

A photograph of a large industrial refinery or chemical plant. The foreground and middle ground are filled with complex steel structures, including tall vertical towers, horizontal pipes, and walkways. The sky above is blue with scattered white clouds. A bright sun is visible in the top right corner, casting long shadows and creating a lens flare effect.

Data Engineering Workshop

Mars Weng

2022-06-18

活動內容

[上午]

- 資料工程的起源
- Modelstorming : 以電商資料為例, 討論 & 設計 Data Warehouse
- 以 SQL+Redash 實作 Data Warehouse

[下午]

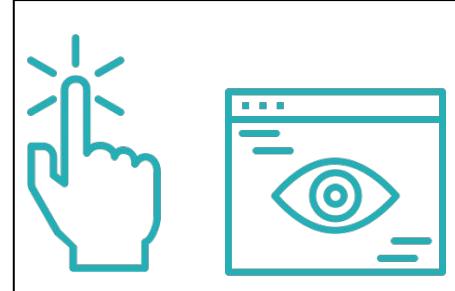
- Pipeline 實作 (Python + Airflow)
 - Modern Data Infrastructure
 - 資料工程這條路
- * 常見的資料工程名詞介紹

A wide-angle photograph of a coastal scene at sunset. In the foreground, there are large, rounded rock formations covered in bright green moss or algae. Small pools of water are trapped in the crevices between the rocks. Beyond the rocks, the ocean is visible with white-capped waves crashing onto the shore. The sky is filled with dramatic, dark clouds, with the sun setting behind them, casting a warm, golden glow over the entire scene.

Start from the beginning



用戶互動

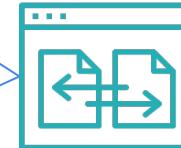


Front-end
iOS
Android

會員操作

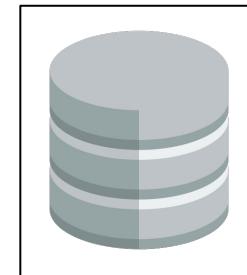


提供所需資料

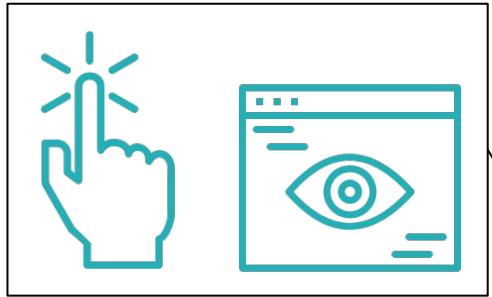


API

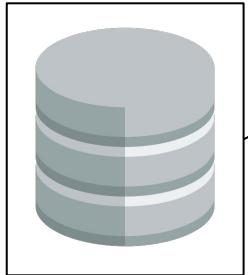
Back-end



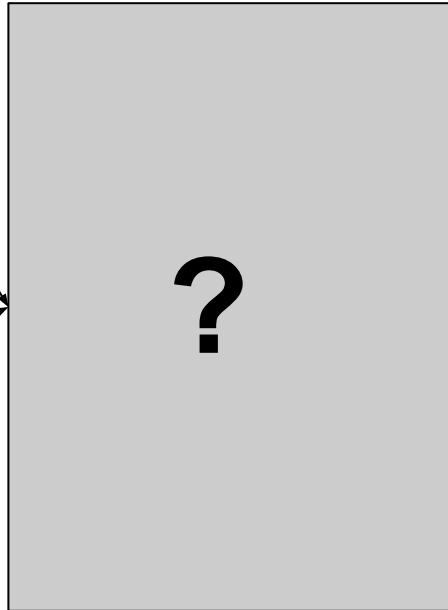
Database



用戶互動



會員操作



Data
Engineer
資料工程師

AI 應用

個人推薦、搜尋排序

Machine Learning
Engineer
機器學習工程師

Data Scientist
資料科學家

BI 應用

BI 決策、實驗性分析

Data Analyst
資料分析師

Why Data Warehousing?

	OLTP Online T ransactional Processing	OLAP Online A nalytical Processing
目的	執行用, 修改資料快速穩定	評估分析用, 讀取資料快速
操作方式	Insert, Update, Delete, Select	Select
操作資料量級	非常少 (兩位數以下)	可到非常巨大 (萬筆以上)
資料操作設計	能事先規劃, 事後也少變動	Ad-hoc
設計方式	Entity-Relationship Modeling (normalization: 3NF)	Dimensional Modeling (denormalization)
更新頻率	即時	每日 (即時)

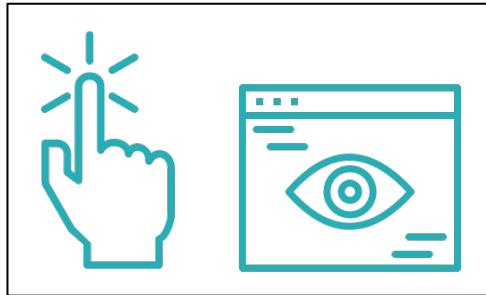
- ⇒ 符合 Query 需求 (eg. 歷史資料)
- ⇒ 增進 Query 效率
- ⇒ 避免與服務資料庫互搶資源

所以～資料工程師簡單來說到底是在做什麼？

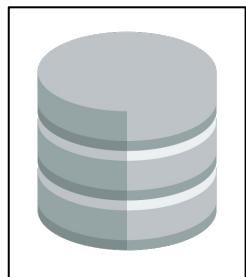
- 讓所有人都能有效率的使用資料

以用水比喻

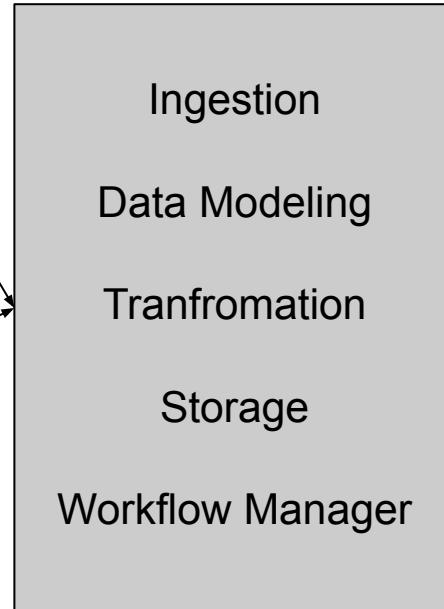
- 過去：
 - 鑿水井
 - 找到水源
 - 引水到農牧地
- 現代：
 - 規劃水庫位置
 - 建立淨水廠
 - 淨水廠到每個區域的水管管線
 - 每個用戶家裡個水龍頭的管線



用戶互動



會員操作



Data
Engineer
資料工程師

AI 應用

個人推薦、搜尋排序

Machine Learning
Engineer
機器學習工程師

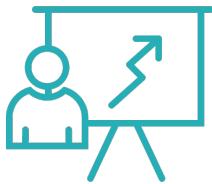
Data Scientist
資料科學家

BI 應用

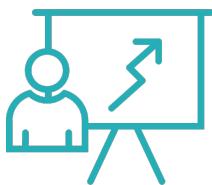
BI 決策、實驗性分析

Data Analyst
資料分析師

情境1



可以提供這個月的營收狀況的報表嗎？



可以分購買的商品類別嗎？



可以看一年嗎？

分析師

好的，馬上辦



好.....



.....



工程師

情境2



那這個金額是包含運費、折扣嗎？



那商品這幾個欄位是什麼意思？

分析師

這裡的訂單資料有日期，
可以自己算年、月、季度
可以自行 JOIN 商品資訊



這個欄位包含
這個欄位單獨運費
另外這個是折扣

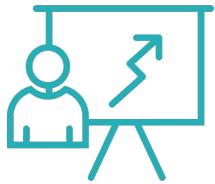


(在忙沒回覆)



工程師

情境3



分析師

ABCDEFG



PM

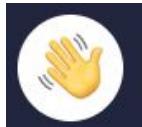
10011010010001010



工程師

大家有經歷過哪些情境呢？

- 1個



1

- 2個



2

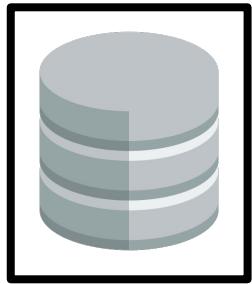
- 全部都中



3



用戶互動



會員操作

Ingestion

Data Modeling

Tranfromation

AI 應用

個人推薦、搜尋排序

BI 應用

BI 決策、實驗性分析

Modelstorming
= Data Modeling + Brainstorming

Why

- Data/Dimensional Modeling
 - 建立 Warehouse 的基礎
 - 讓 Query 效能提升
 - 能有符合報表所需內容的資料
- Brainstorming : 以事件的內容為中心
 - 不會只看報表需要欄位，造成設計彈性不足
 - 產生資料、處理資料、使用的資料的人可以一同參與討論，更有成就感與責任感

Brainstorming

以電商資料為例

如果希望找出來在最火熱的商品，大家覺得哪些可能是火熱的條件？

- for AI 推薦商品
- for MKT 下廣告、站內活動推廣
- for BD 去洽談更好的合作方式

希望透過 Data 的力量，讓營收更高、或提升用戶滿意度(黏著度)

因子

- 瀏覽最多次、瀏覽最久
- 被分享(點擊)
- 被收藏
- 帶來訂單:有付款、沒有取消
- 利潤高
- 評價最高、評價數
- 多少折扣、折扣比率
- 剩餘庫存
- 送貨速度
- 賣家回覆速度
- 駛升最快:購買數

因子 → 事件

Who Subject	verb event	What/Who Object	相關因子
會員	購買	商品	金額、折扣、付款、收到貨、退貨取消、成本
會員	評價	商品	星等
會員	收藏	商品/供應商	
用戶	看到	商品	
用戶	瀏覽	商品	瀏覽多久
用戶	分享	商品	

事件：購買

資料會變動？											
中文(腦力激盪)	會員	商品	訂單建立時間		收貨地址		訂單總金額		折扣活動內容		
7W	Who	What	When	When	Where	Where	How Many	How Many	Why	How 如何辨識 ?	
	Chris	衛生紙	2022-05-01		高雄		1000				
	James	iPhone 13	2022-04-06		桃園		20000		雙十一活動		

資料生成後，是否會更改？

- [FX] 不變 Fixed
- [CV] 會變動，但只需要最新資料 (SCD Type1)
- [HV] 會變動，需要歷史資料 (SCD Type2)
 - Slowly Changing Dimension (SCD Type1~6)

描述資訊:會員

資料會變動?											
中文(腦力激盪)	姓名		生日年		居住城市		購買訂單數		註冊活動/渠道	會員id	
7W	What	What	When	When	Where	Where	How Many	How Many	Why	How	
	Chris		1988-10-10		高雄			10		直接註冊	1
	James		1962-04-17		台北			2		雙十一活動	2

描述資訊：商品

資料會變動？											
中文(腦力激盪)	供應商	商品名稱		上架時間		商品製造地	商品庫存		商品描述	商品代號	
7W	Who	What	What	When	When	Where	How Many	How Many	Why	How	
		衛生紙									
		iPhone 13									

描述資訊：日期

資料會變動？					
中文(腦力激盪)	代碼	日期	星期幾	季度	
	20220101	2022-01-01	6	2022Q1	
	20220102	2022-01-02	7	2022Q1	
	20220603	2022-06-03	5	2022Q2	

```
SELECT create_time,  
       DATE_FORMAT(create_time, '%Y-%m-%d') as date_ymd,  
       DATE_FORMAT(create_time, '%w') as weekday, ...
```

⇒ 提升 Query 效率

事件：評價

資料會變動？							
中文(腦力激盪)	會員	商品	評價時間	星等		圖片、文字描述	How 如何辨識？
7W	Who	What	When	How Many	How Many	Why	How

事件：收藏

資料會變動？	HV	HV	HV	HV	HV
中文(腦力激盪)	會員	收藏對象	商品/供應商	收藏時間	會員+收藏對象+商品代號
7W	Who	What	What	When	How
		商品	商品代號		
		供應商	供應商代號		

Bus Matrix

有哪些共用的維度、讓所有合作對象更能綜觀全貌 (重要性)

	會員	商品	供應商	促銷活動
購買	∨	∨	∨	∨
評價	∨	∨		
收藏	∨	∨	(∨)	

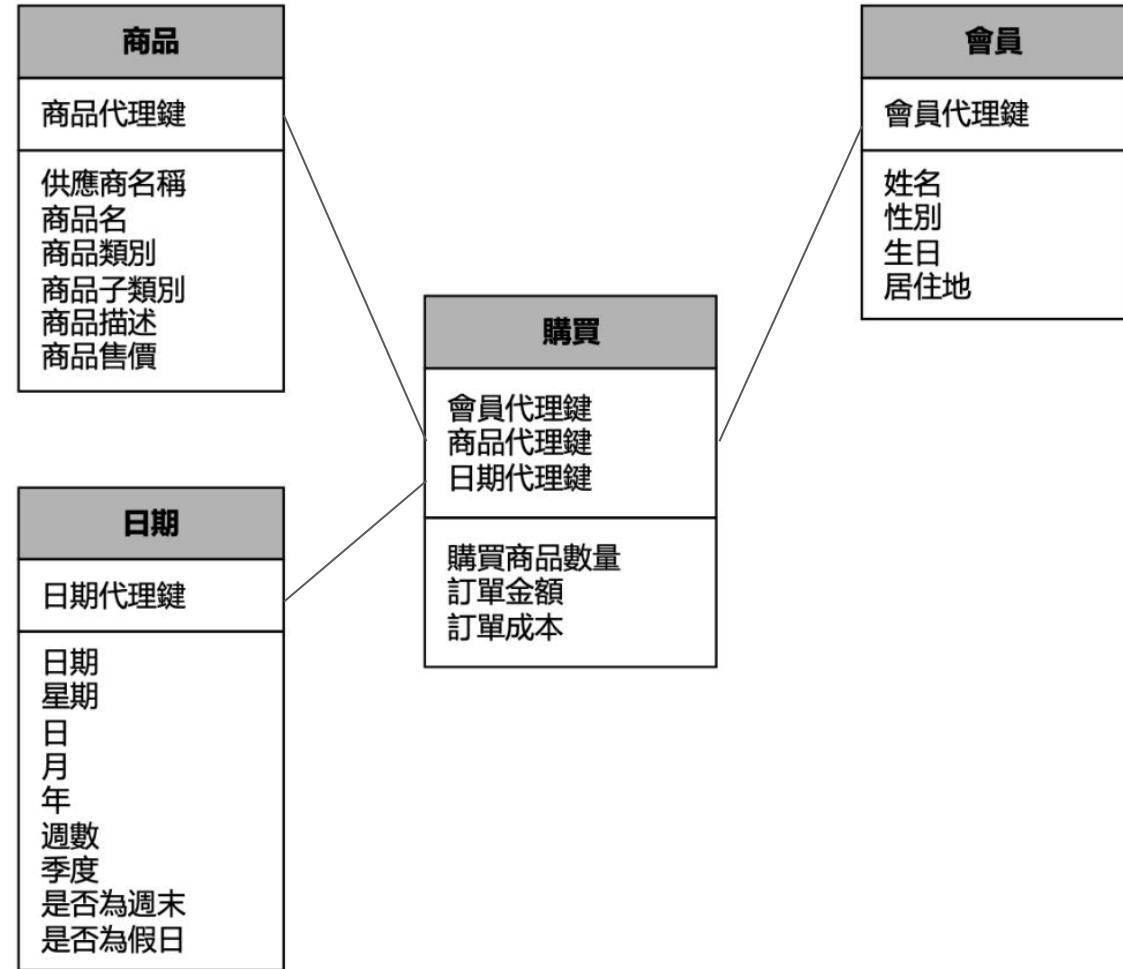
Fact & Dimension

- Fact Table: 事件, 會被 aggregation 計算的 <購買、評價、收藏、留言>
 - How many: 數值 (金額、商品數、訂單數,...)
 - Dimension SK
- Dimension Table: 描述資訊, 會被 filter, group <會員、商品、日期>
 - Who, What, When, Where, Why, How

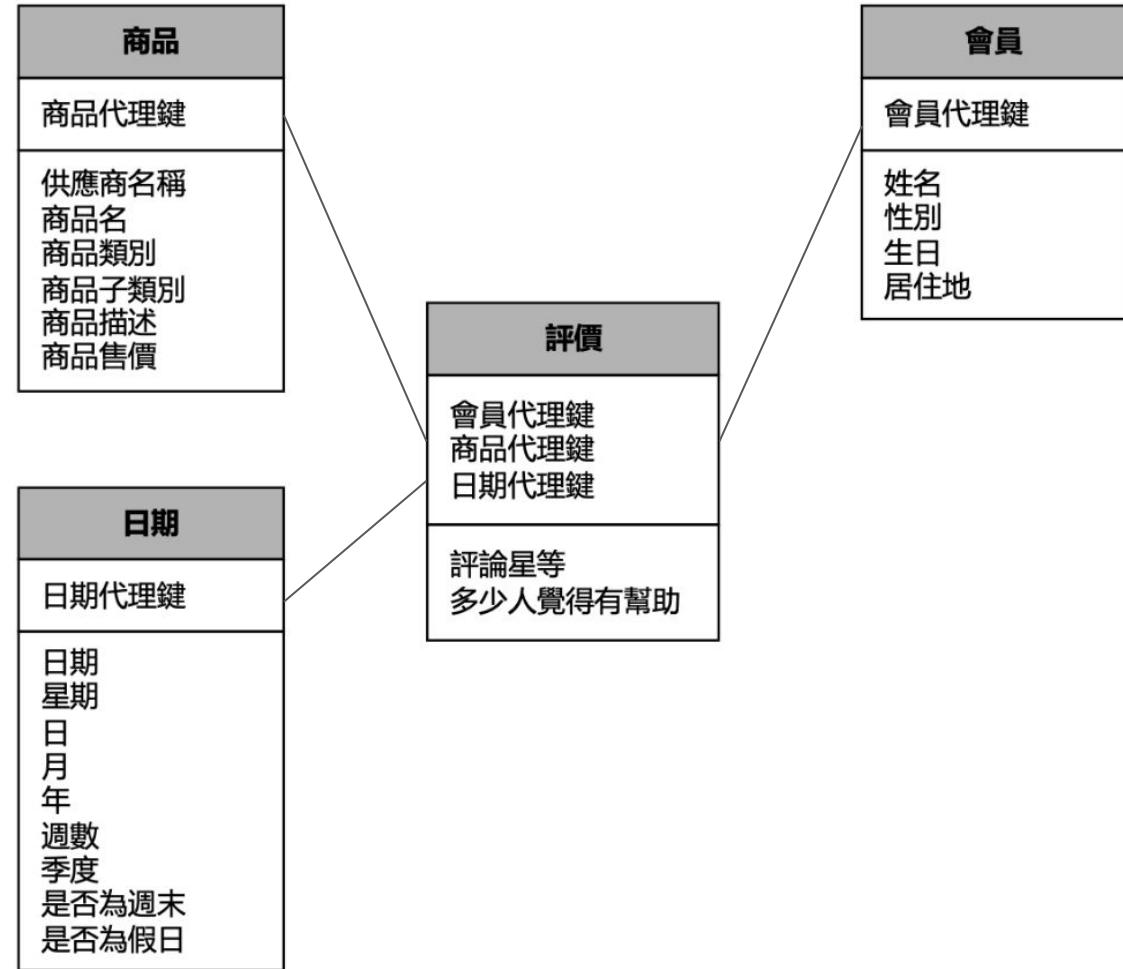
Star Schema - 購買

標準的 Dimensional Modeling 視覺化結果
讓大家可以很容易了解
哪些資料可拿來測量(fact)、
有哪些描述維度(dimension)

