เอกสารประกอบโครงงานการออกแบบและจัดทำ Data Pipeline

รายวิชา 06026239 DATA PIPELINE ARCHITECTURE

กลุ่ม : นึกไม่ออก

ชุดข้อมูล : DataCo SMART SUPPLY CHAIN FOR BIG DATA ANALYSIS

---

1. บริบทของข้อมูล

1.1 บริบททางธุรกิจหรือการบริการ

- ชุดข้อมูลนี้มาจาก "DataCo Smart Supply Chain" ซึ่งเป็นข้อมูลจำลองของธุรกิจค้าปลีก (Retail) หรือ E-commerce ที่มีการดำเนินงานทั่วโลก (Global)
- ตลาดหลักครอบคลุม USCA, LATAM, Europe, Pacific Asia และ Africa
- กลุ่มลูกค้าสามารถแบ่งได้ 3 กลุ่ม ได้แก่ Consumer (ผู้บริโภคทั่วไป), Corporate (องค์กร) และ Home Office (ออฟฟิศขนาดเล็ก)
- กระบวนการหลัก คือ การจัดการโลจิสติกส์และซัพพลายเชน (Supply Chain) ตั้งแต่การจัดการคำสั่งซื้อ (Order Management), การชำระเงิน, การตรวจจับการทุจริต (Fraud Detection) ไปจนถึงการจัดส่งสินค้า
- ข้อมูลดิบแต่ละแถวคือรายละเอียดการสั่งซื้อสินค้าแต่ละชิ้น (order_item) ประกอบด้วย ข้อมูลคำสั่งซื้อ (order), ข้อมูลสินค้า (product), ข้อมูลลูกค้า (customer) และข้อมูลการขนส่ง (shipping)

1.2 เป้าหมายของการออกแบบและจัดทำ Data Pipeline

- สร้าง Data Pipeline ที่ ingest ⟶ store ⟶ process ข้อมูล
- สร้าง Single Source of Truth (SSOT) : พัฒนา Data Warehouse (Core Schema) ที่เป็น 3NF เพื่อให้ทุกแผนกใช้ข้อมูลจากแหล่งเดียวกัน
- สร้าง Data Marts สำหรับการวิเคราะห์เชิงธุรกิจ ในรูปแบบ Star Schema เพื่อความรวมเร็วในการประมวลผล

1.3 ข้อตกลงระดับการบริการ (Service Level Agreement - SLA)

- Data Marts ต้องพร้อมใช้งานภายใน 8:00 น. ของทุกเช้า
- รัน Data Pipeline ทุก 1 ชั่วโมง
- หากแล้วเสร็จ หรือมีข้อขัดข้องให้แจ้งเตือนทีม Data/BI ทางช่องทางที่กำหนด (Email)

1.4 KPI ที่ต้องการวัดจากชุดข้อมูล

- ด้านโลจิสติกส์ (Logistics) :
    - **On-Time Delivery Rate**: อัตราการส่งสินค้าทันภายในเวลาจัดส่งที่คาดการณ์
    - **Late Delivery Rate**: อัตราการส่งสินค้าไม่ทันภายในเวลาจัดส่งที่คาดการณ์
    - **Shipping Days Variance**: ความคลาดเคลื่อนของเวลาจัดส่งที่คาดการณ์กับเวลาจัดส่งจริง
- ด้านการขายและกำไร (Sales & Profit) :
    - **Total Sales** : ผลรวมของยอดขาย
    - **Total Profit** : ผลรวมของกำไร
    - **Profit Margin Rate** : อัตราส่วนการทำกำไรเทียบกับยอดขาย
    - **Average Order Value** : ค่าเฉลี่ยราคาขายของออเดอร์
- ด้านลูกค้า (Customer) :
    - **Sales by Customer Segment** : ยอดขายแบ่งตามกลุ่มลูกค้า
- ด้านความเสี่ยง (Risk) :
    - Fraud Rate : อัตราของออเดอร์ที่สงสัยว่ามีการฉ้อโกง
    - Value at Risk : ผลรวมยอดเงินของออเดอร์ที่สงสัยว่ามีการฉ้อโกง

2. รายละเอียดการออกแบบ Data Pipeline

2.1 เครื่องมือที่เลือกใช้ พร้อมเหตุผลประกอบ

- **Apache Airflow** : ใช้เป็นเครื่องมือหลักในการกำหนดลำดับการทำงาน (Orchestrator), จัดการตารางเวลา (Scheduling) และติดตามผลการรัน Pipeline ทั้งหมด

  เหตุผลที่เลือกใช้ :

  - สามารถกำหนดลำดับการทำงาน, Dependencies และการตั้งเวลาทั้งหมดได้ด้วยโค้ด Python
  - มี Operators หลากหลายที่รองรับการทำงานหลายรูปแบบ ได้แก่ PythonOperator เพื่อรันฟังก์ชัน Python, BashOperator เพื่อรันคำสั่งใน Shell และ EmailOperator เพื่อส่งการแจ้งเตือน
  - มี Web UI ที่ช่วยให้เราติดตามสถานะของ Task ดู Logs และสั่งรัน Pipeline ใหม่ได้ง่าย

- **Python :** ใช้ในการอ่านไฟล์ CSV ขนาดใหญ่เข้าสู่หน่วยความจำด้วย Pandas และโหลดเข้าสู่ฐานข้อมูล PostgreSQL ด้วย SQLAlchemy ในขั้นตอน Ingestion และใช้สำหรับการแก้ไข Data Type ข้อมูลเบื้องต้น

  เหตุผลที่เลือกใช้ :

  - เป็นภาษาพื้นฐานของการใช้เครื่องมือส่วนใหญ่
  - การ Ingestion ด้วย Python ทำได้รวดเร็วกว่า Airbyte ทำให้เหมาะสมสำหรับการทำตาม SLA ที่ “ต้องรัน Data Pipeline ทุก 1 ชั่วโมง” มากกว่า

- **PostgreSQL** : ใช้เป็นฐานข้อมูลหลักในการจัดเก็บข้อมูล ทั้งในส่วน Staging, Core (3NF) และ Data Marts (Star Schema)

  เหตุผลที่เลือกใช้ :

