4. Data Engineering 101

Very basic. For details, take a database class!

Data engineering 101

- Data sources
- Data formats
- Data models
- Data storage engines & processing

Data sources

- User generated
- Systems generated
- Internal databases: users, inventory, customer relationships
- Third-party data

Data sources

Users generated data	Systems generated data
User inputs	Logs, metadata, predictions
Easily mal-formatted	Easier to standardize
Need to be processed ASAP	OK to process periodically (unless to detect problems ASAP)
	Can grow very large very quickly • Many tools to process & analyze logs: Logstash, DataDog, Logz, etc. • OK to delete when no longer useful

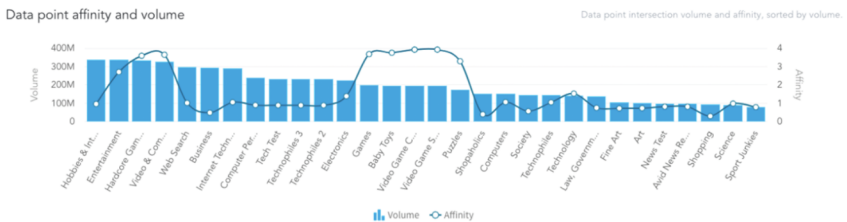
Users' behavioral data (clicks, time spent, etc.) is often system-generated but is considered **user data**

Third-party data: creepy but fascinating

- Types of data
  - social media, income, job
- Demographic group
  - men, age 25-34, work in tech
- More available with Mobile Advertiser ID
- Useful for learning features
  - people who like A also like B

Top interests

They love computing and electronic entertainment. If you want to reach players, try targeting at their top interests.



61 M profiles

Remote working

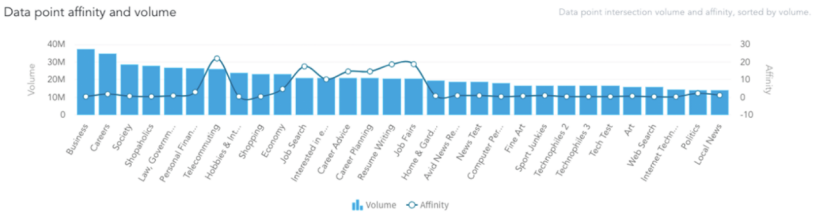
Millions of people decided to #stayhome and work remotely to limit the spread of coronavirus. Use our Remote working segment to easily reach them and show software or products that will help them stay effective.

How did we build the segment?

Our segment includes profiles of users who recently read articles, watched videos or used mobile apps which refers to:

- remote working
- effective ways of working from home
- tools for remote workers
- homeschooling and e-learning

If you want to reach remote workers, try to extend your target group by selecting the top interests, which include Telecommuting, Career Planing or Personal Finance.



How to store your data?

Storing your data is only interesting if you want to access it later

- Storing data: **ser**ialization
- Unloading data: **des**erialization

How to store your data?

Data formats are
agreed upon standards
to serialize your data so that
it can be transmitted & reconstructed later

Data formats: questions to consider

- How to store multimodal data?
  - `{'image': [[200,155,0], [255,255,255], ...], 'label': 'car', 'id': 1}`
- Access patterns
  - How frequently the data will be accessed?
- The hardware the data will be run on
  - Complex ML models on TPU/GPU/CPU

Data formats

Row-major

Column-major

Format	Binary/Text	Human-readable	Example use cases
JSON	Text	Yes	Everywhere
CSV	Text	Yes	Everywhere
Parquet	Binary	No	Hadoop, Amazon Redshift
Avro	Binary primary	No	Hadoop
Protobuf	Binary primary	No	Google, TensorFlow (TFRecord)
Pickle	Binary	No	Python, PyTorch serialization

Row-major vs. column-major

Column-major:

- stored and retrieved column-by-column
- good for accessing features

Row-major:

- stored and retrieved row-by-row
- good for accessing samples

	Column 1	Column 2	Column 3
Sample 1	...	...	...
Sample 2	...	...	...
Sample 3	...	...	...

