ETL для задач BI

На этом уроке

1. Цели Data Engineer'а при подготовке данных для задач BI.
2. Как ETL-процессы решают задачу преобразования данных к многомерной модели.

Теория

1. Общее напоминание о фазах Extract, Transform и Load.
2. О ETL в приложении на многомерную модель: в чем заключается подготовка данных к визуализации (соединение, фильтрация, объединение).
3. Проектирование OLAP-кубов. Bus Matrix.
4. Советы по визуализации данных.

Также в конце занятия будет дано домашнее задание с разбором.

Процессы Extract-Transform-Load

Напомним, ETL-процессы отвечают за движение данных из внешней среды в хранилище и внутри самой инфраструктуры данных.

Материализованное хранилище данных (в широком смысле, то есть не ограничиваясь только концепцией традиционного Data Warehouse) обычно имеет слоистую архитектуру. ETL процесс **преобразует данные и перемещает от слоя к слою**.

Первый слой — это слой временного хранения (т.н. staging или landing layer). В нем данные размещаются **в формате, максимально приближенном к формату источника**: например, json-документы для документных баз данных или CSV/TSV-файлы для реляционных хранилищ.

При размещении данных в следующем слое их **преобразуют к единой модели данных**. Обычно все данные приводятся к табличному виду для дальнейшей работы с ними средствами SQL и подобных технологий. Например, данные из реляционных хранилищ приводятся к такому формату интуитивно (буквально as-is), а вот преобразование документов или данных в объектной модели потребуется спроектировать. Этот слой называется operational data storage (ODS) — данные из него уже можно использовать для операционной аналитики в BI инструментах, не ожидая дальнейших трансформаций.

Однако **целью проектирования хранилища данных выступает создание «единой версии правды»**: данные из нескольких источников требуется взаимно обогатить. Обогащенные данные хранятся в следующем слое хранилища (т.н. enriched layer). Например, домашнее задание будет посвящено визуализации истории банковских транзакций. История транзакций может быть дополнена информацией о категориях покупок через коды MCC: каждая транзакция хранит только численный MCC код, а в справочнике MCC кодов (отдельная таблица) хранится расшифровка категории в название. Когда к каждой транзакции присоединяется расширенная информация о ее категории, история транзакций становится обогащенной.

Создание обогащенного слоя важно: несмотря на то, что каждый аналитик может из данных слоя ODS собрать необходимую ему обогащенную сущность самостоятельно, подходы разных аналитиков могут разойтись в мелочах и в дальнейшем потребуется сверка различных версий.

Далее **на базе слоя обогащенных данных уже можно строить визуализации**. Однако визуализации чаще всего строятся по агрегированным данным: например, число покупок в разбивке по регионам или число уникальных пользователей в разбивке по полу. Требуется **эффективно вычислять такие агрегаты**. Их можно предподсчитать и разместить в отдельном слое хранилища данных на другой технической базе, рассчитанной именно на такую нагрузку — **ROLAP**. Либо подготовить данные для работы с **MOLAP**, который сам возьмёт на себя эти расчеты.

Таким образом, выбор и настройка подходящего решения для использования в качестве источника для BI инструмента — и есть одна из ключевых задач Data Engineer'а в контексте внедрения практики BI в дополнение к организации поставки данных.

Проектирование OLAP-кубов

Другой важной задачей команды поддержки инфраструктуры данных является проектирование сущностей и агрегатов, представленных таблицами в хранилище данных. Напомним, что OLAP-решение является обычно источником данных для BI инструмента.

При этом не всегда удобно поддерживать для каждого отдельного элемента BI дашборда отдельную таблицу: бизнесу могут требоваться десятки визуализаций, и поэтому **разные агрегаты часто объединяют в одну таблицу**.

