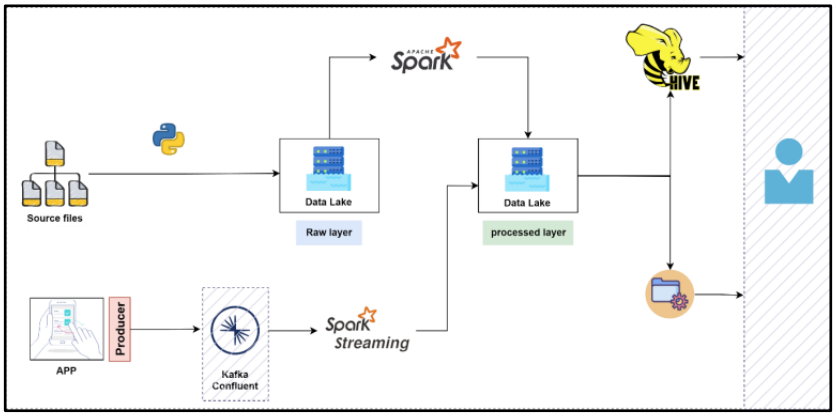
Q company Case Study

1. Introduction

- **Company Overview:** Q company is a retail business with both physical branches and an e-commerce platform. They face challenges integrating data from various sources, including hourly updates on branches, sales agents, and sales transactions, as well as app logs streamed to a Kafka cluster. This disparate data landscape makes it difficult to derive timely insights for marketing and B2B initiatives.
- **Project Goals:**
    - Efficiently ingest raw sales data from multiple sources.
    - Transform and load data into a structured data warehouse (Hive).
    - Enable analysis to support marketing and B2B teams.
    - Process app logs from a Kafka cluster using a Spark streaming job and store the processed data on HDFS.