  - เป็นระบบฐานข้อมูล (RDBMS) แบบ Open-Source ที่ได้รับความน่าเชื่อถือสูง และรองรับ SQL ที่ซับซ้อนได้ดี

- รองรับการจัดเก็บข้อมูลทั้งในรูปแบบ 3NF (Core Database) ที่ต้องการความถูกต้องของข้อมูลสูง และรูปแบบ Star Schema (Data Marts) ที่เน้นความเร็วในการ Query เพื่อทำ Analytics
- ทำงานร่วมกับเครื่องมืออื่นได้อย่างราบรื่น

- **dbt (Data Build Tool)** : เป็นเครื่องมือหลักในการทำ Data Transformation

  เหตุผลที่เลือกใช้ :

  - dbt สามารถสร้าง Data Pipeline ด้วยการเขียน SQL ซึ่งสะดวกและเข้าใจง่ายสำหรับ Database แบบ Relational
  - สามารถจัดการลำดับการสร้างตารางได้สะดวก
  - สามารถทดสอบคุณภาพข้อมูล (Data Quality) ได้ในตัวโดยการใช้ dbt test

- **Great Expectations (GX)** : ใช้ในการตรวจสอบและรับประกันคุณภาพข้อมูล (Data Quality) ในจุดสำคัญของ Pipeline โดยเฉพาะหลังจากการ Ingest ข้อมูลดิบ (Staging) และหลังจากสร้าง Data Marts เพื่อให้มั่นใจว่าข้อมูลถูกต้อง

  เหตุผลที่เลือกใช้ :

  - สามารถสร้าง Expectations ซึ่งเป็นการทดสอบคุณภาพข้อมูลที่ซับซ้อนกว่า dbt test
  - สามารถใช้งานได้ในทุกจุดของ Data Pipeline ทำให้การทดสอบคุณภาพข้อมูลทำได้สะดวก

- **Mailtrap** : ใช้สำหรับทดสอบการส่ง Email แจ้งเตือนสถานะของ Pipeline เมื่อทำงานสำเร็จ ตามที่ระบุใน SLA

  เหตุผลที่เลือกใช้ :

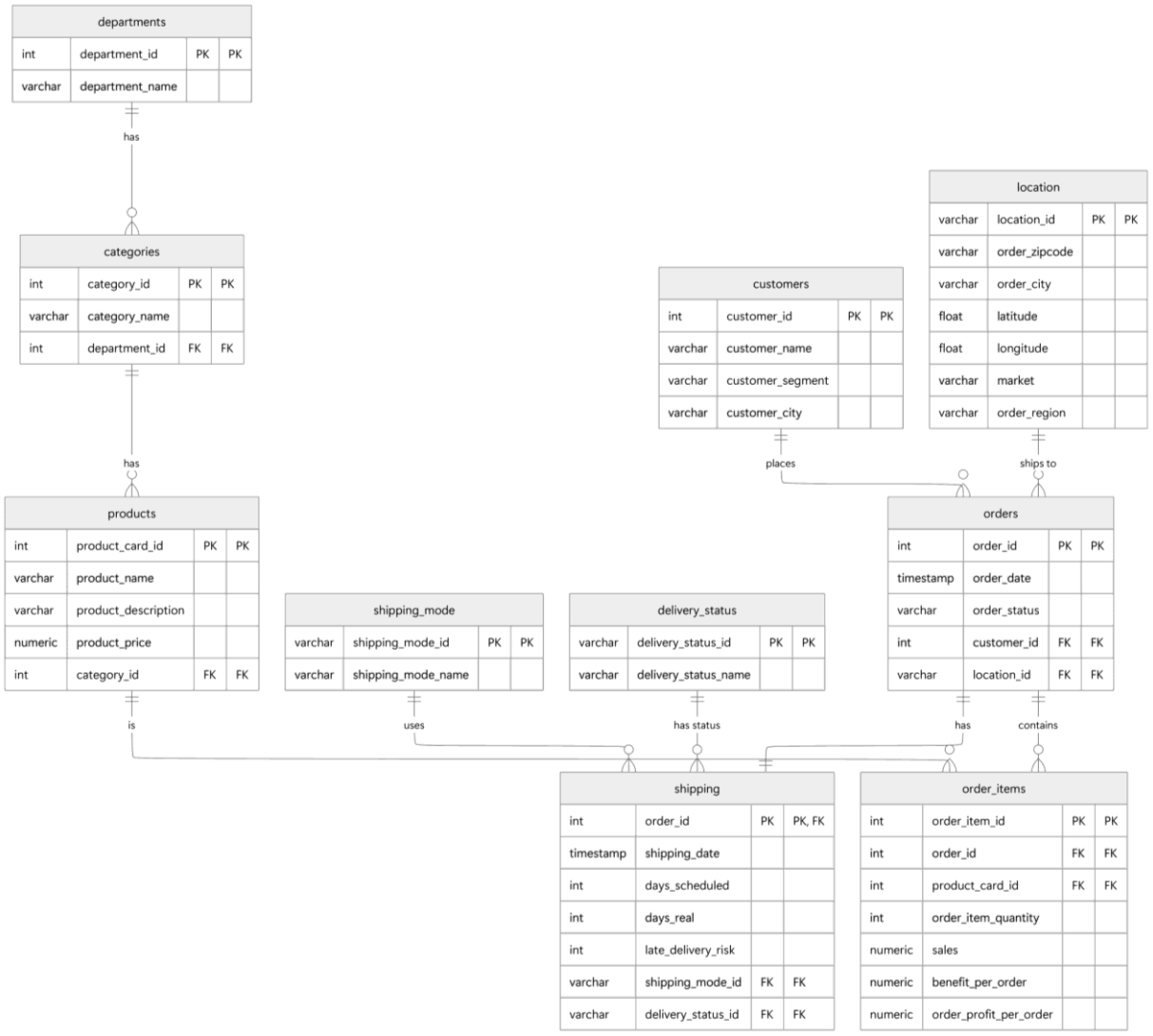
  - สามารถสร้าง Sandbox ที่ใช้ทดสอบได้ง่ายและรวดเร็ว

