Du RAW au schéma en étoile (DW) : génération, staging, dimensions, fait, agrégations

Programme Data Engineer

ECE

23 octobre 2025

Objectifs du Mini-TP et plan

Objectifs :

- Simuler des données **RAW** (clients, produits, commandes).
- Nettoyer/normaliser (*staging*).
- Construire des **dimensions** (date, produit, client) avec SK.
- Construire la **table de faits** `fact_sales` et réaliser des agrégations.
- Produire un **graphique** des ventes mensuelles.

Plan (12 diapositives) : Contexte
$\rightarrow$ *RAW* $\rightarrow$ *Staging* $\rightarrow$ *Dimensions* $\rightarrow$ *Fait* $\rightarrow$ *Persist/Analytics* $\rightarrow$ *Graphique* $\rightarrow$ *TP*.

Contexte et setup (imports, reproductibilité)

```
import numpy as np, pandas as pd
from datetime import datetime, timedelta
from pathlib import Path
import matplotlib.pyplot as plt

# Reproductibilite et sortie
rng = np.random.default_rng(42)
out_dir = Path("dw_example"); out_dir.mkdir(exist_ok=True)
```

Génération RAW : clients et produits

```python
# Clients (avec un peu de "salete")
countries = ["FR","DE","ES","IT","UK","US"]
raw_customers = pd.DataFrame({
  "customer_id": range(1,121),
  "name": [f"Cust {i}" for i in range(1,121)],
  "email": [f"user{i}@Example.com" for i in range(1,121)],
  "country": rng.choice(countries, 120)
})

# Produits
categories = ["Electronics","Home","Sports","Toys"]
raw_products = pd.DataFrame({
  "product_id": range(1,61),
  "product_name": [f"Item {i}" for i in range(1,61)],
  "category": rng.choice(categories, 60),
  "brand": rng.choice(["Acme","Globex","Umbrella","Initech","Soylent
     "], 60),
  "unit_price": np.round(rng.uniform(10,300,60),2)
```

Génération RAW : lignes de commande (order lines)

```
start, end = datetime(2025,1,1), datetime(2025,9,30)
N = 2500
raw_orders = pd.DataFrame({
  "order_id": 10000 + rng.integers(0,10000, N),
  "order_line_id": range(1, N+1),
  "order_ts": [start + timedelta(days=int(rng.integers(0,(end-start)
     .days+1))) for _ in range(N)],
  "customer_id": rng.integers(1,121, N),
  "product_id": rng.integers(1,61, N),
  "quantity": rng.integers(1,6, N)
})

# Persist RAW (optionnel)
raw_customers.to_csv(out_dir/"raw_customers.csv", index=False)
raw_products.to_csv(out_dir/"raw_products.csv", index=False)
raw_orders.to_csv(out_dir/"raw_orders.csv", index=False)
```

Staging : nettoyage et standardisation

```
stg_customers = raw_customers.assign(
  email = raw_customers["email"].str.strip().str.lower(),
  country = raw_customers["country"].str.upper()
)

stg_products = raw_products.assign(
  category = raw_products["category"].str.title(),
  brand    = raw_products["brand"].str.title()
)

stg_orders = raw_orders.assign(order_ts = pd.to_datetime(raw_orders[
   "order_ts"]))
```

Dimensions : dim_date, dim_product, dim_customer

```
# dim_date (SK sur calendrier des commandes)
dim_date = (pd.DataFrame({"date": pd.to_datetime(stg_orders["
    order_ts"].unique())})
            .sort_values("date").reset_index(drop=True))
dim_date["date_sk"] = range(1, len(dim_date)+1)
dim_date = dim_date.assign(
  day = dim_date["date"].dt.day,
  month = dim_date["date"].dt.month,
  quarter = dim_date["date"].dt.quarter,
  year = dim_date["date"].dt.year
)

# dim_product & dim_customer (SK)
dim_product  = stg_products.drop_duplicates("product_id").
    reset_index(drop=True)
dim_product["product_sk"] = range(1, len(dim_product)+1)

dim_customer = stg_customers.drop_duplicates("customer_id").
```

Table de faits : fact_sales (quantité, montant, coût, marge)

```
fact = (stg_orders
 .merge(dim_customer[["customer_id","customer_sk"]], on="customer_id")
 .merge(dim_product[["product_id","product_sk","unit_price","unit_cost"]], on="product_id")
 .merge(dim_date[["date","date_sk"]], left_on="order_ts", right_on="date"))

fact["amount"] = fact["quantity"] * fact["unit_price"]
fact["cost"]   = fact["quantity"] * fact["unit_cost"]
fact["margin"] = fact["amount"] - fact["cost"]

fact_sales = fact[[
  "order_id","order_line_id","date_sk","product_sk","customer_sk",
  "quantity","unit_price","amount","cost","margin"
]].sort_values(["order_id","order_line_id"])
```

