

Lab 8: Transactions and permissions

Course: SQL Server Database Development (ITE-5223)

Database: AdventureWorks2019

Schema Used: Reporting

Student: Yi-Chun Lien – N01745009

1. Executive Summary

This report documents the design, implementation, and validation of transactional integrity, concurrency control, and role-based access control for the AdventureWorks IoT Operations Platform.

The platform ingests high-volume telemetry from SmartBike consumer devices and industrial manufacturing sensors.

To meet enterprise requirements, the database solution ensures:

- ACID-compliant multi-step telemetry ingestion
- Protection against partial writes and corrupted packets
- Concurrency handling under high-throughput workloads
- Least-privilege access control for IoT departments
- Full auditability and error logging

The report summarizes schema creation, transactional workflow logic, failure scenario handling, concurrency behavior under different isolation levels, and validation of role-based permissions.

2. IoT Business Case Overview

AdventureWorks expanded its business into IoT-enabled devices in two major divisions:

1. SmartBike

- Consumer bicycles with GPS, cadence, temperature, crash detection, and battery sensors
- Sends telemetry every 30 seconds

2. Manufacturing IoT

- Factory-floor devices monitoring temperature, vibration, operational status
- Used for predictive maintenance and real-time monitoring

The backend database processes more than 250,000 telemetry messages per hour.

To ensure reliability and regulatory compliance (GDPR / CCPA), the system must:

- Prevent partial telemetry ingestion
- Detect invalid or corrupted sensor data
- Maintain consistent device state under concurrent operations
- Enforce strict security boundaries between IoT teams

This lab simulates a production IoT backend for both consumer and industrial devices.

3. Transaction Workflow

The stored procedure `IoT.usp_ProcessTelemetryCycle` performs atomic processing of a complete telemetry cycle.

It ensures that all five dependent records are created successfully, or the entire operation rolls back.

Processing Steps:

1. Validate Device Registration:
 - Reject unregistered, invalid, or decommissioned devices.
2. Validate Sensor Inputs
 - Temperature range -40 to 85°C
 - Battery level $0-100\%$
 - Latitude/Longitude cannot be NULL and must be within valid ranges
3. Insert Raw Telemetry into `IoT.Telemetry`.
4. Update LastHeartbeat in `IoT.Device`.
5. Insert DeviceHealth Summary
 - Computes TemperatureStatus, BatteryStatus, VibrationStatus
 - Marks device as Healthy/Unhealthy
6. Insert Alerts into `IoT.Alert`
 - Low/Critical battery
 - High temperature
7. Insert TelemetryAudit Record
 - Processing duration
 - Number of records inserted
 - Status (Completed or CompletedWithAlerts)
8. Commit Transaction

Rollback Conditions

If any failure occurs:

- Missing or invalid DeviceID
- Sensor readings outside acceptable ranges
- Null or invalid coordinates
- Any conversion error or unexpected exception

Then:

- Entire transaction rolls back
- Detailed error entry is written to `IoT.TelemetryErrorLog`

This ensures 100% data integrity during telemetry ingestion.

4. Failure Scenarios

Three failure scenarios were executed to verify correct rollback and error logging.

Failure Case 1 – Invalid DeviceID

- DeviceID = 99999 (not registered)
- Stored procedure throws error "Device not registered"

- Result:
 - No telemetry, health, alert, audit rows created
 - IoT.TelemetryErrorLog contains an error entry with ErrorStep = "Validate Device"

The image displays two screenshots of the SQL Server Enterprise Manager interface, illustrating a failed query execution and its results.

Top Screenshot: Shows the execution of a query in the "Messages" pane. The query is a stored procedure call: `EXEC IoT_usp_ProcessTelemetryCycle`. The query parameters are: `@DeviceID = 99999`, `@Speed = 10`, `@Cadence = 80`, `@Temperature = 30`, `@BatteryLevel = 50`, `@Latitude = 43.6532`, `@Longitude = -79.3832`, and `@RawPayload = N'Case1_InvalidDeviceID'`. The query execution failed with the error: "Msg 61001, Level 16, State 1, Procedure IoT_usp_ProcessTelemetryCycle, Line 34 (Batch Start Line 304): Device not registered or is decommissioned." The completion time is 2025-11-19T10:40:47.1953982-05:00.

Bottom Screenshot: Shows the results of the query execution in the "Results" pane. The query execution was successful, but no rows were returned. The results pane displays the following columns: `TelemetryID`, `DeviceID`, `Timestamp`, `Speed`, `Cadence`, `Temperature`, `BatteryLevel`, `GPSTLatitude`, `GPSTLongitude`, `HealthID`, `DeviceID`, `Timestamp`, `IsHealthy`, `TemperatureStatus`, `BatteryStatus`, `VibrationStatus`, `AlertID`, `DeviceID`, `Timestamp`, `AlertType`, `Severity`, `Description`, `AuditID`, `DeviceID`, `Timestamp`, `ProcessingDuration`, `RecordsInserted`, and `Status`.

Object Explorer: Connect - localhost (SQL Server 16.0.1000.6 - Allison\j8903 (51))

Query: lab8.sql - Ad...N\j8903 (51)*

```

313 @RawPayload = N'Case1_InvalidDeviceID';
314
315 SELECT TOP 5 * FROM IoT_Telemetry ORDER BY TelemetryID DESC;
316 SELECT TOP 5 * FROM IoT_DeviceHealth ORDER BY HealthID DESC;
317 SELECT TOP 5 * FROM IoT_Alert ORDER BY AlertID DESC;
318 SELECT TOP 5 * FROM IoT_TelemetryAudit ORDER BY AuditID DESC;
319
320 SELECT TOP 5 * FROM IoT_TelemetryErrorLog ORDER BY ErrorID DESC;
321
322
323
324
325

```

ErrorID	DeviceID	Timestamp	ErrorMessage	ErrorStep	RawPayload
1	4	99999	2025-11-19 15:40:47.167	Device not registered or is decommissioned.	Validate Device
2	3	99999	2025-11-19 15:40:29.200	Device not registered or is decommissioned.	Validate Device

Query executed successfully. localhost (16.0 RTM) | ALLISON\j8903 (51) | AdventureWorks2019 | 00:00:00 | 2 rows

Error List: Entire Solution | 0 Errors | 0 Warnings | 0 Messages | Build Only | Search Error List