2.2 ข้อกำหนดพื้นฐานของ DAG

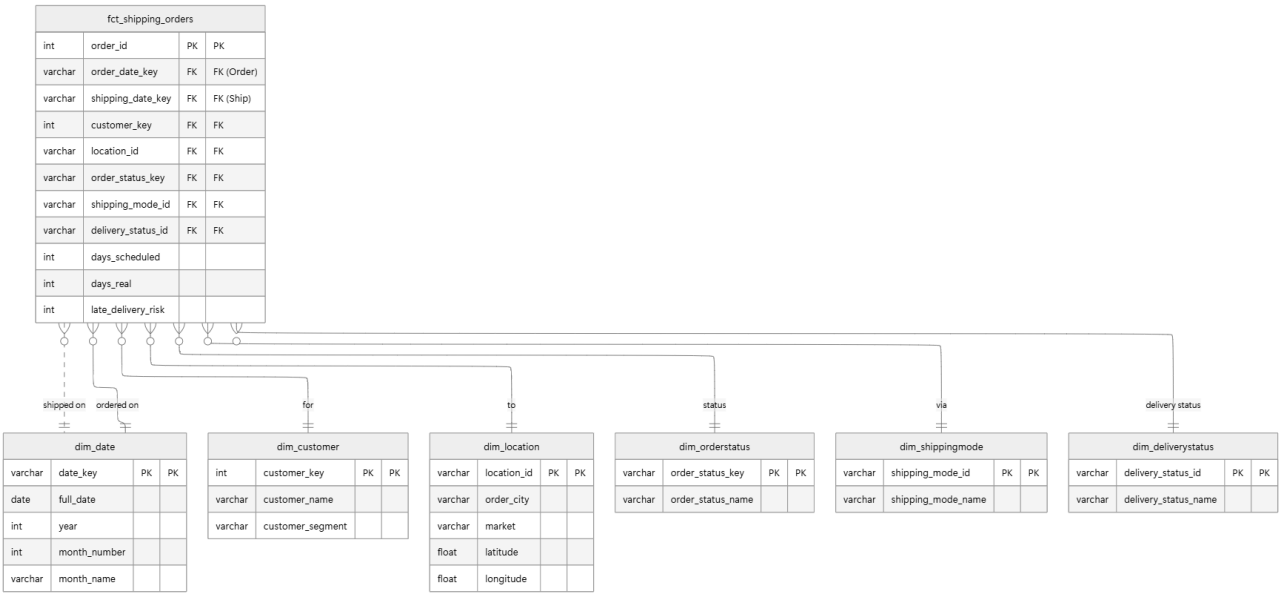
- ทำงานทุก ๆ 1 ชั่วโมง (schedule_interval="@hourly")
- retries = 2 ครั้ง, retry_delay = 1 นาที
- หาก fail ⟶ mark DAG failed และส่งแจ้งเตือนไปที่ Email

2.3 โครงสร้างฐานข้อมูล

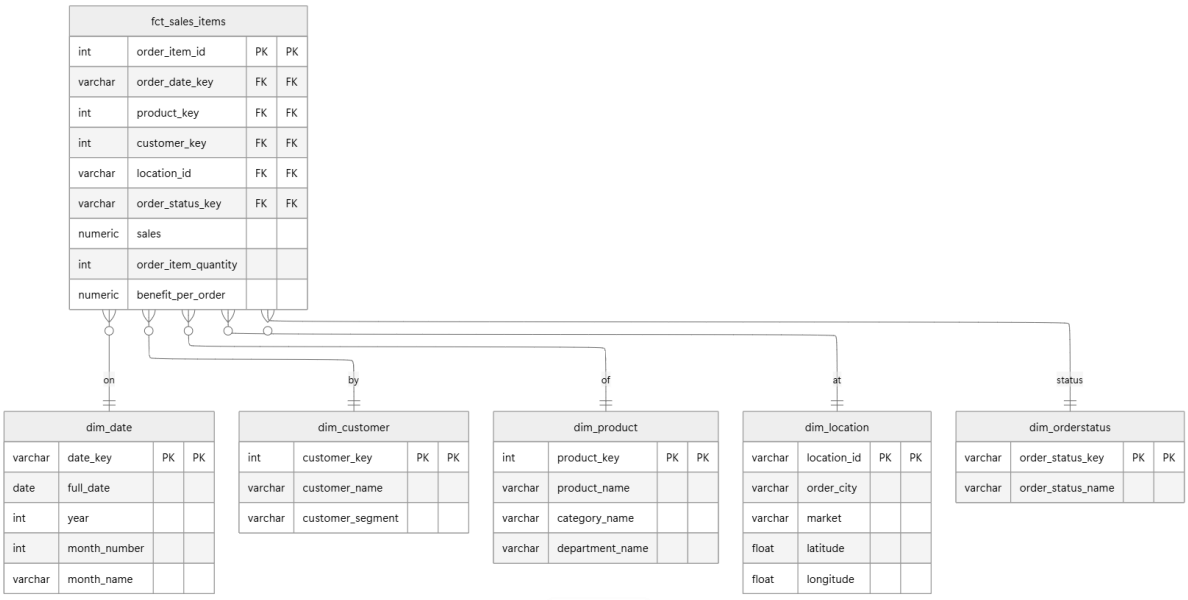
- Core Schema (3NF)



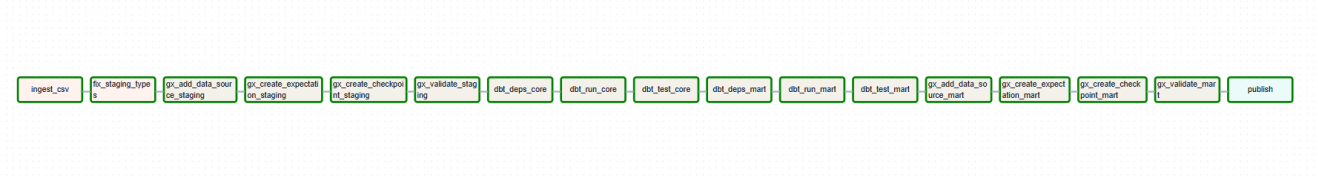
- **Logistics Mart (Star Schema):**
    - Fact Table : fct_shipping_orders
    - Grain (ความละเอียด) : 1 แถว คือ 1 คำสั่งซื้อ (order_id)
    - Description : ใช้ตอบ KPI ด้านโลจิสติกส์ (Logistics)