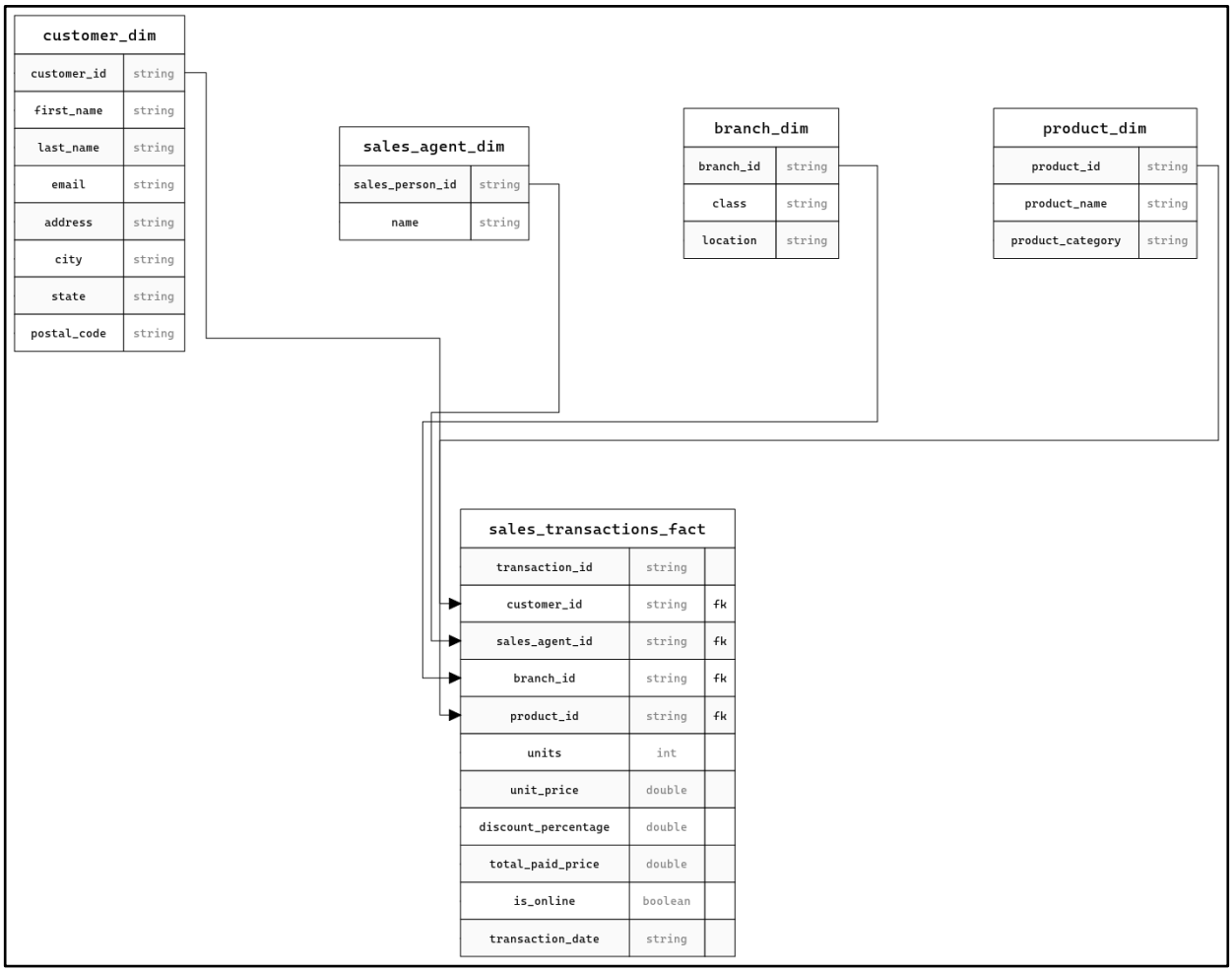
2. Architecture Diagram

- **Local File System:** Where raw CSV files are initially placed.
- **HDFS (Hadoop Distributed File System):** The scalable storage layer for raw and processed data.
- **Spark:** The processing engine for data transformation and loading.
- **Hive:** The data warehouse for structured storage and querying.
- **Cron:** The scheduler for automating batch jobs.
- **Kafka :** The streaming platform used to handle real-time data feeds



3. Data Model (Star Schema)

- **Explanation:** the star schema design:
    - **Fact Table:** `sales_transactions_fact` (stores transactional data).
    - **Dimension Tables:** `customer_dim`, `product_dim`, `sales_agent_dim`, `branch_dim` (store descriptive attributes).



- **Table Details (Kafka)**

kafka	
eventType	string
customerId	string
productid	string
timestamp	string
category	string
source	string
quantity	int
totalAmount	double
paymentMethod	string
recommendedProductId	string
algorithm	string
event_date	string
event_hour	int

4. Data Ingestion and Processing

- `ingestCode.py`:
    - **Functionality:** This script is designed to move files from a local file system to HDFS (Hadoop Distributed File System). It does this in a controlled manner, processing one directory at a time and maintaining state to avoid re-processing.
    - **Key Features:**
        - **State Management:** Uses a JSON file to track which directories have been processed, ensuring fault tolerance and idempotence.
        - **Directory Structure:** Creates a date-based directory structure in HDFS to organise the raw data.
        - **Error Handling:** Includes basic error handling to log issues during the HDFS put operation.
- `processing.py`:
    - **Functionality:** This Spark script is the core of the data transformation process. It reads raw CSV files from HDFS, applies transformations, and writes the processed data into Hive tables.
    - **Key Features:**
        - **Data Cleaning:** Handles missing values, standardises email formats, and trims whitespace.
        - **Data Transformation:** Calculates derived metrics like `total_paid_price` and `discount_percentage`.
        - **Dimensional Modelling:** Creates dimension tables (customer, product, sales agent, branch) and a fact table (sales transactions) following a star schema design.
        - **Incremental Loading:** Checks for existing records in dimension tables to avoid duplicates.
        - **Hive Integration:** Leverages Hive's partitioning for efficient querying by `transaction_date`.
- `Daily_dump_hive.py`:
    - **Functionality:** This script generates a daily CSV report of sales by agent and product.
    - **Key Features:**
        - **Date Parameterization:** Allows for querying data for a specific date.
        - **Hive Query:** Executes a SQL query against the Hive tables to aggregate sales data.
        - **CSV Output:** Writes the results to a CSV file in the local file system.
- `producer.py`:
    - **Functionality:** This script generates a streaming data send them to kafka cluster.
- `streaming.ipynb`:
    - **Functionality:** This notebook take the data from kafka and save them on hdfs partitioned by event_date , event_hour.
- `compaction.py`:

    - **Functionality:** This script compact the data on hdfs of the last hour in just one file.
    - **Key Features:**
        - **Small Files Problem:** Fixes the problem of small file problem on hdfs.
- `hiveRepairPartition.py`**:**
    - **Functionality:** This script repair the hive table to make it see the new partitions.
    - **Key Features:**
        - **Partitions:** fixed the hive problem to see the new partitions because new partitions is added not from hive so we need to update them.