Row-major vs. column-major: DataFrame vs. ndarray

Pandas DataFrame: column-major

- accessing a row much slower than accessing a column and NumPy

```
# Get the column `date`, 1000 loops
%timeit -n1000 df["Date"]

# Get the first row, 1000 loops
%timeit -n1000 df.iloc[0]
```

```
1.78 µs ± 167 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
145 µs ± 9.41 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

NumPy ndarray: row-major by default

- can specify to be column-based

```
df_np = df.to_numpy()
%timeit -n1000 df_np[0]
%timeit -n1000 df_np[:,0]
```

```
147 ns ± 1.54 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
204 ns ± 0.678 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```


https://github.com/chiphuyen/just-pandas-things

Text vs. binary formats

	Text files	**Binary files**
Examples	CSV, JSON	Parquet
Pros	Human readable	Compact
Store the number *1000000*?	7 characters -> 7 bytes	If stored as int32, only 4 bytes

You can unload the result of an Amazon Redshift query to your Amazon S3 data lake in Apache Parquet, an efficient open columnar storage format for analytics. Parquet format is up to 2x faster to unload and consumes up to 6x less storage in Amazon S3, compared with text formats. This enables you to save data transformation and enrichment you have done in

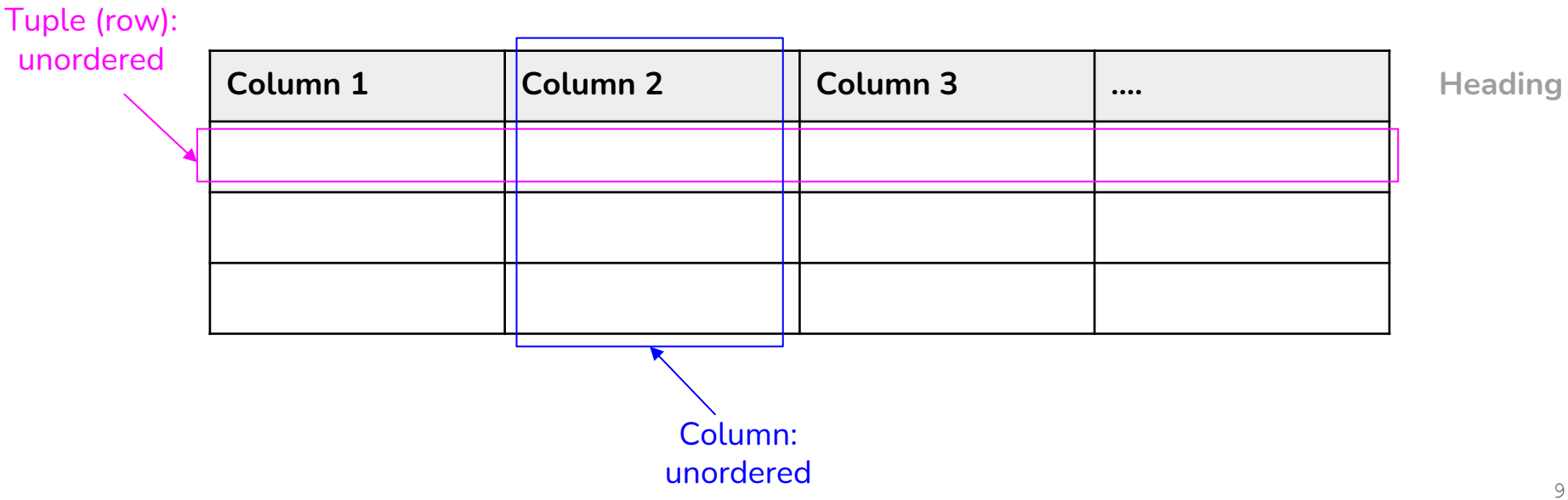
amazon REDSHIFT

Data models

- Describe how data is represented
- Two main paradigms:
  - Relational model
  - NoSQL

Relational model (est. 1970)

- Similar to SQL model
- Formats: CSV, Parquet



Relational model: normalization

What if we change “Banana Press” to “Pineapple Press”?