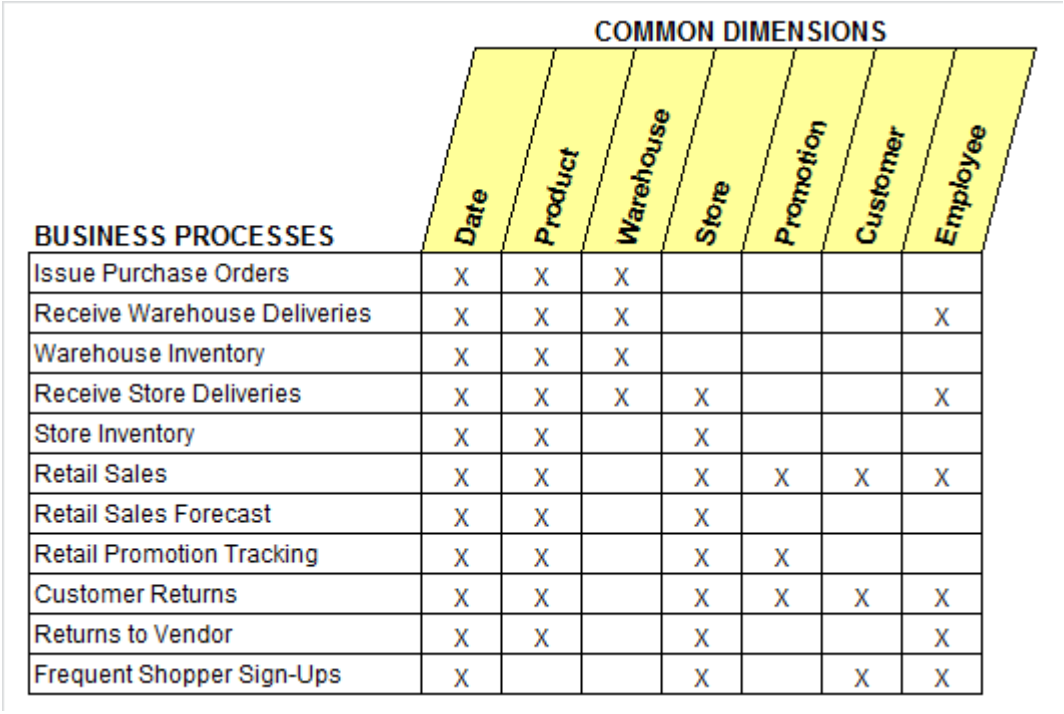
Например, на дашборде может требоваться вывести два графика:

- Суммарная выручка и количество заказов в разбивке по месяцу, тарифу и региону.
- Количество заказов и количество успешных заказов в разбивке по месяцу и тарифу (без региона).

У этих графиков совпадают два измерения и одна метрика (количество заказов). Для снижения сложности поддержки мы можем строить эти две визуализации по одной таблице, измерениями в которой будут месяц, тариф и регион, а показателями — все требуемые показатели. Мы знаем, что OLAP решение сможет быстро агрегировать количество заказов с группировкой по месяцу и тарифу (исключая регион), поэтому если регионов не слишком много (а обычно это так), пользователь не заметит ощутимой разницы во времени отрисовки первого (для которого ничего не агрегируется — таблица в точности соответствует его задаче) и второго графиков (с агрегацией).

Рассмотренный выше пример описывает случай с двумя общими измерениями. В реальных проектах число измерений может доходить до десяти и даже больше, поэтому использование этого подхода для проектирования OLAP-кубов вполне оправдано.

Для упрощения проектирования таких агрегатов существует метод Bus Matrix:



BUSINESS PROCESSES	Date	Product	Warehouse	Store	Promotion	Customer	Employee
	COMMON DIMENSIONS						
Issue Purchase Orders	X	X	X				
Receive Warehouse Deliveries	X	X	X				X
Warehouse Inventory	X	X	X				
Receive Store Deliveries	X	X	X	X			X
Store Inventory	X	X		X			
Retail Sales	X	X		X	X	X	X
Retail Sales Forecast	X	X		X			
Retail Promotion Tracking	X	X		X	X		
Customer Returns	X	X		X	X	X	X
Returns to Vendor	X	X		X			X
Frequent Shopper Sign-Ups	X			X		X	X

В столбцах таблицы выписываются измерения. В строках — потенциальные агрегаты (обычно один агрегат решает задачу одного бизнес-процесса). Показатели в таблице не участвуют (т.к. зависят от измерений и не влияют на гранулярность агрегата).

В ячейке в теле таблицы ставится отметка, если данному агрегату требуется заданное измерение. Например, отделу логистики может быть важно видеть показатели в разбивке по складу, а вот отделу продаж склады могут быть не важны, но важна рекламная кампания.