- Sales Mart (Star Schema):
    - Fact Table : fct_sales_items
    - Grain (ความละเอียด) : 1 แถว คือ 1 รายการสินค้า (order_item_id)
    - Description : ใช้ตอบ KPI ด้านการขายและกำไร (Sales & Profit), ด้านลูกค้า (Customer) และด้านความเสี่ยง (Risk)



3. ขั้นตอนการดำเนินการเพื่อสร้างและใช้งาน Data Pipeline



ขั้นตอนการทำงานของ Pipeline นี้ถูกกำหนดโดยลำดับการทำงาน (Dependencies) ใน Airflow DAG โดยมีทั้งหมด 17 Task ที่ทำงานเป็นลำดับขั้นดังนี้

1. **Ingestion (ingest_csv)** : ใช้ PythonOperator เรียกใช้ฟังก์ชัน ingest_csv_to_postgres เพื่ออ่านไฟล์ DataCoSupplyChainDataset.csv ด้วย Pandas และโหลดข้อมูลดิบทั้งหมดลงใน Schema “stagging” ในตาราง “datacosupplychaindataset_tmp” (staging.datacosupplychaindataset_tmp) ใน PostgreSQL

```
ingest_csv = PythonOperator(
task_id="ingest_csv",
python_callable=ingest_csv_to_postgres,
)
```

```
def ingest_csv_to_postgres():
    csv_path = "/opt/airflow/dags/data/DataCoSupplyChainDataset.csv"
    df = pd.read_csv(csv_path)

    engine = create_engine("postgresql+psycopg2://dpuser:dppass@dp_postgres:5432/dp_course")

    df.to_sql(
    "datacosupplychaindataset_tmp",
    con=engine,
    schema="staging",
    if_exists="replace",
    index=False
    )
```

2. **Fix Data Types (fix_staging_types)** : ใช้ PythonOperator เรียกใช้ฟังก์ชัน fix_data_types_in_staging เพื่อรันคำสั่ง SQL ALTER TABLE ใน PostgreSQL ทำการแก้ไขประเภทข้อมูล (Data Types) ในตาราง staging.datacosupplychaindataset_tmp ให้ถูกต้อง เช่น แปลงข้อความวันที่เป็น TIMESTAMP, แปลงตัวเลขเป็น NUMERIC

```
fix_staging_types = PythonOperator(
        task_id="fix_staging_types",
        python_callable=fix_data_types_in_staging,
    )
```

```python
def fix_data_types_in_staging():
    """
    Connects to Postgres and runs ALTER TABLE
    to fix data types in the staging table.
    """
    engine = create_engine("postgresql+psycopg2://dpuser:dppass@dp_postgres:5432/dp_course")


    sql_command = text("""
    ALTER TABLE staging.datacosupplychaindataset_tmp
    ALTER COLUMN "order date (DateOrders)" TYPE TIMESTAMP USING "order date (DateOrders)"::timestamp,
    ALTER COLUMN "shipping date (DateOrders)" TYPE TIMESTAMP USING "shipping date (DateOrders)"::timestamp,
    ALTER COLUMN "Sales" TYPE NUMERIC(10, 5) USING "Sales"::numeric,
    ALTER COLUMN "Product Price" TYPE NUMERIC(10, 5) USING "Product Price"::numeric,
    ALTER COLUMN "Latitude" TYPE DOUBLE PRECISION USING "Latitude"::double precision,
    ALTER COLUMN "Longitude" TYPE DOUBLE PRECISION USING "Longitude"::double precision,
    ALTER COLUMN "Order Item Discount Rate" TYPE NUMERIC(3, 2) USING "Order Item Discount Rate"::numeric,
    ALTER COLUMN "Order Item Total" TYPE NUMERIC(10, 5) USING "Order Item Total"::numeric,
    ALTER COLUMN "Order Item Discount" TYPE NUMERIC(10, 5) USING "Order Item Discount"::numeric,
    ALTER COLUMN "Sales per customer" TYPE NUMERIC(10, 5) USING "Sales per customer"::numeric,
    ALTER COLUMN "Benefit per order" TYPE NUMERIC(10, 5) USING "Benefit per order"::numeric,
    ALTER COLUMN "Order Profit Per Order" TYPE NUMERIC(10, 5) USING "Order Profit Per Order"::numeric,
    ALTER COLUMN "Order Item Profit Ratio" TYPE NUMERIC(5, 4) USING "Order Item Profit Ratio"::numeric,
    ALTER COLUMN "Order Item Product Price" TYPE NUMERIC(10, 5) USING "Order Item Product Price"::numeric,
    ALTER COLUMN "Order Item Quantity" TYPE INTEGER USING "Order Item Quantity"::integer,
    ALTER COLUMN "Late_delivery_risk" TYPE SMALLINT USING "Late_delivery_risk"::smallint,
    ALTER COLUMN "Product Status" TYPE SMALLINT USING "Product Status"::smallint,
    ALTER COLUMN "Days for shipment (scheduled)" TYPE INTEGER USING "Days for shipment (scheduled)"::integer,
    ALTER COLUMN "Days for shipping (real)" TYPE INTEGER USING "Days for shipping (real)"::integer;
    """)

    try:
        with engine.begin() as conn:
            conn.execute(sql_command)

        print("Successfully fixed data types in staging.")
    except Exception as e:
        print(f"Error fixing data types: {e}")
        raise
```

3. **GX - Add Staging Data Source (gx_add_data_source_staging)** : ใช้ BashOperator เพื่อรันสคริปต์ Python (gx_scripts/add_datasource_staging.py) เพื่อเพิ่มฐานข้อมูล PostgreSQL ที่มีตารางใน Staging เป็น Data Source ให้ Great Expectations (GX) รู้จัก

```python
gx_add_data_source_staging = BashOperator(
    task_id="gx_add_data_source_staging",
    bash_command=(
    "cd /workspace && "
    "python gx_scripts/add_datasource_staging.py"
    )
)
```