Title	Author	Format	Publisher	Country	Price
Harry Potter	J.K. Rowling	Paperback	Banana Press	UK	$20
Harry Potter	J.K. Rowling	E-book	Banana Press	UK	$10
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	$30
The Hobbit	J.R.R. Tolkien	Paperback	Banana Press	US	$30
Sherlock Holmes	Conan Doyle	Paperback	Guava Press	US	$15

Original Book Relation

Relational model: normalization

Title	Author	Format	Publisher ID	Price
Harry Potter	J.K. Rowling	Paperback	1	$20
Harry Potter	J.K. Rowling	E-book	1	$10
Sherlock Holmes	Conan Doyle	Paperback	2	$30
The Hobbit	J.R.R. Tolkien	Paperback	1	$30
Sherlock Holmes	Conan Doyle	Paperback	2	$15

Updated Book Relation

Publisher ID	Publisher	Country
1	Banana Press	UK
2	Guava Press	US

Publisher Relation

Relational model: normalization

Title	Author	Format	Publisher ID	Price
Harry Potter	J.K. Rowling	Paperback	1	$20
Harry Potter	J.K. Rowling	E-book	1	$10
Sherlock Holmes	Conan Doyle	Paperback	2	$30
The Hobbit	J.R.R. Tolkien	Paperback	1	$30
Sherlock Holmes	Conan Doyle	Paperback	2	$15

Publisher ID	Publisher	Country
1	Banana Press	UK
2	Guava Press	US

Pros:

- **Less mistakes (standardized spelling)**
- **Easier to update**
- **Easier localization**

Cons:

- **Slow to join across multiple large tables**

Relational Model & SQL Model

- SQL model slightly differs from relational model
  - e.g. SQL tables can contain row duplicates. True relations can’t.
- SQL is a query language
  - How to specify the data that you want from a database
- SQL is declarative
  - You tell the data system what you want
  - It’s up to the system to figure out how to execute
    - Query optimization

SQL

- SQL is an essential data scientists' tool

LEARN SQL!

NoSQL: No SQL -> Not Only SQL

- Document model
- Graph model

NoSQL

- Document model
  - Central concept: document
  - Relationships between documents are rare
- Graph model
  - Central concept: graph (nodes & edges)
  - Relationships are the priority

