Lab 8 – IoT Operations Platform Technical Report

NAME – PARWINDER SINGH

HUMBER ID – N01730928

1. Executive Summary

This lab focused on building a realistic IoT Operations Platform inside SQL Server. The main goals were to ensure reliable telemetry ingestion, protect data integrity, manage concurrency, and apply principle-based access control. I designed a set of IoT tables, implemented a fully transactional telemetry ingestion stored procedure, tested multiple failure conditions, evaluated concurrency and isolation levels, and created a complete role-based permission model. The outcome is an IoT backend that is safe, consistent, secure, and compliant with enterprise standards.

2. IoT Business Case Summary

AdventureWorks is rolling out two major IoT product lines:

1. **SmartBike IoT** – consumer devices tracking GPS, cadence, battery level, speed, and crash indicators.
2. **Manufacturing IoT** – factory-floor sensors monitoring temperature, vibration, and machine performance.

Each device sends telemetry every **30 seconds**, resulting in **250,000+ records per hour**.
The database must:

- Store telemetry reliably
- Prevent partial data writes
- Ensure accurate device state
- Support thousands of concurrent operations
- Enforce strict access control between IoT teams
- Protect customer-linked device information

The lab simulated building this backend.

3. Transactional Workflow Description

A stored procedure (IoT.usp_IngestTelemetry) was created to process a full telemetry cycle. The workflow uses **explicit transactions**, **TRY/CATCH**, and **rollback logic** to guarantee data integrity.

Transactional Steps

1. **Validate Device**
    - Ensure DeviceID exists
    - Prevent ingestion for unregistered devices
2. **Validate Sensor Inputs**
    - Reject impossible values (battery >100, temperature outside design range)
    - Reject NULL GPS coordinates
    - This protects data quality
3. **Insert Telemetry Record**
    - Speed, Cadence, Temperature, BatteryLevel, GPS
4. **Update Device LastHeartbeat**
    - Tracks most recent communication
5. **Insert DeviceHealth Summary**
    - TemperatureStatus (Normal/High/Low)
    - BatteryStatus
    - VibrationStatus = Normal (for this lab)
    - IsHealthy flag derived from statuses
6. **Generate Alerts** (if required)
    - High Temperature
    - Low Battery
    - Invalid Coordinates
    - Alerts stored in IoT.Alert
7. **Insert TelemetryAudit Record**
    - Processing duration
    - Number of inserted records
    - Status = “Success”

Rollback Logic

If ANY failure occurs:

- Entire transaction is rolled back
- A detailed error is logged in IoT.TelemetryErrorLog
- No partial writes occur

This ensures **atomicity**, **consistency**, and **auditability**.

4. Concurrency Findings

In Task 4, several concurrency scenarios were tested using two SSMS windows.

A. Concurrent Updates

- When Window A opened a transaction and updated a Device row,
- Window B trying to update the same row became **blocked**.
- SQL Server placed an **exclusive lock (X)** on the row until commit.
  Finding: IoT.Device updates must be designed to avoid long transactions.

B. Long-Running Transaction Blocking Ingestion

- During a WAITFOR DELAY update on IoT.Device,
- A telemetry ingestion call was blocked.
  Finding: Telemetry ingestion depends on quick access to IoT.Device, and long maintenance operations can block ingestion. The platform needs operational planning or row-versioning strategies.

C. Isolation Level Testing

Each isolation level produced different behavior:

Isolation Level	**Expected Behavior**	**Observation**
READ UNCOMMITTED	Dirty reads allowed	Returned uncommitted data instantly
READ COMMITTED (default)	No dirty reads	Most stable for normal reads
REPEATABLE READ	Holds read locks	Prevented changes to scanned rows
SNAPSHOT	Reads versioned data	Worked without blocking
SERIALIZABLE	Highest isolation	Most blocking, safest but slowest

Finding:
For high-volume IoT workloads, **SNAPSHOT** or **READ COMMITTED SNAPSHOT** would reduce blocking without sacrificing consistency.

5. Role-Based Permissions Design

To satisfy separation of duties and privacy requirements, five roles were designed.

1. IoTDeviceAgent

- Can register or update device metadata
- Cannot insert telemetry
- Cannot view device-owner data

2. TelemetryIngestionService

- Automated account
- Can insert Telemetry, DeviceHealth, TelemetryAudit
- Can update LastHeartbeat
- Cannot read customer-linked tables

3. IoTAnalyst

- Read-only access to Telemetry and DeviceHealth
- Blocked from modifying IoT operational tables

4. IoTFieldTechnician

- Can insert/update MaintenanceLog
- Cannot view owner or customer data

5. SecurityComplianceOfficer

- Read-only access across IoT tables
- No modification permissions
- Supports auditing & regulatory investigations

Overall:
The permission model enforces **least privilege**, **data privacy**, and **operational safety**.

6. Validation of Permissions (Task 6)

Each role was validated using EXECUTE AS USER.
The following behaviors were tested:

Allowed Actions (Success Screenshots)

- DeviceAgent → Update IoT.Device
- TelemetryService → Insert IoT.Telemetry
- Analyst → Select from IoT.Telemetry
- FieldTech → Insert IoT.MaintenanceLog
- ComplianceOfficer → Select IoT.Device

Denied Actions (Error Screenshots)

- DeviceAgent → Insert Telemetry
- TelemetryService → Read DeviceOwner
- Analyst → Delete Telemetry
- FieldTech → Read DeviceOwner
- ComplianceOfficer → Update Telemetry

Findings:
All roles behaved exactly as designed:

- Allowed actions executed successfully
- Denied actions produced correct permission errors

This proves that the RBAC model was implemented correctly.

7. Conclusion

This lab successfully demonstrated how to build a secure and reliable IoT database platform in SQL Server. Through transactional controls, the system prevents partial ingestion and maintains complete integrity of telemetry cycles. Concurrency testing showed how locking affects high-volume IoT workloads, and isolation levels were compared to help identify suitable strategies.

The role-based security model enforces strict separation of duties and supports privacy regulations like GDPR and CCPA by preventing unauthorized access to customer-linked device data. Overall, the complete solution reflects real-world IoT backend requirements and follows professional database engineering practices.