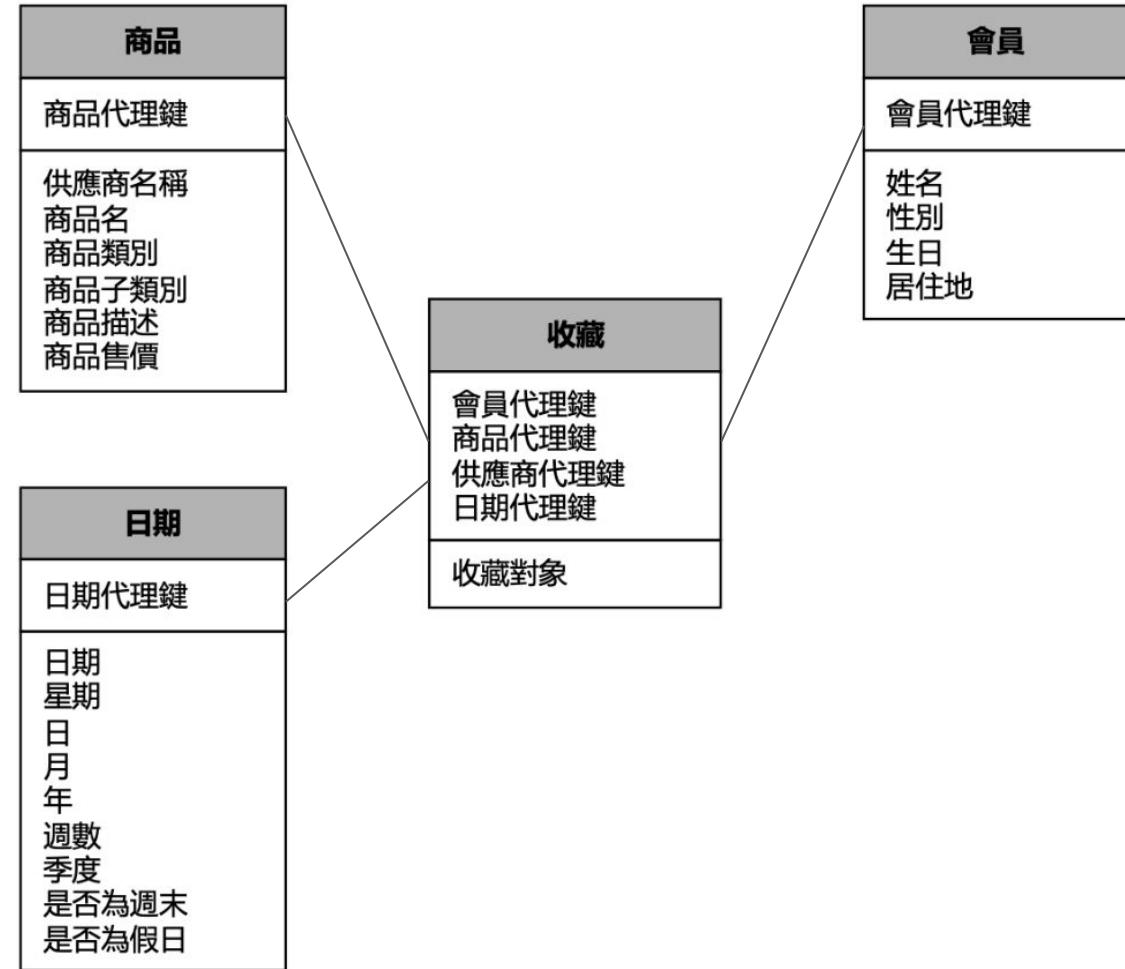
一個 schema
以一個事件(fact)為中心



Star Schema - 評價

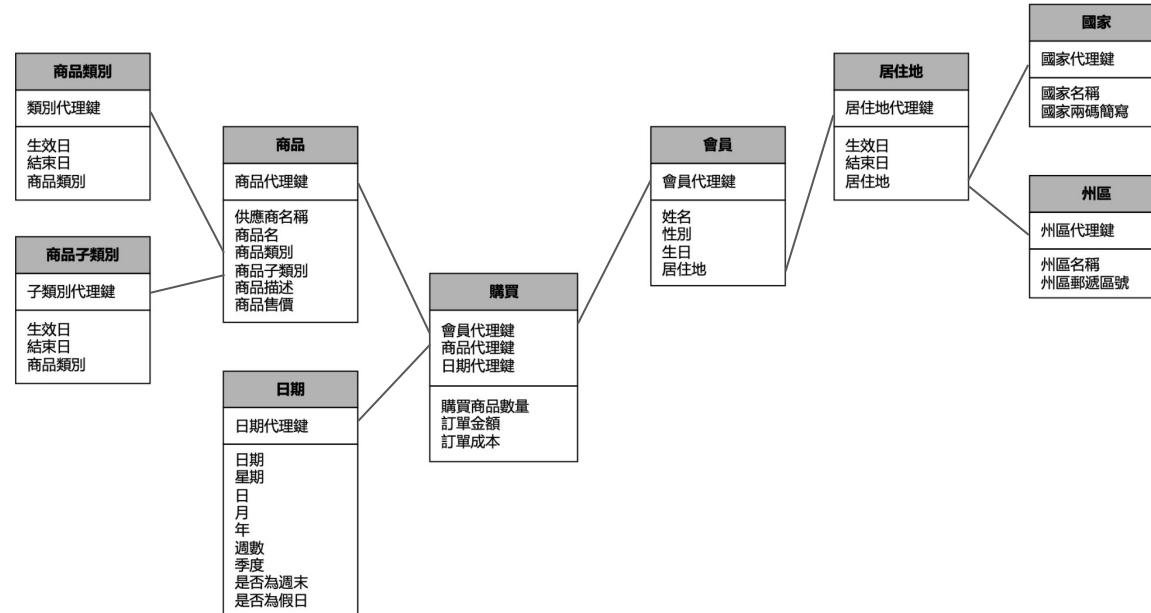


Star Schema - 收藏



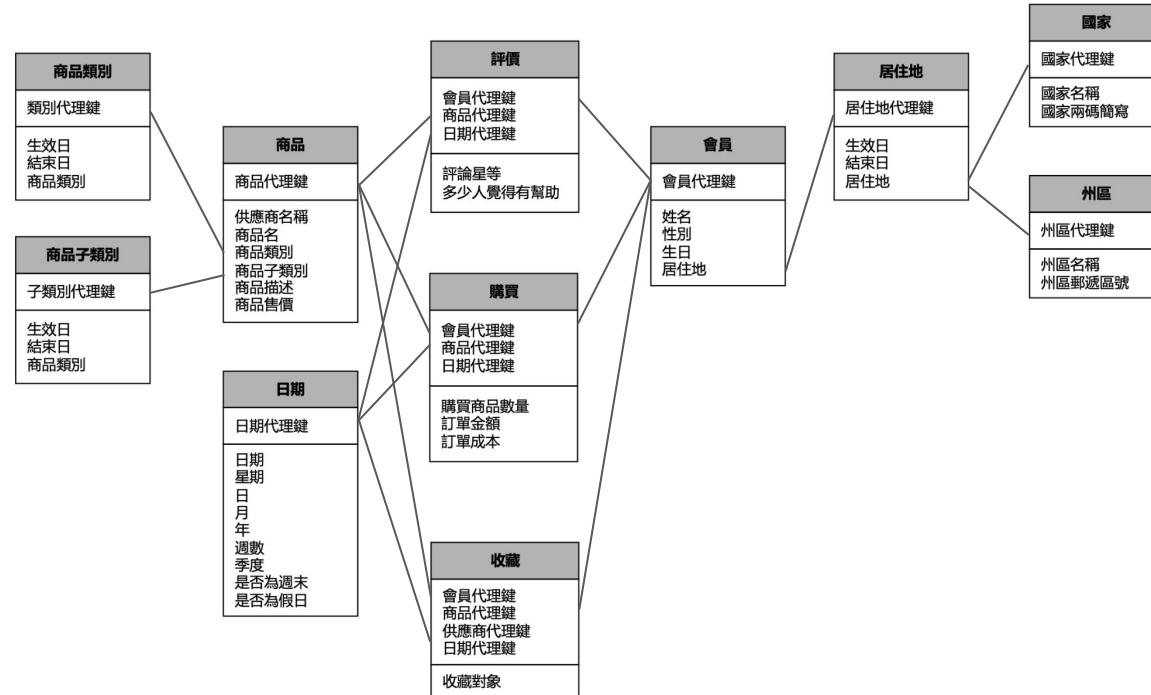
Snowflake Schema

當 Dimension Table 太過巨大，只好做部分 normalization。



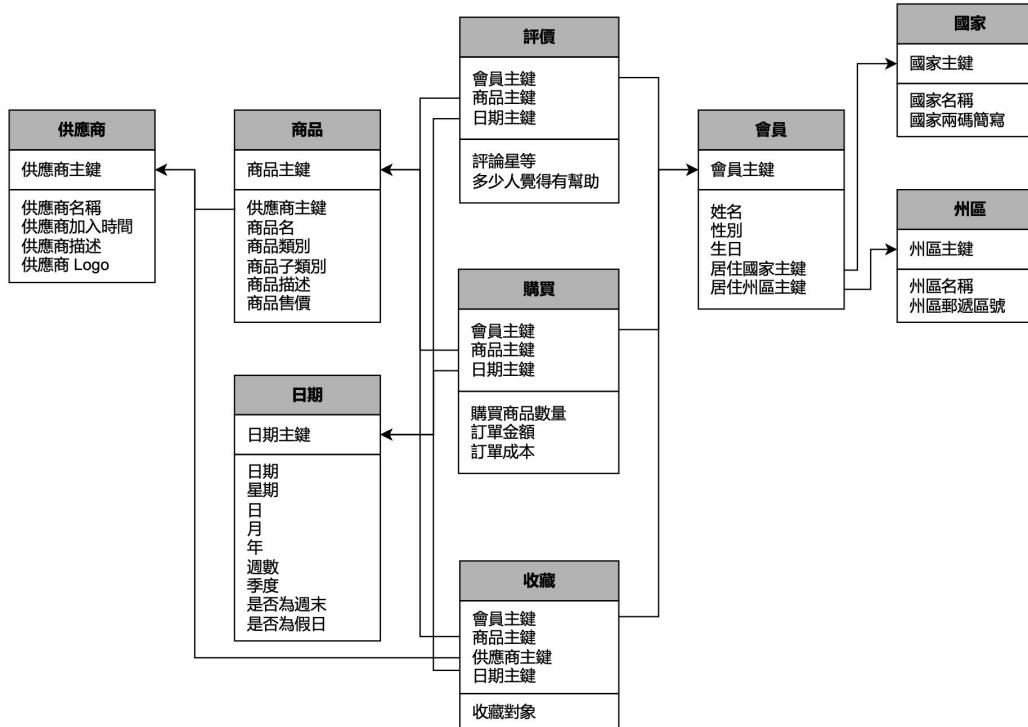
Fact Constellation Schema

Data Warehouse 真實情況：多張 Fact Table 共用 Dimension Table



vs. Entity-Relationship Diagram (ERD)

OLTP 使用：更重表之間的關聯，也只有最新資料



代理鍵(Surrogate Key, SK) 的好處

- 是整數值, 比主鍵(Primary Key, PK) 是字串在 JOIN 操作上有更高效能
- 避免原生資料意義改動
 - eg. 類別分類是 1,2,3 新版變成 11,12,30
- 處理「[HV] 會變動, 需要歷史資料」類型

6/10

商品主鍵	商品名	類別
A	iPhone 13	3C

6/15

商品主鍵	商品名	類別
A	iPhone 13	iPhone

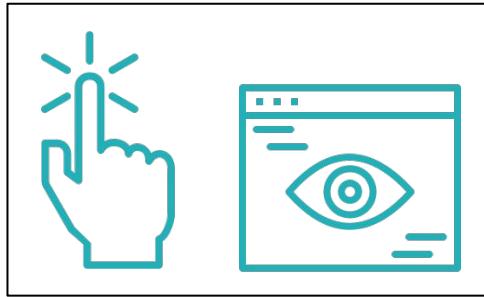
原生的商品資料表

Fact Table

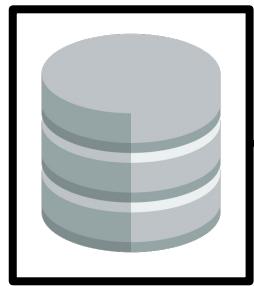
訂單代號	商品代理鍵	購買日期
060901	1	6/9
061605	2	6/16

Dimension Table

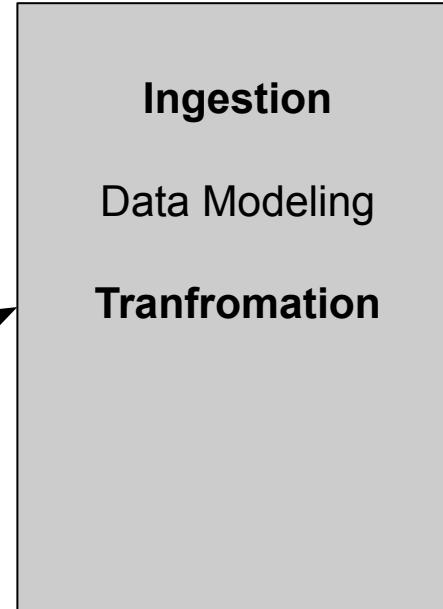
商品代理鍵	商品主鍵	商品名	類別
1	A	iPhone 13	3C
2	A	iPhone 13	iPhone



用戶互動



會員操作



AI 應用

個人推薦、搜尋排序

BI 應用

BI 決策、實驗性分析

Data Profiling

- 資料的來源是哪張表
- 有沒有缺值
 - 可從現有資料補
 - 統一用預設值
 - 棄用

Redash

- 可以連結多樣的資料來源
- 方便會 SQL、Python 的使用者，拿到資料可快速建立視覺化圖表
 - 在 Redash 中，處理資料與視覺化兩個是完全分開的操作
- 分享功能多元
- 自架：可客製化、資安風險可控

Redash - MySQL

The screenshot shows the Redash web application interface. At the top, there is a navigation bar with 'Dashboards', 'Queries', and 'Alerts' dropdowns, a 'Create' button with a dropdown menu ('Query', 'Dashboard', 'Alert'), a search bar ('Search queries...'), and a user profile icon ('MarsW').

The main area is titled 'New Query'. On the left, there is a 'source_sample' input field with a yellow border, a 'Search schema...' dropdown, and a list of available data sources:

- Accidents.nes... (selected)
- Accidents.oseba
- Accidents.upravna_enota
- Ad.ad
- AdventureWorks2014.AWBuildVersion
- AdventureWorks2014.Address
- AdventureWorks2014.AddressType
- AdventureWorks2014.BillOfMaterials

The central part of the screen has a red border around the '撰寫 SQL' (Write SQL) section, which contains a large empty text area for writing SQL queries. At the bottom right of this section are three small icons: {}, {}, and *.

At the bottom right of the entire interface are two buttons: 'Save' and 'Execute'.

Redash - Google Sheet

XXXXXX | n-1

- 試算表要開共享
 - email
- 格式處理

Google Sheets sample Unpublished

1 [1dLS_s0T_5yXe-IdedhIqtox-xvXxdX7VqVJEDfPgzac|1](#)

n-1
(從零開始數)

Save Exec

Table + New Visualization

Subject	event	Object/Subject	column_D
Who	verb	What	相關因子
會員	購買	商品	金額、折扣、付款、收到貨、退貨取消
會員	評價	商品	星等
會員	收藏(關注)	商品	
會員	收藏(關注)	供應商	

.../spreadsheets/d/XXXXXX/edit...

Dimensional Modeling

docs.google.com/spreadsheets/d/1dLS_s0T_5yXe-IdedhIqtox-xvXxdX7VqVJEDfPgzac/edit#gid=711845910

	A	B	C	D	E
1	Subject	event	Object/Subject		
2	Who	verb	What	相關因子	
3	會員	購買	商品	金額、折扣、付款、收到貨、退貨取消	
4	會員	評價	商品	星等	
5	會員	收藏(關注)	商品		
6	會員	收藏(關注)	供應商		
7	用戶	看到	商品		
8	用戶	瀏覽	商品		
9	用戶	分享	商品		
10					
11					
12					
13					
14					
15					
16					

第 n 張工作表

source RDB Table 概覽

資料 sample: <https://relational.fit.cvut.cz/dataset/TPCDS>

- fact: 訂單
- dim: 商品、會員、日期

值得注意的點:

- Star Schema 的好處
- Surrogate Key (SK): 歷史資料處理方式、Data Warehouse 省 Query 成本

Data Warehouse 實作 & 分組

- 所有人
 - 購買
- 分組：
 - 商品
 - 會員
 - 日期
 - 評價
 - 收藏
- 實作說明
- 實作完後請跟你做同樣資料的 PyLadies 們討論遇到缺值的處理方式

實作時間

實作成果應用

前面我們做的算是第一層原生資料，後面就可以做更進一步的組合

- 補上出貨天數、是否是假日、商品名稱
- 訂單銷售、出貨狀況、商品銷售狀況
- 每日：訂單銷售、出貨報表、商品銷售狀況
- 商品銷售前五名、特殊節日銷售
 - vs. 特殊節日銷售 無 layering 版

Redash 實作成果應用

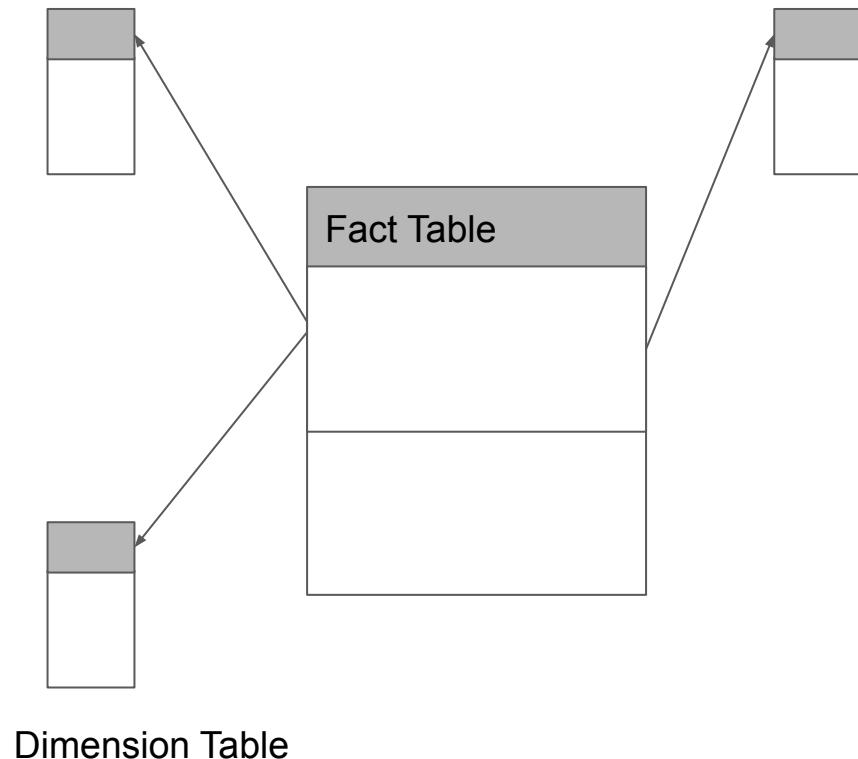
Data Layering 資料分層

數據結構清晰、邏輯簡單化

- 資料可重複應用
- 容易追蹤資料來源
- 降低 pipeline 每次執行的負擔
- 增加 query 效能

Dimension Table

Dimension Table



資料分層-各公司架構

通用	電商 1	電商 2	阿里巴巴	愛奇藝	實例
Source	RDB	RDB	RDB	RDB	
Data Warehouse	RDB denormalize	ingest	ODS	ODS	原生表
			DWD	DWD	Fact Table - 購買 (原生資訊)
Data Mart	stag	DWS	DWS	orders + date + product ⇒ 出貨時間、特殊節日、商品名	
			DWT	訂單銷售、出貨報表、商品銷售狀況	
	bi		DWA	每日：訂單銷售、出貨報表、商品銷售狀況	
		ADS	APP	商品銷售前五名、特殊節日銷售	

小小市調

大家的工作是拿哪層的資料分析的？感想如何？

- 原生表，需要自己 JOIN、GROUP BY
- 已經有 JOIN 完畢的表，但要自己 SELECT、GROUP BY
- 不需要 JOIN、GROUP BY，直接 SELECT * 就好
- 其他



1



2



3



4

Common Table Expression (CTE)

- 可讀性高
- 在某些平台效能較快

```
SELECT
    is_holiday,
    avg(product_price) as avg_product_price,
    avg(product_cost) as avg_product_cost,
    avg(total_price) as avg_total_price,
    avg(profit) as avg_profit
FROM (
    SELECT
        created,
        is_holiday,
        sum(product_price) as product_price,
        sum(product_cost) as product_cost,
        sum(total_price) as total_price,
        sum(profit) as profit
    FROM (
        SELECT
            sold.d_date as created,
            IF(sold.d_holiday = 'Y',1,0) as is_holiday,
            ws_quantity*ws_sales_price as product_price,
            ws_quantity*ws_wholesale_cost as product_cost,
            ws_net_paid as total_price,
            ws_net_profit as profit,
            ws_order_number as order_sk
        FROM web_sales
        INNER JOIN date_dim sold ON ws_sold_date_sk = sold.d_date_sk
        WHERE ws_sold_date_sk >= 2451666 and ws_sold_date_sk <= 2451818
    ) as A
    GROUP BY created
) as B
GROUP BY is_holiday
```

```
WITH orders_date as (
    SELECT
        sold.d_date as created,
        IF(sold.d_holiday = 'Y',1,0) as is_holiday,
        ws_quantity*ws_sales_price as product_price,
        ws_quantity*ws_wholesale_cost as product_cost,
        ws_net_paid as total_price,
        ws_net_profit as profit,
        ws_order_number as order_sk
    FROM web_sales
    INNER JOIN date_dim sold ON ws_sold_date_sk = sold.d_date_sk
    WHERE ws_sold_date_sk >= 2451666 and ws_sold_date_sk <= 2451818
),
daily_orders_date as (
    SELECT
        created,
        is_holiday,
        sum(product_price) as product_price,
        sum(product_cost) as product_cost,
        sum(total_price) as total_price,
        sum(profit) as profit
    FROM orders_date
    GROUP BY created
)
SELECT
    is_holiday,
    avg(product_price) as daily_avg_product_price,
    avg(product_cost) as daily_avg_product_cost,
    avg(total_price) as daily_avg_total_price,
    avg(profit) as daily_avg_profit
FROM daily_orders_date
GROUP BY is_holiday
```

Redash - Query Result

- Add Data Source
- SELECT * FROM query_N
 - cached_query_N
- SQLite 語法

A screenshot of the Redash interface showing a query result table. The URL in the browser bar is `pyladies-dews.marsw.tw/queries/61/source`, with the 'source' part highlighted by a red box. The table is titled 'orders' and has 14 columns: member_sk, product_sk, created_sk, shipped_sk, product_unit, product_unit_price, product_unit_cost, discount, shipping_fee, tax, total_price, profit, and order_sk. There are four rows of data displayed.

member_sk	product_sk	created_sk	shipped_sk	product_unit	product_unit_price	product_unit_cost	discount	shipping_fee	tax	total_price	profit	order_sk
7,598	23	2,451,666	2,451,667	55	51.88	36.49	656.28	650.10	109.85	2,197.12	190.17	1,701
59,756	121	2,451,666	2,451,675	74	88.16	66.79	0.00	4,348.98	521.90	6,523.84	1,581.38	5,703
22,757	151	2,451,666	2,451,770	62	43.77	40.88	0.00	0.00	244.23	2,713.74	179.18	3,872
84,809	167	2,451,666	2,451,777	39	53.30	44.10	0.00	1,106.82	0.00	2,078.70	358.80	53,689

New Query

A screenshot of the Redash interface showing a new query page. The 'Query Results' tab is selected, indicated by a red box. The query code is `1 select * from cached_query_61`. Below the code, the results of the query are displayed in a table titled 'orders'. The table has 14 columns: member_sk, product_sk, created_sk, shipped_sk, product_unit, product_unit_price, product_unit_cost, discount, shipping_fee, tax, total_price, profit, and order_sk. There are four rows of data displayed.