Document model: example

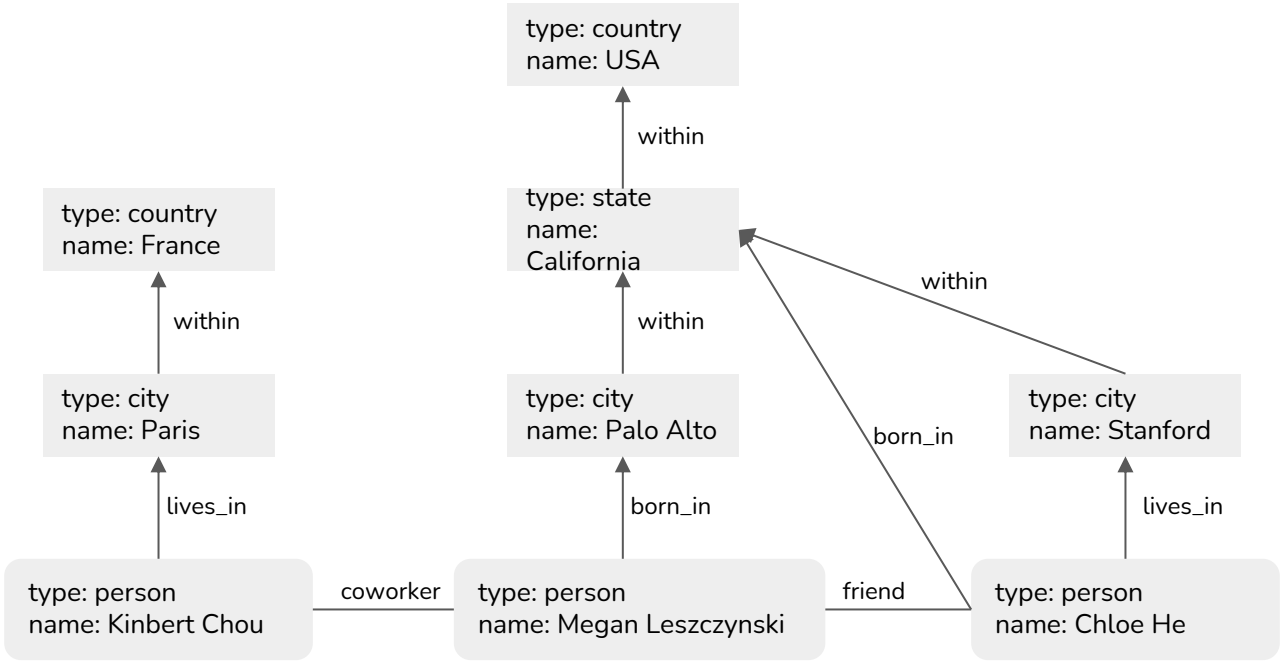
- Book data in the document model
- Each book is a document

```
# Document 1: harry_potter.json
{
    "Title": "Harry Potter",
    "Author": "J.K. Rowling",
    "Publisher": "Banana Press",
    "Country": "UK",
    "Sold as": [
        {"Format": "Paperback", "Price": "$20"},
        {"Format": "E-book", "Price": "$10"}
    ]
}

# Document 2: sherlock_holmes.json
{
    "Title": "Sherlock Holmes",
    "Author": "Conan Doyle",
    "Publisher": "Guava Press",
    "Country": "US",
    "Sold as": [
        {"Format": "Paperback", "Price": "$30"},
        {"Format": "E-book", "Price": "$15"}
    ]
}

# Document 3: the_hobbit.json
{
    "Title": "The Hobbit",
    "Author": "J.R.R. Tolkien",
    "Publisher": "Banana Press",
    "Country": "UK",
    "Sold as": [
        {"Format": "Paperback", "Price": "$30"},
    ]
}
```