4. **GX - Create Staging Expectations (gx_create_expectation_staging)** : ใช้ BashOperator เพื่อรันสคริปต์ Python gx_scripts/create_supply_chain_expectations.py ทำการสร้างชุดกฎการตรวจสอบข้อมูล (Expectation Suite) สำหรับตารางใน Staging

```python
gx_create_expectation_staging = BashOperator(
    task_id="gx_create_expectation_staging",
    bash_command=(
    "cd /workspace && "
    "python gx_scripts/create_supply_chain_expectations.py"
    )
)
```

โดยจะทำการตรวจสอบข้อมูลดังนี้:

- Order Item Id ต้องไม่ซ้ำกัน (Unique) และห้ามเป็นค่าว่าง (Not Null)
- Order Id ห้ามเป็นค่าว่าง
- Customer Id ห้ามเป็นค่าว่าง
- Order Status ต้องเป็นหนึ่งใน 9 สถานะที่กำหนด (เช่น 'COMPLETE', 'PENDING_PAYMENT', 'SUSPECTED_FRAUD' ฯลฯ)
- Type ต้องเป็น 'DEBIT', 'PAYMENT', 'TRANSFER', หรือ 'CASH'
- Shipping Mode ต้องเป็นหนึ่งใน 4 ประเภท ('First Class', 'Same Day', 'Second Class', 'Standard Class')
- Delivery Status ต้องเป็นหนึ่งใน 4 สถานะ ('Advance shipping', 'Late delivery', 'Shipping canceled', 'Shipping on time')
- Market ต้องเป็นหนึ่งใน 5 ตลาดที่กำหนด ('Pacific Asia', 'USCA', 'Africa', 'Europe', 'LATAM')
- Customer Segment ต้องเป็น 'Consumer', 'Corporate', หรือ 'Home Office'
- Late_delivery_risk ต้องเป็น 0 หรือ 1
- Product Status ต้องเป็น 0 หรือ 1
- Financials Product Price , Sales , Order Item Product Price , Order Item Total , Order Item Discount , และ Sales per customer ทั้งหมดต้องมีค่าไม่ติดลบ (>= 0)
- Order Item Quantity ต้องมีค่าอย่างน้อย 1
- Order Item Discount Rate ต้องอยู่ระหว่าง 0 ถึง 1
- Latitude ต้องอยู่ระหว่าง -90 ถึง 90 และ Longitude ต้องอยู่ระหว่าง -180 ถึง 180
- Days for shipment (scheduled) และ Days for shipping (real) ต้องไม่ติดลบ (>= 0)
- order date (DateOrders) ต้องไม่เป็นวันที่ในอนาคต (ต้องน้อยกว่าหรือเท่ากับวันปัจจุบัน)
- shipping date (DateOrders) (วันที่ส่งของ) ต้องมาทีหลังหรือตรงกับ order date (DateOrders) (วันที่สั่งของ)
- ตรวจสอบว่าคอลัมน์สำคัญอื่นๆ อีกหลายคอลัมน์ (เช่น Customer City, Category Id, Department Id, Order Region, Customer Email, Benefit per order ฯลฯ) ห้ามเป็นค่าว่าง

5. **GX - Create Staging Checkpoint (gx_create_checkpoint_staging)** : ใช้ BashOperator เพื่อรันสคริปต์ Python gx_scripts/create_supply_chain_checkpoint.py ทำการสร้าง Checkpoint ชื่อ stg_supply_chain_checkpoint ใน GX ซึ่งเป็นการจับคู่ตาราง Staging เข้ากับ Expectation Suite ที่สร้างไว้ในขั้นตอนก่อนหน้า

```
gx_create_checkpoint_staging = BashOperator(
    task_id="gx_create_checkpoint_staging",
    bash_command=(
    "cd /workspace && "
    "python gx_scripts/create_supply_chain_checkpoint.py"
    )
)
```

6. **GX - Validate Staging Data (gx_validate_staging)** : ใช้ BashOperator เพื่อรันคำสั่ง great_expectations checkpoint run stg_supply_chain_checkpoint เรียกใช้ Checkpoint เพื่อทำการตรวจสอบข้อมูลจริงในตาราง Staging เทียบกับ Expectation Suite ที่ตั้งไว้ เพื่อวัดว่าข้อมูลดิบผ่านการตรวจสอบคุณภาพหรือไม่

```
gx_validate_staging = BashOperator(
    task_id="gx_validate_staging",
    bash_command=(
    "cd /workspace/gx && "
    "great_expectations checkpoint run stg_supply_chain_checkpoint"
    )
)
```

7. **dbt - Install Core Dependencies (dbt_deps_core)** : ใช้ BashOperator เพื่อรันคำสั่ง dbt deps ในโปรเจค supply_chain_core เพื่อทำการติดตั้ง dbt packages ในไฟล์ packages.yml ที่จำเป็นก่อนที่จะรันโมเดล

```
dbt_deps_core = BashOperator(
    task_id="dbt_deps_core",
    bash_command="cd /usr/app/supply_chain_core && dbt deps",
)
```

8. **dbt - Run Core Models (dbt_run_core)** : ใช้ BashOperator เพื่อรันคำสั่ง dbt run --select core ทำการแปลงข้อมูล (Transform) จากตารางใน Staging ไปเป็นตารางใน Core Database ในรูปแบบ 3NF โดยสร้างตารางทั้งหมดไว้ใน Schema ชื่อ core_core

```
dbt_run_core = BashOperator(
    task_id="dbt_run_core",
    bash_command="cd /usr/app/supply_chain_core && dbt run --select core",
)
```

9. **dbt - Test Core Models (dbt_test_core)** : ใช้ BashOperator เพื่อรันคำสั่ง dbt test --select core ทำการทดสอบคุณภาพข้อมูล (เช่น not_null, unique) ของตารางใน Core Database ที่เพิ่งสร้างเสร็จ

```
dbt_test_core = BashOperator(
    task_id="dbt_test_core",
    bash_command="cd /usr/app/supply_chain_core && dbt test --select core",
)
```

10. **dbt - Install Mart Dependencies (dbt_deps_mart)** : ใช้ BashOperator เพื่อรันคำสั่ง dbt deps ในโปรเจค supply_chain_mart เพื่อทำการติดตั้ง dbt packages ในไฟล์ packages.yml ที่จำเป็นก่อนที่จะรันโมเดล

```
dbt_deps_mart = BashOperator(
    task_id="dbt_deps_mart",
    bash_command="cd /usr/app/supply_chain_mart && dbt deps",
)
```