Затем бизнес-процессы группируются, чтобы объединить как можно больше общих измерений в одном агрегате и иметь как можно меньше пересечений с другими агрегатами — по сути, кластеризуются.

Появление новых измерений в агрегатах — это достаточно нечастое явление, поэтому пересматривать архитектуру хранилища регулярно не приходится.

Домашнее задание

В рамках домашнего задания вам предлагается построить аналитику по своей истории трат с разбиением по категориям.

Инструменты

Выполнять домашнее задание рекомендуется в BI инструменте Tableau, который имеет бесплатную двухнедельную пробную версию. По своему желанию вы можете выбрать другой, в том числе open source, инструмент, если сможете обеспечить преподавателя доступом к нему для проверки домашнего задания.

Например, Apache Superset или Redash можно в качестве полезного упражнения развернуть на виртуальной машине и обеспечить к ней (желательно — защищенный) доступ для внешних пользователей.

Или использовать бесплатный managed веб-инструмент, такой как Google Data Studio.

Источники

Визуализацию требуется построить на базе двух таблиц: таблицы-фактов с историей ваших транзакций, экспортированной (можно вручную) из онлайн-банка, и таблицы-измерения, сопоставляющей численные MCC-коды их названиям. Скачать MCC-коды можно, например, отсюда.

Эти таблицы требуется соединить в один источник данных (датасет). Сделать это можно удобным вам способом: или написать код, обрабатывающий данные и загрузить в BI сразу результат его работы, или соединить таблицы средствами самого BI инструмента. Не забудьте проверить типы ключа соединения на совместимость: не получится «сджойнить» ключ-число на ключ-строку.

Пример кода для чтения и соединения таблиц на языке программирования Python, версия 3.9:

```
import csv
from itertools import tee
from typing import Any, Callable, Generator, Iterable, Mapping

def read_records(file_name: str) -> Generator[dict, None, None]:
    """ Read records from CSV file; ``mcc_code`` field is required """
    with open(file_name) as transactions_file:
        yield from map(
            make_converter(mcc_code=int),
            csv.DictReader(transactions_file),
        )

def make_converter(**types_mapping: Callable[[Any], Any]) \
        -> Callable[[dict[str, Any]], dict[str, Any]]:
    def convert(record: dict[str, Any]) -> dict[str, Any]:
        """ Convert record fields to a specified type """
        return record | {
            key: _convert(record[key])
            for key, _convert in types_mapping.items()
        }
    return convert

def left_join_mcc_codes_to_transactions(
        transactions_records: str,
        mcc_codes_records: str,
) -> Generator[dict, None, None]:
    mcc_codes_mapping = {
        mcc_codes_record['mcc_code']: mcc_codes_record['category_name']
        for mcc_codes_record in mcc_codes_records
    }
    for transaction_record in transactions_records:
        transaction_mcc_code = transaction_record['mcc_code']
        yield transaction_record | \
            dict(category=mcc_codes_mapping[transaction_mcc_code])

def main() -> None:
    enriched_transactions = left_join_mcc_codes_to_transactions(
        transactions_records=read_records('transactions.csv'),
        mcc_codes_records=read_records('mcc_codes.csv'),
    )
    enriched_transactions, header = infer_header(enriched_transactions)
    with open('enriched_transactions.csv', 'w') as enriched_transactions_file:
```

```
        writer = csv.DictWriter(enriched_transactions_file, header)
        writer.writeheader()
        writer.writerows(enriched_transactions)

def infer_header(records: Iterable[dict]) \
        -> Tuple[Iterable[dict], typing.KeysView]:
    records, records_copy = tee(records)
    try:
        return records, next(records_copy).keys()
    except StopIteration:
        return records, dict().keys()

if __name__ == '__main__':
    main()
```

Код выше основан на генераторах Python и обрабатывает данные из входного файла с транзакциями построчно. Такой подход оправдан для больших датасетов, которые могут не поместиться в оперативную память, если загрузить их целиком. В своем домашнем задании вы вряд ли столкнетесь с критически большим объемом данных, но использование генераторов является в целом хорошей практикой, которая позволяет масштабировать ваш код.

Если вы используете версию Python <3.9, замените конструкции `d1 | d2` на `{**d1, **d2}`.