member_sk	product_sk	created_sk	shipped_sk	product_unit	product_unit_price	product_unit_cost	discount	shipping_fee	tax	total_price	profit	order_sk
7598	23	2,451,666	2451667	55	51.88	36.49	656.28	650.1	109.85	2,197.12	190.17	1,701
59756	121	2,451,666	2451675	74	88.16	66.79	0	4,348.98	521.9	6,523.84	1,581.38	5,703
22757	151	2,451,666	2451770	62	43.77	40.88	0	0	244.23	2,713.74	179.18	3,872
84809	167	2,451,666	2451777	39	53.3	44.1	0	1,106.82	0	2,078.7	358.80	53,689

Redash - Parameter: partial result

- 不能用在最原始資料 Query Result
 - 此例 queries/65 不能有 Parameter
- Type: Date, Dropdown List 很好用
- 選擇性使用參數小技巧: 原始 SQL 是字串的要保留「」

```
FROM cached_query_ 65
WHERE
  ('{{useDate}}' = 'Y' and created = '{{specific_date}}')
  or '{{useDate}}' = 'N'
```

useDate

Title: useDate

Type: Dropdown List

Values:
Y
N

Dropdown list values (newline delimited)

Allow multiple values

Cancel OK



specific_date

Title: specific_date

Type: Date

Cancel OK

```
1 SELECT
2     created,
3     product_name,
4     SUM(product_unit) as product_unit
5 FROM cached_query_65
6 WHERE '{{useDate}}' = 'Y' and created = '{{specific_date}}'
7     or '{{useDate}}' = 'N'
8 GROUP BY created, product_name
```

{{ }}

≡

⚡

Save

Execute

useDate 

Y 

specific_date 

2000-05-02 

Table

+ New Visualization

created	product_name	product_unit
2000-05-02	ableableableleeseought	98
2000-05-02	ableablebarought	59
2000-05-02	ableationablecally	89

 Edit Visualization

⋮

208 rows 1 second runtime

Updated 4 minutes ago

Redash - Filter: full result with partial display

- name alias:『"filter_name": "filter_type"』
- filter_type:
 - filter
 - multi-filter

```
1 SELECT
2     created as 'filter_date::filter',
3     product_name as 'filter_product::multi-filter',
4     SUM(product_unit) as product_unit
5 FROM cached_query_65
6 GROUP BY created, product_name
```

Save Execute

Table + New Visualization

Filter_date

2000-05-02 00:00:00

Filter_product

ableableableleeseought x ableablebarought x
ableationablecally x

filter_date	filter_product	product_unit
2000-05-02 00:00:00	ableableableleeseought	98
2000-05-02 00:00:00	ableablebarought	59
2000-05-02 00:00:00	ableationablecally	89

Edit Visualization : 50679 rows 18 seconds runtime Updated 6 minutes ago

Redas - Visualization : Table

- 調整顯示格式(不影響原始資料)
- Use column for search
- Display as Link(只有單欄資料)、Allow HTML content

```
SELECT
  *,
  CONCAT("<a href='https://www.imdb.com/find?q=",movies.name," ",year,'" target='_blank'>search on IMDB</a>") as search
FROM imdb_small.movies
```

Columns Grid

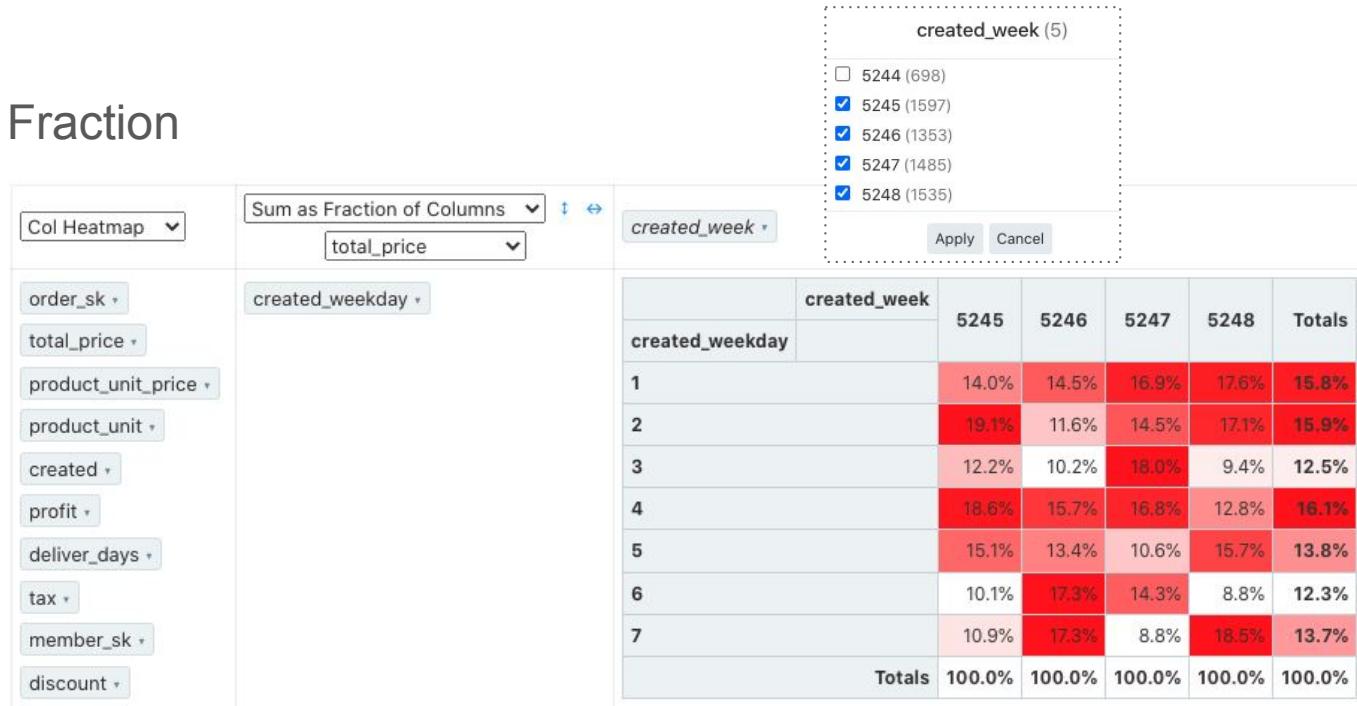
<input checked="" type="checkbox"/> id	<input checked="" type="checkbox"/> name	<input checked="" type="checkbox"/> year	<input checked="" type="checkbox"/> rank
<input type="checkbox"/> Use for search	<input checked="" type="checkbox"/> Use for search	<input type="checkbox"/> Use for search	<input type="checkbox"/> Use for search
Display as:	Display as:	Display as:	Display as:
Number	Link	Date/Time	Number
Number format	URL template	Date/Time format	Number format
0	:imdb.com/find?q={{ @ }}	YYYY	0,0,0
Text template	<code>{{ @ }}</code>		
Title template	<code>{{ @ }}</code>		
<input checked="" type="checkbox"/> Open in new tab			

id	name	year	rank	search
254943	Pi	1998	7.5	search on IMDB
256630	Pirates of the Caribbean	2003		search on IMDB

Redas - Visualization : Pivot Table

因為是在圖表做運算，有效能瓶頸：rows * columns \leq 50000

- Heatmap
- Count as Fraction
- Filter



Redas - Visualization : Chart

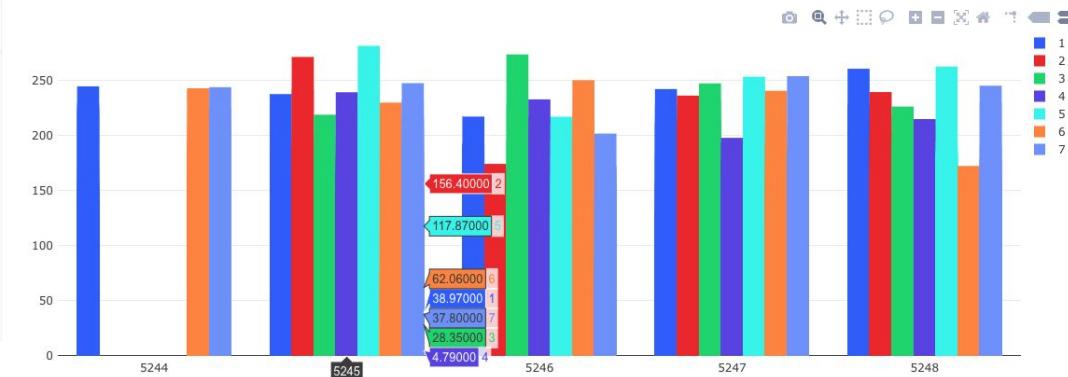
!!! 請先在 Query Editor 處理 Dimensions, Measures !!!

- Y: Measures
- X, Group by: Dimensions
 - 沒先處理就 Group by 只會出現 one row (很奇怪的結果)
- Chart Type: Line, Bar, Area, Pie, Scatter

Sum
total_price

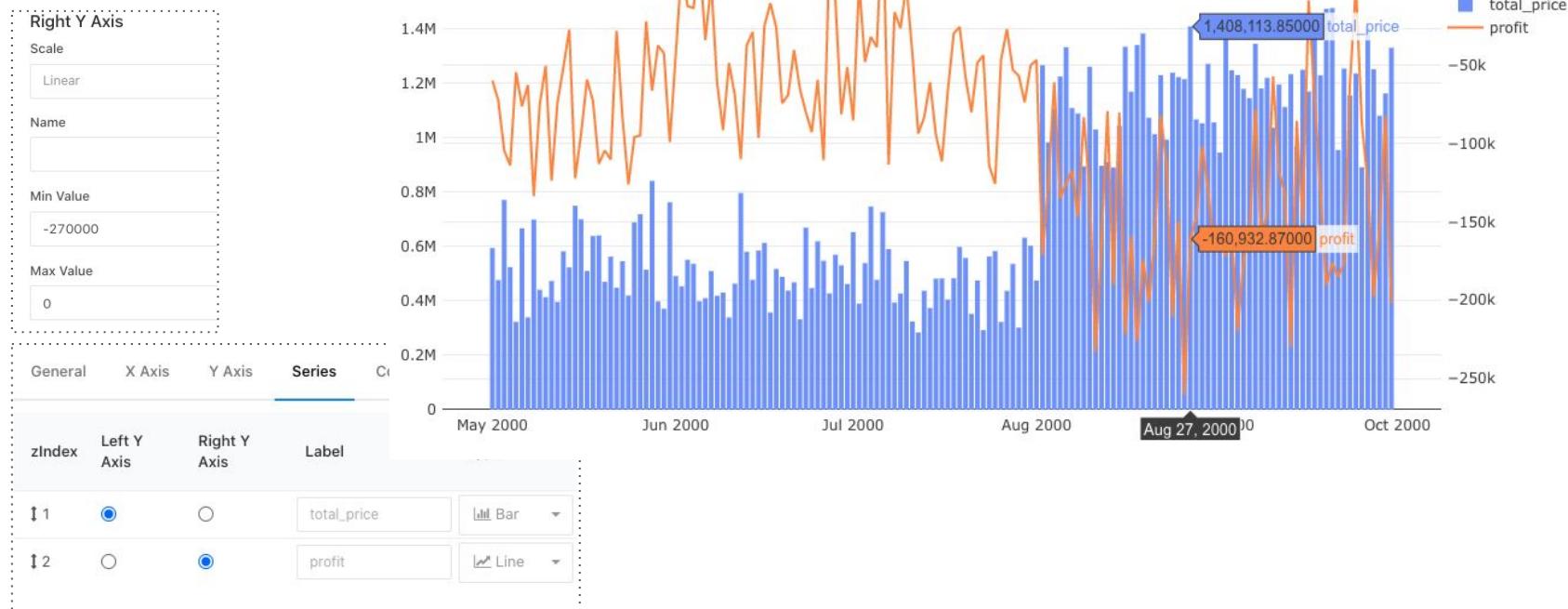
created_weekday

	created_weekday	5244	5245	5246	5247	5248	Totals
	created_weekday	538,614.31	546,475.24	404,978.58	562,817.18	602,507.44	2,655,382.76
1		746,679.14	324,160.74	482,953.03	583,180.41	2,136,973.32	
2		476,787.08	283,898.73	597,898.19	322,560.24	1,681,144.24	
3		725,995.32	437,175.00	557,645.10	436,683.81	2,157,499.23	
4		590,251.77	373,886.10	352,212.49	535,465.29	1,851,815.65	
5		652,682.77	393,766.26	481,599.78	474,804.88	301,852.65	2,304,706.34
6		389,812.39	427,079.53	481,370.56	292,446.25	632,068.98	2,223,777.50
7		Totals	1,581,109.47	3,907,034.34	2,788,069.28	3,320,777.12	3,414,318.82
							15,011,309.03



Redas - Visualization : Chart - Series

- Left/Right Y Axis
- Type



Redash - Fork

所有 Parameters, filters, visualizations 都能一起複製

The screenshot shows a Redash visualization interface. At the top, there's a title bar with a star icon and the word "Visualization". Below it is a code editor containing the following SQL query:

```
1 SELECT * FROM cached_query_62
2 WHERE created >= '{{start_date}}' AND created <= '{{end_date}}'
3
```

Below the code editor are two input fields for date parameters: "start_date" set to "2000-07-01" and "end_date" set to "2000-07-31".

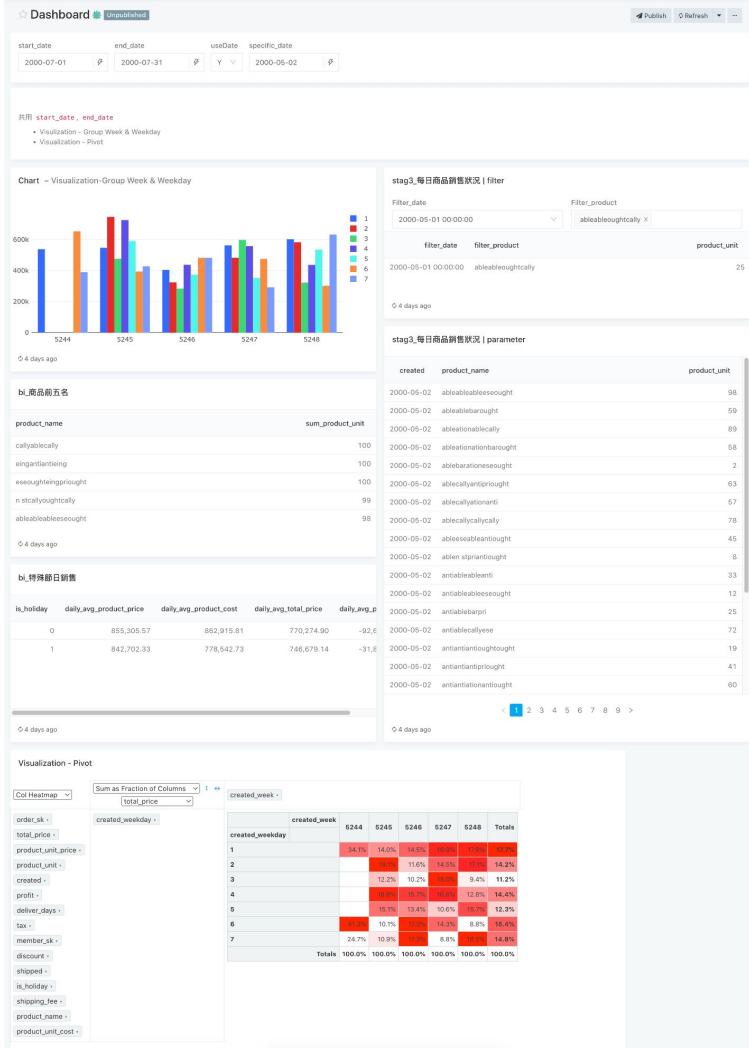
The main content area displays a table with the following columns: member_sk, product_name, created, created_weekday, created_week, is_holiday, shipped, deliver_days, and product_u. The table contains five rows of data:

member_sk	product_name	created	created_weekday	created_week	is_holiday	shipped	deliver_days	product_u
36,792	priable	2000-07-01 00:00:00	6	5,244	0	2000-08-31 00:00:00	61.00	
68,424	antin st	2000-07-01 00:00:00	6	5,244	0	2000-09-22 00:00:00	83.00	
82,604	oughtn stought	2000-07-01 00:00:00	6	5,244	0	2000-07-05 00:00:00	4.00	
29,747	oughtationable	2000-07-01 00:00:00	6	5,244	0	2000-07-25 00:00:00	24.00	
17,277	ationeingable	2000-07-01 00:00:00	6	5,244	0	2000-07-13 00:00:00	12.00	

On the right side of the interface, there's a "Fork" button with a dropdown menu containing "Show Data Only", "Archive", "Unpublish", and "Show API Key".

Redash - Dashboard

- Parameter: 全域使用
 - 所以同名的 Dimension attributes 很重要
- Filter: 單張 visualization 使用
- Markdown Textbox



其他常見的自架 BI 服務

	Redash	Metabase	Superset
處理資料&視覺化步驟	處理資料優先	處理資料優先	兩者皆可
處理資料工具	SQL、Python	SQL、GUI	SQL、GUI
視覺化圖表	基本	較 Redash 豐富一點	最為豐富 (但需要有時間維度)
資料源	最豐富	基本	只支援 SQLAlchemy
講師個人評論	適合工程師操作習慣 快速視覺化的好工具 但不適合非技術人員探索	適合非技術人員探索 很容易上手	適合非技術人員探索 需要一些訓練入門 視覺化功能非常強大

小結

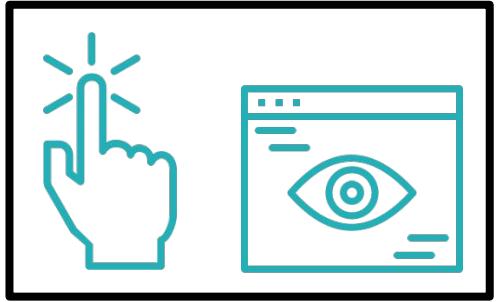
- Data/Dimensional Modeling
 - 建立 Warehouse 的基礎
 - 讓 Query 效能提升
 - 能有符合報表所需內容的資料
- Brainstorming : 以事件的內容為中心
 - 不會只看報表需要欄位，造成設計彈性不足
 - 先處理最為重要的事件
 - 產生資料、處理資料、使用的資料的人可以一同參與討論，更有成就感與責任感
- prototype 工具 : 表格、Redash

“Dimensional Modeling is a design technique for databases intended to support end-user queries in a data warehouse”

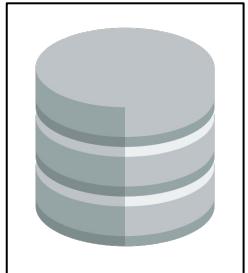
Ralph Kimball



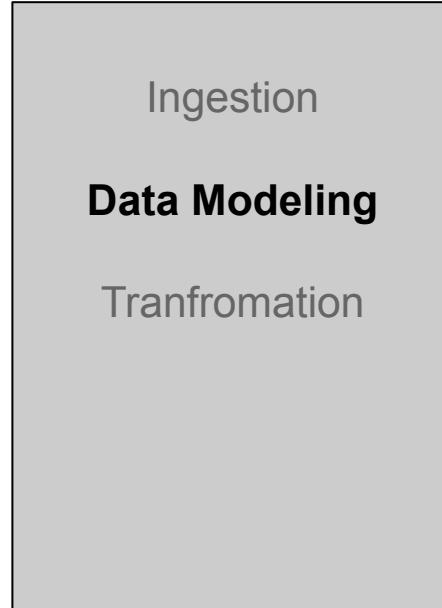
Break Time



用戶互動



會員操作



AI 應用

個人推薦、搜尋排序

BI 應用

BI 決策、實驗性分析

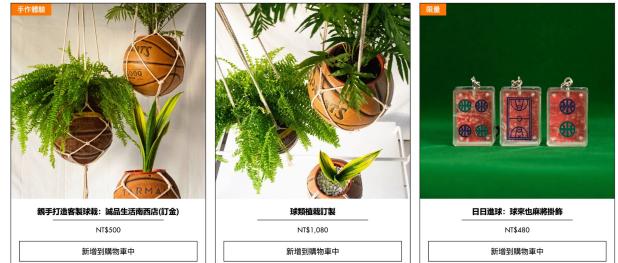
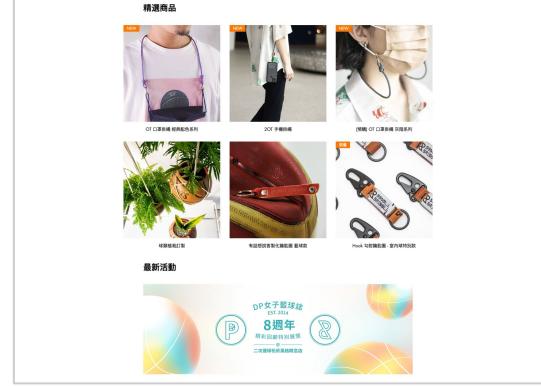
用戶事件

Who Subject	verb event	What/Who Object	相關因子
用戶	瀏覽	商品頁面	瀏覽多久
用戶	分享	商品	

用戶互動

當你想要在網路上購物的時候你會做什麼事呢？

- Google 搜尋:「籃球 再製」
- 選官網:<https://www.2doubledribble.com/>
- 稍微理解一下內容:在首頁往下滑
- 可能想看有沒有更多商品:點選更多商品
- 點選某一個有興趣的商品