11. **dbt - Run Mart Models (dbt_run_mart)** : ใช้ BashOperator เพื่อรัน dbt run --select mart ทำการแปลงข้อมูล (Transform) จาก Core Database (3NF) ไปเป็นตารางใน Data Marts (ในรูปแบบ Star Schema) สำหรับการวิเคราะห์ตาม KPI โดยสร้างตารางทั้งหมดไว้ใน Schema ชื่อ mart_mart

```
dbt_run_mart = BashOperator(
    task_id="dbt_run_mart",
    bash_command="cd /usr/app/supply_chain_mart && dbt run --select mart",
)
```

12. **dbt - Test Mart Models (dbt_test_mart)** : ใช้ BashOperator เพื่อรัน dbt test --select mart ทำการทดสอบคุณภาพข้อมูลของตารางใน Data Marts ที่เพิ่งสร้างเสร็จ

```
dbt_test_mart = BashOperator(
    task_id="dbt_test_mart",
    bash_command="cd /usr/app/supply_chain_mart && dbt test --select mart",
)
```

13. **GX - Add Mart Data Source (gx_add_data_source_mart)** : ใช้ BashOperator เพื่อรันสคริปต์ Python (gx_scripts/add_datasource_mart.py) เพื่อเพิ่มฐานข้อมูล PostgreSQL ที่มีตารางใน Data Marts เป็น Data Source ให้ Great Expectations (GX) รู้จัก

```
gx_add_data_source_mart = BashOperator(
    task_id="gx_add_data_source_mart",
    bash_command=(
    "cd /workspace && "
    "python gx_scripts/add_datasource_mart.py"
    )
)
```

14. **GX - Create Mart Expectations (gx_create_expectation_mart)** : ใช้ BashOperator เพื่อรันสคริปต์ Python gx_scripts/create_analytics_expectations.py ทำการสร้างชุดกฎการตรวจสอบข้อมูล (Expectation Suite) สำหรับตารางใน Data Marts

```
gx_create_expectation_mart = BashOperator(
    task_id="gx_create_expectation_mart",
    bash_command=(
    "cd /workspace && "
    "python gx_scripts/create_analytics_expectations.py"
    )
)
```

15. **GX - Create Mart Checkpoint (gx_create_checkpoint_mart)** : ใช้ BashOperator เพื่อรันสคริปต์ Python gx_scripts/create_analytics_checkpoint.py ทำการสร้าง Checkpoint ชื่อ analytics_checkpoint สำหรับตารางใน Data Marts

```
gx_create_checkpoint_mart = BashOperator(
    task_id="gx_create_checkpoint_mart",
    bash_command=(
    "cd /workspace && "
    "python gx_scripts/create_analytics_checkpoint.py"
    )
)
```

16. **GX - Validate Mart Data (gx_validate_mart)** : ใช้ BashOperator เพื่อรัน great_expectations checkpoint run analytics_checkpoint ทำการตรวจสอบข้อมูลจริงในตาราง Data Marts เทียบกับ Expectation Suite ที่ตั้งไว้ เพื่อวัดว่าข้อมูลผ่านการตรวจสอบคุณภาพหรือไม่

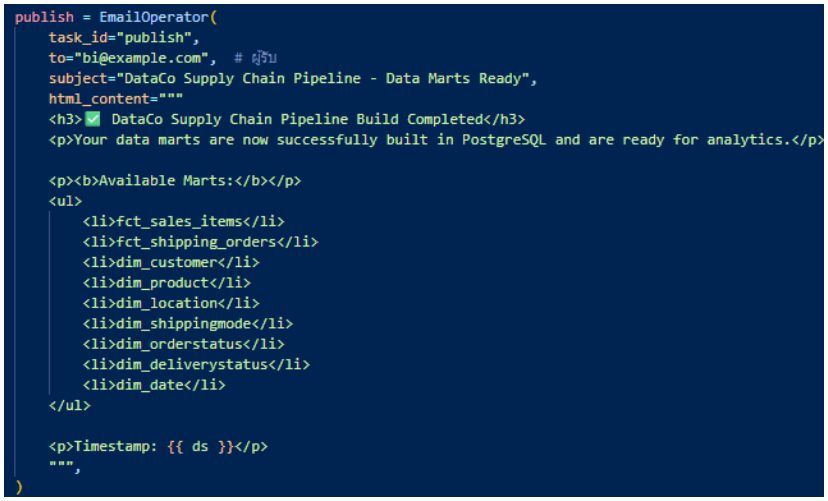
```
gx_validate_mart = BashOperator(
    task_id="gx_validate_mart",
    bash_command=(
    "cd /workspace/gx && "
    "great_expectations checkpoint run analytics_checkpoint"
    )
)
```

โดยจะทำการตรวจสอบข้อมูลดังนี้:

- customer_key ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- customer_fname (ชื่อ) ห้ามเป็นค่าว่าง
- customer_segment ต้องเป็นค่าที่ถูกต้อง ('Consumer', 'Corporate', 'Home Office')
- product_key ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- product_name, category_name, department_name ห้ามเป็นค่าว่าง
- location_id ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- order_city ห้ามเป็นค่าว่าง
- market ต้องเป็นค่าที่ถูกต้อง ('Pacific Asia', 'USCA', 'Africa', 'Europe', 'LATAM')
- latitude และ longitude ต้องมีประเภทข้อมูลเป็น DOUBLE_PRECISION
- date_key ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- full_date ต้องมีประเภทข้อมูลเป็น DATE
- year ต้องอยู่ในช่วงปีที่สมเหตุสมผล (ตั้งแต่ปี 2000 ถึงปีปัจจุบัน)
- month_number ต้องอยู่ระหว่าง 1-12
- month_name ต้องเป็นชื่อเดือนที่ถูกต้อง (เช่น 'January', 'February' ฯลฯ)
- order_status_key ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- order_status_name ต้องเป็นค่าสถานะที่ถูกต้อง (เช่น 'COMPLETE', 'SUSPECTED_FRAUD' ฯลฯ)
- shipping_mode_id ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- shipping_mode_name ต้องเป็นค่าที่ถูกต้อง ('First Class', 'Same Day', 'Second Class', 'Standard Class')