Failure Case 2 – BatteryLevel > 100

- Violates sensor rules (0–100%)
- Stored procedure throws custom error
- Result:
 1. Transaction rolled back
 2. Error logged with ErrorStep = "Validate input ranges"

Object Explorer: Connect - localhost (SQL Server 16.0.1000.6 - Allison\j8903 (51))

Query: lab8.sql - Ad...N\j8903 (51)*

```

325 VALUES ('SN-001', 'SmartBike', '1.0.0');
326
327 EXEC IoT.usp_ProcessTelemetryCycle
328     @DeviceID = 1,
329     @Speed = 20,
330     @Cadence = 60,
331     @Temperature = 25,
332     @BatteryLevel = 150,
333     @Latitude = 43.6532,
334     @Longitude = -79.3832,
335     @RawPayload = N'Case2_BatteryAbove100';
336
337
338

```

Msg 51003, Level 16, State 1, Procedure IoT.usp_ProcessTelemetryCycle, Line 43 [Batch Start Line 326]
BatteryLevel must be between 0 and 100.

Completion time: 2025-11-19T10:46:17.0657203-05:00

Query completed with errors. localhost (16.0 RTM) | ALLISON\j8903 (51) | AdventureWorks2019 | 00:00:00 | 0 rows

Error List: Entire Solution | 0 Errors | 0 Warnings | 0 Messages | Build Only | Search Error List

The top screenshot shows a SQL query executed successfully. The query is as follows:

```

332  @BatteryLevel = 150,
333  @Latitude = 43.6532,
334  @Longitude = -79.3832,
335  @RawPayload = N'Case2_BatteryAbove100';
336
337  SELECT TOP 5 * FROM IoT_Telemetry ORDER BY TelemetryID DESC;
338  SELECT TOP 5 * FROM IoT_DeviceHealth ORDER BY HealthID DESC;
339  SELECT TOP 5 * FROM IoT_Alert ORDER BY AlertID DESC;
340  SELECT TOP 5 * FROM IoT_TelemetryAudit ORDER BY AuditID DESC;
341
342  SELECT TOP 5 * FROM IoT_TelemetryErrorLog ORDER BY ErrorID DESC;
343
344

```

The bottom screenshot shows the same query executed, but with an error message displayed in the Results pane. The error message is:

ErrorID	DeviceID	Timestamp	ErrorMessage	ErrorStep	RawPayload
1	6	2025-11-19 15:46:17.066	BatteryLevel must be between 0 and 100.	Validate input ranges	Case2_BatteryAbove100
2	5	2025-11-19 15:43:10.455	Device not registered or is decommissioned.	Validate Device	Case2_BatteryAbove100
3	4	2025-11-19 15:40:47.167	Device not registered or is decommissioned.	Validate Device	Case1_InvalidDeviceID
4	3	2025-11-19 15:40:29.200	Device not registered or is decommissioned.	Validate Device	Case1_InvalidDeviceID

Failure Case 3 – Missing GPS Coordinates

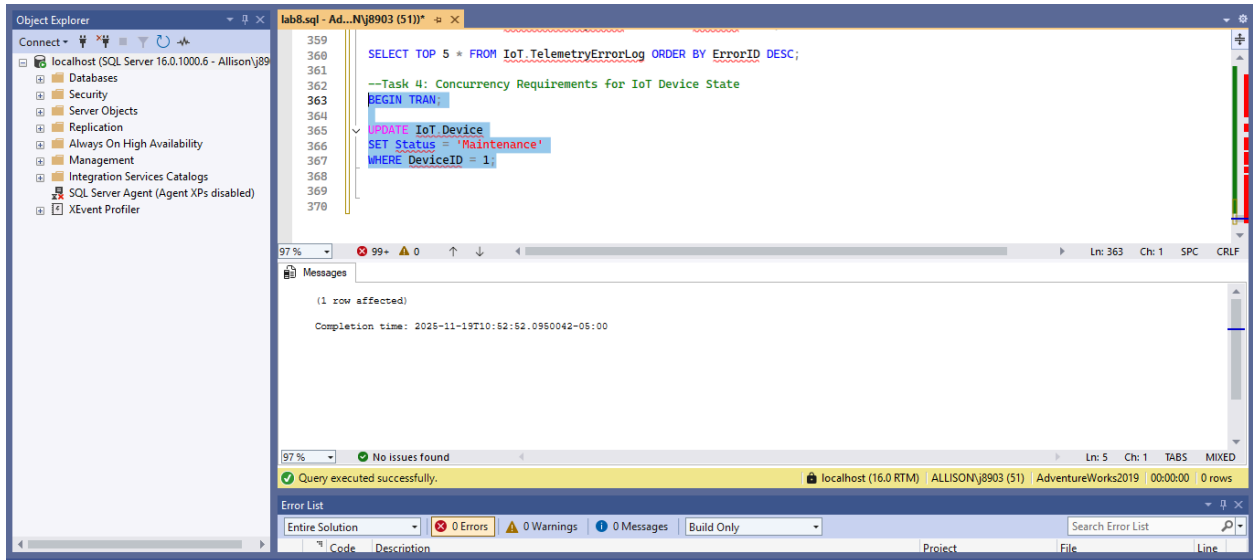
- Latitude = NULL
- Procedure rejects NULL GPS
- Result:
 1. The transaction was rolled back to prevent partial telemetry ingestion.
 2. TelemetryErrorLog contains correct error and RawPayload