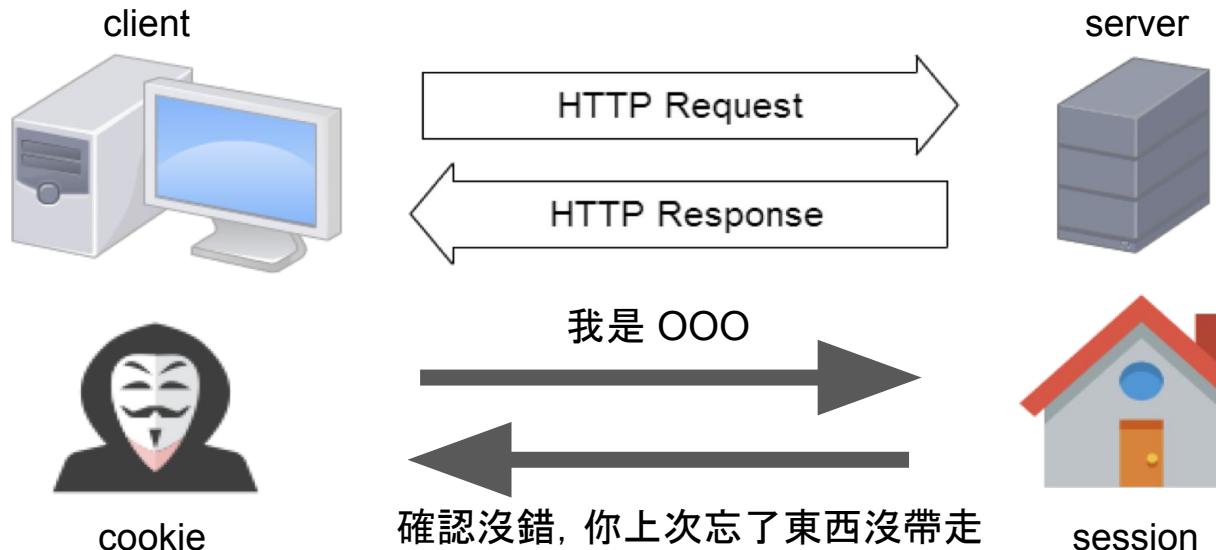


事件: 瀏覽頁面

資料會變動?									
中文(腦力激盪)	Who 如何辨識?	Who	頁面	頁面特有內容	什麼時候瀏覽的	頁面網址	停留時間	從哪裡來的網址	How 如何辨識?
7W	What	When	Where	How Many	Why				
	一個未知的使用者A		搜尋列表頁	搜尋關鍵字	2022-05-29 12:05:00	http://搜尋?搜尋!	121	Google 搜尋	
	一個未知的使用者A		商品A		2022-05-29 12:07:01	http://商品A			http://搜尋?搜尋關鍵字
	一個未知的使用者B		活動頁面		2022-05-29 12:05:00	http://活動			FB 廣告

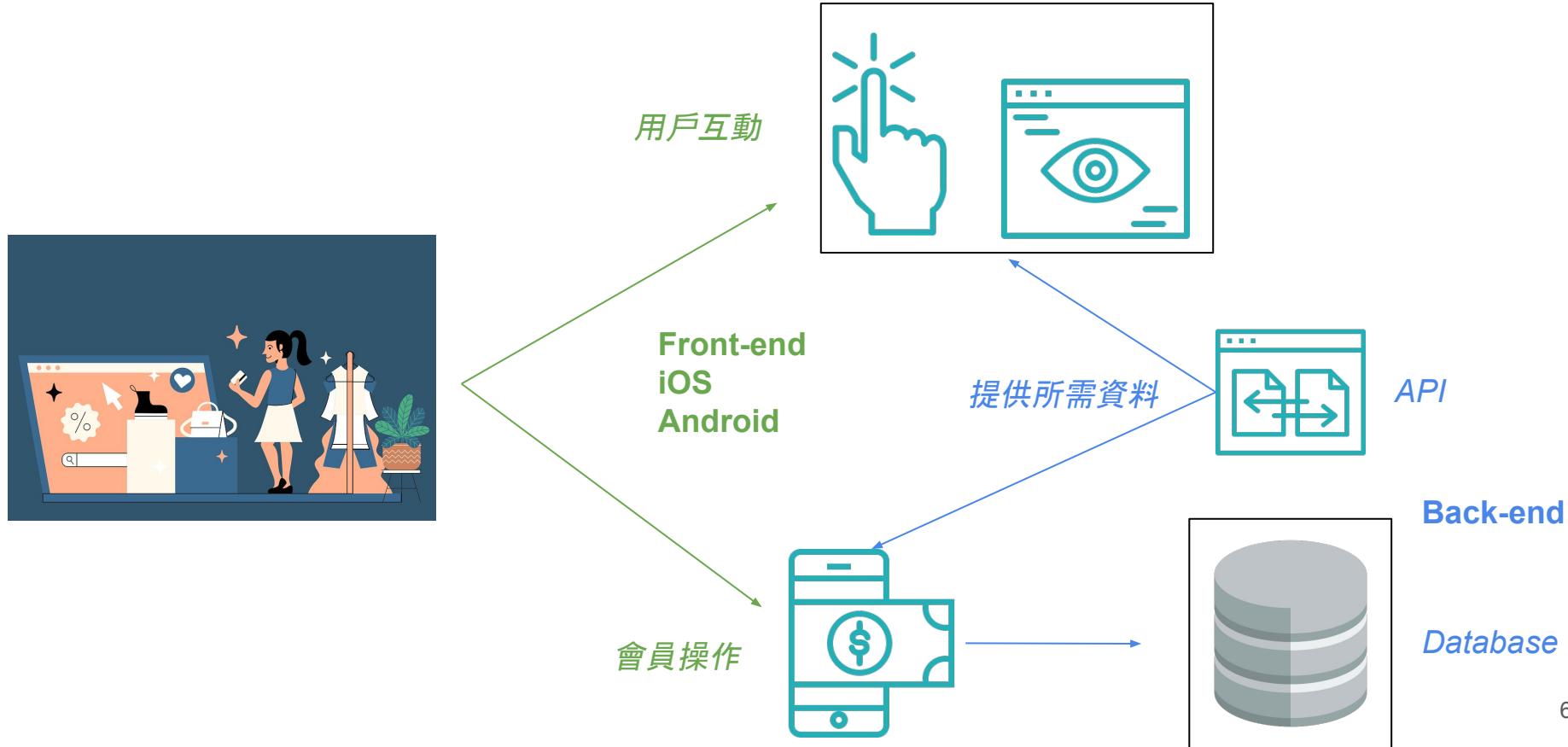
網頁運作原理-session/cookie

可以想像是沒有登入的人去領一張號碼牌的概念，
沒有身分證但櫃檯人員利用你的穿著辨認 ⇒ session
自己保管好號碼牌 ⇒ cookie
(同樣穿著弄丟號碼牌可以重領一樣的)



Request Cookies	
Name	Value
_ga	GA1.2.349...
cqid	GA1.2.361...
cf_bm	G3qyaM_I...

埋點/埋碼方式 - client/server side



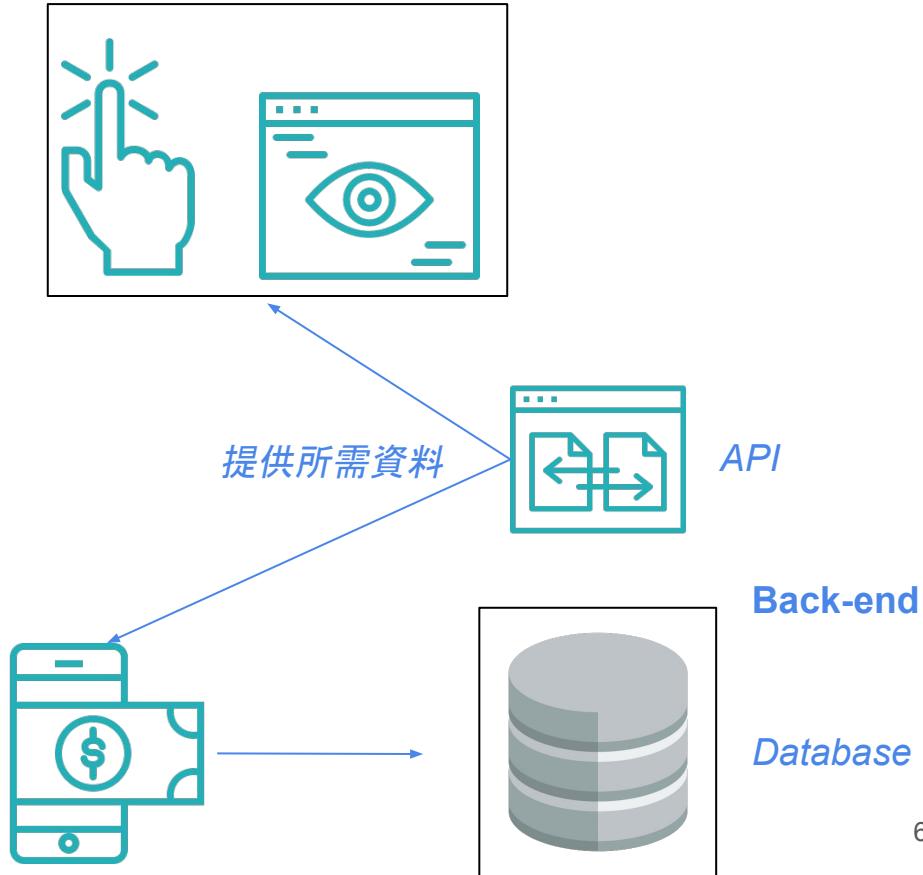
server side 埋點/埋碼

優點

- 只要在後端維護邏輯
- 後端記錄 log 的工具成熟

缺點

- 沒有牽涉資料的互動追蹤不到
 - 回到頂部
 - 關閉視窗
- 呼叫 API 跟用戶互動方式需要一樣
 - 整頁的資料是先取出還是滾動到再取？
- 會有網頁爬蟲干擾



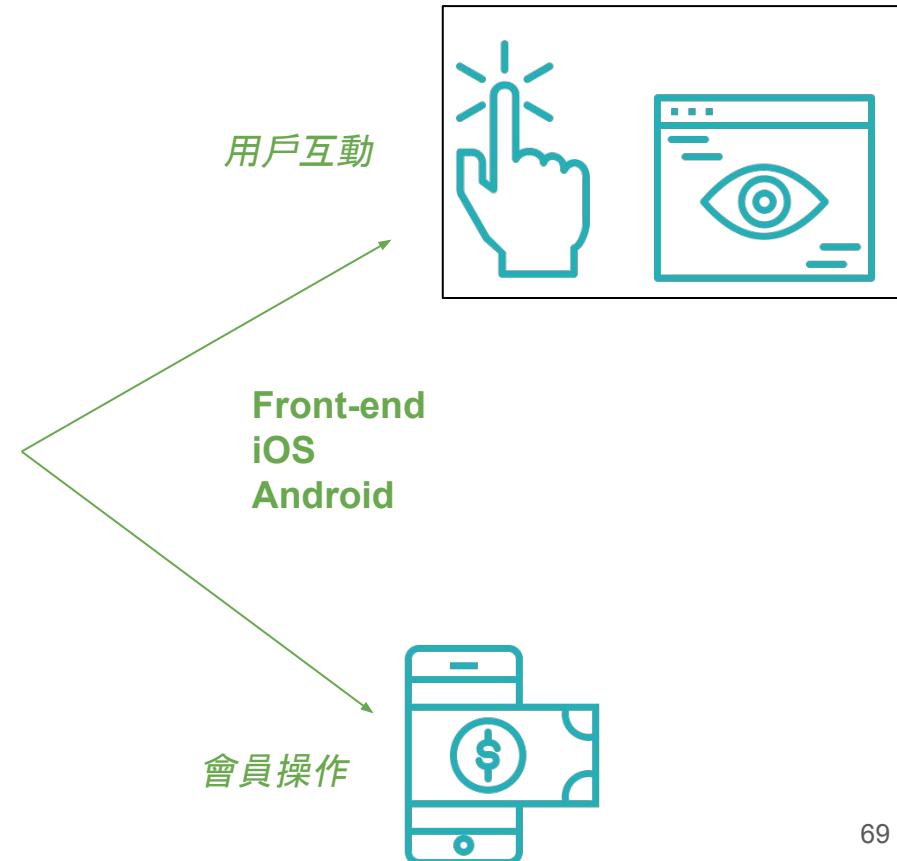
client side 埋點/埋碼

優點

- 使用者做什麼都可以追蹤得到

缺點

- 在網站外的行為追蹤不到
 - 發送推播是否有接收
- 每個行為都要定義
 - 不同裝置會有不同的畫面與操作方式
 - 多少比例的畫面算看到？
 - App 背景執行要繼續計算嗎？



經驗分享

- 電商1:只有 server side
 - 人手不足, 實作上最快的方式
- 電商2:server & client side 兩者皆有, 以 client side 為主
 - client side 沒定義或無法做到的, 再用 server side
 - 兩者資料可以交互驗證

計算瀏覽時間方式

需考慮資料需求、資料量大小：

- Front-End/iOS/Android：
 - 使用者的行為可以最清楚知道
 - 不同裝置以及畫面行為要詳細定義
 - eg. 開新視窗、有彈跳視窗出來要繼續計算嗎？
- Back-End：
 - 所有上站使用者的所有行為都只由幾台機器即時處理，又同時要處理使用者操作的服務 (API)，會不堪負荷
- Data 團隊：
 - 資料量跟 Back-End 一樣，但可以在背後專心批次 / 即時處理
 - 分頁間的跳轉難以計算

事件：點擊

資料會變動？						
中文(腦力激盪)	頁面	內容	什麼時候瀏覽的	頁面網址		
7W	Who	What	When	Where	How 如何辨識？	
	領了號碼牌的A	搜尋列表頁	以價格低->高排序	2022-05-29 12:05:00	http://搜尋?搜尋關鍵字	
	領了號碼牌的A	商品A		2022-05-29 12:07:10	http://商品A	
	領了號碼牌的B	活動頁面		2022-05-29 12:05:02	http://活動	

使用者看了哪些東西，哪些有點擊但哪些沒有？

表示沒有興趣，只是「看看」（「滑動」並不是重點動作）

- 同樣的內容，同樣的頁面會不會重複出現？
- 看到某個物件多少比例算是「看到」？
- 重複看到某個物件，算是多看到嗎？

舉例：

- A開啟了首頁(滑到底，又滑到最上緣)，
看到上下區塊兩個，點擊了上區塊
- B開啟了首頁，看到上區塊，點擊了上區塊
- C開啟了首頁，看到上區塊

上區塊的點擊率 = $2/3$ or $2/4$ (上區塊要被算兩次嗎？)

下區塊的點擊率 = $0/1$



事件:看到

資料會變動?								
中文(腦力激盪)	頁面	內容	什麼時候瀏覽的	頁面網址	在頁面的位置	在哪個頁面		
7W	Who	What	When	Where	Where	Where	How	如何辨識?
	領了號碼牌的A	搜尋列表頁	商品卡A	2022-05-29 12:05:00	http://搜尋?搜尋關鍵字			
	領了號碼牌的A	搜尋列表頁	商品卡B	2022-05-29 12:05:00	http://搜尋?搜尋關鍵字			
	領了號碼牌的A	商品A		2022-05-29 12:07:01	http://商品A			
	領了號碼牌的B	活動頁面		2022-05-29 12:05:01	http://活動			

辨別頁面的方式

- 靠事件時間 & 頁面網址/來源網址 是否足夠?
 - 同一個頁面內容可能不一樣 eg. 個人化推薦
 - 時間的精度有多細？
 - 新分頁、新視窗
- 應用：
 - 哪次的推薦效果比較好
 - 使用者的實際操作路徑

⇒ 每一次瀏覽頁面都給一個識別碼 view_sk



補足架構

- 點擊
 - 同樣頁面按鈕是不是也有可能會有重複 ⇒ 也需要位置資訊
 - 也可以用同樣頁面的辨識資訊
- 瀏覽
 - 本來要知道前一頁要利用時間 + 網址 ⇒ 可以用 ref_view_sk

常用網頁互動名詞

Who	verb	What	
用戶	瀏覽	頁面	view page
用戶	看到	某個物件	impression (某個商品被曝光)
用戶	點擊	某個物件	click

電商(互聯網)是個滿成熟的產業，幾乎都有現成實作方案，名詞很常沿用(eg. Google)

小結

先從事件出發，比較不會只專注在報表內容，忽略使用者真實行為樣貌

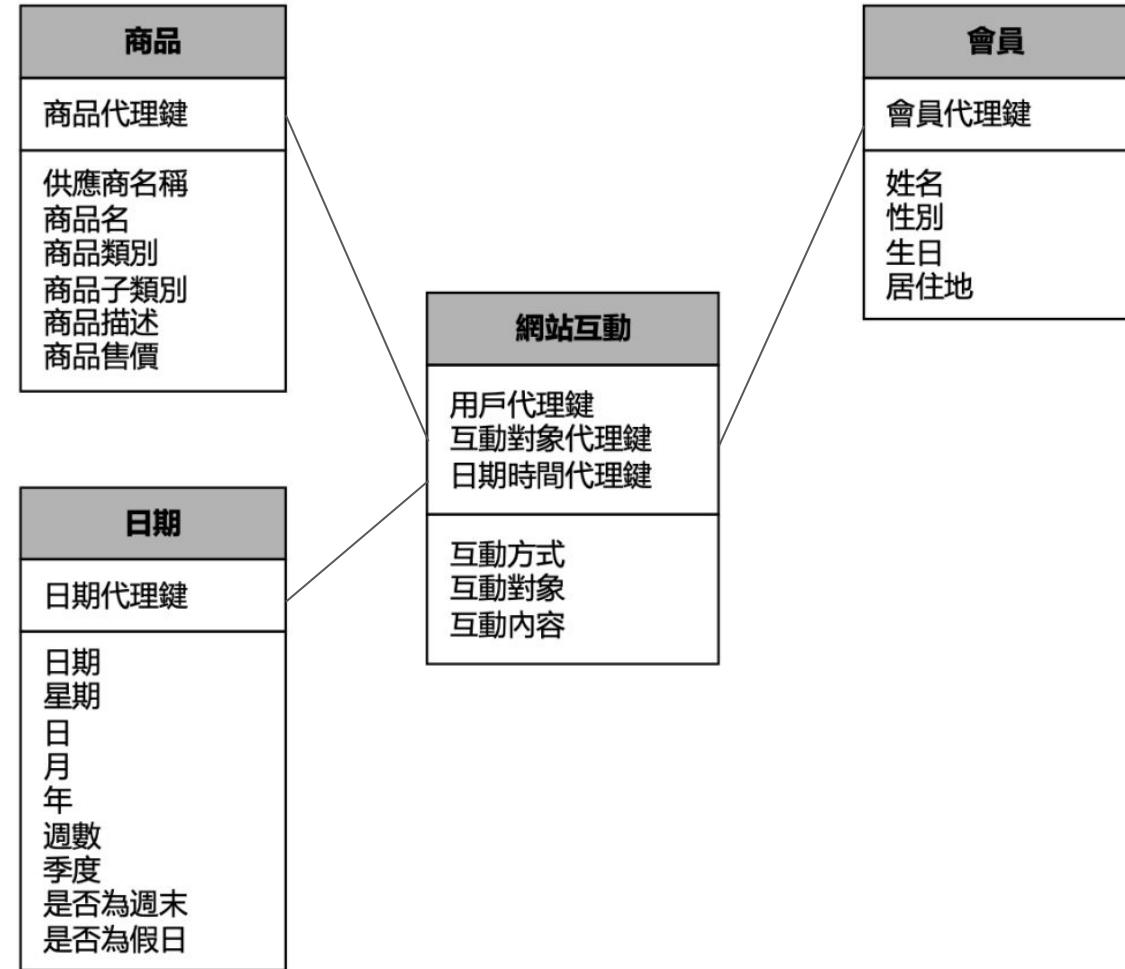
⇒

不知道怎麼設計的時候，可以去想你身為使用者，實際上做了什麼事，
把這些細節記下來，配合畫面截圖甚至是影片，
可以讓工程師在埋碼的時候就會是你想要的結果

用戶互動事件

Who	verb	What	When	Where	How Many	How 辨識方式
用戶	瀏覽	商品頁 活動頁	開啟頁面時間	網址、從哪個網址來 ref_view_sk	停留時間	view_sk
用戶	看到	商品卡 一列商品 一列活動	看到時間	view_sk 網址 相對/絕對位置		
用戶	點擊	按鈕 區塊	點擊時間	view_sk 相對/絕對位置		

Star Schema



資料分層-各公司架構

通用	電商 1		電商 2	愛奇藝	實例
Source	RDB	log	RDB, log	RDB, log	
Data Lake		未命名	ingest	ODS	用戶互動、原生表
Data Warehouse	RDB denormalize		DWD	Fact Table - 用戶互動、購買 (原生資訊)	
Data Mart		stag	DWS	網頁停留時間、補齊登入資訊 用戶「 <u>工作階段</u> 」切分、是否產生訂單、 <u>屬於哪個渠道</u>	
			DWT	訂單銷售、渠道貢獻	
		bi	DWA	每日：訂單銷售、渠道貢獻、用 戶互動	
			APP	特定站內外活動各頁面轉換率表現	

資料處理方式

資料是否會變動 / 需求	資料處理方式
[FX] 不變	full dump、incremental (建立時間)
[CV] 會變動，但只需要最新資料	full dump、incremental (建立時間+修改時間)
[HV] 會變動，需要歷史資料	incremental (修改時間)