- delivery_status_id ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- delivery_status_name ต้องเป็นค่าที่ถูกต้อง ('Advance shipping', 'Late delivery', 'Shipping canceled', 'Shipping on time')
- order_item_id (Key หลัก) ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- Foreign Keys (เช่น order_date_key, customer_key, product_key) ใน fct_sales_items ทั้งหมดห้ามเป็นค่าว่าง เพื่อรับประกัน Referential Integrity
- sales (ยอดขาย) ต้องไม่ติดลบ (>= 0)
- order_item_quantity (จำนวน) ต้องมีค่าอย่างน้อย 1
- benefit_per_order (กำไร) ห้ามเป็นค่าว่าง
- order_id (Key หลัก) ต้องไม่ซ้ำและไม่เป็นค่าว่าง
- Foreign Keys (เช่น order_date_key, customer_key, shipping_date_key) ใน fct_shipping_orders ทั้งหมดห้ามเป็นค่าว่าง
- late_delivery_risk ต้องเป็น 0 หรือ 1
- days_for_shipment_scheduled และ days_for_shipping_real ต้องไม่ติดลบ (>= 0)

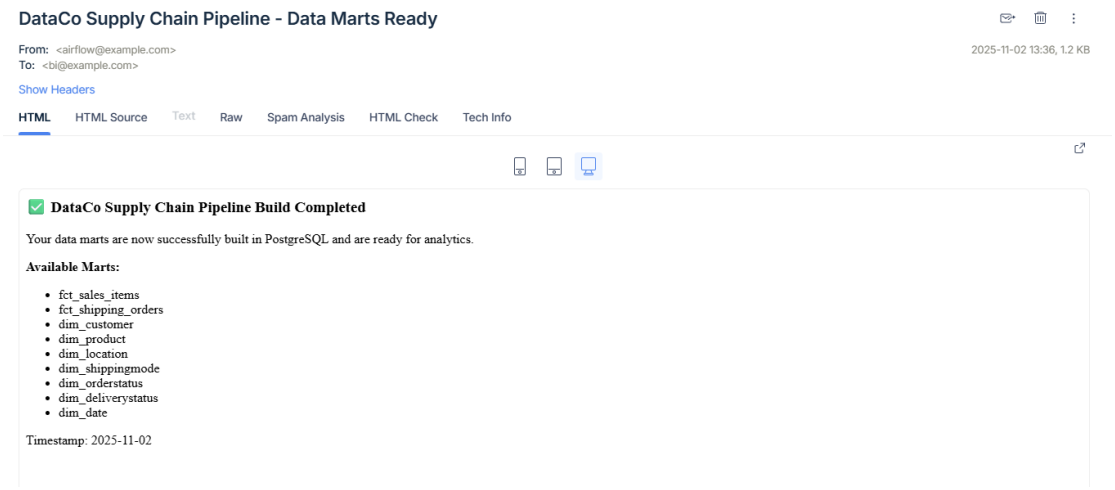
17. **Publish (publish)** : ใช้ EmailOperator เพื่อส่ง Email แจ้งเตือนว่า Pipeline ทำงานเสร็จสมบูรณ์ และตาราง Data Marts พร้อมใช้งานสำหรับทำ Analytics แล้ว โดยให้ส่งไปที่ Mailtrap เป็นการทดสอบการใช้งาน



```
publish = EmailOperator(
    task_id="publish",
    to="bi@example.com",  # ผู้รับ
    subject="DataCo Supply Chain Pipeline - Data Marts Ready",
    html_content="""
    <h3>✅ DataCo Supply Chain Pipeline Build Completed</h3>
    <p>Your data marts are now successfully built in PostgreSQL and are ready for analytics.</p>

    <p><b>Available Marts:</b></p>
    <ul>
        <li>fct_sales_items</li>
        <li>fct_shipping_orders</li>
        <li>dim_customer</li>
        <li>dim_product</li>
        <li>dim_location</li>
        <li>dim_shippingmode</li>
        <li>dim_orderstatus</li>
        <li>dim_deliverystatus</li>
        <li>dim_date</li>
    </ul>

    <p>Timestamp: {{ ds }}</p>
    """,
)
```



4. คำตอบของ KPI ที่ตั้งไว้

- ด้านโลจิสติกส์ (Logistics) :
    - **On-Time Delivery Rate**: อัตราการส่งสินค้าทันภายในเวลาจัดส่งที่คาดการณ์
        - มีค่าอยู่ที่ **17.83%** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    ROUND(
        100.0 * SUM(CASE WHEN ds.delivery_status_name = 'Shipping on time' THEN 1 ELSE 0 END)
        / COUNT(*),
        2
    ) AS on_time_delivery_rate

FROM mart_mart.fct_shipping_orders AS fs
LEFT JOIN mart_mart.dim_deliverystatus AS ds
    ON fs.delivery_status_id = ds.delivery_status_id;
```

	on_time_delivery_rate numeric
1	17.83

- **Late Delivery Rate**: อัตราการส่งสินค้าไม่ทันภายในเวลาจัดส่งที่คาดการณ์
  - มีค่าอยู่ที่ **54.82%** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    ROUND(
        100.0 * SUM(CASE WHEN ds.delivery_status_name = 'Late delivery' THEN 1 ELSE 0 END)
        / COUNT(*),
        2
    ) AS late_delivery_rate

FROM mart_mart.fct_shipping_orders AS fs
LEFT JOIN mart_mart.dim_deliverystatus AS ds
    ON fs.delivery_status_id = ds.delivery_status_id;
```

	late_delivery_rate numeric
1	54.82

- **Shipping Days Variance**: ความคลาดเคลื่อนของเวลาจัดส่งที่คาดการณ์กับเวลาจัดส่งจริง
  - มีค่าอยู่ที่ **±0.57** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    ROUND(
        AVG(fs.days_for_shipping_real - fs.days_for_shipment_scheduled),
        2
    ) AS avg_shipping_days_variance