Рекомендуется также исключить из списка транзакций те, которые вы не считаете тратами. Например, пополнения, переводы между своими счетами или на счета родственникам. Изучите файл с транзакциями: есть ли общие паттерны, по которым можно отфильтровать переводы себе? Фильтрацию можно выполнить на удобном вам этапе:

1. Или добавьте логику фильтрации в приведенный выше код. Например, так: `transactions_records = filter(lambda transaction_record: 'перевод' not in transaction_record['description'].lower(), transactions_records)`.
2. Или используйте эти правила для фильтрации в Tableau.

Объедините MCC-коды в категории самостоятельно (например, MCC-категории «фастфуд» и «рестораны» в категорию «еда вне дома»). Это можно сделать или средствами BI-инструмента (рекомендуемый способ, в Tableau можно использовать функциональность групп или вычислимых полей), или на этапе подготовки данных, объединив некоторые MCC коды под одним именем в словаре `mcc_codes_mapping`.

Если вы не можете или не хотите использовать историю транзакций в качестве источника для домашнего задания, воспользуйтесь встроенным в Tableau датасетом Superstore.

Задание

Постарайтесь с использованием средств визуализации обнаружить какие-либо паттерны в вашем финансовом поведении. Возможно, сценарии оптимизации расходов.

Например, визуализируйте данные следующим образом:

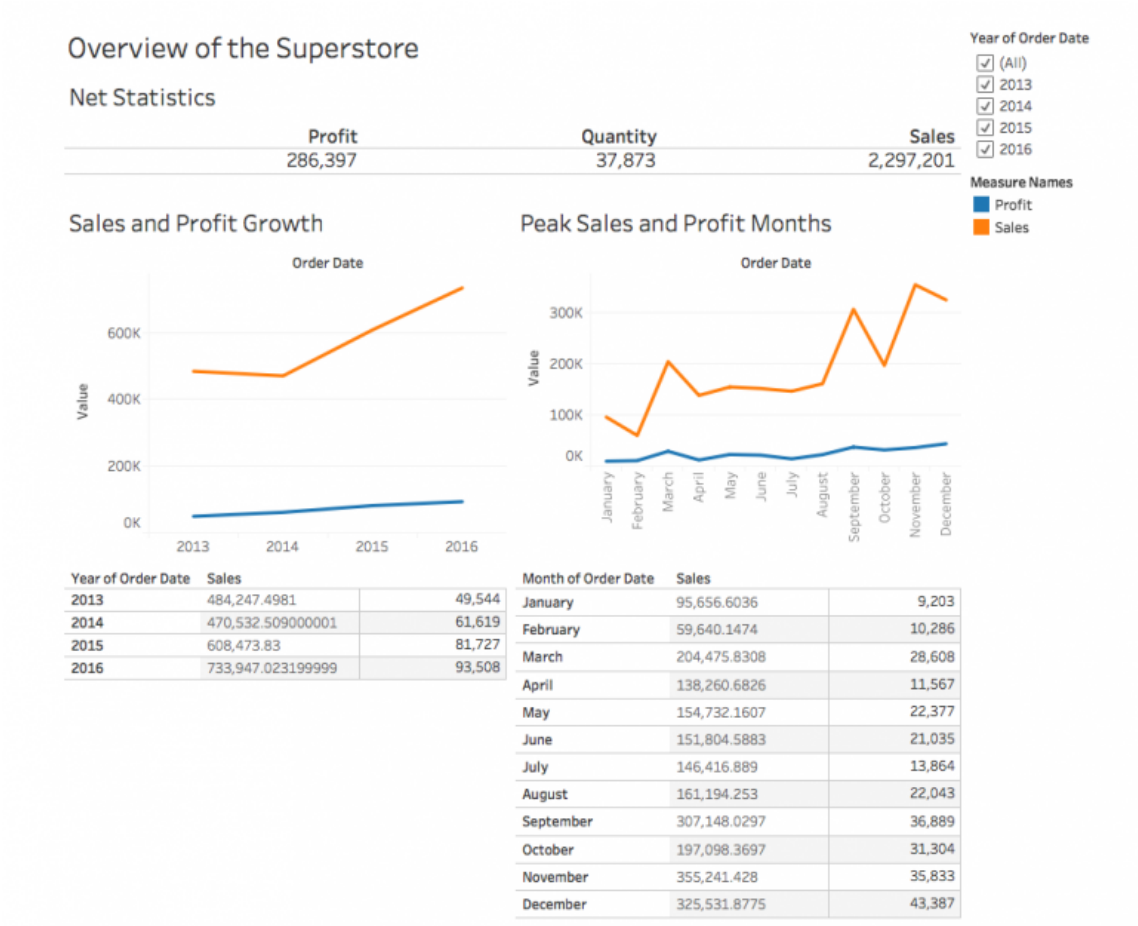
1. Топ категорий трат (по сумме) в течение последних 3, 6 или 12 месяцев.
2. Ваши среднемесячные траты как сумма средних по всем категориям (`sum(avg(spendings) for category in categories)`).
3. Разбивка трат по месяцам и категории.
4. Изобразите четырехмерный график на плоскости с использованием средств визуализации. Например, зависимость объема трат от месяца, категории и того, является ли дата траты выходным днем или нет.
5. Найдите категории трат с наибольшей долей в сравнении с другими месяцами.

