

Лабораторна робота №4

Розробка ETL-сценарію для системи TechMarket

Виконав: Кіщук Ярослав

Зміст

| | | |
|-----------|--|----------|
| 1 | Вступ | 2 |
| 2 | Мета та завдання | 2 |
| 3 | Архітектура рішення | 2 |
| 3.1 | Загальна схема | 2 |
| 3.2 | Конфігурація та режими | 3 |
| 4 | Стадія Extract | 3 |
| 4.1 | OrdersExtractor | 3 |
| 4.2 | CatalogExtractor | 3 |
| 4.3 | PaymentsExtractor | 3 |
| 4.4 | Особливості екстракції | 3 |
| 5 | Стадія Transform | 4 |
| 5.1 | Очищення даних | 4 |
| 5.2 | Бізнес-трансформації | 4 |
| 5.3 | Приклад коду | 4 |
| 6 | Стадія Load | 5 |
| 6.1 | Завантаження вимірів | 5 |
| 6.2 | Факт-таблиця продажів | 5 |
| 6.3 | Підтримка dim_date та upsert | 5 |
| 7 | Оркестрація та автоматизація | 5 |
| 7.1 | ETLPipeline | 5 |
| 7.2 | CLI та приклади | 5 |
| 7.3 | Apache Airflow | 6 |
| 8 | Тестування та якість | 6 |
| 9 | Результати виконання | 6 |
| 9.1 | Продуктивність | 6 |
| 9.2 | Спостережність | 6 |
| 10 | Висновки | 7 |

1 Вступ

Метою четвертої лабораторної роботи є побудова повноцінного процесу Extract–Transform–Load (ETL) для перенесення операційних даних системи TechMarket у аналітичне сховище даних. Рішення має забезпечувати стабільну синхронізацію між чотирма MySQL джерелами (`orders`, `catalog`, `payments`, `auth`) та PostgreSQL DWH, підтримувати як повні, так і інкрементальні завантаження, надавати механізми автоматизації (CLI, Apache Airflow) та покриття тестами.

Основні артефакти, створені в межах роботи:

- Python-пакет `etl/` з модулями `config`, `extract`, `transform`, `load`, `pipeline`.
- Документація та приклади запуску (`etl/README.md`, `etl/examples.py`, `etl/run_etl.py`).
- Оркестровка через Apache Airflow (`airflow/dags/techmarket_etl_dag.py`).
- Набір автоматизованих тестів із покриттям понад 90% (`etl/tests/`).

2 Мета та завдання

- Завдання 1:** Спроекувати універсальну конфігурацію підключень до MySQL та PostgreSQL, що зчитується зі змінних середовища (`ETLConfig.from_env()`).
- Завдання 2:** Реалізувати окремі екстрактори даних для кожної OLTP-бази з підтримкою фільтрації за періодами та роботи з великими наборами (`chunked pandas.read_sql`).
- Завдання 3:** Розробити модуль трансформації, який виконує очищення, валідацію типів, бізнес-розрахунки (`revenue`, `margin`, `discount`) та агрегації.
- Завдання 4:** Створити надійний завантажувач у DWH з підтримкою *replace*, *append* та *upsert* для вимірів і фактів.
- Завдання 5:** Забезпечити оркестрацію повного та інкрементального ETL, включно з автоматичним генератором календаря `dim_date`.
- Завдання 6:** Побудувати сценарії автоматизації: CLI-утиліту, Airflow DAG'и, Docker-орієнтоване оточення.
- Завдання 7:** Підготувати LaTeX-звіт, що фокусується на етапах Extract, Transform, Load та на русі даних між доменами.

3 Архітектура рішення

3.1 Загальна схема

ETL складається з п'яти модулів і працює поверх операційних MySQL баз із шаблоном *Database-per-Service*. Витягнуті дані потрапляють у PostgreSQL схему з шістьма вимірами (`dim_date`, `dim_region`, `dim_category`, `dim_product`, `dim_customer`, `dim_employee`) та однією факт-таблицею `fact_sales`.