5. Concurrency Analysis

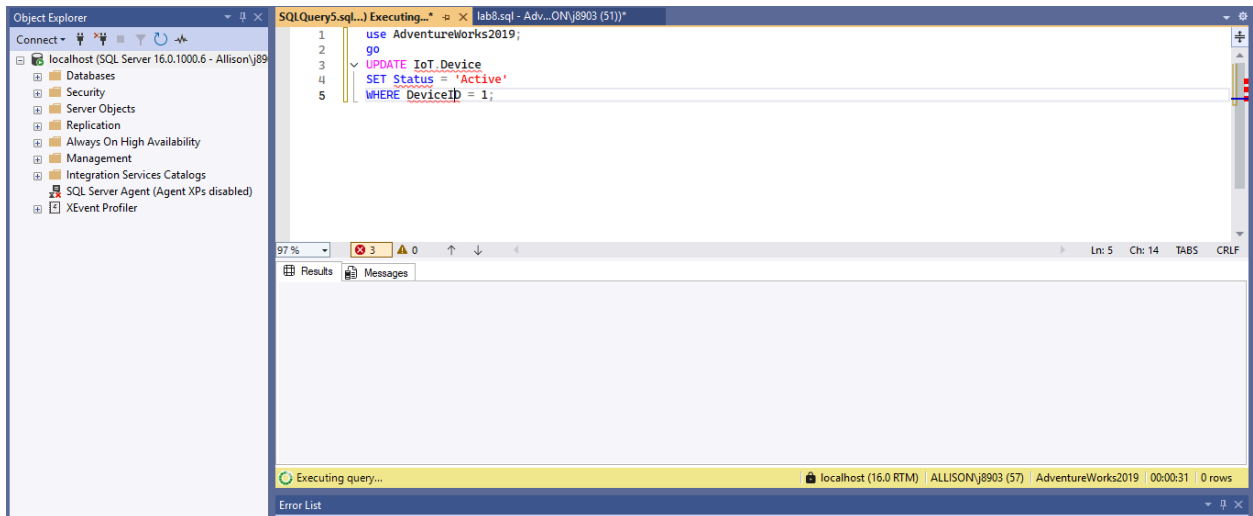
Multiple concurrency scenarios were executed using two SQL Server sessions to observe blocking behavior and isolation level differences.

1 Blocking Scenario

A:



B:



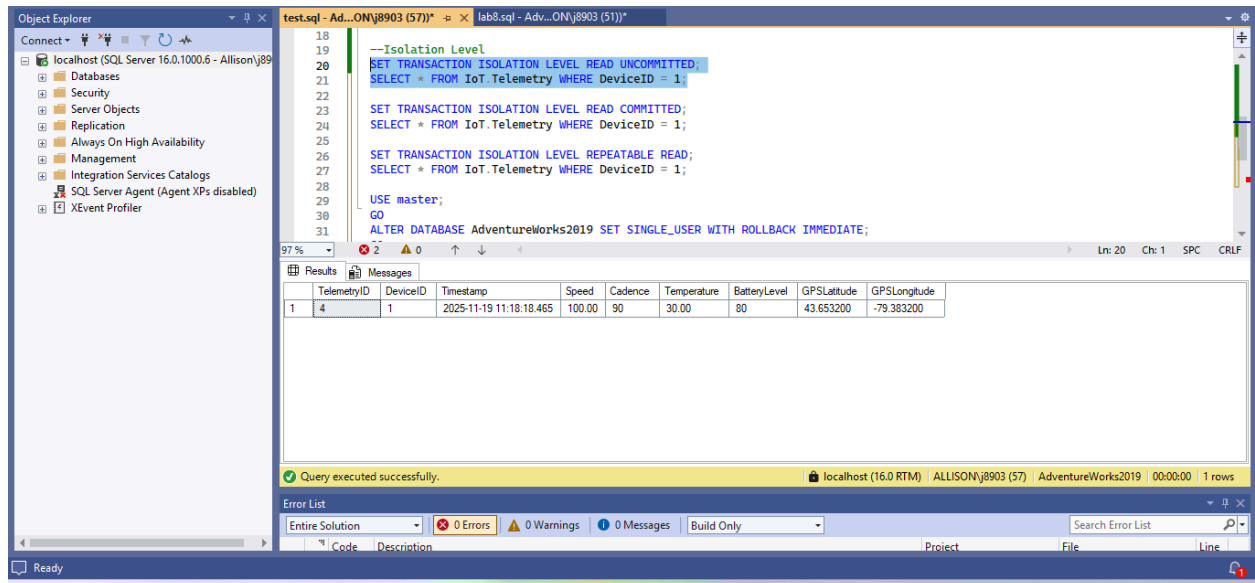
Observation:

B becomes BLOCKED

This simulates high-volume ingestion conflicting with maintenance actions.

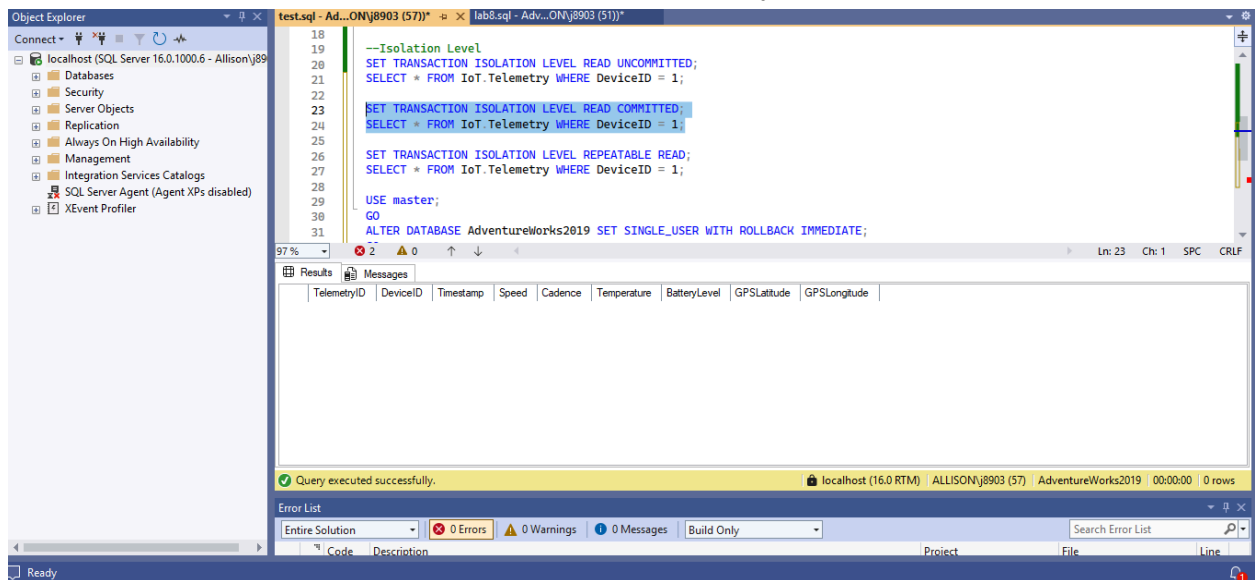
2 Dirty Read – READ UNCOMMITTED

- B can see uncommitted telemetry inserts
- Demonstrates dirty reads allowed under this isolation level



