Question 1: How do you scale this solution to 10x or 100x, taking into account different potential latencies?

Scaling an ETL pipeline for significantly higher data volumes requires **parallelism, optimization, and resilience**. Below are possible ways of scaling that I have used so far:

1. Database Optimization

- **Indexing:** Ensure key query columns (*customer_id, transaction_date*, etc.) are indexed for faster retrieval.

Partitioning: Partition the *TransactionsOptimized* table by *transaction_date* to speed up queries.

```
ALTER TABLE Customer.TransactionsOptimized
PARTITION BY RANGE (YEAR(transaction_date)) (
   PARTITION p2023 VALUES LESS THAN (2024),
   PARTITION p2024 VALUES LESS THAN (2025)
);
```

- **Sharding:** If handling billions of rows, distribute data across multiple MySQL instances.

2. Efficient Data Loading

Batch Inserts: Instead of row-by-row inserts, insert in bulk, for example using the python script below:

```
insert_query = """
INSERT INTO Customer.TransactionsOptimized (...)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
"""
cursor.executemany(insert_query, data_chunk)
```

- **Parallel Processing:** Use *ThreadPoolExecutor* or *Dask* to parallelize data ingestion.
- **Asynchronous Processing:** Move data from MySQL to a **data warehouse (Amazon Redshift, Snowflake)** for faster analytics.

3. Handling Latencies & Data Delays

- **Message Queues (Kafka, AWS SQS):** If data arrives in bursts, a message queue can smooth ingestion.
- **Auto-scaling compute resources:** In **AWS**, services like **Glue & Lambda** can scale automatically.

Question 2: How to handle changes within the source data, assuming some transactions would be modified or backdated?

Handling **late-arriving or changed transactions** requires **Change Data Capture (CDC)** techniques:

1. Track Changes in the Source

- **Use Last Modified Timestamp:** If the source system has a *last_updated* column, use it to detect changes.
- **Implement a Change Log Table:** Keep historical records in an *audit_log* table.

2. Handle Backdated Transactions

- **Merge Historical & New Data:** Use ***ON DUPLICATE KEY UPDATE*** in MySQL to update existing records.

Insert New & Update Existing Transactions:

```
INSERT INTO Customer.TransactionsOptimized (...)

VALUES (...)

ON DUPLICATE KEY UPDATE

  transaction_amount = VALUES(transaction_amount),

  transaction_category = VALUES(transaction_category);
```

- **Event-Driven Updates:** Use Kafka or AWS DMS CDC to track and apply changes dynamically.

Question 3: Assuming the company is getting this data for the first time and has never had it before in a usable format for analytics or automations - what would be the single most important thing to build from it or take from it to deliver value?

1. Customer Lifetime Value (CLV) Prediction

- Helps **target high-value customers** and **optimize pricing & promotions**.
- Requires: **Transaction history + Spending trends + Recency/Frequency/Monetary (RFM) analysis**

2. Real-time Financial Reporting

- Daily & Monthly **Revenue Dashboards** for Finance Teams.
- Requires: **Aggregations & Query Optimization** in MySQL or a Data Warehouse.