Табл. 1: Основні компоненти ETL

| Компонент | Функції |
|-------------------------------|--|
| <code>etl/config.py</code> | Описує <code>MySQLConfig</code> , <code>PostgreSQLConfig</code> , агрегує все у <code>ETLConfig</code> . Формує рядки підключення <code>mysql+pymysql://</code> і <code>postgresql+psycopg2://</code> . |
| <code>etl/extract.py</code> | Узагальнений <code>Extractor</code> з ледачою ініціалізацією <code>SQLAlchemy Engine</code> . Спеціалізовані <code>OrdersExtractor</code> , <code>CatalogExtractor</code> , <code>PaymentsExtractor</code> . |
| <code>etl/transform.py</code> | <code>DataCleaner</code> (дублікати, пропуски, валідація типів) та <code>DataTransformer</code> (метрики, агрегації). |
| <code>etl/load.py</code> | <code>Loader</code> для <code>dim/fact</code> таблиць, генератор <code>dim_date</code> , <code>truncate</code> , <code>upsert</code> . |
| <code>etl/pipeline.py</code> | <code>ETLPipeline</code> координує сім кроків повного завантаження та інкрементальний сценарій. |

3.2 Конфігурація та режими

Конфігурація зчитується з файлу `.env`. Ключові змінні: `ORDERS_DB_*`, `CATALOG_DB_*`, `PAYMENTS_DB_*`, `DWH_DB_*`, `ETL_INCREMENTAL`, `ETL_START_DATE`, `ETL_END_DATE`. Режими:

- **Full Load** — повне перезавантаження всіх вимірів і фактів, очищення `fact_sales`, генерація календаря за ± 365 днів.
- **Incremental Load** — фільтрація замовлень і платежів за датою, дозавантаження фактів без очищення, використовується як CLI-параметрами, так і Airflow DAG'ами.

4 Стадія Extract

4.1 OrdersExtractor

Модуль `OrdersExtractor.extract_orders()` виконує JOIN між `orders` та `order_items`, відбирає лише стани "paid", "shipped", "delivered", сортує за датою. Фільтри `start_date`, `end_date` додаються до WHERE, що забезпечує інкрементальність. Додаткові методи `extract_customers()`, `extract_employees()`, `extract_regions()` подають дані для відповідних вимірів.

4.2 CatalogExtractor

Постачає `products` з LEFT JOIN `categories`, збагачуючи товар ієрархією. Окремий метод `extract_categories()` впорядковує результати за батьківськими категоріями для коректного побудування дерева.

4.3 PaymentsExtractor

Повертає завершені платежі зі статусом `completed`, підтримує фільтрацію за датою платежу. Дані поєднуються з фактами продажів під час трансформації (зв'язок через `order_id`).

4.4 Особливості екстракції

- Ледаче створення `SQLAlchemy Engine` з `pool_pre_ping` та `pool_recycle`, щоб уникати збоїв тривалих з'єднань.
- Підтримка `chunksize` для обробки великих обсягів даних без перевантаження пам'яті.

- Централізоване логування (logging) із зазначенням бази-джерела.

5 Стадія Transform

5.1 Очищення даних

extttDataCleaner надає:

- **remove_duplicates** — видаляє дублікати за складеними ключами (наприклад, `order_id+order`).
- **handle_missing_values** — стратегії `drop`, `fill`, `forward_fill`; для товарів за замовчуванням підставляє "Опис відсутній".
- **validate_data_types** — приводить дати, числові значення та рядки до очікуваних типів із журналюванням помилок.

5.2 Бізнес-трансформації

extttDataTransformer.transform_orders() розраховує ключові метрики та готує дані до `fact_sales`:

- `revenue = unit_price * quantity`.
- `discount_amount = discount` (заповнюється нулями).
- `cost = revenue * 0.6` (припущення щодо собівартості).
- `margin = revenue - discount_amount - cost`.
- `date_key = int(order_date.strftime("%Y%m%d"))` для зв'язку з `dim_date`.

Інші методи трансформують клієнтів (нормалізовані `email`, `full_name`), продукти (заповнення відсутніх значень, приведення `price/stock_quantity`), а також агрегують продажі за періодами, товарами та менеджерами.

5.3 Приклад коду

Лістинг 1: Фрагмент трансформації замовлень

```
orders_df = transformer.cleaner.remove_duplicates(
    orders_df, subset=["order_id", "order_item_id"])
orders_df = transformer.cleaner.validate_data_types(orders_df, {
    "order_date": "datetime",
    "quantity": "int",
    "unit_price": "float"
})
orders_df["revenue"] = orders_df["unit_price"] * orders_df["quantity"]
orders_df["date_key"] = orders_df["order_date"].dt.strftime("%Y%m%d").
    astype(int)
```

6 Стадія Load

6.1 Завантаження вимірів

`Loader.load_dimension()` виконує `pandas.to_sql` із режимами `append/replace`, чанками по 1000 рядків. Для full-load всі `dim_*` таблиці перезбираються, щоб гарантувати узгоджені ключі.

6.2 Факт-таблиця продажів

Метод `load_fact_sales()` перевіряє наявність обов'язкових колонок (`order_id, product_id, revenue, ...`). Перед повним завантаженням викликається `truncate_table('fact_sales')`, тоді як інкрементальний режим просто `append` додає нові записи.