5. Business Queries and Insights

- **Marketing Team:**
    - **Most Selling Products:**

product_id	product_name	total_units_sold
22	Coffee Maker	379
25	Washing Machine	365
9	Boots	337
7	Dress	336
27	Iron	322
17	Blouse	319
28	Hair Dryer	308
1	Laptop	306
24	Blender	305
6	Jeans	302
29	Hair Straightener	296
19	Sandals	296
20	Heels	293
5	T-Shirt	283
11	TV	277

    - **Most Redeemed Offers:**

discount_percentage	number_of_sales
0.0	747
20.0	168
15.0	162
10.0	151
5.0	138
25.0	134

- **Lowest Performing Cities (Online Sales):**

city	total_online_sales
Port Charlotte	151.962
Colchester	151.962
Scituate	299.99
Lawrence	299.99
San Luis Obispo	323.4805
Concord	339.83
Mexico Beach	569.905
Duxbury	573.809
Dummerston	573.809
Richmond	925.8910000000001
Alameda	925.8910000000001
Plymouth	1441.853
Palm Bay	1517.3159999999998
Saugus	1582.323

- **Most Redeemed Offers per Product:**

product_id	product_name	discount_percentage	purchase_count
4	Headphones	10.0	20
11	TV	20.0	19
24	Blender	10.0	19
26	Vacuum Cleaner	15.0	19
2	Smartphone	20.0	18
19	Sandals	20.0	17
14	Camera	15.0	17
22	Coffee Maker	15.0	16
29	Hair Straightener	20.0	15
8	Sneakers	20.0	15
29	Hair Straightener	5.0	15
5	T-Shirt	5.0	14
25	Washing Machine	10.0	14
6	Jeans	5.0	14
29	Hair Straightener	15.0	14
18	Boots	15.0	14
20	Heels	15.0	13
6	Jeans	15.0	13
13	Printer	20.0	13
22	Coffee Maker	10.0	13

only showing top 20 rows

- **B2B Team:**
    - **Daily Sales Dump:**

sales_agent_name	product_name	total_sold_units
Michael Johnson	Blouse	51
Jane Smith	Sandals	50
Emma Taylor	Headphones	46
Christopher Miller	Sandals	45
Christopher Miller	Coffee Maker	43
John Doe	Sandals	41
Sophia Moore	Boots	41
Christopher Miller	T-Shirt	38
Olivia Davis	Vacuum Cleaner	35
Jane Smith	Printer	34
Emily Brown	T-Shirt	34
Christopher Miller	Hoodie	33
David Wilson	Blender	32
David Wilson	Dress	32
Sophia Moore	Electric Kettle	31
Emma Taylor	Washing Machine	31
David Wilson	Sneakers	30
Emma Taylor	Vacuum Cleaner	30

- **Streaming Queries:**
    - Total Sales Amount by Payment Method

paymentMethod	total_sales
Debit Card	58476.46000000004
Credit Card	48251.62
PayPal	47463.930000000015

    - Most Popular Categories by Number of Events

category	total_events
null	1944
Home & Kitchen	205
Books	182
Electronics	166
Clothing	134

- Revenue by Customer

customerId	total_revenue
40568	798.8
24184	499.3
36839	497.62
32730	497.26
66714	496.19
65429	494.8
41784	493.9
38514	493.74
32457	491.46
45241	491.37

- Hourly Sales Analysis

event_hour	total_sales
3	2350.4
4	2468.7
5	1005.14
6	3056.5499999999997
11	11588.650000000001
12	4466.29
13	1436.93
15	16607.089999999997
16	76627.44