FROM mart_mart.fct_shipping_orders AS fs
LEFT JOIN mart_mart.dim_deliverystatus AS ds
    ON fs.delivery_status_id = ds.delivery_status_id;
```

	avg_shipping_days_variance numeric
1	0.57

- ด้านการขายและกำไร (Sales & Profit) :
    - **Total Sales** : ผลรวมของยอดขาย
        - มีค่าอยู่ที่ **36,784,735.13$** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    TO_CHAR(SUM(sales), 'FM999,999,999,999.00') AS total_sales
FROM mart_mart.fct_sales_items;
```

	total_sales text
1	36,784,735.13

    - **Total Profit** : ผลรวมของกำไร
        - มีค่าอยู่ที่ **3,966,902.98$** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    TO_CHAR(SUM(benefit_per_order), 'FM999,999,999,999.00') AS total_profit
FROM mart_mart.fct_sales_items;
```

	total_profit text
1	3,966,902.98

    - **Profit Margin Rate** : อัตราส่วนการทำกำไรเทียบกับยอดขาย
        - มีค่าอยู่ที่ **10.78%** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    ROUND(
        100.0 * SUM(benefit_per_order) / NULLIF(SUM(sales), 0),
        2
    ) AS profit_margin_rate
FROM mart_mart.fct_sales_items;
```

	profit_margin_rate numeric
1	10.78

- Average Order Value : ค่าเฉลี่ยราคาขายของออเดอร์
  - มีค่าอยู่ที่ **559.45$/ออเดอร์** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    TO_CHAR(
        SUM(sales) / COUNT(DISTINCT order_id),
        'FM999,999,999.00'
    ) AS average_order_value
FROM mart_mart.fct_sales_items;
```

	average_order_value text
1	559.45

- ด้านลูกค้า (Customer) :
  - Sales by Customer Segment : ยอดขายแบ่งตามกลุ่มลูกค้า
    - กลุ่ม Home Office มียอดขายรวมอยู่ที่ **6,520,538.03$** มีทั้งหมด **11,777 ออเดอร์** คิดเป็นค่าเฉลี่ย **553.67$/ออเดอร์**
    - กลุ่ม Consumer มียอดขายรวมอยู่ที่ **19,095,790.21$** มีทั้งหมด **34,119 ออเดอร์** คิดเป็นค่าเฉลี่ย **559.68$/ออเดอร์**
    - กลุ่ม Corporate มียอดขายรวมอยู่ที่ **11,168,406.88$** มีทั้งหมด **19,856 ออเดอร์** คิดเป็นค่าเฉลี่ย **562.47$/ออเดอร์**
    - คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    dc.customer_segment,
    --Total Sales (ยอดขายรวม)
    TO_CHAR(SUM(fsi.sales), 'FM999,999,999,999.00') AS total_sales,

    --Total Orders (จำนวนออเดอร์)
    TO_CHAR(COUNT(DISTINCT fsi.order_id), 'FM999,999,999,999') AS total_orders,

    --Average Order Value (AOV)
    ROUND(SUM(fsi.sales) / COUNT(DISTINCT fsi.order_id), 2) AS avg_order_value
FROM mart_mart.fct_sales_items AS fsi
LEFT JOIN mart_mart.dim_customer AS dc
    ON fsi.customer_key = dc.customer_key
GROUP BY dc.customer_segment
ORDER BY total_sales DESC;
```

	customer_segment text	total_sales text	total_orders text	avg_order_value numeric
1	Home Office	6,520,538.03	11,777	553.67
2	Consumer	19,095,790.21	34,119	559.68
3	Corporate	11,168,406.88	19,856	562.47

- ด้านความเสี่ยง (Risk) :
    - **Fraud Rate** : อัตราของออเดอร์ที่สงสัยว่ามีการฉ้อโกง
        - มีค่าอยู่ที่ **2.26%** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    -- Total Orders (จำนวนออเดอร์ทั้งหมด)
    COUNT(DISTINCT fsi.order_id) AS total_orders,

    -- Fraud Orders (จำนวนออเดอร์ที่สงสัยเป็น Fraud)
    COUNT(DISTINCT CASE
        WHEN dos.order_status_name = 'SUSPECTED_FRAUD' THEN fsi.order_id
    END) AS fraud_orders,

    -- Fraud Rate (%)
    TO_CHAR(
        100.0 * COUNT(DISTINCT CASE
            WHEN dos.order_status_name = 'SUSPECTED_FRAUD' THEN fsi.order_id
        END) / NULLIF(COUNT(DISTINCT fsi.order_id), 0),
        'FM999,990.00'
    ) AS fraud_rate
FROM mart_mart.fct_sales_items AS fsi
LEFT JOIN mart_mart.dim_orderstatus AS dos
    ON fsi.order_status_key = dos.order_status_key;
```

	total_orders bigint	fraud_orders bigint	fraud_rate text
1	65752	1488	2.26

    - **Value at Risk** : ผลรวมยอดเงินของออเดอร์ที่สงสัยว่ามีการฉ้อโกง
        - มีค่าอยู่ที่ **825,934.96$** คำนวณจากคำสั่ง SQL ดังนี้

```sql
SELECT
    -- Value at Risk (ยอดเงินรวมของ Fraud Orders)
    TO_CHAR(
        SUM(CASE
            WHEN dos.order_status_name = 'SUSPECTED_FRAUD' THEN fsi.sales
            ELSE 0
        END),
        'FM999,999,999,999.00'
    ) AS value_at_risk
FROM mart_mart.fct_sales_items AS fsi
LEFT JOIN mart_mart.dim_orderstatus AS dos
    ON fsi.order_status_key = dos.order_status_key;
```

	value_at_risk text
1	825,934.96

สมาชิกในกลุ่ม

1. นาง สาวณัฐกมล ใจเมธา 65070068
2. นาย ณัฐภัทร ทองมีขวัญ 65070080
3. นาย ธิติวุฒิ มะลิวัลย์ 65070107
4. นางสาว พิชญา เดือนแรม 65070154
5. นางสาว ภัสสร จิวะเริงสวัสดิ์ 65070172
6. นาย รามิล ธรรมจินดา 65070198
7. นาย วิศรุต หงษ์ทอง 65070210
8. นาย สันติภาพ ธรรมรงค์ 65070234