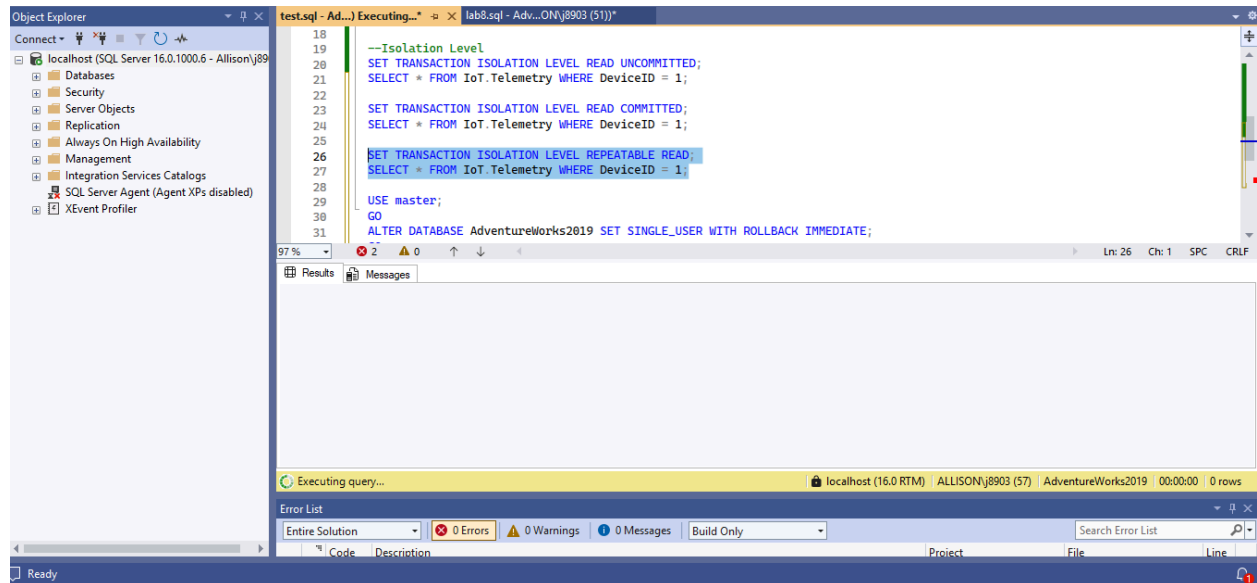
3 READ COMMITTED

- B waits for A to commit
- Prevents dirty reads
- Default and safest isolation level for transactional systems



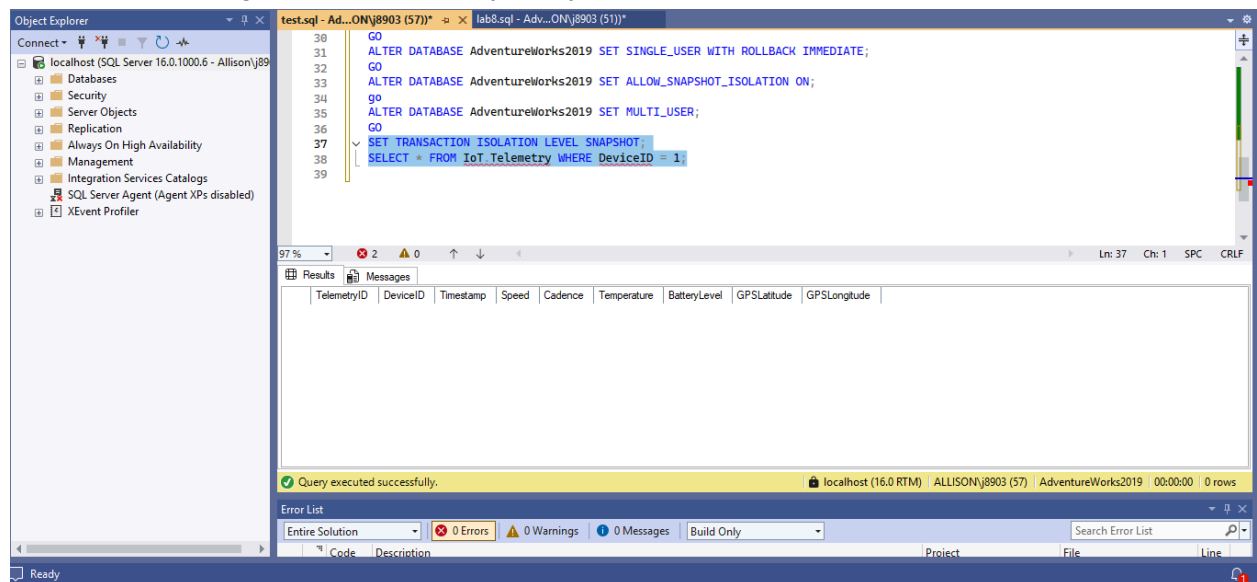
4 REPEATABLE READ

- Prevents changes to rows read by B
- A cannot update those rows until B commits



