1. Write an SQL query to find the top 3 accounts with the highest total transaction volume for each month.

```
WITH MonthlyTransactionVolume AS (
    SELECT
        account_id,
        DATE_TRUNC('month', transaction_date) AS transaction_month,
        SUM(transaction_amount) AS total_volume,
        RANK() OVER (PARTITION BY DATE_TRUNC('month', transaction_date)
                     ORDER BY SUM(transaction_amount) DESC) AS rank
    FROM transactions
    GROUP BY account_id, DATE_TRUNC('month', transaction_date)
)
SELECT account_id, transaction_month, total_volume
FROM MonthlyTransactionVolume
WHERE rank <= 3;SELECT account_id, transaction_month, total_volume
FROM MonthlyTransactionVolume
WHERE rank <= 3;
```

This query calculates the total transaction volume per account for each month and ranks them based on the highest transaction amount. The final result retrieves only the top 3 accounts for each month.

---

2. Design a database schema to securely store and manage API keys, user details, and transaction data for a payment processing system.

Schema Design:

- Users Table: Stores user information.
- API_Keys Table: Stores API keys securely with hashing.
- Transactions Table: Stores transaction details with proper indexing and encryption.

sql

CopyEdit

```sql
CREATE TABLE Users (
    user_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE API_Keys (
    api_key_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES Users(user_id),
    hashed_api_key TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Transactions (
    transaction_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES Users(user_id),
    account_id VARCHAR(50) NOT NULL,
    transaction_amount DECIMAL(10,2) NOT NULL,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) CHECK (status IN ('Pending', 'Completed', 'Failed'))
);
```

Security measures:

- Store API keys as hashed values using bcrypt or HMAC-SHA256.
- Encrypt sensitive transaction details using AES-256 encryption.

---

3. Describe a data project you worked on. What were some of the challenges you faced?

One of my projects was processing and analyzing financial transactions for fraud detection using Azure Data Factory and Databricks.

Challenges & Solutions:

- Data Inconsistency: Different sources had different data formats. Used schema validation and transformation in Databricks.
- Performance Issues: Large-scale transaction data caused slow processing. Optimized queries and used Spark caching.
- Security Compliance: Needed to comply with PCI-DSS. Implemented encryption, access control, and anonymization.

---

4. What are the advantages and disadvantages of using a star schema versus a snowflake schema in a data warehouse?

Schema Type	Advantages	Disadvantages
Star Schema	Simple structure, faster query performance due to fewer joins, easy to understand.	Higher data redundancy, increased storage.
Snowflake Schema	Less redundancy, better data integrity, normalized tables reduce update anomalies.	Complex queries, slower performance due to multiple joins.

Use star schema for high-performance OLAP queries and snowflake schema for normalized data models with better maintainability.

---

5. Explain the differences between Hadoop, Spark, and Flink. In what scenarios would you choose one over the others?

Technology	Key Features	When to Use
Hadoop	Batch processing, fault-tolerant, disk-based storage.	Large-scale batch processing, historical data analysis.
Spark	In-memory processing, faster than Hadoop, supports batch & streaming.	Real-time analytics, iterative machine learning tasks.

Technology	Key Features	When to Use
Flink	True stream processing, event-driven architecture.	Low-latency, real-time transaction processing.

Use Hadoop for batch jobs, Spark for mixed workloads, and Flink for real-time transaction analysis.

---

6. How would you design a scalable data pipeline to process and analyze streaming transaction data in real-time?

Architecture:

- Data Ingestion: Kafka or Azure Event Hubs.
- Processing Layer: Apache Flink or Spark Streaming for real-time transformations.
- Storage: Azure Blob Storage for raw data, Snowflake for structured analysis.
- Analytics & Visualization: Power BI or Looker for reporting.

---

7. How do you handle data quality issues in a data pipeline?

- Schema Validation: Enforce data type constraints using Spark schema.
- Data Deduplication: Use window functions or hash checks to remove duplicates.
- Error Logging & Monitoring: Implement Azure Monitor and Databricks logs.

---

8. Given two sorted lists, write a function to merge them into one sorted list.

python

CopyEdit

```
def merge_sorted_lists(list1, list2):
    merged = []
    i, j = 0, 0
    while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            merged.append(list1[i])
            i += 1
```

```
        else:
            merged.append(list2[j])
            j += 1
    merged.extend(list1[i:])
    merged.extend(list2[j:])
    return merged
```

9. Design a system that can detect fraudulent transactions in real-time for a global payment network.

Architecture:

1. Data Ingestion:
    - Use Kafka to stream transaction data from multiple sources (banks, merchants, payment processors).
    - Store raw transactions in Azure Data Lake or Amazon S3 for historical analysis.
2. Feature Engineering:
    - Identify risk factors such as:
        - Sudden large withdrawals.
        - Unusual transaction locations (e.g., someone transacts in India and USA within 5 minutes).
        - Multiple transactions within a short timeframe.
        - Transactions from new devices/IPs.
3. Modeling:
    - Use Machine Learning (ML) & Rule-Based Detection:
        - Random Forest/XGBoost for pattern-based fraud detection.
        - LSTM (Long Short-Term Memory) for anomaly detection in sequential transaction data.
    - Leverage real-time streaming analytics in Apache Flink or Spark Structured Streaming.

4. Action Layer:
    - If a transaction is flagged as high-risk, trigger:
        - Multi-factor authentication (MFA) for the user.
        - Alerts to fraud detection teams.
        - Temporary account freeze for highly suspicious activities.
5. Logging & Monitoring:
    - Use ELK Stack (Elasticsearch, Logstash, Kibana) or Datadog for monitoring fraud detection effectiveness.