Попробуйте использовать разные способы визуализации, чтобы инсайты в ваших данных были наглядными.

Затем из созданных вами графиков создайте дашборд. Дашборд — это набор визуализаций, обычно объединенных общими фильтрами. Например, фильтр по дате транзакции может быть применим ко всем визуализациям. Также вы можете использовать фильтр на признак выходного дня, категории и даже параметры категорий — например, отображать только те категории, в которых совершено не менее 10 покупок. Добавьте фильтр по своему усмотрению на дашборд.

Задание для датасета Superstore

Воссоздайте следующий дашборд:



Year of Order Date	Sales	
2013	484,247.4981	49,544
2014	470,532.509000001	61,619
2015	608,473.83	81,727
2016	733,947.023199999	93,508

Month of Order Date	Sales	
January	95,656.6036	9,203
February	59,640.1474	10,286
March	204,475.8308	28,608
April	138,260.6826	11,567
May	154,732.1607	22,377
June	151,804.5883	21,035
July	146,416.889	13,864
August	161,194.253	22,043
September	307,148.0297	36,889
October	197,098.3697	31,304
November	355,241.428	35,833
December	325,531.8775	43,387

Компоненты:

1. Фильтр по годам.
2. Таблица Net Statistics: суммарные значения Profit, Quantity и Sales.
3. Суммарные значения Sales и Profit в разбивке по годам.
4. Суммарные значения Sales и Profit в разбивке по месяцам.
5. Графики 3–4 в табличном виде.

Добавьте другие интересные вам компоненты по своему усмотрению. Например, в разбивке по географии. Посмотреть пример можно здесь.

Рекомендации по визуализациям

Визуализация (например, линейный график или столбчатая диаграмма) обычно имеет две оси (иногда — три): обычно ось-измерение и ось-показатель. Чтобы повышать размерность изображаемой на плоскости функции можно использовать следующие приемы:

1. Каждый столбец столбчатой диаграммы можно разбить на несколько. Причем подстолбцы могут быть размещены друг на друге (вертикально) или рядом (горизонтально). Каждый подстолбец — это разбиение отдельного столбца по еще одному измерению. Например, столбец изображает общую сумму трат за месяц, а подстолбцы — траты в каждой категории в этом месяце.
2. Можно совместить способы, описанные выше: разместить столбцы как рядом друг с другом, так и друг на друге. Таким образом можно изобразить до трех измерений для одной метрики.
3. Сводная таблица может изображать условно неограниченное число измерений и/или метрик. Однако с ростом числа измерений читаемость таблицы снижается.
4. Точечная диаграмма позволяет изобразить зависимость двух метрик друг от друга (например, зависимости количества покупок в категории от суммарных трат в ней), а добавление точкам размеров и цветов позволяет добавить еще один показатель (кодируемый размером точки) и измерение (например, признак выходного дня).

При выборе способа визуализации, стремитесь расположить как можно ближе те ее фрагменты, которые непосредственно требуется сравнить. Например, сравнить два соседних столбца визуально проще, чем два подстолбца, расположенных друг над другом.

Источники

1. Антихрупкость архитектуры хранилищ данных
2. Для вдохновения: Tableau Public Gallery
3. Tableau for Beginners – Data Visualisation made easy
4. Как начать работать с Tableau
5. Installing Superset from Scratch
6. Setting up a Redash Instance