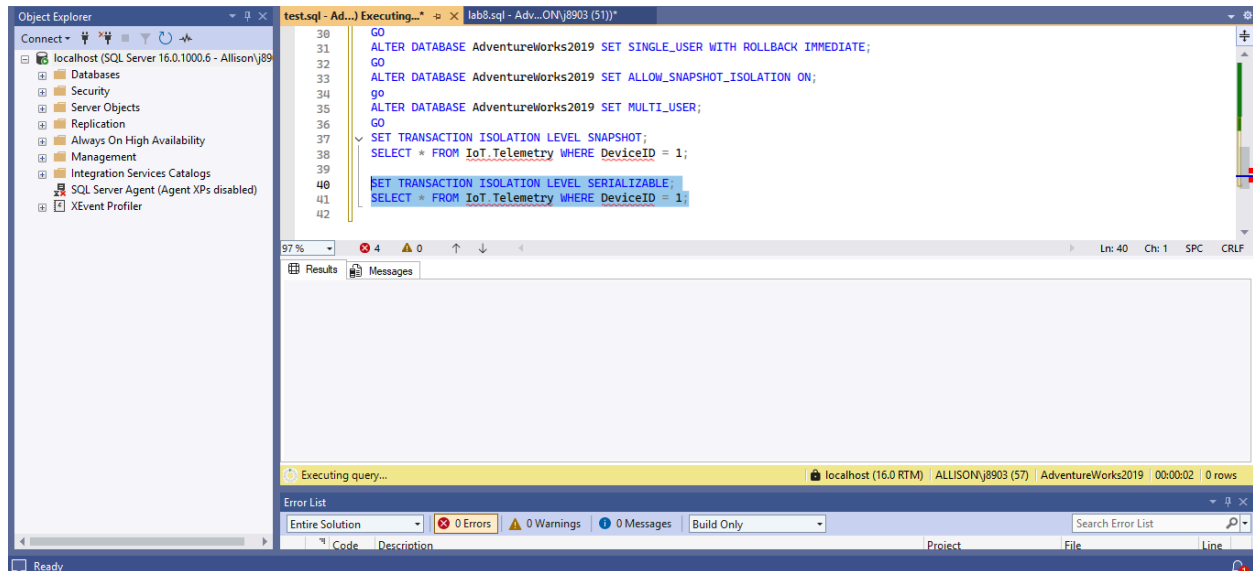
5 SNAPSHOT Isolation

- B reads a consistent version of the data
- Does not block nor get blocked
- Ideal for high-volume telemetry analytics



6 SERIALIZABLE

- Under SERIALIZABLE isolation, the transaction in B logically restricts concurrent inserts into the same key range.
- Although the INSERT in A was not visibly blocked due to the lack of an index on DeviceID, SERIALIZABLE is still designed to prevent phantom reads and enforce the strictest ACID guarantees.
- In a properly indexed environment, the Session A insert would be blocked until Session B completes.



6. Role-Based Access Control

Roles were created to match real-world IoT departmental responsibilities while following the Least Privilege Principle.

1 IoTDeviceAgent

Allowed:

- INSERT / UPDATE on IoT.Device
- SELECT on IoT.Device

Denied:

- Any INSERT/UPDATE/DELETE on telemetry-related tables

Used for device registration staff.

2 TelemetryIngestionService

Allowed:

INSERT Telemetry, DeviceHealth, Alert, Audit

UPDATE Device.LastHeartbeat

SELECT Device

Denied:

UPDATE Device metadata (SerialNumber, FirmwareVersion, etc.)

Represents automated ingestion microservices.

3 IoTAnalyst

Allowed:

- SELECT all IoT tables

Denied:

- INSERT / UPDATE / DELETE any IoT table

Analysts can read data but cannot affect operations.

4 IoTFieldTechnician

Denied:

- SELECT DeviceOwner or any customer-linked data
- SELECT all telemetry data (Per privacy restrictions)

5 SecurityComplianceOfficer

Allowed:

- SELECT Device, DeviceOwner, Telemetry, Alerts, ErrorLog

Denied:

- All modification operations

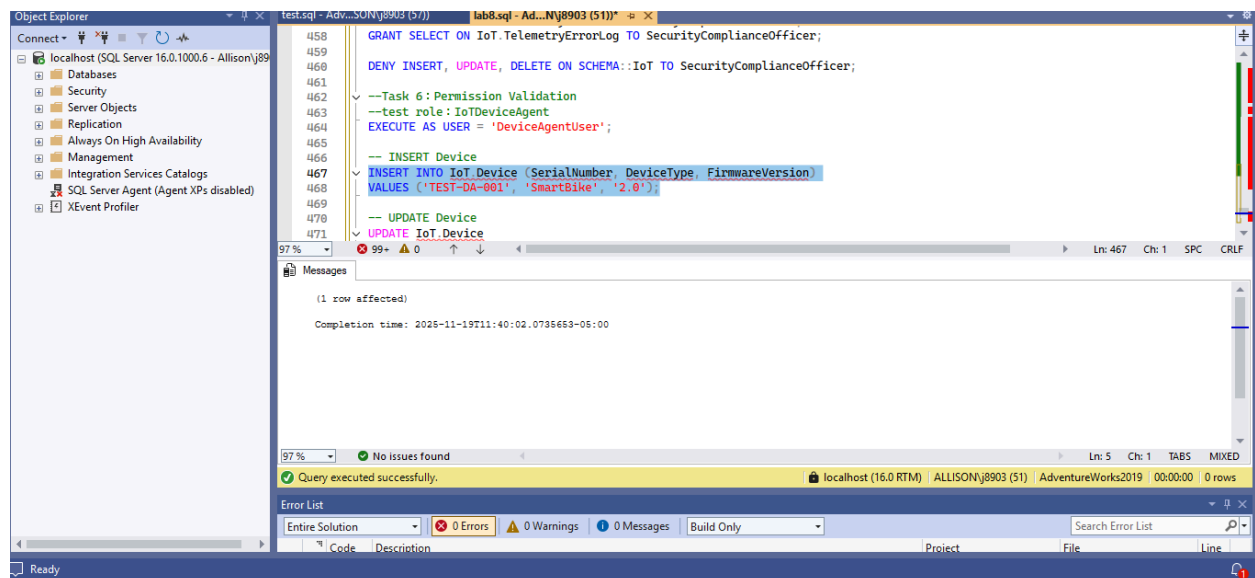
Used for GDPR/CCPA regulatory review.