---

10. What factors would you consider when choosing between Amazon S3, Google Cloud Storage, and Azure Blob Storage for storing transaction data?

Factor	Amazon S3	Google Cloud Storage	Azure Blob Storage
Security & Compliance	Supports encryption, IAM, bucket policies	Strong security, fine-grained IAM controls	Best for enterprise compliance (PCI-DSS, GDPR, HIPAA)
Integration	Best for AWS ecosystem (Redshift, Athena)	Best for GCP ecosystem (BigQuery, AI services)	Best for Azure stack (Synapse, ADLS, Power BI)
Cost & Performance	Flexible pricing (S3 Standard, Glacier)	Lower retrieval latency, fast reads/writes	Optimized for Microsoft workloads
Data Lifecycle Management	S3 lifecycle policies (auto-tiering)	Similar lifecycle policies	Built-in Data Lake Storage Gen2

Decision:

- Use Azure Blob Storage if compliance & enterprise security are critical.
- Use Google Cloud Storage for low-latency analytics.
- Use AWS S3 for flexible, cost-effective storage.

---

11. How would you ensure PCI-DSS compliance while storing and processing transaction data?

PCI-DSS (Payment Card Industry Data Security Standard) compliance ensures the secure handling of card transactions.

Key Practices:

1. Encryption:
    - Use AES-256 encryption for sensitive data (card details, CVV).
    - Encrypt data at rest (using TDE, SSE-S3, or SSE-Azure) and in transit (TLS 1.2+).
2. Access Control & Authentication:
    - Implement Role-Based Access Control (RBAC) using IAM policies.
    - Use multi-factor authentication (MFA) for admin access.
    - Rotate API keys and credentials periodically.
3. Data Masking & Tokenization:
    - Replace card numbers with tokens to prevent exposure.
    - Only store the last 4 digits of a card number for reference.
4. Logging & Monitoring:
    - Use SIEM tools (Splunk, Azure Sentinel, AWS GuardDuty) to detect unauthorized access.
    - Maintain audit logs and set up real-time alerts for suspicious activity.
5. Network Security:
    - Deploy firewalls and intrusion detection systems (IDS/IPS).
    - Isolate payment data in a dedicated Virtual Private Cloud (VPC).

---

12. What are some best practices for optimizing the performance of data processing jobs?

1. Partitioning & Bucketing:
    - Partition large datasets in Spark, Snowflake, or BigQuery based on time-based fields (e.g., transaction_date).
    - Use bucketing to improve join performance in Spark.
2. Indexing:
    - Create indexes on frequently queried columns (e.g., account_id, transaction_id).

   - Use clustering keys in Snowflake for faster retrieval.
3. Caching Intermediate Results:
   - Persist frequently used datasets in Spark using .cache() or .persist().
   - Materialized views for pre-aggregated results.
4. Parallel Processing:
   - Use distributed computing (Spark, Databricks, Flink) instead of single-node processing.
   - Optimize shuffle operations by reducing data skew.
5. Monitoring & Auto-Scaling:
   - Use Datadog, Prometheus, or AWS CloudWatch to monitor job execution time.
   - Enable auto-scaling in cloud-based clusters (Databricks, EMR, GCP Dataflow).

---

13. How would you handle data ingestion from multiple sources with different schemas?

1. Schema Evolution Handling:
   - Use Delta Lake, Avro, or Parquet to handle schema changes without breaking pipelines.
2. Schema Mapping & Standardization:
   - Convert different source formats (CSV, JSON, XML) into a unified schema using ETL tools like Azure Data Factory, Apache NiFi.
   - Use Data Contracts to ensure consistency between producers & consumers.
3. Data Validation & Cleaning:
   - Use Great Expectations or PyDeequ for data quality checks.
   - Apply schema validation rules using Spark or Pandas.
4. Metadata Management:
   - Store metadata in a central repository like Hive Metastore, AWS Glue, or Databricks Unity Catalog.

---

14. Talk about a time when you had trouble communicating with stakeholders. How were you able to overcome it?

Scenario:
In a past project, a business stakeholder required daily transaction reports in Power BI. However, their requirements kept changing, causing delays and confusion.

Challenges:

- Frequent scope changes made previous work obsolete.
- Lack of clear documentation of expectations.
- Misalignment between technical feasibility and business needs.

Solution:

1. Set up weekly sync meetings to clarify priorities.
2. Created a version-controlled Power BI dashboard, allowing stakeholders to test features before finalizing.
3. Used Agile methodology – delivered small, incremental changes rather than waiting for a full revamp.

Outcome:

- Improved stakeholder engagement and feedback loop.
- Reduced rework time by 40%.

---

15. Given the rise of contactless payments, how can Mastercard ensure security without compromising user experience?

1. Tokenization:
   - Replace card details with one-time-use tokens.
   - Reduce risk by storing tokens instead of raw card data.
2. Biometric Authentication:
   - Use fingerprint, facial recognition, or voice authentication to validate high-value transactions.
3. AI-Powered Fraud Detection:
   - Use machine learning models to analyze transaction behavior.
   - Identify patterns of fraud in real-time.
4. Risk-Based Authentication (RBA):
   - If a transaction seems suspicious, prompt for extra authentication.

    - Allow low-risk transactions to proceed without friction.
5. End-to-End Encryption (E2EE):
    - Ensure contactless transactions are encrypted at every step.
    - Protect data from man-in-the-middle (MITM) attacks.