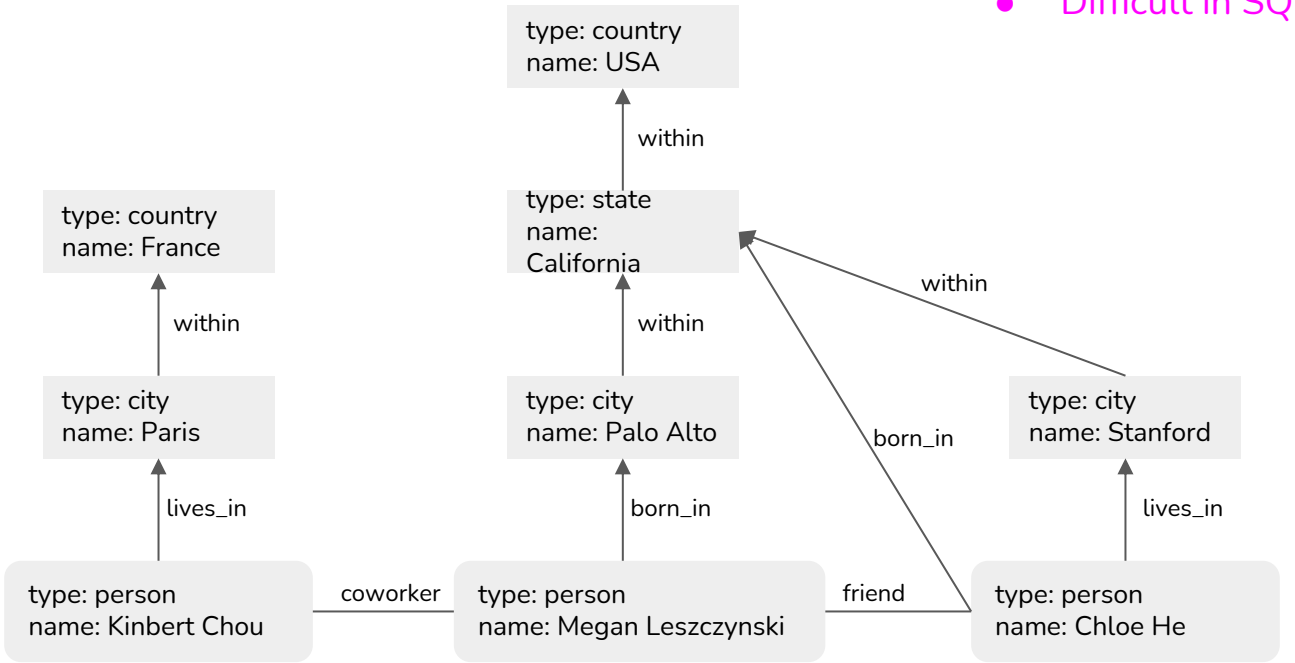
Graph model


type: country
name: USA
within
type: state
name:
California
within
type: country
name: France
within
type: city
name: Paris
type: city
name: Palo Alto
type: city
name: Stanford
born_in
lives_in
born_in
lives_in
type: person
name: Kinbert Chou
coworker
type: person
name: Megan Leszczynski
friend
type: person
name: Chloe He

Graph model

Query: show me everyone who was born in the USA?

- Easy in graph
- Difficult in SQL



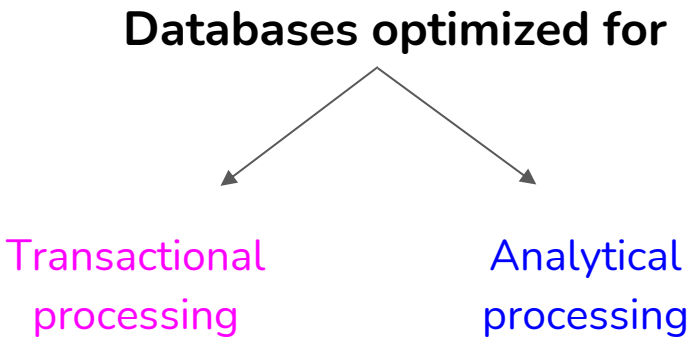
Structured vs. unstructured data

Structured	Unstructured
Schema clearly defined	Whatever
Easy to search and analyze	Fast arrival (e.g. no need to clean up first)
Can only handle data with specific schema	Can handle data from any source
Schema changes will cause a lot of trouble	No need to worry about schema changes
Data warehouses	Data lakes

Structured vs. unstructured data

Structured	Unstructured
Structure is assumed at **write**	Structure is assumed at **read**

Data Storage Engines & Processing



OnLine Transaction Processing (OLTP)

- Transactions: tweeting, ordering a Lyft, uploading a new model, etc.
- Operations:
  - Insert when generated
  - Occasional update/delete

OnLine Transaction Processing

- Transactions: tweeting, ordering a Lyft, uploading a new model, etc.
- Operations:
  - Inserted when generated
  - Occasional update/delete
- Requirements
  - Low latency
  - High availability