Persistance du schéma en étoile (CSV) et aperçu

```
# Sauvegarde des dimensions et du fait
dim_date.to_csv(out_dir/"dim_date.csv", index=False)
dim_product[["product_sk","product_id","product_name","category","
    brand",
              "unit_price","unit_cost"]].to_csv(out_dir/"dim_product.
                 csv", index=False)

dim_customer[["customer_sk","customer_id","name","email","country"]]
  .to_csv(out_dir/"dim_customer.csv", index=False)

fact_sales.to_csv(out_dir/"fact_sales.csv", index=False)
```

Fichiers produits : `dim_date.csv`, `dim_product.csv`, `dim_customer.csv`, `fact_sales.csv`

Agrégations : ventes mensuelles, top 10 produits

```
# Ventes mensuelles
sales_by_month = (fact_sales
  .merge(dim_date[["date_sk","month","year"]], on="date_sk")
  .groupby(["year","month"], as_index=False)["amount"].sum()
  .sort_values(["year","month"]))

# Top 10 produits par CA
top_products = (fact_sales
  .merge(dim_product[["product_sk","product_name"]], on="product_sk")
  .groupby(["product_sk","product_name"], as_index=False)["amount"].sum()
  .sort_values("amount", ascending=False).head(10))
```

Graphique : chiffre d'affaires mensuel (matplotlib)

```
import matplotlib.pyplot as plt
x = range(len(sales_by_month))
labels = [f"{int(y)}-{int(m):02d}" for y,m in zip(sales_by_month["
    year"], sales_by_month["month"])]
plt.figure(); plt.plot(x, sales_by_month["amount"], marker="o")
plt.title("Chiffre d'affaires mensuel (exemple)")
plt.xlabel("Mois"); plt.ylabel("Montant")
plt.xticks(x[::max(1, len(x)//10)], [labels[i] for i in range(0,len(
    labels), max(1,len(x)//10))],
           rotation=45, ha="right")
plt.tight_layout()
plt.savefig(out_dir/"monthly_sales.png")
plt.show()
```

TP : exercices proposés

- Ajouter une dimension **Promotion** et lier au fait.
- Créer une table **fact_inventory** (stock journalier) et comparer CA vs stock.
- Mettre en place une **dimension SCD2** sur `dim_customer` (`valid_from/to`, `is_current`).
- Construire une vue matérialisée des **ventes hebdomadaires**.
- Ajouter des **règles DQ** (emails valides, valeurs non nulles) et un rapport d'anomalies.

Astuce : séparer RAW, STAGING, DIM, FACT en modules pour réutiliser le code.

Exemple flocon : dimension Géographie

Objectif : réduire la redondance des attributs de localisation (ville, région, pays) en normalisant la dimension `Magasin`. **Tables proposées** :

- `dim_magasin(sk_magasin, id_ville, nom_magasin, type_magasin, ...)`
- `dim_ville(id_ville, ville, id_region)`
- `dim_region(id_region, region, id_pays)`
- `dim_pays(id_pays, pays, code_iso2)`

Avantages : référentiels centralisés (pays/régions), mise à jour cohérente, moindre stockage.
Inconvénients : plus de jointures pour la BI, modèle moins intuitif qu'une étoile plate.

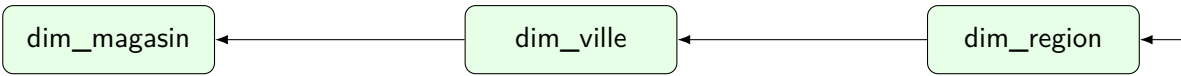
DDL (extrait) — flocon Géographie

```sql
CREATE TABLE dim_pays (
  id_pays INT PRIMARY KEY,
  pays VARCHAR(80) NOT NULL,
  code_iso2 CHAR(2) UNIQUE
);

CREATE TABLE dim_region (
  id_region INT PRIMARY KEY,
  region VARCHAR(80) NOT NULL,
  id_pays INT NOT NULL REFERENCES dim_pays(id_pays)
);

CREATE TABLE dim_ville (
  id_ville INT PRIMARY KEY,
  ville VARCHAR(80) NOT NULL,
  id_region INT NOT NULL REFERENCES dim_region(id_region)
);
```

Schéma en flocon — vue Géographie



Remarque : pour la consommation BI, on peut matérialiser une vue aplatie `dim_magasin_flat` (magasin+ville+region+pays) pour limiter les jointures.