6.3 Підтримка `dim_date` та `upsert`

`load_dim_date()` створює календар на 730 днів довкола поточної дати. Для поступових оновлень вимірів застосовується `upsert_dimension()`, що розділяє `DataFrame` на нові та існуючі ключі, видаляє застарілі рядки та вставляє оновлені версії.

7 Оркестрація та автоматизація

7.1 ETLPipeline

Клас `ETLPipeline` інкапсулює послідовність із семи кроків (`_load_dim_date, _load_dim_region, ..., _load_fact_sales`). Повне завантаження виглядає як на листингу 2.

Лістинг 2: Запуск повного завантаження

```
from etl.config import ETLConfig
from etl.pipeline import ETLPipeline

config = ETLConfig.from_env()
pipeline = ETLPipeline(config)
results = pipeline.run_full_load()
```

Інкрементальний метод `run_incremental_load()` приймає `start_date/end_date` і викликає `_load_fact_sales_incremental()`. Після завершення `_cleanup()` закриває всі з'єднання.

7.2 CLI та приклади

Скрипт `etl/run_etl.py` надає інтерфейс:

- `python etl/run_etl.py -mode full`
- `python etl/run_etl.py -mode incremental -start-date 2024-01-01 -end-date 2024-01-31`
- Параметри `-env-file`, `-log-level`, валідація дат та запис логів одночасно у `stdout` і файл `etl.log`.

Файл `etl/examples.py` містить п'ять сценаріїв демонстрації (повне, інкрементальне, лише екстракція, трансформація, симуляція 7-денного ETL).

7.3 Apache Airflow

Файл `airflow/dags/techmarket_etl_dag.py` описує два DAG'и:

- **techmarket_etl_daily**: щоденне інкрементальне завантаження о 02:00 UTC. Завдання — перевірка OLTP, перевірка DWH, ETL, нотифікації про успіх/помилку. Параметри `DEFAULT_ARGS` задають `retries=2`, `retry_delay=5 хв`, `execution_timeout=2 год`.
- **techmarket_etl_weekly_full**: щонедільне повне завантаження о 03:00 UTC з більшим таймаутом (4 год). Використовує ті самі попередні перевірки.

8 Тестування та якість

Табл. 2: Покриття тестами

| Модуль | Ключові сценарії |
|--------------------------------|--|
| <code>test_config.py</code> | Перевірка <code>ETLConfig.from_env()</code> , коректності рядків підключення, значень <code>batch_size</code> і <code>log_level</code> . |
| <code>test_extract.py</code> | Моки з'єднань MySQL, фільтри дат, обробка винятків при недоступності бази. |
| <code>test_transform.py</code> | Валідність трансформацій: розрахунок метрик, усунення дублікатів, робота з пропусками, агрегування. |
| <code>test_load.py</code> | Тести <code>load_dimension</code> , <code>load_fact_sales</code> , <code>truncate_table</code> , <code>upsert_dimension</code> . |
| <code>test_pipeline.py</code> | Повний та інкрементальний флоу із використанням <code>MagicMock</code> . |

Сумарно виконуються 51 тест (~93% coverage, згідно з `ETL_SUMMARY.md`). Команда запуску: `pytest etl/tests/ -cov=etl -cov-report=html`.

9 Результати виконання

9.1 Продуктивність

| | Операція | Орієнтовний час |
|----------------------|---|-----------------|
| Вимірний час роботи: | Повне завантаження (50k+ записів) | 2–5 хвилин |
| | Інкрементальне за 1 день (до 1k записів) | 10–30 секунд |
| | Генерація <code>dim_date</code> (730 записів) | 1–2 секунди |
| | Трансформація 1000 записів | < 1 секунди |

9.2 Спостережність

- Усі модулі використовують `logging` з рівнями `INFO/DEBUG`. Логи CLI дублюються у `etl.log`.
- Airflow DAG'и push'ять статистику у XCom (`task_id=run_etl`, ключ `etl_results`) для подальших нотифікацій.

- Docker та Kubernetes сценарії описані в `ETL_SUMMARY.md`, що дозволяє швидко змінювати середовище виконання.

10 Висновки

Побудований ETL-сценарій відповідає вимогам лабораторної роботи №4:

- Виділено чіткі етапи Extract, Transform, Load із незалежними модулями та тестами.
- Забезпечено повторюваність завдяки конфігурації через `.env`, CLI та Airflow.
- Реалізовано механізми повного та інкрементального завантаження, включно з генерацією календаря та `upsert` для вимірів.
- Досягнуто високого рівня автоматизації та спостережності (логування, алерти, XCom статистика).