T15:00:22.4736927-05:00

```

A status bar at the bottom indicates 'Query completed with errors.'

```

96 -- SELECT DeviceOwner
97 SELECT TOP 10 * FROM IoT.DeviceOwner;
98
99 -- SELECT Telemetry
100 SELECT TOP 10 * FROM IoT.Telemetry;
101
102 -- INSERT
103 INSERT INTO IoT.Device (SerialNumber, DeviceType, FirmwareVersion, Status)
104 VALUES ('TEST01', 'SmartBike', '1.0', 'Active');
105
106 --UPDATE
107 UPDATE IoT.Device
108 SET Status = 'Inactive'
109 WHERE DeviceID = 1;
110
111 --DELETE
112 DELETE FROM IoT.Telemetry WHERE TelemetryID = 1;
113

```

Messages

Msg 229, Level 14, State 5, Line 112
The DELETE permission was denied on the object 'Telemetry', database 'AdventureWorks2019', schema 'iot'.
Completion time: 2025-11-23T15:09:39.0460240-05:00

Query completed with errors.

```

92
93 -- SELECT Device
94 SELECT TOP 10 * FROM IoT.Device;
95
96 -- SELECT DeviceOwner
97 SELECT TOP 10 * FROM IoT.DeviceOwner;
98
99 -- SELECT Telemetry
100 SELECT TOP 10 * FROM IoT.Telemetry;
101
102 -- INSERT
103 INSERT INTO IoT.Device (SerialNumber, DeviceType, FirmwareVersion, Status)
104 VALUES ('TEST01', 'SmartBike', '1.0', 'Active');
105
106 --UPDATE
107 UPDATE IoT.Device
108 SET Status = 'Inactive'
109 WHERE DeviceID = 1;
110

```

Messages

Msg 229, Level 14, State 5, Line 101
The UPDATE permission was denied on the object 'Device', database 'AdventureWorks2019', schema 'iot'.
Completion time: 2025-11-23T15:09:01.4071926-05:00

Query completed with errors.

(Reference: Task6.sql)

All results matched the expected behavior for each role.

7. Conclusion

This lab gave me hands-on experience with building an IoT-oriented SQL Server backend using real database engineering principles.

I learned how to design a schema that supports high-volume telemetry, how to write a transactional ingestion procedure that ensures data integrity, and how to handle concurrency issues that appear when multiple sessions access the same data.

I also gained practical experience with SQL Server's security model by creating customized roles with specific permissions and testing each one individually.

The combination of transactions, isolation levels, and RBAC helped me understand how real IoT systems protect data consistency and enforce privacy rules.

Overall, the lab strengthened my understanding of SQL Server in a realistic and meaningful way.