7. Permission Validation

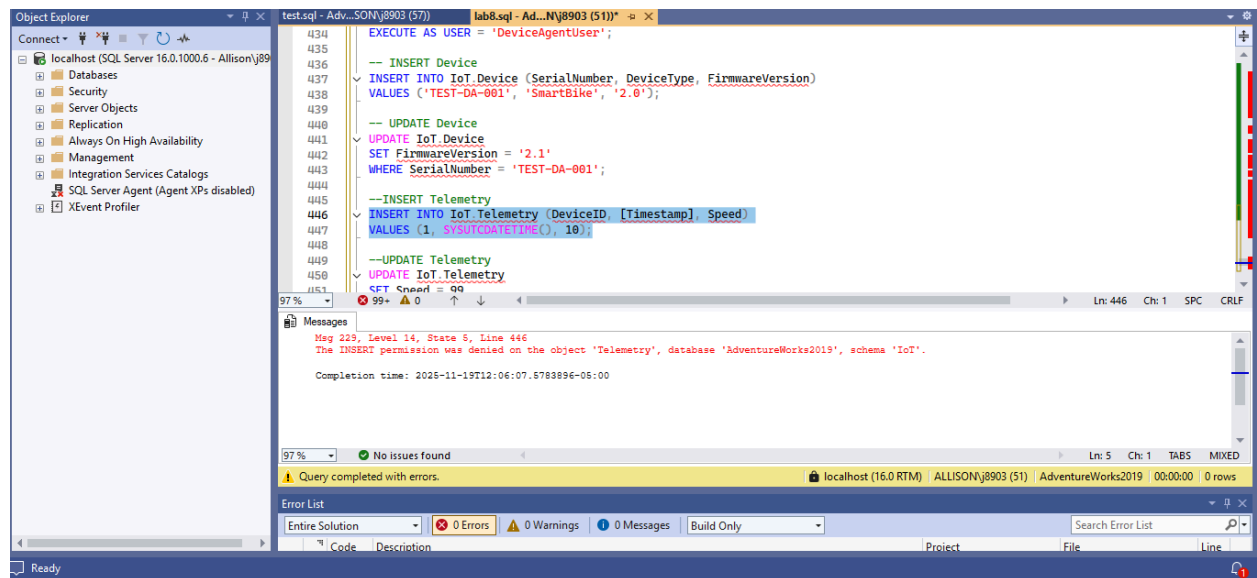
Each role was tested using EXECUTE AS USER.