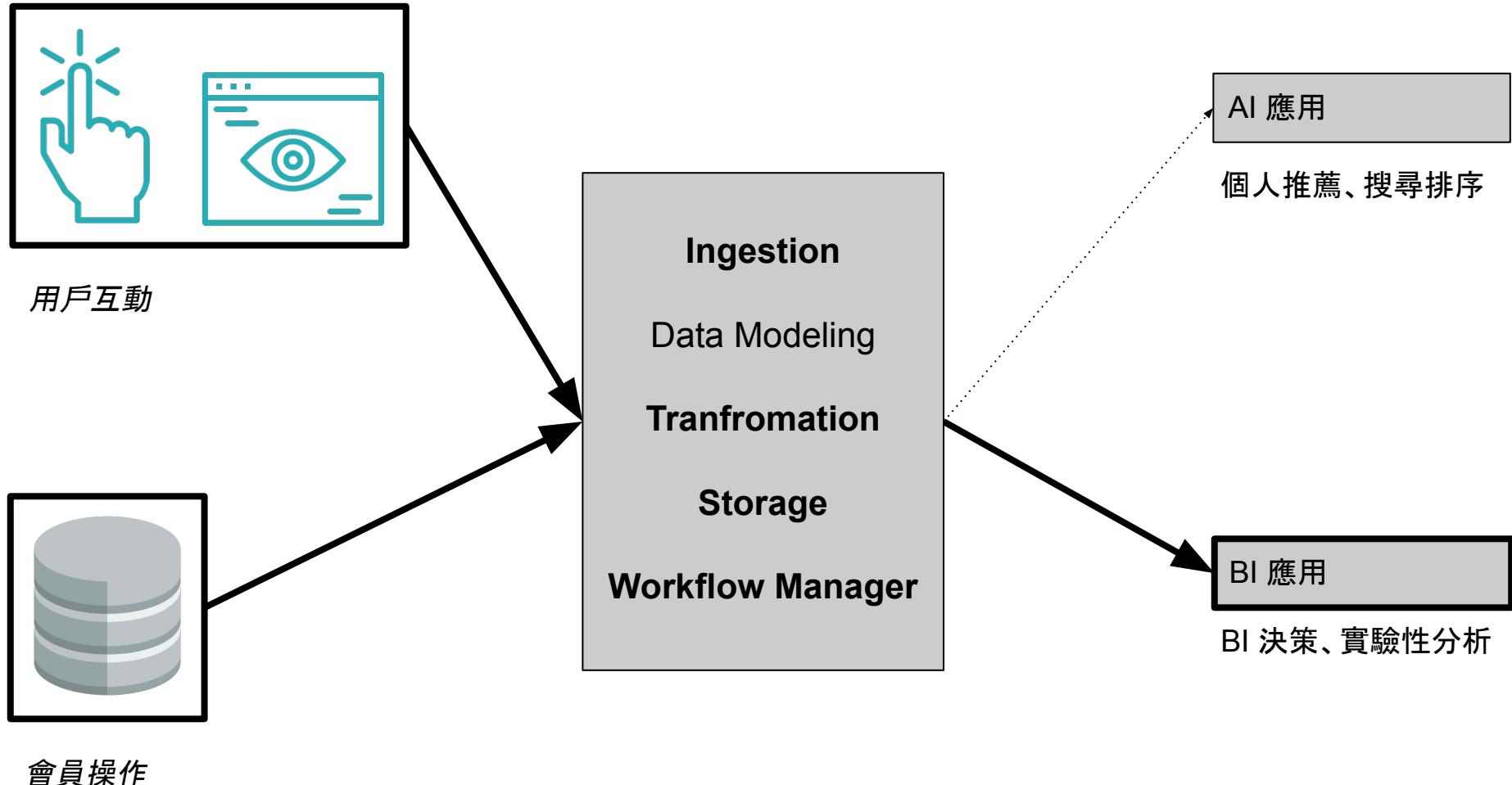
通用處理方式：Change Data Capture

資料處理頻率

- batch: 容易開發維護、資料較穩定 [FX, CV]
- streaming: 即時資料、每次處理的資料筆數較少 [FX, CV, HV]

幾種常見的資料處理架構:

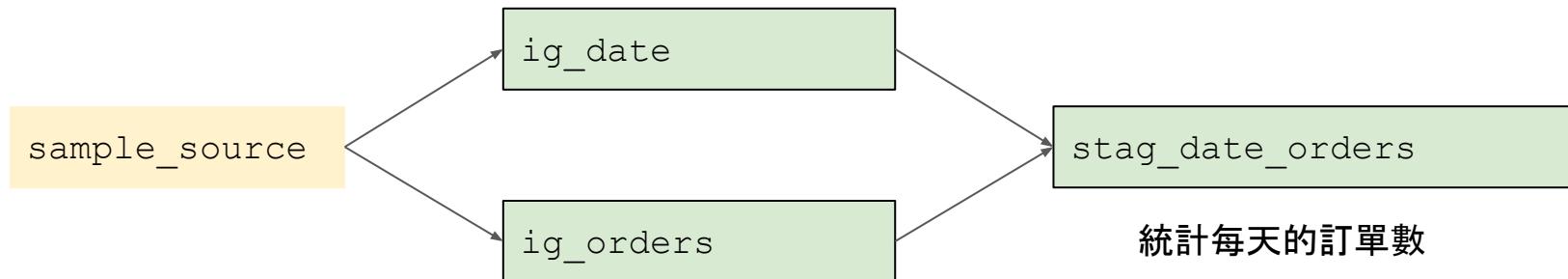
- Lambda: 同一資料源 batch + streaming 兩條路單獨處理 & Query
- Delta: 同一資料源 batch + streaming 兩條路單獨處理, 但可統一 Query
- Kappa: streaming only
- Zeta: streaming only + 其他資源管理/運算模組



Airflow

管理 pipeline (由多個 Tasks 構成) 的開源工具

- batch 管理「事件發生時間」與「處理該事件數據時間」有落差的 Tasks
- DAG (Directed Acyclic Graph) : 定義每個 Task 的執行順序、彼此間關係



ig:
 ingest 把資料落地不做 aggregation
stag:
 stage 資料的各種 aggregation,
 window function 計算

DAG - Data Mart / BI 應用

需求有

- 統計每天的訂單數、顯示該天是否為假日
- 補齊會員最後登入時間
- 統計會員累積購買訂單數
- 統計商品累積收藏數
- 統計商品累積購買訂單數
- 統計商品累積評價數、平均評分星等
- 統計商品累積瀏覽數

stag_date_orders

stag_member_interacts

stag_member_orders

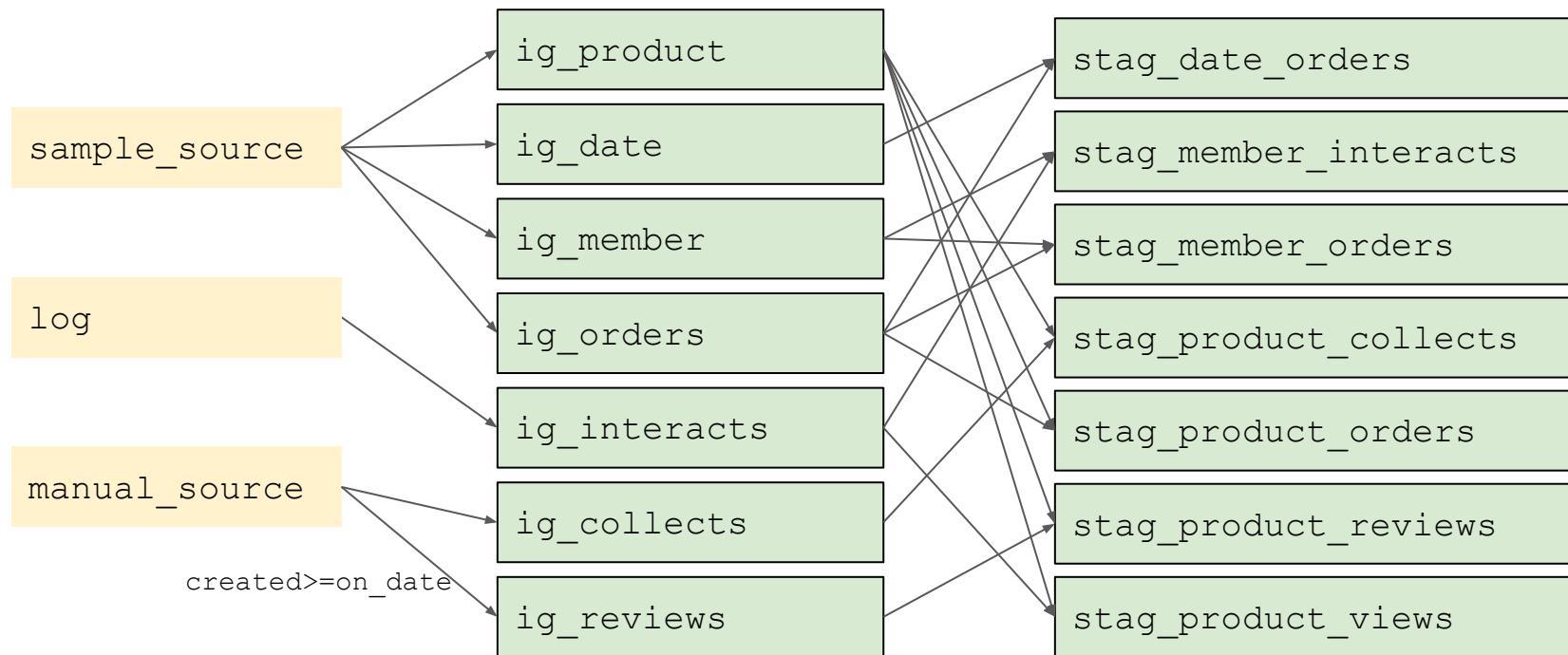
stag_product_collects

stag_product_orders

stag_product_reviews

stag_product_views

DAG - Graph



DAG - Task Dependency Graph

```
ig_product >> [
    stag_product_collects,
    stag_product_orders,
    stag_product_reviews,
    stag_product_views
]

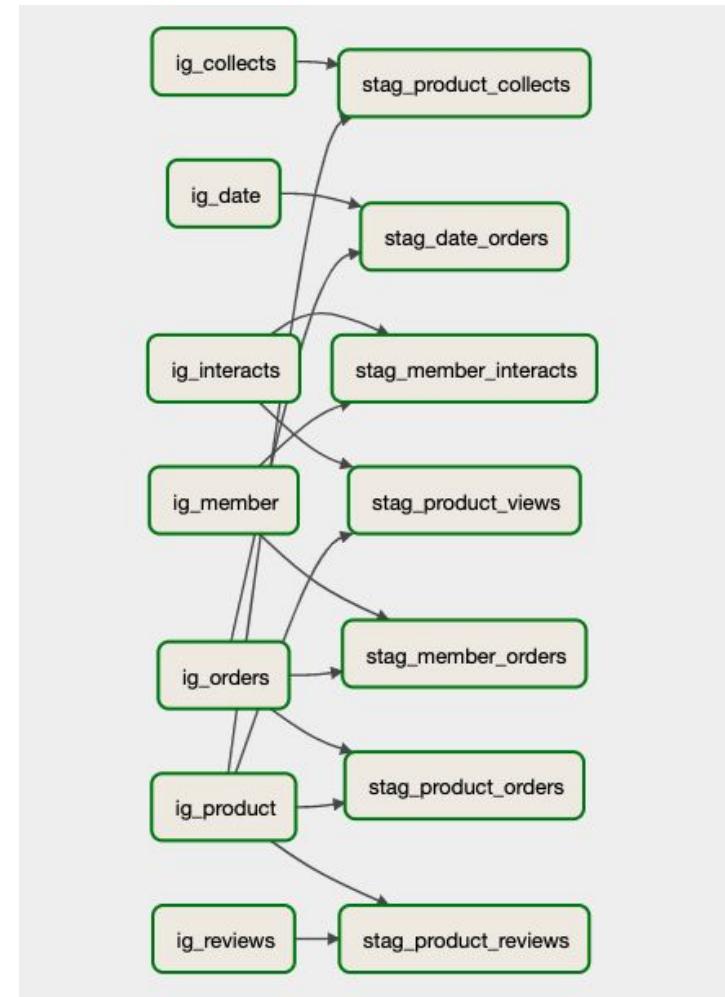
ig_date >> stag_date_orders
ig_member >> [
    stag_member_interacts,
    stag_member_orders
]

ig_orders >> [
    stag_date_orders,
    stag_member_orders,
    stag_product_orders
]

ig_interacts >> [
    stag_member_interacts,
    stag_product_views
]

ig_collects >> stag_product_collects
ig_reviews >> stag_product_reviews
```

dags.py



DAG - Operators

- Operators: 定義 Task 的工作內容,
推薦使用 BashOperator
 - 降低 python package 的 dependency

```
tasks = [
    'ig_product',
    'ig_date',
    'ig_member',
    'ig_orders',
    'ig_interacts',
    'ig_collects',
    'ig_reviews',
    'stag_date_orders',
    'stag_member_interacts',
    'stag_member_orders',
    'stag_product_collects',
    'stag_product_orders',
    'stag_product_reviews',
    'stag_product_views',
]
for task_name in tasks:
    op = BashOperator(
        task_id=task_name,
        bash_command=f'cd {dags_folders} ; python -m do_task {task_name} ' + "{{execution_date}}",
    )
    globals()[task_name] = op
```

dags.py

```
import click
from pipeline.utils import conn

@click.group(name='do_task')
@click.help_option('-h', '--help')
def do_task_cli():
    pass

@do_task_cli.command(name='ig_orders')
@click.argument('on_date', metavar='DATE')
@click.help_option('-h', '--help')
def ig_orders(on_date):
    from pipeline.tasks import ig_orders
    results = ig_orders.get_data(on_date='')
    table_name = 'ig_orders'
    conn.overwrite_dw(results, table_name)

@do_task_cli.command(name='ig_collects')
@click.argument('on_date', metavar='DATE')
@click.help_option('-h', '--help')
def ig_collects(on_date):
    on_date = pdl.parse(on_date, tz='Asia/Taipei').date()
    from pipeline.tasks import ig_collects
    results = ig_collects.get_data(on_date)
    table_name = 'ig_collects'
    conn.clear_dw_by_date(on_date, table_name)
    conn.insert_dw(results, table_name)

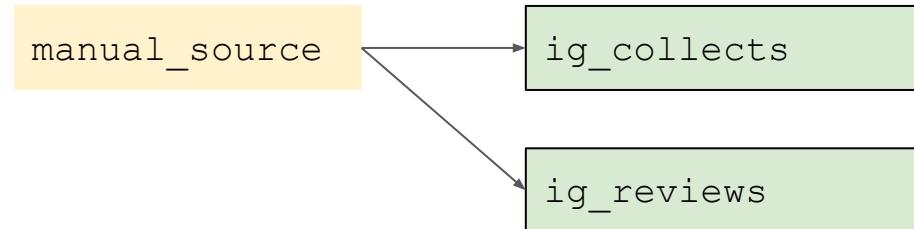
@do_task_cli.command(name='stag_date_orders')
@click.argument('on_date', metavar='DATE')
@click.help_option('-h', '--help')
def stag_date_orders(on_date):
    from pipeline.tasks import stag_date_orders
    results = stag_date_orders.get_data(on_date='')
    table_name = 'stag_date_orders'
    conn.overwrite_dw(results, table_name)

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

Airflow 重要觀念

執行最新的完整區段

- DAG:schedule_interval
- DAG:start_date
 - 第一次執行是在 start_date + schedule_interval 的時候, 執行的是 start_date 的資料
- task:execution_date
 - eg. :execution_date =2022-06-11, schedule_interval 是每月 1 號的時候, 會是拿 2022-05-01 的資料來做處理 (第一次執行是 2022-06-01)



使用者足跡 - json:key-value 記錄共同、特有內容

```
✉ [  
  ↳ {  
    "user_props": {  
      "device": "iphone",  
      "token": "tokenA",  
      "member_sk": 2,  
      "browser": "chrome"  
    },  
    "events": [5]  
  },  
  ↳ {  
    "user_props": {  
      "device": "web",  
      "token": "tokenB",  
      "browser": "safari"  
    },  
    "events": [1]  
  }  
]
```

```
✉ {  
  ↳ {  
    "event_timestamp": "2022-05-28 12:00:01",  
    "action": "view",  
    "entity": "home",  
    "entity_sk": "home",  
    "http_url": "url1",  
    "http_refer": "url_google",  
    "view_sk": 1,  
    "interact_sk": 1  
  },  
  ↳ {  
    "event_timestamp": "2022-05-28 12:00:07",  
    "action": "view",  
    "entity": "search",  
    "entity_sk": "DE book",  
    "http_url": "url2",  
    "http_refer": "url1",  
    "view_sk": 2,  
    "ref_view_sk": 1,  
    "interact_sk": 2  
  },  
  ↳ {  
    "event_timestamp": "2022-05-28 12:00:08",  
    "action": "impress",  
    "entity": "product",  
    "entity_sk": "3",  
    "position": 1,  
    "view_sk": 2,  
    "interact_sk": 3  
  }  
}
```

Pipeline 實作 & 分組

- 所有人
 - 網頁互動: ig_interacts
- 個人: 分別完成自己負責的 tasks (其他的都幫你寫完了)
 - ig_* : 跟上午的大同小異
 - stag_* : 統計資料 (無統計值可不處理)
- 實作說明

實作時間

Backfill

把過去的資料回填

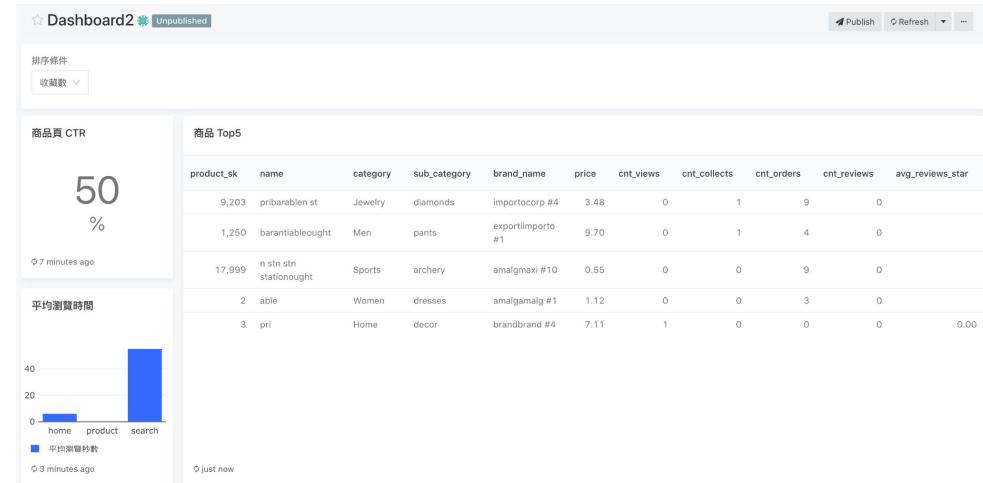
- 可直接在開發環境手動處理
- 或利用 Airflow, 把 DAG `start_date` 設在過去
 - 但如果讀取原始資料的 transform 流程很複雜、耗時很久, 新邏輯很單純, 還是會走手動處理