OnLine Transaction Processing

- Transactions: tweeting, ordering a Lyft, uploading a new model, etc.
- Operations:
  - Inserted when generated
  - Occasional update/delete
- Requirements
  - Low latency
  - High availability
  - ACID not necessary
    - **Atomicity**: all the steps in a transaction fail or succeed as a group
      - If payment fails, don't assign a driver
    - **Isolation**: concurrent transactions happen as if sequential
      - Don't assign the same driver to two different requests that happen at the same time

See ACID: Atomicity, Consistency, Isolation, Durability

OnLine Transaction Processing

- Transactions: tweeting, ordering a Lyft, uploading a new model, etc.
- Operations:
  - Inserted when generated
  - Occasional update/delete
- Requirements
  - Low latency
  - High availability
- Typically row-major

Row

```
INSERT INTO RideTable(RideID, Username, DriverID, City, Month, Price)
VALUES ('10', 'memelord', '3932839', 'Stanford', 'July', '20.4');
```

OnLine Analytical Processing (OLAP)

- How to get aggregated information from a large amount of data?
  - e.g. what's the average ride price last month for riders at Stanford?
- Operations:
  - Mostly SELECT

OnLine Analytical Processing

- Analytical queries: aggregated information from a large amount of data?
  - e.g. what's the average ride price last month for riders at Stanford?
- Operations:
  - Mostly SELECT
- Requirements:
  - Can handle complex queries on large volumes of data
  - Okay response time (seconds, minutes, even hours)

OnLine Analytical Processing

- Analytical queries: aggregated information from a large amount of data?
  - e.g. what's the average ride price last month for riders at Stanford?
- Operations:
  - Mostly SELECT
- Requirements:
  - Can handle complex queries on large volumes of data
  - Okay response time (seconds, minutes, even hours)
- Typically column-major

Column

```
SELECT AVG(Price)
FROM RideTable
WHERE City = 'Stanford' AND Month = 'July';
```