1.IoTDeviceAgent

INSERT Device → success

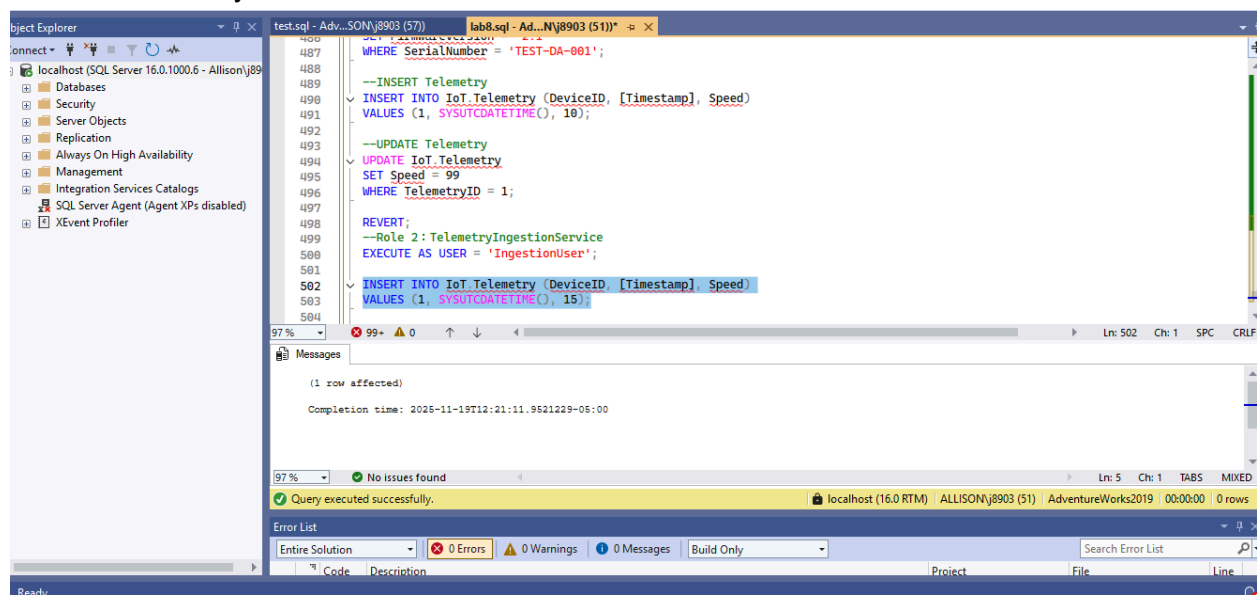


INSERT Telemetry → denied

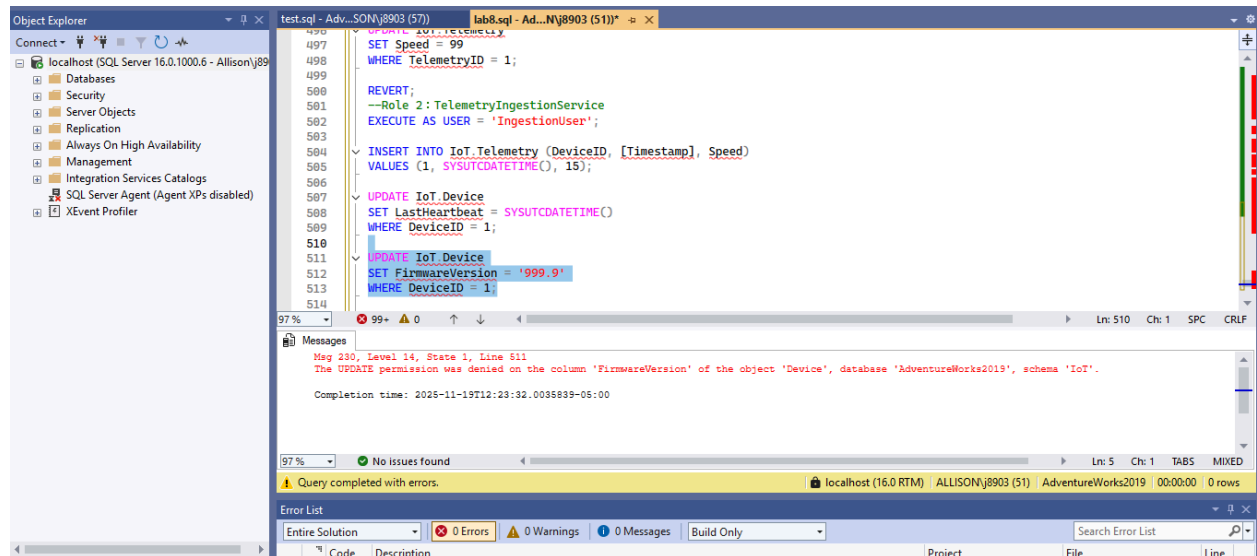


2,TelemetryIngestionService

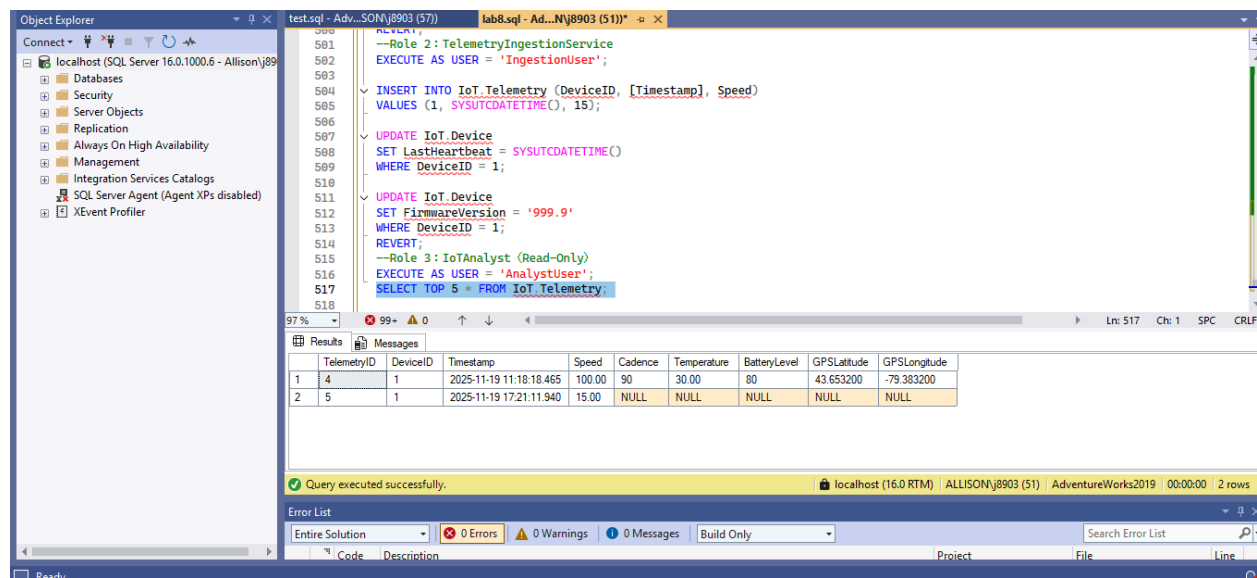
INSERT Telemetry → success



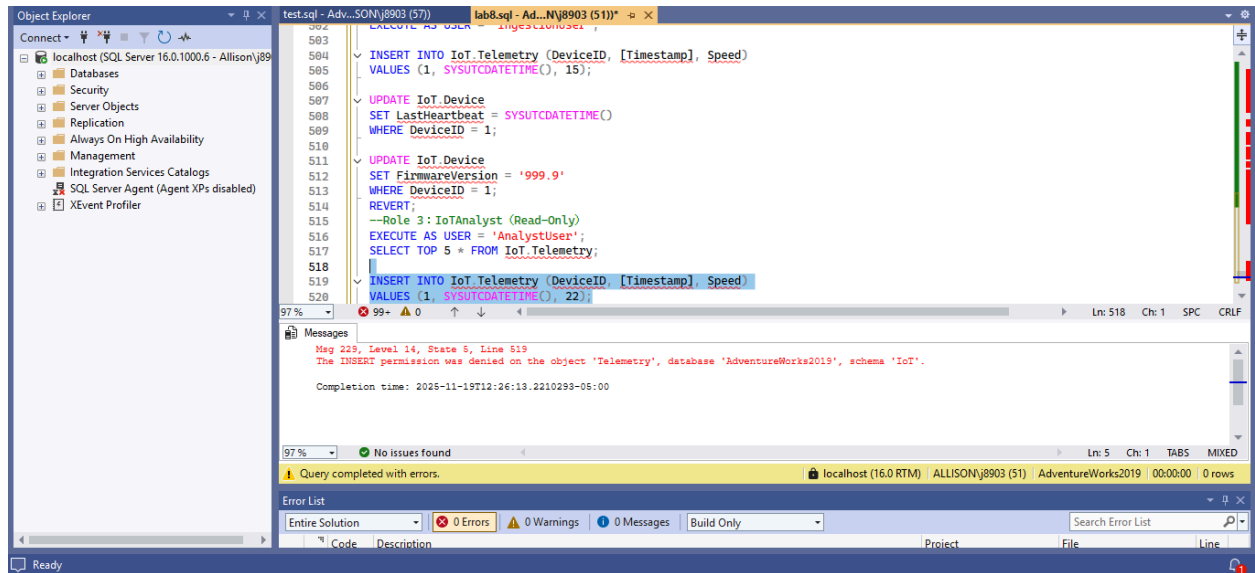
UPDATE FirmwareVersion → denied



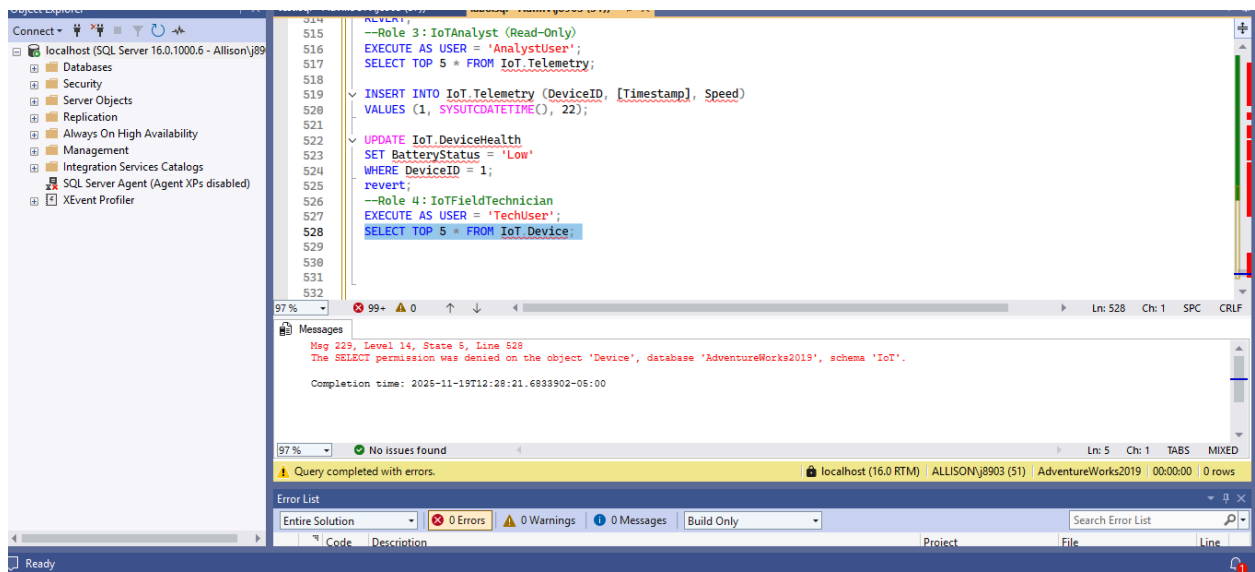
3.IoTAnalyst SELECT → success



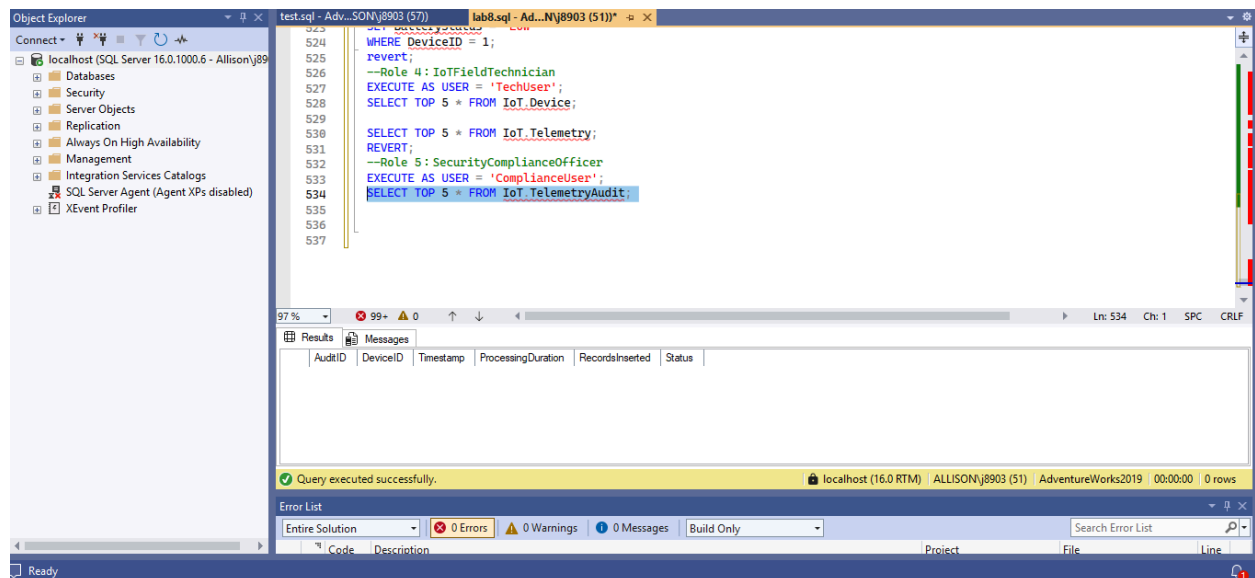
INSERT → denied



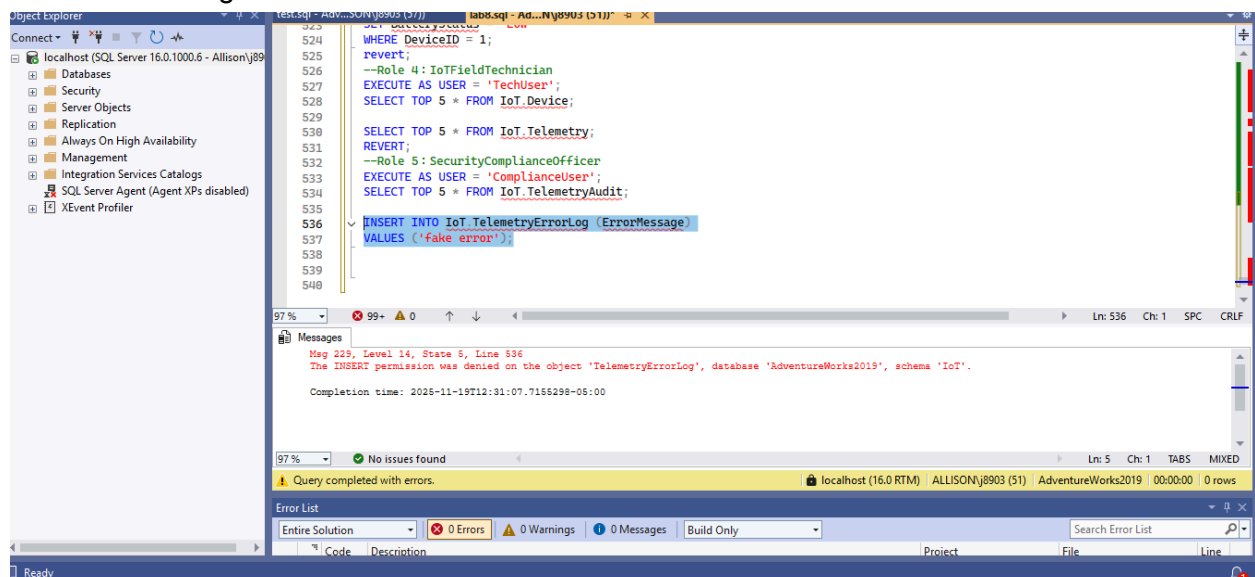
4. IoTFieldTechnician SELECT Device → denied



5. SecurityComplianceOfficer SELECT Audit → success



INSERT ErrorLog → denied



All tests confirmed permissions were correctly enforced.

8. Conclusion

This lab successfully implemented a production-grade IoT backend aligned with enterprise database engineering standards.

Key achievements include:

1.ACID-Compliant Telemetry Pipeline

Prevents partial sensor data ingestion and ensures reliable device health computation.

2.Advanced Concurrency Control

Blocking, dirty reads, repeatable reads, and snapshot isolation behaviors were validated across real SQL Server sessions.

3. Secure Role-Based Access Control

Strict least-privilege permissions ensure compliance with privacy regulations (GDPR/CCPA).

4. Professional Logging & Auditing

TelemetryErrorLog and TelemetryAudit provide operational traceability for monitoring, debugging, and compliance review.

Overall, the system is secure, reliable, and scalable for large-scale real-time IoT deployments.