```
from pipeline.tasks import ig_collects
results = ig_collects.get_data(on_date='2020-01-01')
table_name = 'ig_collects'
conn.overwrite_dw(results, table_name)
```

實作成果 - Data Mart / BI 應用

- stag
 - stag_views: 補上網頁瀏覽時間
- BI
 - bi_product_status: 商品瀏覽、收藏、購買、評價 (stag)
 - view_product_ctr: 商品轉換率 (ig)

Airflow 實作成果應用



Sources

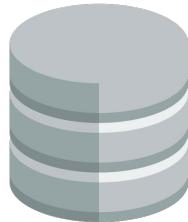
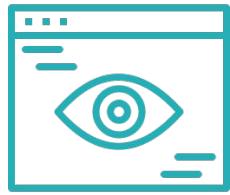
Ingestion and Transformation

Storage

Historical

Predictive

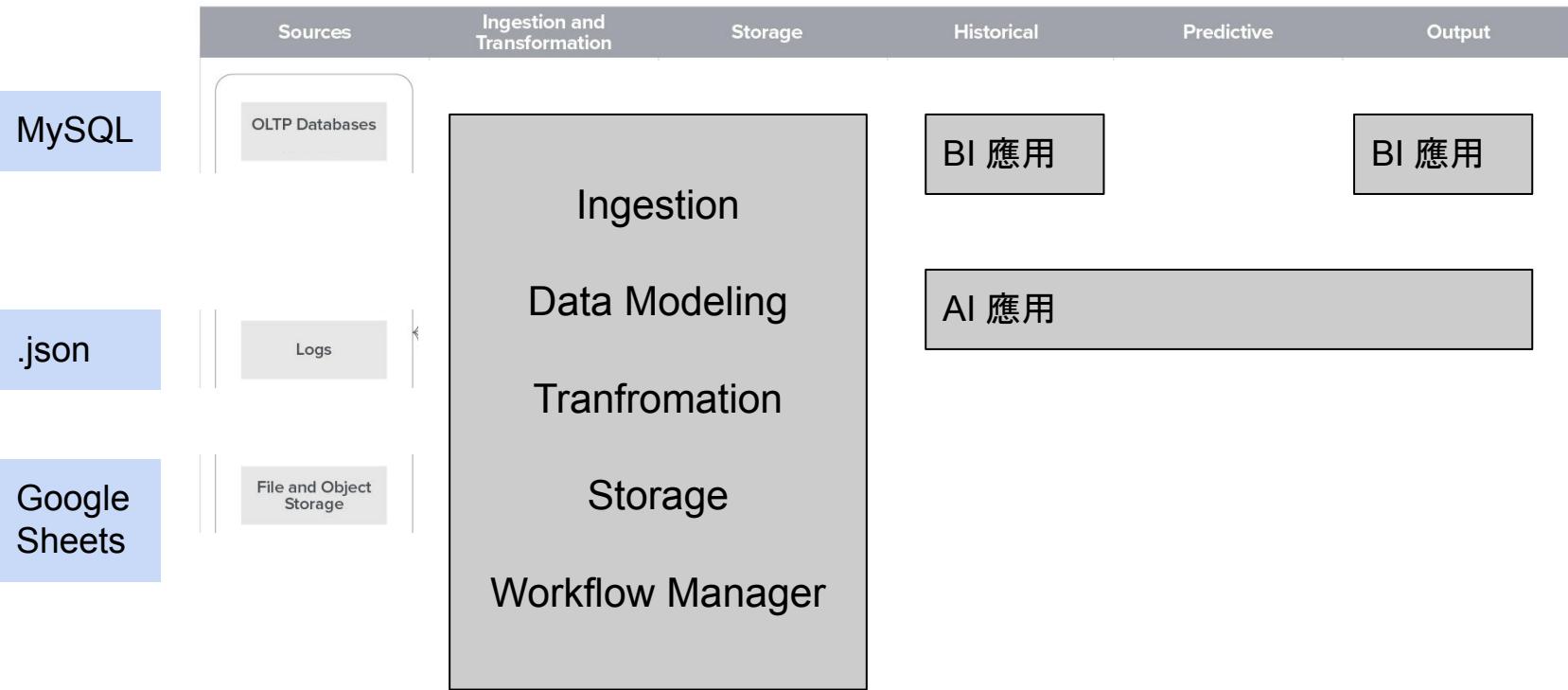
Output

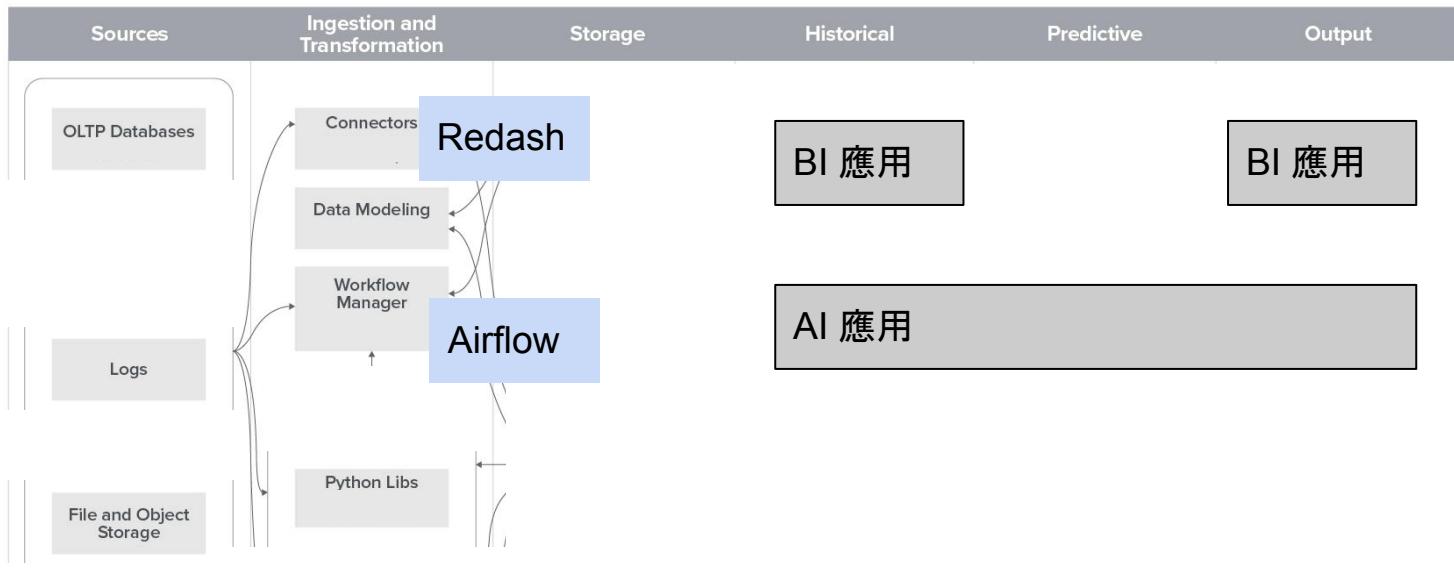


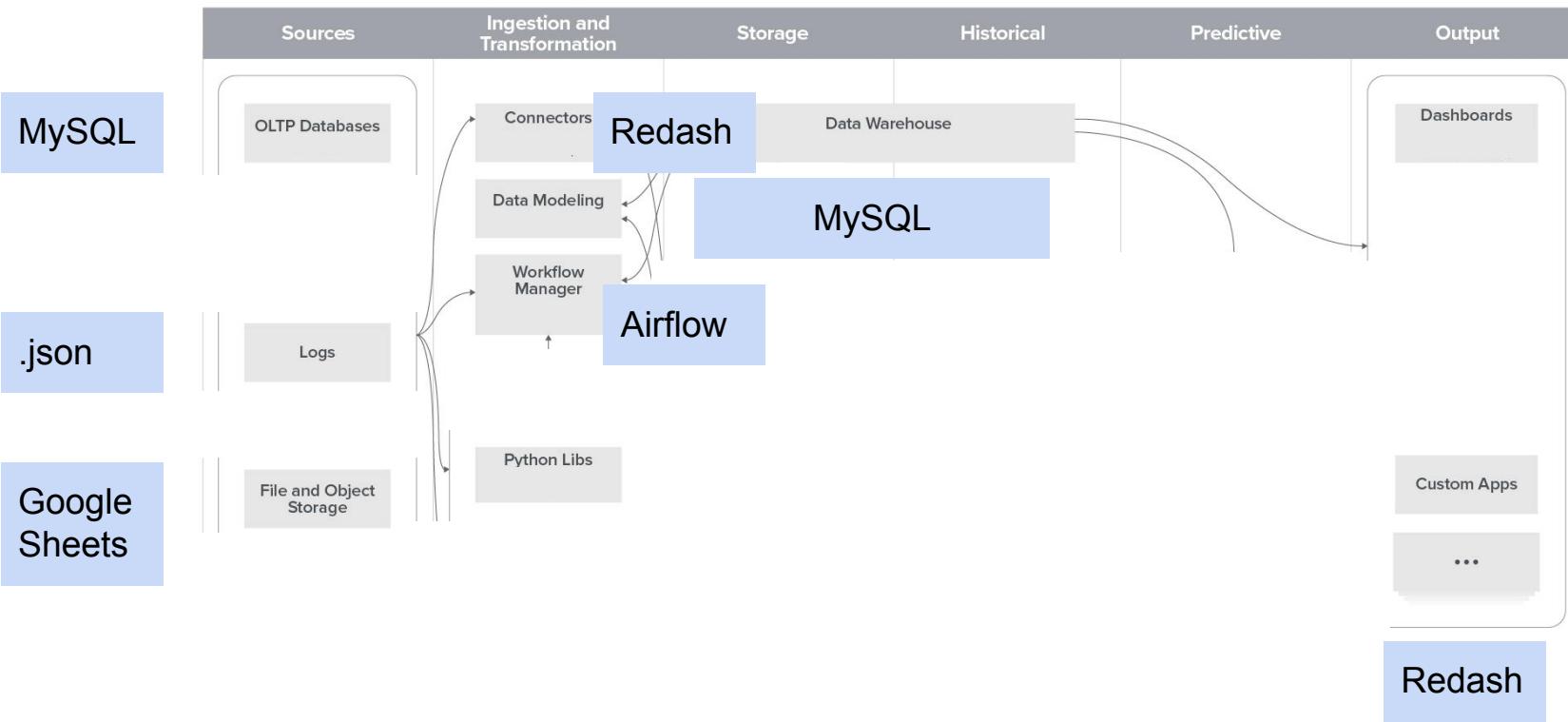
BI 應用

BI 應用

AI 應用





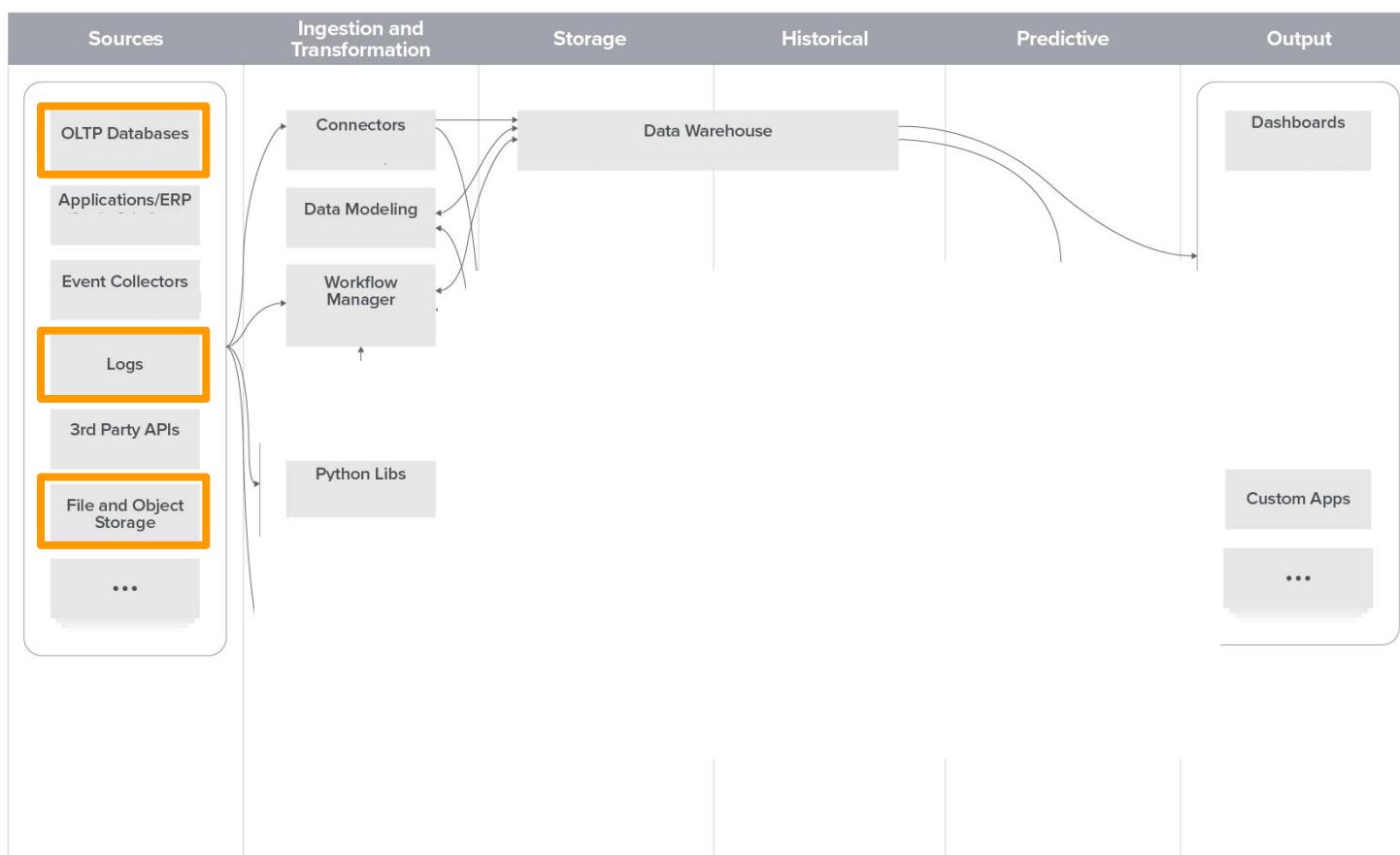


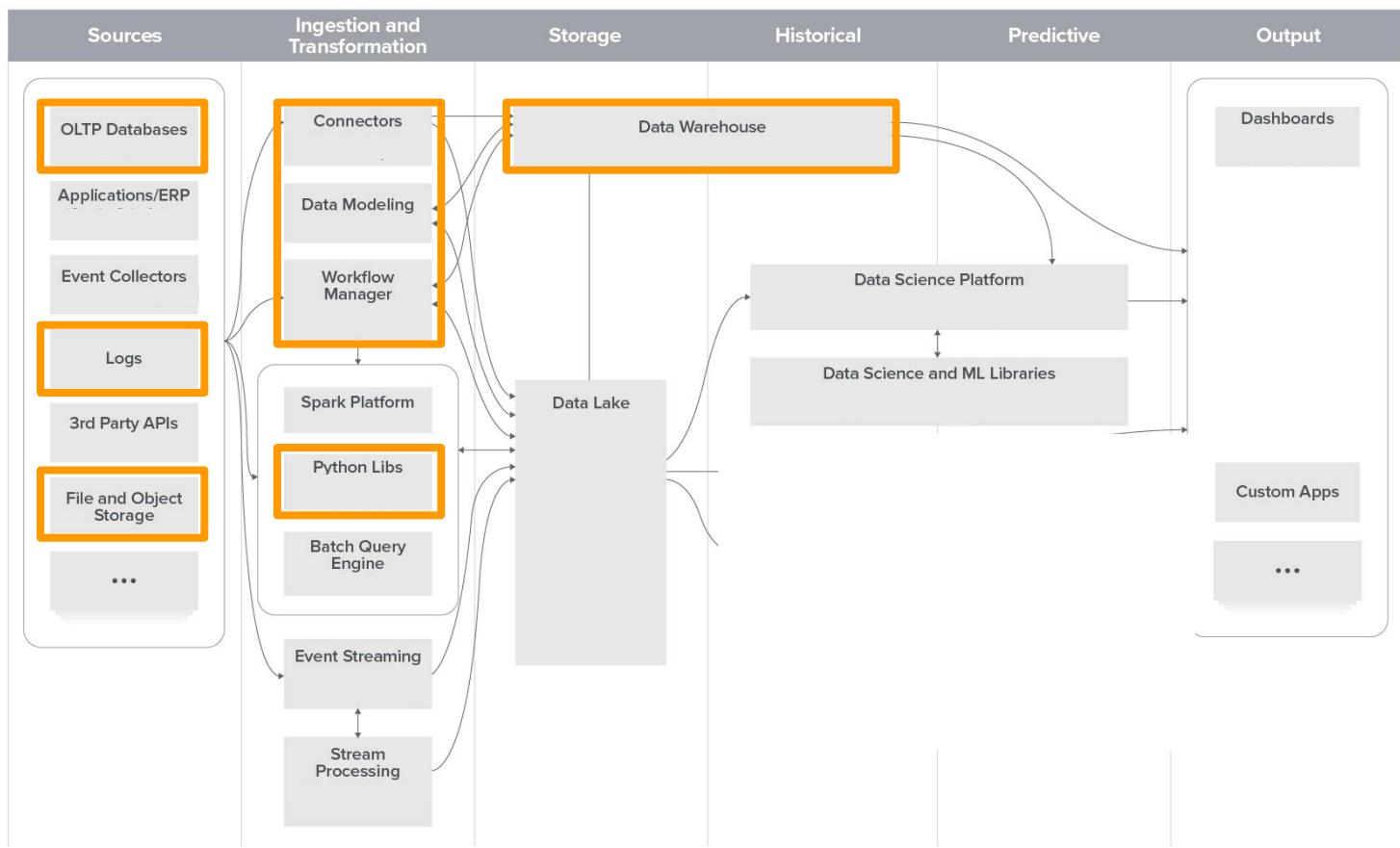
A photograph of a long, narrow concrete pier or breakwater extending from the bottom left towards the horizon. The pier is dark grey and shows signs of weathering and small white spots. The ocean is a calm, light blue. In the distance, a small, flat island or rock formation is visible on the right. The sky is a clear, pale blue with a few wispy white clouds.

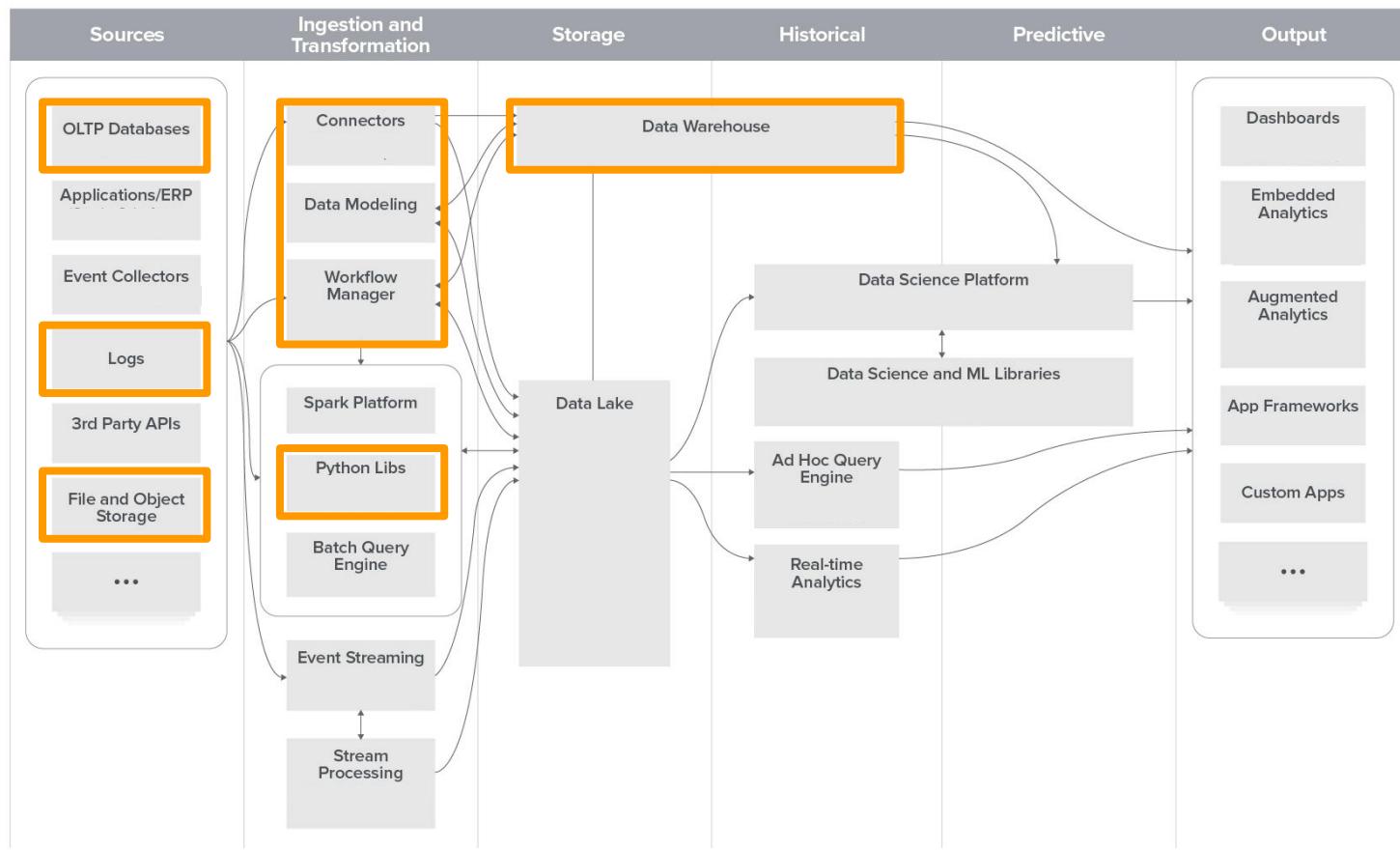
The End?

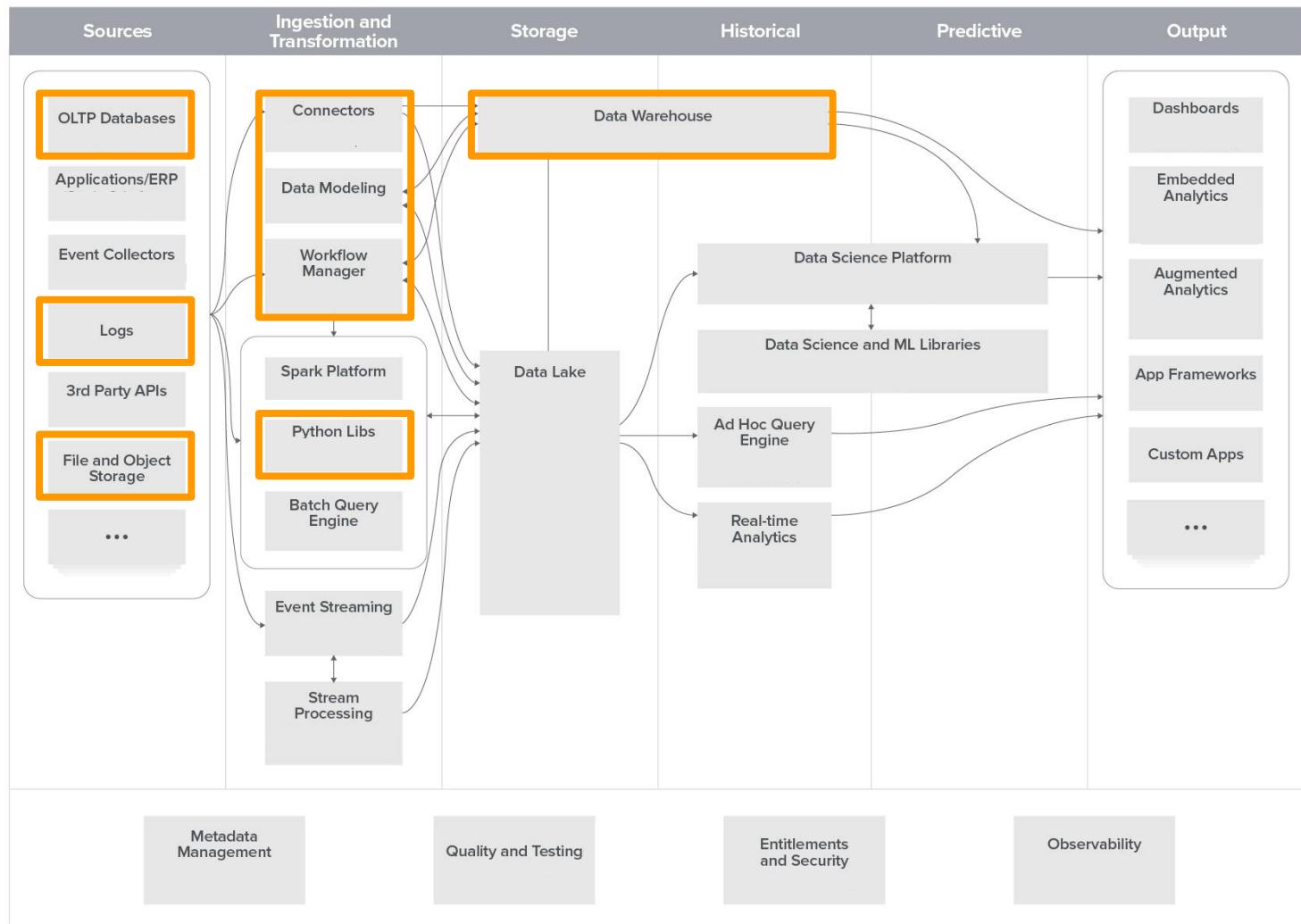
Modern Data Infrastructure/Stack

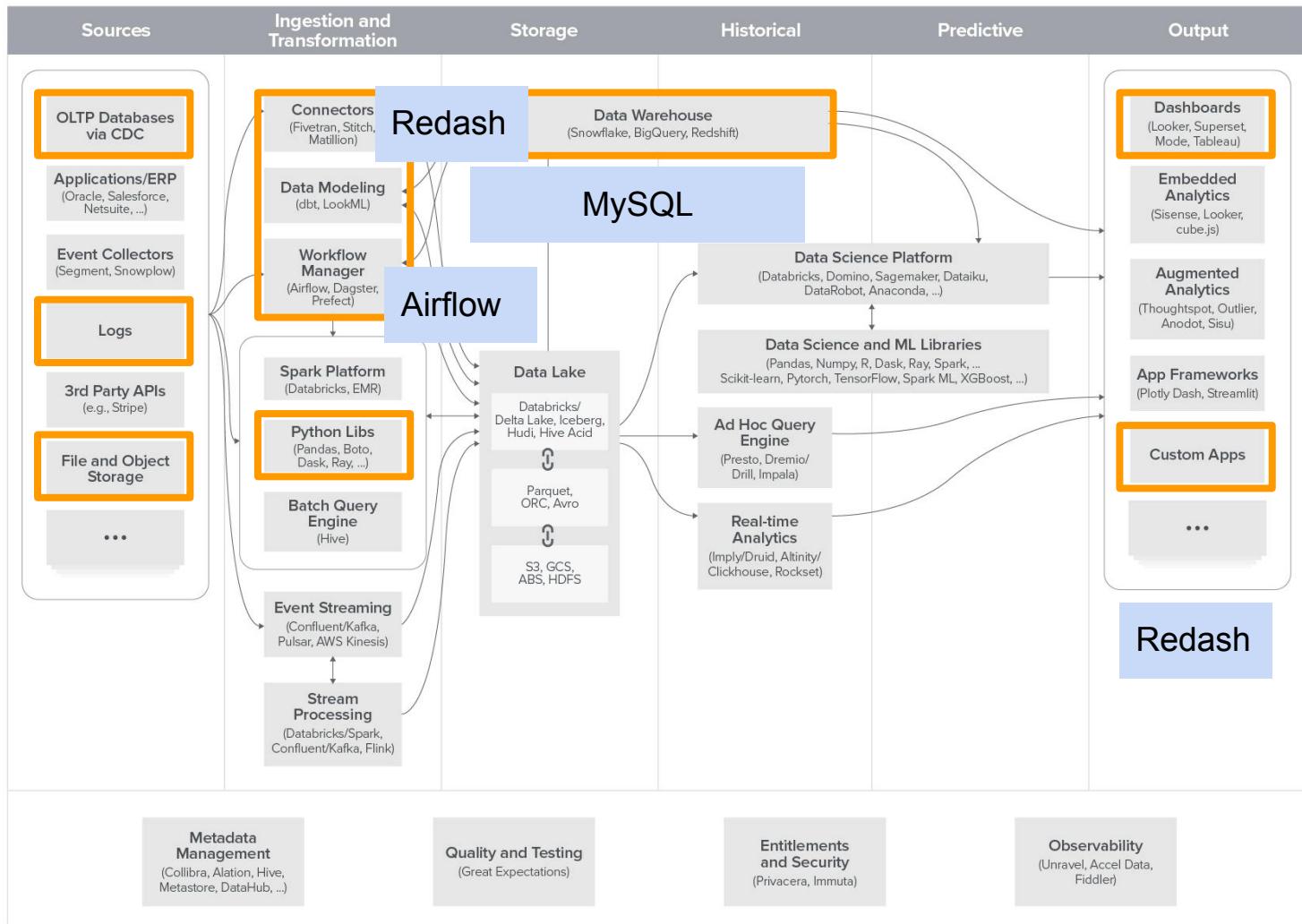
- 1970s Bill Inmon 提出 Data Warehouse
- 1996 Ralph Kimball 建立 Dimensional Modeling 方法
- 2010 Spark open sourced
- 2011 BigQuery released
- 2016 Airflow released



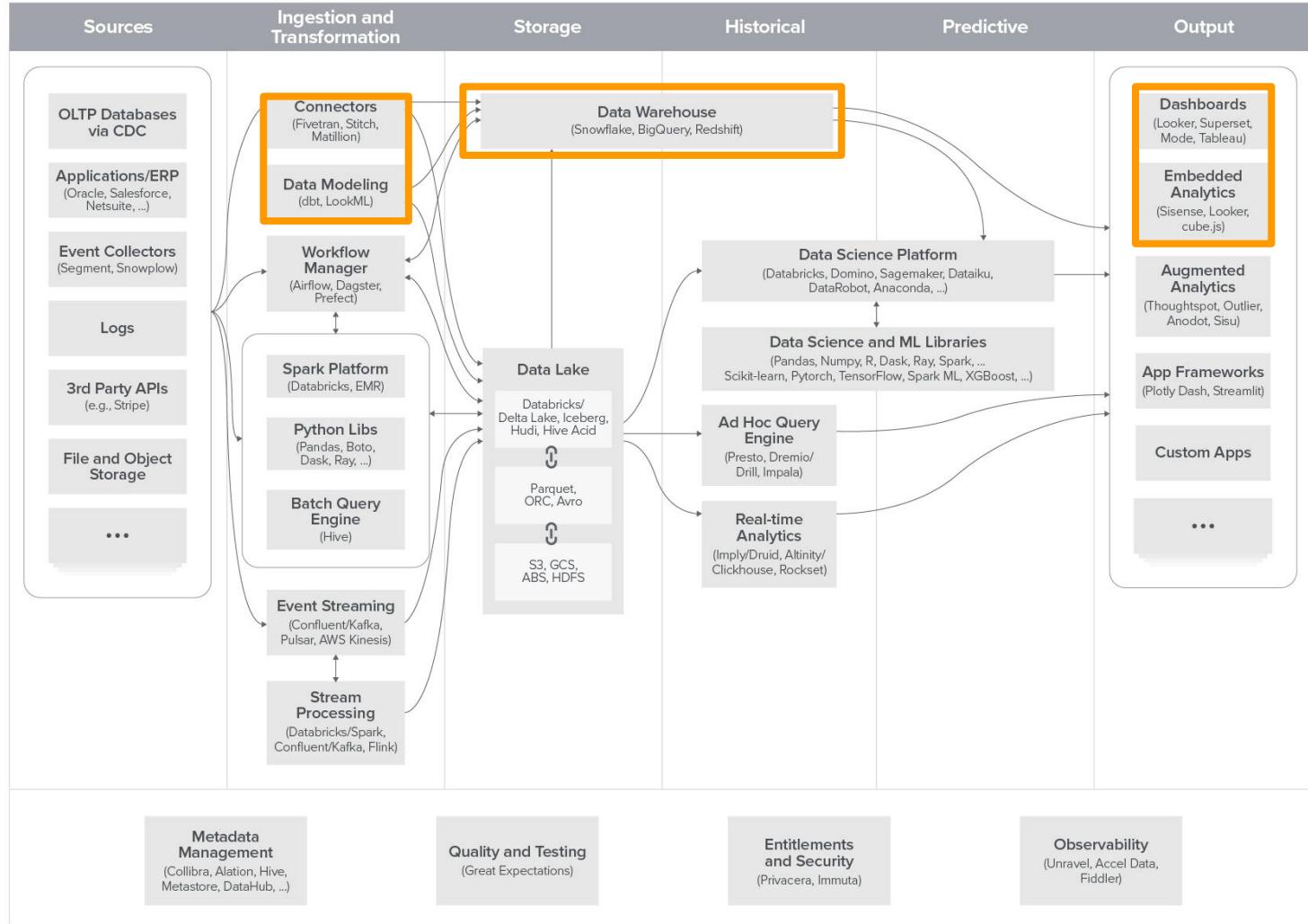




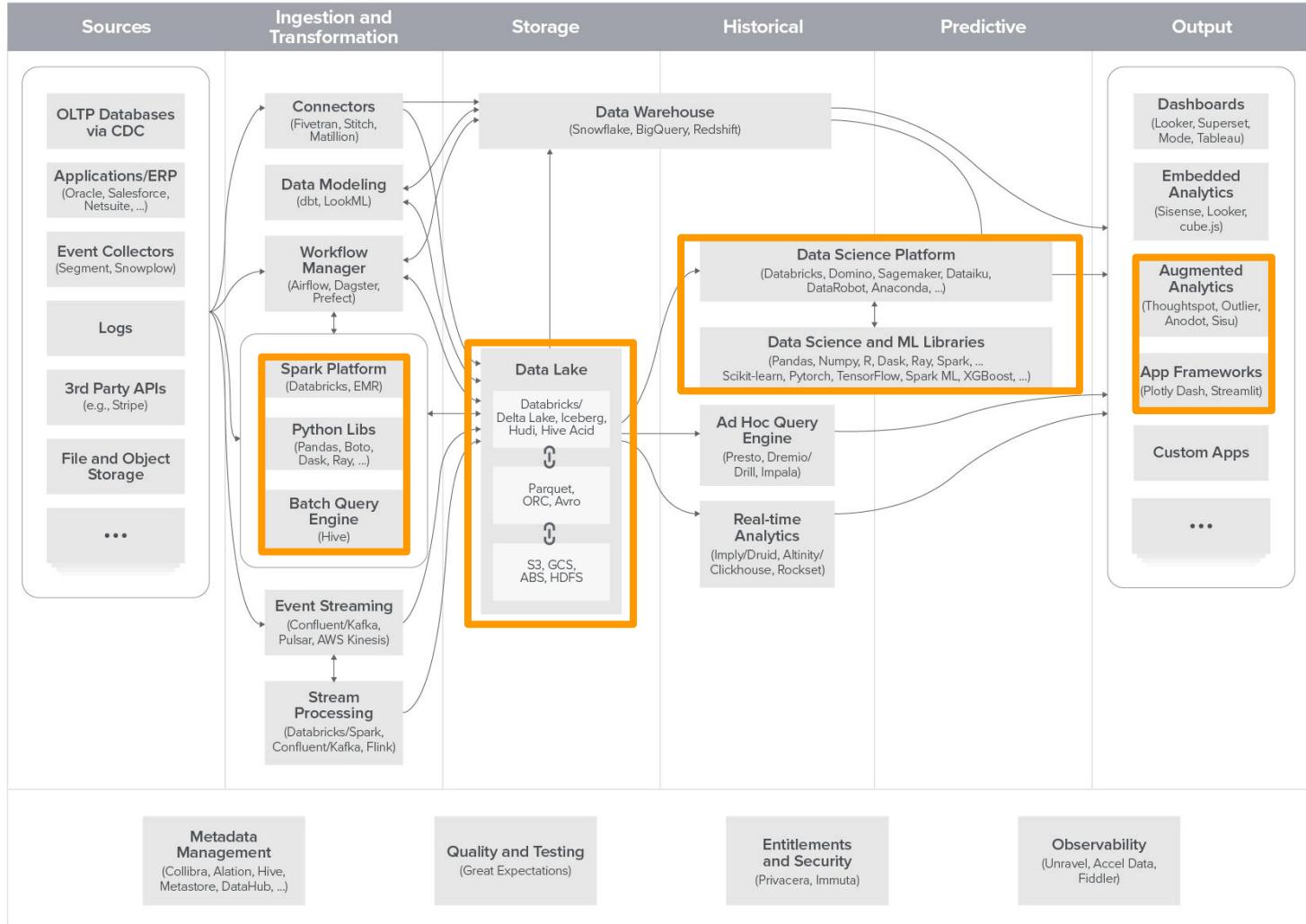




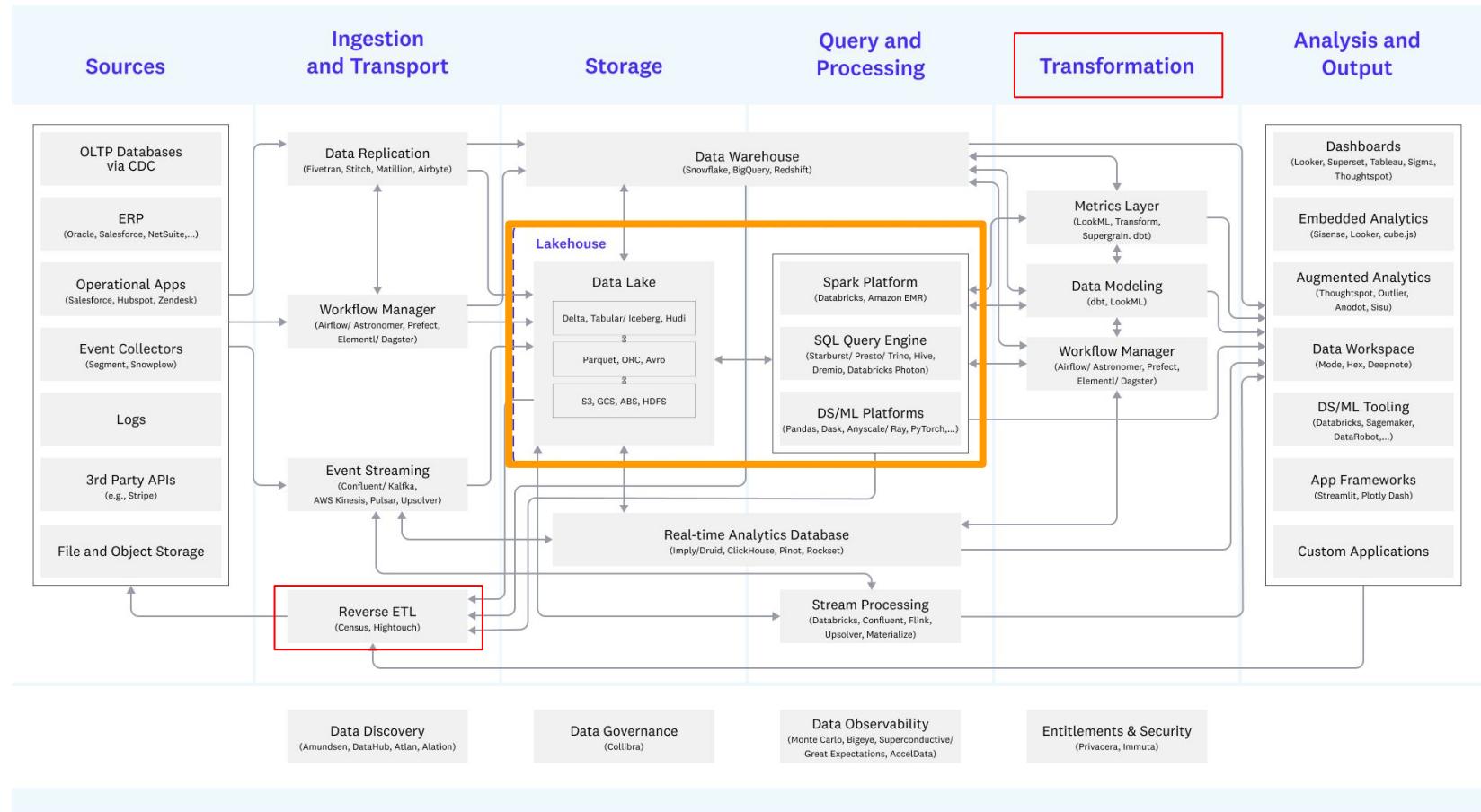
BI
2020



AI 2020



2022



ETL vs. ELT vs. EtLT

- Extract: 將 Source 資料拉下
- Load: 將資料存到 Storage
- Transform: 將資料依照需求(商業邏輯) 轉換 (加工、組合)
- t: 簡易版的 T, 延伸 E 的工作, 多是 cleaning
 - 刪除重複數據
 - 對隱私資訊做加工
 - 把 URL 參數拆開
 - ...

對應到架構圖的「Ingestion (E)」、「Transport (L)」、「Transformation (T)」

Data Lakehouse (2019~2020)

一套系統同時讓 AI/BI 都能有效率的使用，結合傳統 Data Warehouse (ACID, update/delete 快速) 與 Data Lake (Unstructured) 的好處

- [Apache Hudi](#): Uber, Alibaba Cloud
- [Apache Iceberg](#): Netflix, Tabular, Starburst
- [Delta Lake](#): Databricks
- AWS
- Dremio

延伸閱讀：[Hudi, Iceberg, Delta Lake 比較](#)

Data Discovery

- Data Lineage
- Metadata

相關系統: [DataHub](#)



About