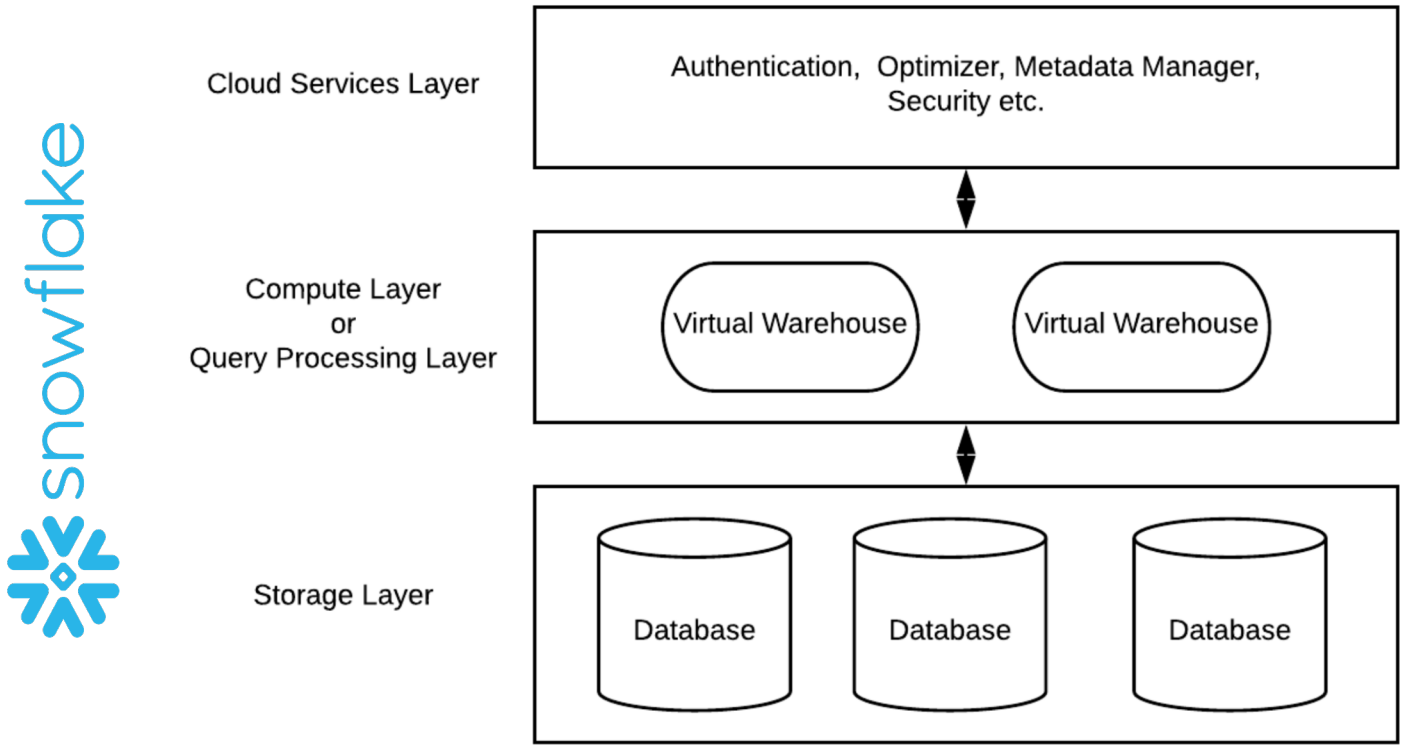
OLTP & OLAP are outdated terms

Decoupling storage & processing

- OLTP & OLAP: how data is stored is also how it's processed
  - Same data being stored in multiple databases
  - Each uses a different processing engine for different query types
- New paradigm: storage is decoupled from processing
  - Data can be stored in the same place
  - A processing layer on top that can be optimized for different query types



