This capstone project is designed to transition entry-level trainees from theoretical AWS knowledge to practical data engineering. You will build a **Banking Analytics Lakehouse** that processes daily transaction files and stores them in **Amazon S3 Tables**—a purpose-built storage tier for Apache Iceberg that provides high-performance transactional consistency.

---

1. Project Architecture

The pipeline follows a standard "Medallion" architecture (Bronze → Silver/Gold) but leverages the new S3 Tables feature for the final analytics layer.

Component	AWS Service	Purpose
Data Source	S3 (Standard)	Landing zone for raw CSV files (Bronze layer).
ETL Engine	AWS Glue	PySpark job to clean, join, and transform data.
Storage	S3 Table Bucket	Managed Iceberg tables for optimized queries (Silver/Gold layer).
Catalog	Glue Data Catalog	Metadata management and integration with Athena.
Query Engine	Amazon Athena	SQL interface to verify results.

---

2. Dataset Schema & Sample Data

The project uses three core banking datasets. You can use the following Python module to generate realistic synthetic data.

Schema Definition

1. **Customers (customers.csv):** customer_id, name, email, country.
2. **Accounts (accounts.csv):** account_id, customer_id, account_type (Savings/Current), balance.
3. **Transactions (transactions.csv):** tx_id, account_id, amount, tx_type (Credit/Debit), timestamp.

Data Generation Module

Run this script locally to generate the initial CSV files for upload to your "Landing" S3 bucket.

Python

```
import pandas as pd
import random
from datetime import datetime, timedelta

def generate_banking_data(num_records=100):
    # 1. Customers
    cust_ids = [f"CUST_{i:03}" for i in range(1, 21)]
    customers = pd.DataFrame({
        'customer_id': cust_ids,
        'name': [f"User_{i}" for i in range(1, 21)],
        'country': [random.choice(['USA', 'UK', 'India', 'Germany']) for _ in range(20)]
    })

    # 2. Accounts
    acc_ids = [f"ACC_{i:03}" for i in range(1, 21)]
    accounts = pd.DataFrame({
        'account_id': acc_ids,
        'customer_id': cust_ids,
        'account_type': [random.choice(['Savings', 'Current']) for _ in range(20)]
    })

    # 3. Transactions
    transactions = []
    for _ in range(num_records):
        transactions.append({
            'tx_id': f"TX_{random.randint(10000, 99999)}",
            'account_id': random.choice(acc_ids),
            'amount': round(random.uniform(10.0, 5000.0), 2),
            'tx_type': random.choice(['Credit', 'Debit']),
```

```
        'timestamp': (datetime.now() - timedelta(days=random.randint(0, 30))).strftime('%Y-%m-%d %H:%M:%S')
    })
tx_df = pd.DataFrame(transactions)

customers.to_csv('customers.csv', index=False)
accounts.to_csv('accounts.csv', index=False)
tx_df.to_csv('transactions.csv', index=False)
print("Sample files generated: customers.csv, accounts.csv, transactions.csv")

generate_banking_data()
```

---

3. Implementation Steps

Phase 1: Storage Setup

1. **Standard S3 Bucket:** Create a bucket named banking-raw-data-<your-id>. Upload the 3 CSVs into folders: /customers, /accounts, and /transactions.
2. **S3 Table Bucket:** * Navigate to **S3 Console > Table buckets**.
    - Create a bucket named banking-lakehouse.
    - Create a **Namespace** named analytics_db.
    - **Note:** S3 Tables automatically handle Apache Iceberg metadata.

Phase 2: Glue ETL Job

Create a Glue Spark Job (Python) with the following logic:

1. **Extract:** Read the CSVs from the standard S3 bucket.
2. **Transform:**
    - Join Transactions with Accounts on account_id.
    - Join the result with Customers on customer_id.
    - Cast amount to Decimal and timestamp to Date.
3. **Load:** Write the final "Enriched Transactions" table to the **S3 Table Bucket**.

Glue Script Snippet:

Python

```
# Configure S3 Tables Catalog
```

```
# Note: Use the S3 Tables catalog endpoint in your Glue connection settings
df_enriched = tx_df.join(acc_df, "account_id").join(cust_df, "customer_id")

# Write to S3 Tables (Iceberg format)
df_enriched.writeTo("s3tablescatalog.analytics_db.enriched_transactions") \
    .tableProperty("format", "parquet") \
    .createOrReplace()
```

---

4. Verification & Success Criteria

To pass the capstone, trainees must execute the following SQL queries in **Amazon Athena** and provide the results.

Test 1: Data Integrity

Verify that the record count in S3 Tables matches the sum of the input files.

SQL

```
SELECT count(*) FROM "s3tablescatalog"."analytics_db"."enriched_transactions";
```

Test 2: Business Logic

Identify the top 3 customers by total transaction volume in the last 30 days.

SQL

```
SELECT name, sum(amount) as total_volume
FROM "s3tablescatalog"."analytics_db"."enriched_transactions"
WHERE tx_type = 'Credit'
GROUP BY name
ORDER BY total_volume DESC
LIMIT 3;
```

Test 3: S3 Table Maintenance

Navigate to the S3 Tables console and verify that **Automatic Compaction** is enabled. This shows the trainee understands the "managed" aspect of S3 Tables compared to standard S3.

Deliverables

- **Infrastructure:** CloudFormation/Terraform template or a screenshot of the S3 Table Bucket.

- **Code:** The Python Glue script.
- **Report:** Results of the verification queries in Athena.