No documentation yet. Share your knowledge by adding documentation and links to helpful resources.

[Add Documentation](#)

[Add Link](#)

Tags

No tags added yet. Tag entities to help make them more discoverable and call out their most important attributes.

[Add Tags](#)

Glossary Terms

No terms added yet. Apply glossary terms to entities to classify their data.

[Add Terms](#)

Owners

No owners added yet. Adding owners helps you keep track of who is responsible for this data.

[Add Owners](#)

補充閱讀：數據中臺 (2018)、Data Mesh (2019)

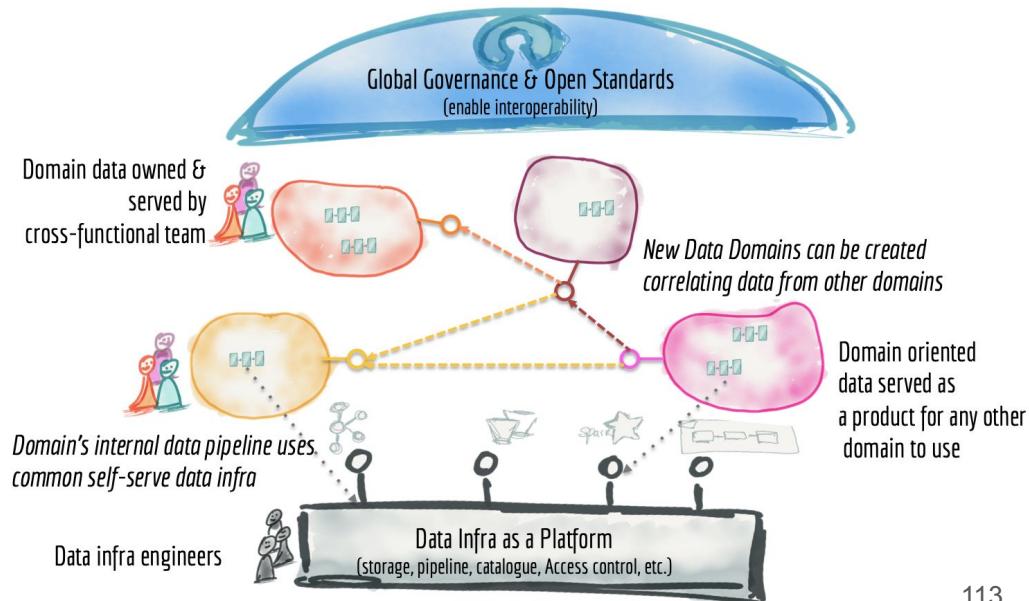
2018 Alibaba 倡導 數據中臺戰略：
能在前台(創新)、後台(穩定)間的整合系統

2019 Zhamak Dehghani Data Mesh 概念：
靠統一的標準去合作

- Distributed
- Self-served
- Data as Product

延伸 :*Data contracts* 是標準實踐概念

以上是 Data Democratization 的實踐方式



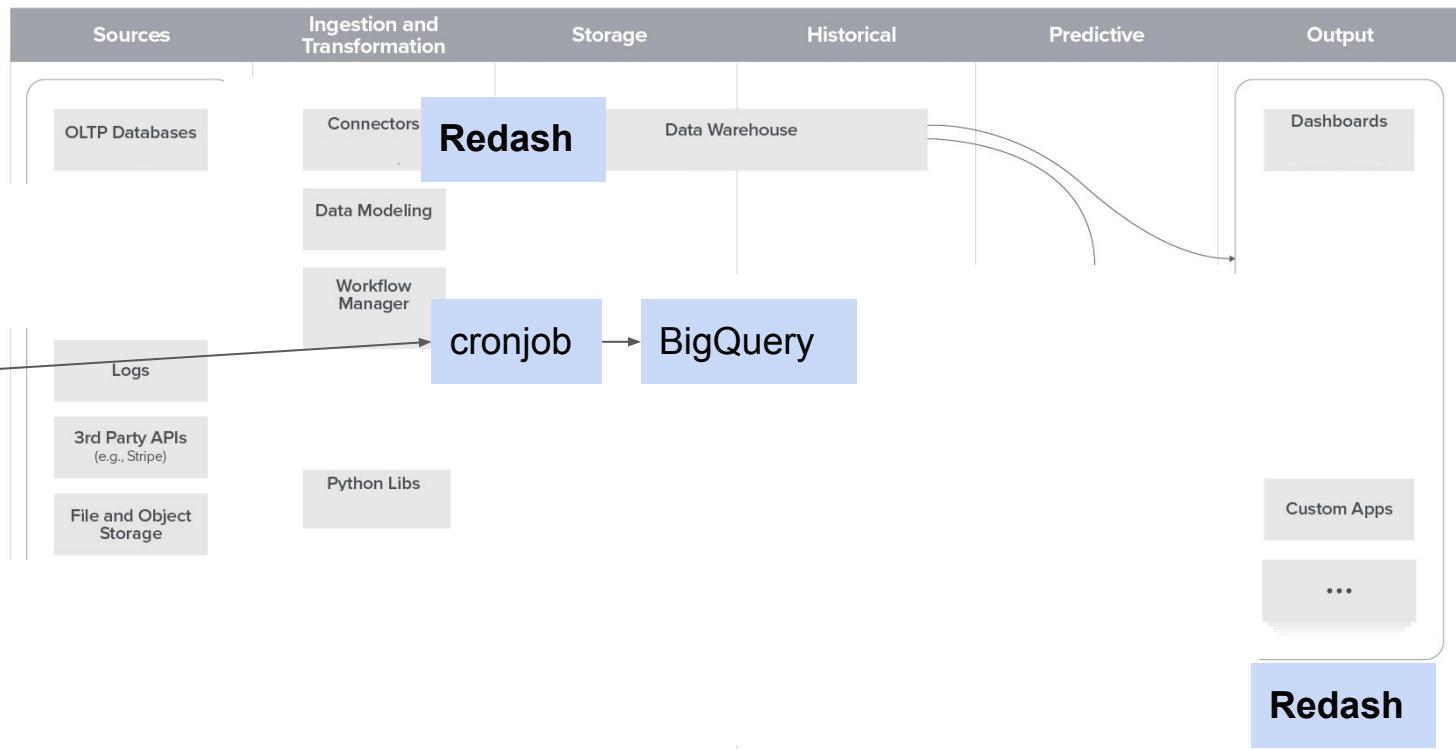
補充閱讀 : reverse ETL

通常我們會把各種資料收集到 Storage,

但現在 Sources 的系統功能越來越強大, 有辦法讓我們反向匯入整理好的「資訊」
這也有賴於現在做 ETL 的工具越來越方便 (eg. [dbt](#))

常見情境：

- 客服
- 投放廣告



電商 1: 資料量不大, 無專職 DE, 無 DA/ML, 公司人數少



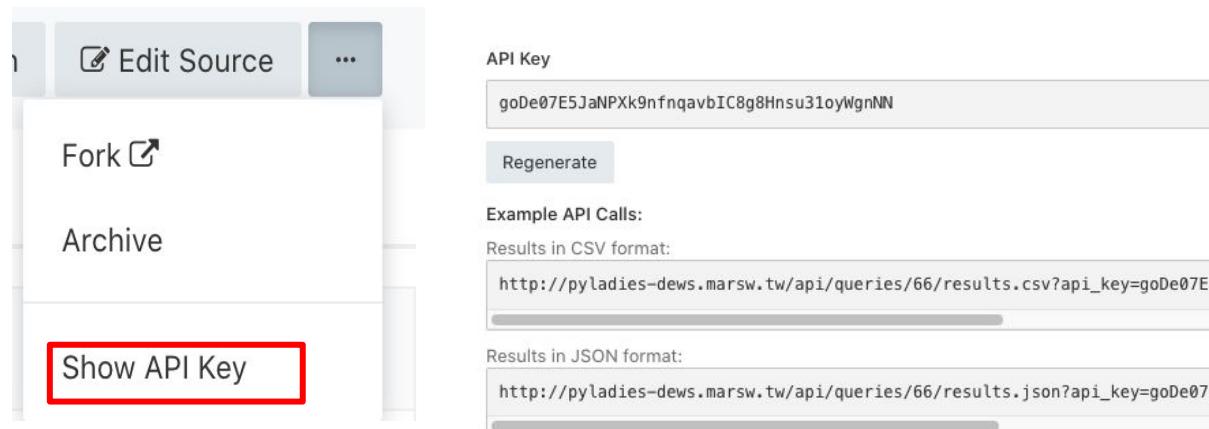
Redash - Share / Download

- 自行驗證資料
- 直接將圖表嵌入到其他產品中
- 把商業邏輯當成 API 接口
 - 在 server 裡用高權限帳號 refresh & get data

created	is_holiday	product_price	product_cost
2000-05-01	0	639,273.44	653,633.38
2000-05-02	0	524,568.07	548,658.38
2000-05-03	0	866,085.61	875,149.72
2000-05-04	0	638,003.73	
2000-05-05	0	377,383.49	
2000-05-06	0	742,897.28	

Edit Visualization

153 rows 2 seconds runtime



The screenshot shows the Redash interface for managing an API key. On the left, there's a sidebar with buttons for 'Edit Source' (highlighted), 'Fork', 'Archive', and a red-bordered 'Show API Key'. The main area displays the API key value 'goDe07E5JaNPXk9nfqavbIC8g8Hnsu31oyWgnNN', a 'Regenerate' button, and examples of API calls. It also shows results in CSV and JSON formats.

API Key

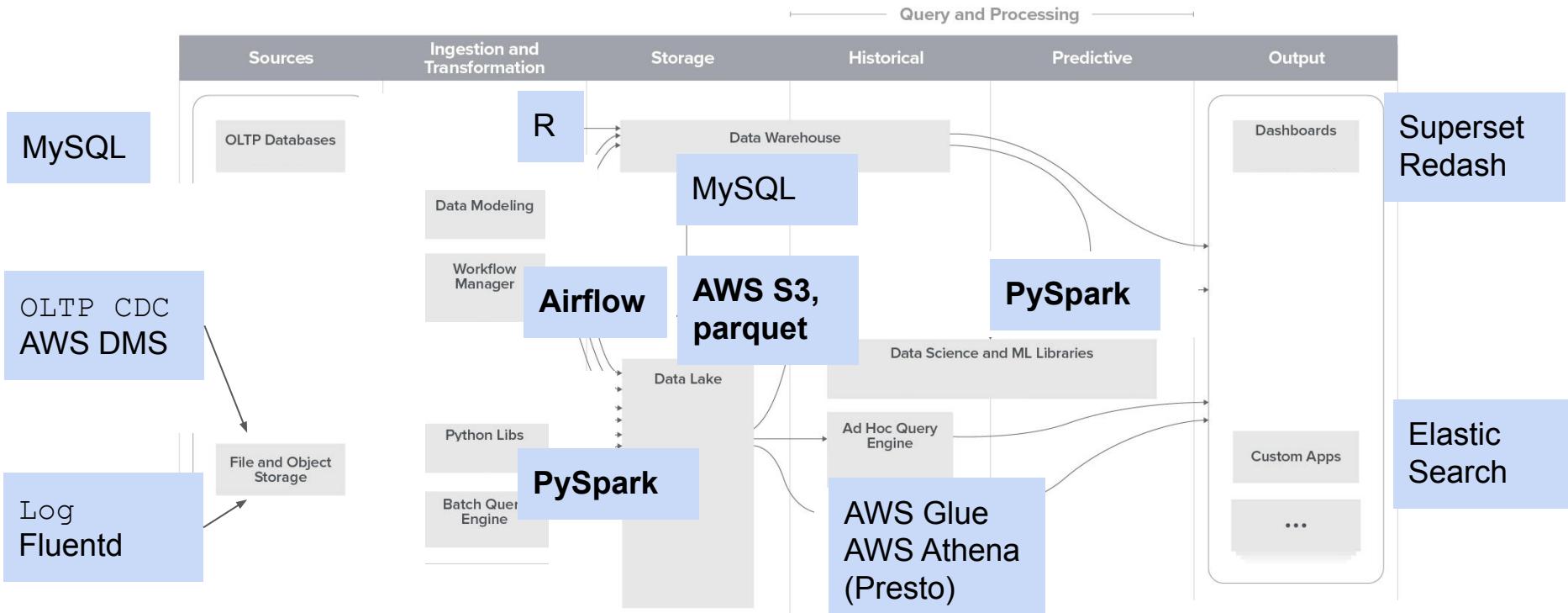
goDe07E5JaNPXk9nfqavbIC8g8Hnsu31oyWgnNN

Regenerate

Example API Calls:

Results in CSV format:
http://pyladies-dews.marsw.tw/api/queries/66/results.csv?api_key=goDe07E5J

Results in JSON format:
http://pyladies-dews.marsw.tw/api/queries/66/results.json?api_key=goDe07E5



電商 2: 資料量稍大, 稍缺 DE 資源, 有 DA/ML, 公司人數多

html files from
codebase comment