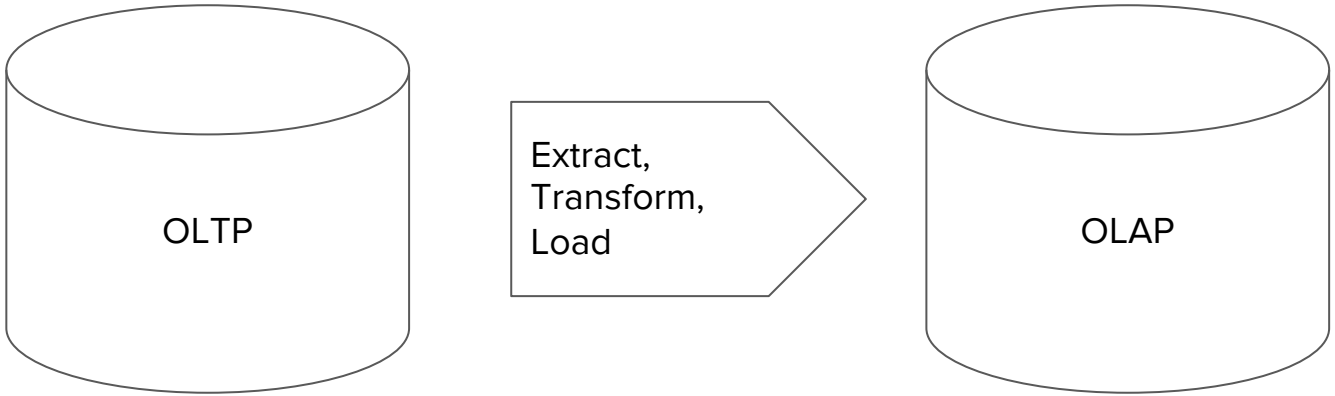
Decoupling storage & processing



https://hevodata.com/blog/snowflake-architecture-cloud-data-warehouse/

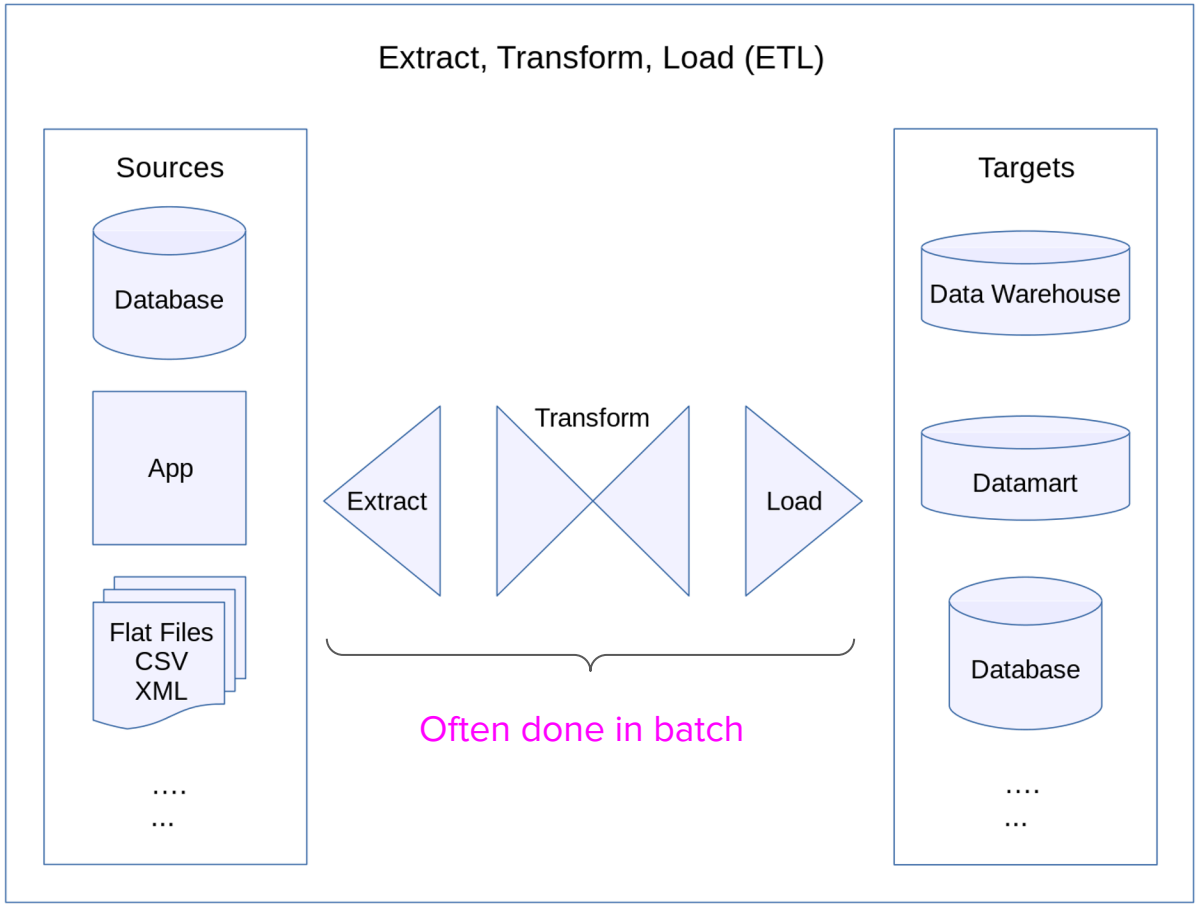

ETL
LIGHTYEAR
ETL EVERYWHERE

ETL (Extract, Transform, Load)



Transform: the meaty part

- cleaning, validating, transposing, deriving values, joining from multiple sources, deduplicating, splitting, aggregating, etc.



https://commons.wikimedia.org/wiki/File:Extract,_Transform,_Load_Data_Flow_Diagram.svg

ETL -> ELT

Structured -> unstructured -> structured

want more flexibility tools & infra

standardized

ETL -> ELT -> ETL