6. Automation (Cron)

- `My_crontab.txt`**:Batch**
  - **Functionality:** This configuration file automates the execution of the Python scripts using cron.
  - **Key Features:**
    - **Hourly Ingestion:** Runs `ingestCode.py` every hour.
    - **Hourly Processing:** Runs `processing.py` every hour at the 20th minute.
    - **Daily Reporting:** Runs `daily_dump_hive.py` daily at 1:00 AM.
    - **Logging:** Redirects output and errors to log files for monitoring.
- `Crontab.txt`**:Streaming**

- **Functionality:** This configuration file automates the execution of the Python scripts using cron.
- **Key Features:**
  - **Hourly Compaction:** Runs `compaction.py` every hour.
  - **Hourly hiveRepairPartition:** Runs `hiveRepairPartition.py` every hour at 1 minute

7. Future Enhancements

- **Schema Evolution**

  The Challenge: Raw data schemas can change over time (e.g., new columns added, column names modified, data types altered). If your processing code isn't prepared, these changes can cause failures.

  Strategies:

  1. **Schema Inference (Current Approach):** The `processing.py` script uses `inferSchema=True` when reading CSV files. This is a simple approach where Spark tries to determine the schema based on the data. It's convenient but can be unreliable if the data is inconsistent.
  2. **Explicit Schema Definition:** A more robust approach is to define the expected schema explicitly using `StructType` (as done for `sales_transactions_schema` in `processing.py`). This provides stricter validation but requires you to update the schema in your code whenever the raw data schema changes.
  3. **Schema Registry:** For more complex scenarios, consider using a schema registry (e.g., Confluent Schema Registry). This allows you to store and manage schemas centrally, providing versioning and compatibility checks.

  Example (Explicit Schema):

```
branches_schema = StructType([
    StructField("branch_id", StringType(), True),
    StructField("class", StringType(), True),
    StructField("location", StringType(), True)
])
branches = spark.read.csv(file_path, header=True,
schema=branches_schema)
```

- **Data Quality Checks**

  The Challenge: Ensuring data quality is crucial for accurate analysis. The current code has some basic checks (e.g., handling nulls in `offer` columns), but more comprehensive validation is needed.

Strategies:

1. **Column-Level Checks:**
    - **Data Type Validation:** Ensure columns have the expected data types (e.g., integers for IDs, dates for transaction dates).
    - **Range Constraints:** Check if numeric values fall within valid ranges (e.g., discount percentages between 0 and 100).
    - **Pattern Matching:** Validate string formats (e.g., email addresses, postal codes).
2. **Business Rule Checks:**
    - **Referential Integrity:** Verify that foreign keys in the fact table match values in the dimension tables.
    - **Uniqueness:** Check for duplicate records, especially in dimension tables.
    - **Consistency:** Ensure data consistency across tables (e.g., total sales in the fact table should match aggregated sales in other tables).
3. **Custom Validation Functions:** Write Spark UDFs (User Defined Functions) to implement complex validation logic.

Example (Column-Level Check):

```
# Check if unit_price is positive
sales_transactions = sales_transactions.filter(col("unit_price") > 0)
```

- **Logging and Monitoring**

  The Need: Robust logging helps you troubleshoot issues, track job progress, and understand data patterns. Monitoring allows you to proactively detect failures and performance bottlenecks.

  Enhancements:

  1. **Structured Logging:** Use a structured logging format (e.g., JSON) to make logs easier to parse and analyze.
  2. **Log Levels:** Use different log levels (e.g., DEBUG, INFO, WARN, ERROR) to categorize log messages.
  3. **Metrics:** Log key metrics (e.g., number of records processed, processing time) to track performance.

8. Conclusion

- **Summary:** The batch processing pipeline successfully addresses Q company's data integration and analysis challenges by consolidating raw sales data from disparate sources into a structured Hive data warehouse. This enables the marketing and B2B teams to gain valuable insights into customer behaviour, product performance, and sales trends.
- **Impact:**
    - **Marketing Team:** The ability to identify top-selling products, underperforming regions, and popular offers empowers the marketing team to refine strategies, optimise campaigns, and allocate resources more effectively. Understanding customer preferences and offer redemption patterns can lead to more targeted and successful marketing initiatives.
    - **B2B Team:** The daily sales report provides the B2B team with a granular view of sales performance by agent and product. This information can be used to identify high-performing sales agents, track product sales, and make informed decisions about inventory management and distribution.