Metadata
Management

補充閱讀：其他公司架構

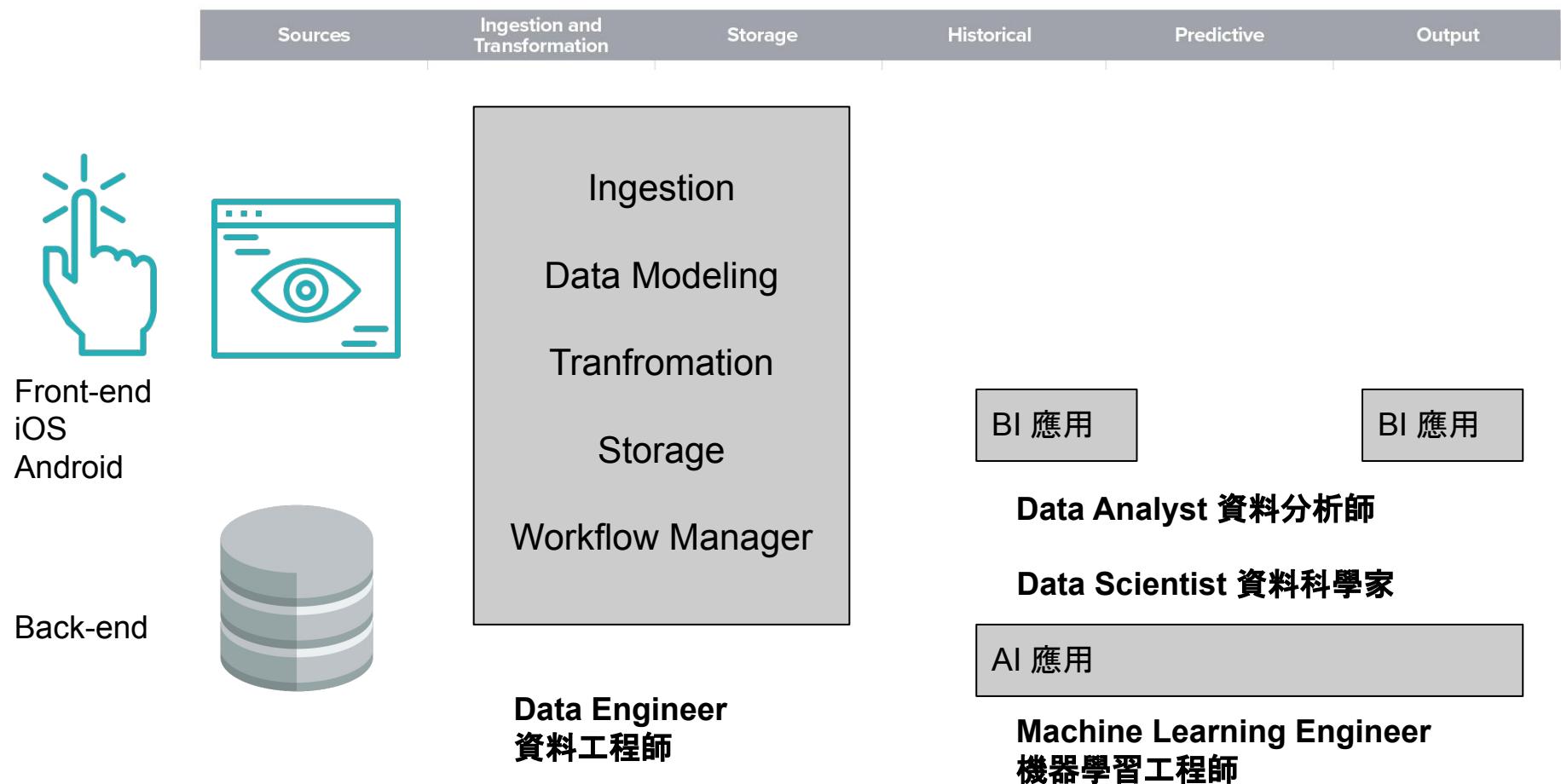
- Airbnb
 - <https://medium.com/airbnb-engineering>
 - <https://databricks.com/session/airstream-spark-streaming-at-airbnb>
 - <https://databricks.com/session/building-data-product-based-on-apache-spark-at-airbnb>
- 阿里巴巴
 - <https://www.alibabacloud.com/blog/593988>
- Netflix
 - <https://netflixtechblog.com/1a52526a7977>
- WePay
 - <https://www.infoq.com/articles/future-data-engineering-riccomini/>
 - <https://www.youtube.com/watch?v=ZZr9oE4Oa5U>
- Others
 - <https://github.com/andkret/Cookbook/blob/master/sections/05-CaseStudies.md>

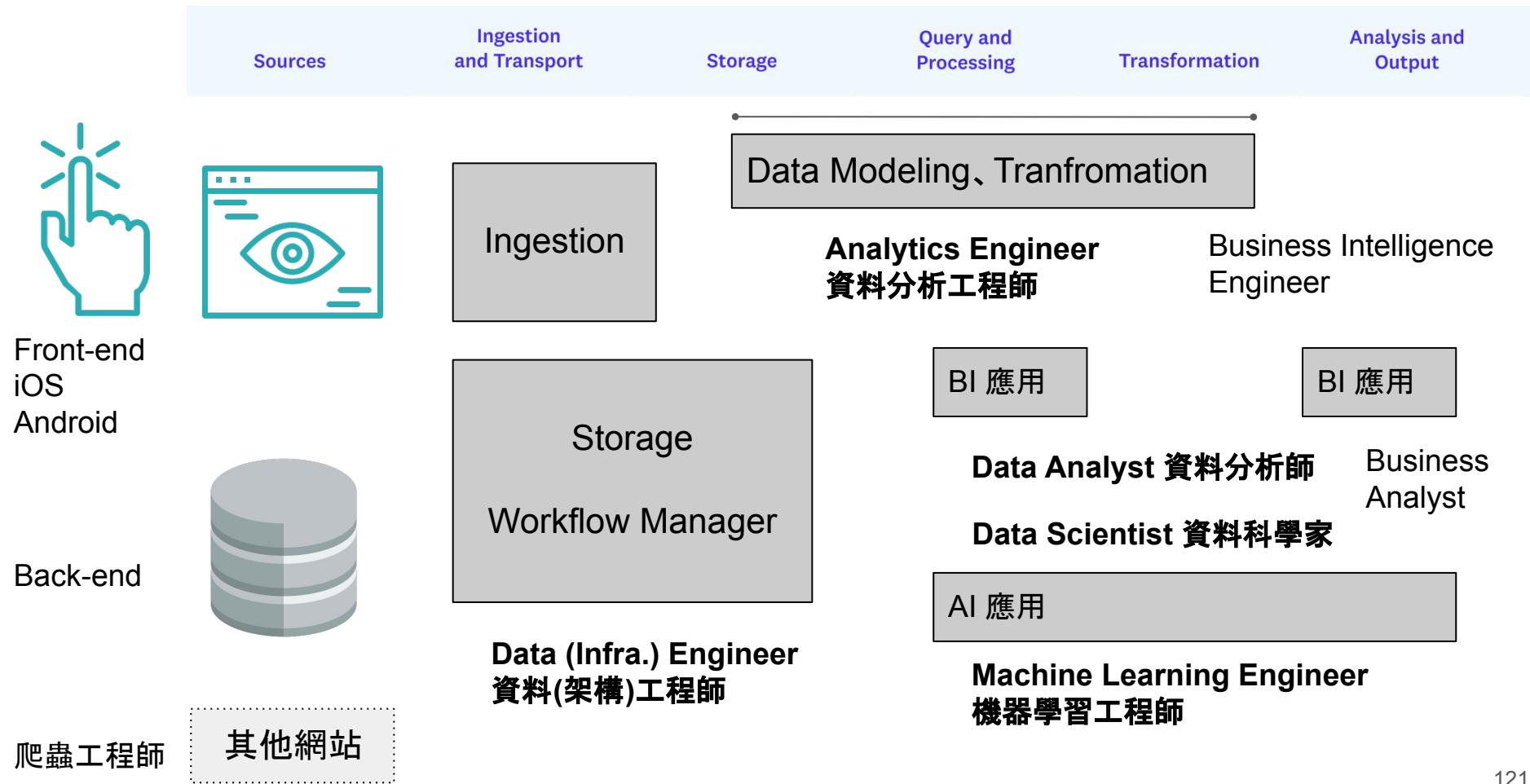
資料工程師又切分得更細了

- 讓所有人都能有效率的使用資料

以用水比喻

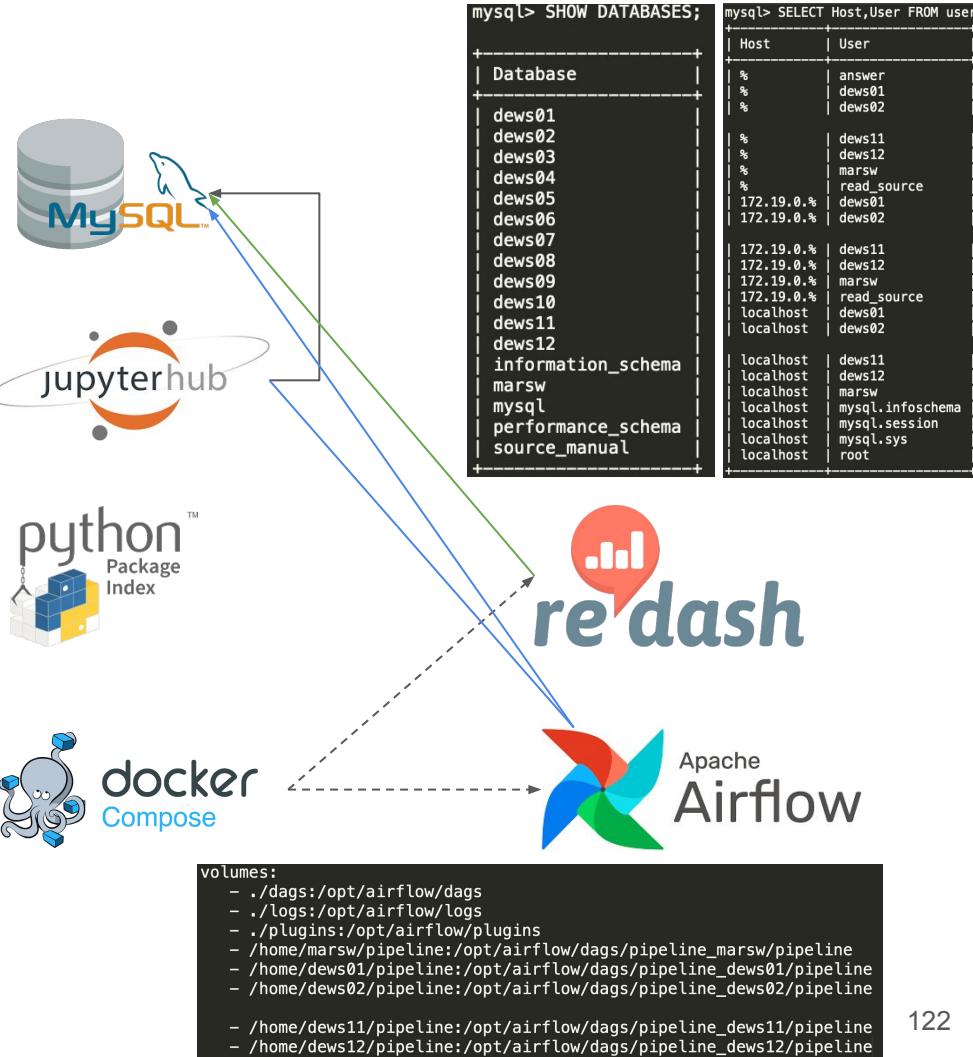
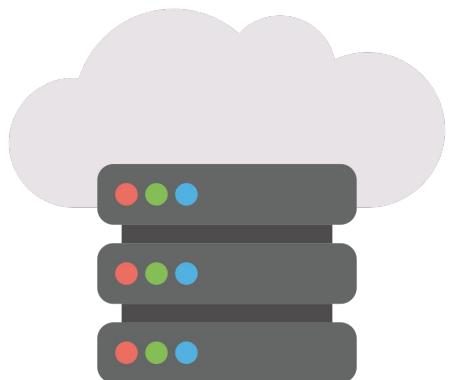
- 規劃水庫位置
- 建立淨水廠
 - 現在可能只建立，但不負責執行淨水過程
- ~~淨水場到每個區域的水管管線~~ 改由 下游使用者 (AE/DA/DS/MLE) 自行處理
- ~~每個用戶家裡個水龍頭的管線~~





資料工程這條路

- conn.py
 - 可以輕鬆存取指定的 Database
 - 可以讀取 json



Resources

- 入門磚: 概覽 (知識+工具)
 - [datastacktv](#) (2021): 提供關鍵字
 - [Data Pipelines Pocket Reference](#) (2021)
 - [SeattleDataGuy](#) (2021)
- 資料工程知識
 - [Agile Data Warehouse Design](#) (2011)
 - Ralph Kimball [The Data Warehouse Toolkit](#) (2013)
 - [CAP 理論](#)
- 資料工程技能工具
 - [Data Engineering Essentials using SQL, Python, and PySpark](#)
 - [PySpark](#)
 - [\[BI工具\] 以Redash為資料視覺化方案之選擇與實踐系列](#)
 - [一段 Airflow 與資料工程的故事](#)
- 綜合知識+技能工具
 - [CS50](#)
 - [鳥哥的私房菜:Linux](#)
 - [演算法](#)

Feeds 蓋樓

- 社群
- blog / medium
- 線上學習資源

Recap

- 資料團隊的角色分工 (古今)
- Data Infrastructure (古今)
- Modelstorming : 以電商資料為例, 討論 & 設計 Data Warehouse
- 實作體驗 : Data Warehouse、Pipeline
 - SQL
 - Redash
 - Python
 - Airflow
- 資料工程這條路

工程師時期
爬蟲 單機→分散



管理時期 跨團隊溝通、BI、Tracking Log



工程師時期 資料工程 : PySpark、Airflow、Data Infra.



The background image shows a vibrant night scene of a city skyline. Two large, multi-colored fireworks displays burst in the upper left and lower left areas of the frame. In the foreground, numerous smaller lights from buildings and streetlights create a dense grid pattern. A prominent skyscraper with a distinctive purple and white illuminated facade stands on the right side. The overall atmosphere is festive and celebratory.

Thanks for listening