



# Pipeline & Workflow

"TBD."

# Table of Contents

01

Kursplan &  
Översikt

02

ETL vs ELT &  
Data Pipeline  
Theory

03

Workshop

04

Uppgifter  
&  
Övningar

# Overview



## Moduler:

- ETL
- ELT
- Filetypes
- Workshop with Data Quality



## Utbildningsmoment

- Dataplatfformar, bakgrund och syfte ✓
- Git och github i teamkontext
- Komponenter och teknologier i en data platform ✓
- ETL vs ELT ✓
- Utveckling av mjukvara mot databaser
- Använda Python mot relationsdatabaser och andra datakällor såsom csv, http xml/json
- Använda Python mot realtidsdataströmmar såsom message queues och/eller event streaming platforms
- Använda Python och för att rensa, validera och transformera data ✓
- Workflow processer ✓



02

ETL vs ELT

# Project Setup using uv



## BONUS

Ignorera detta om du inte vill veta mer om hur *'activate'* fungerar!

In a uv project, do NOT use this command:

```
bash
```

```
source .venv/bin/activate
```

venv/bin/activate. If you use uv, you actually should avoid to use `source .venv/bin/activate`. uv makes it superfluous to activate your environment. 15 juli 2025

*Source fungerar, men är onödigt i .venv projekt*


<https://stackoverflow.com/questions/79701540/pip-and-uv-point-to-local-bin-even-after-activating-virtual-environment>




demo\_7 ~/PycharmProjects/demo\_7


> .venv


app

 \_init\_.py

 main.py

 .gitignore

 .python-version

 pyproject.toml

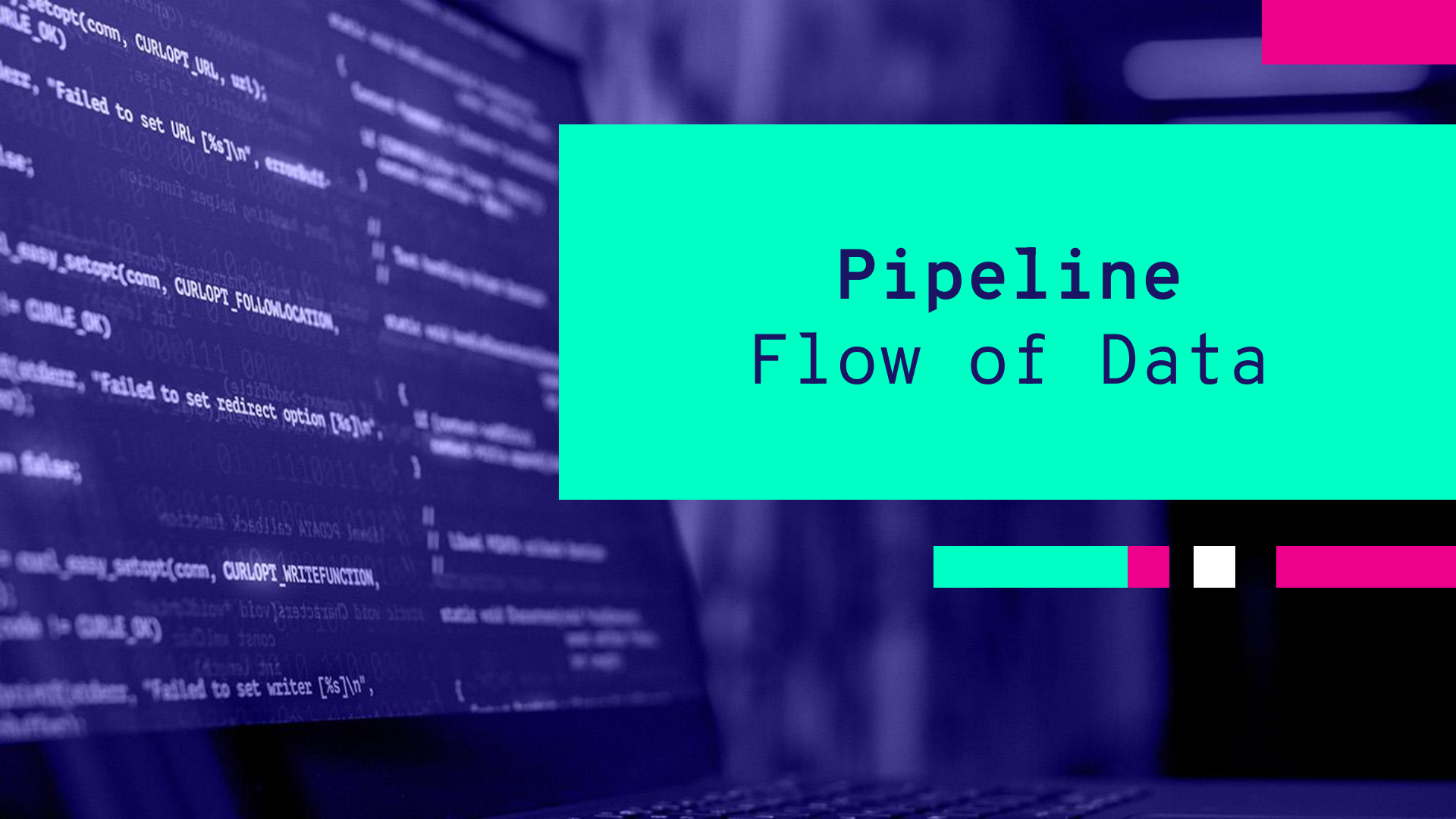
 README.md

 uv.lock

```
$ uv init
$ uv add "fastapi[standard]"
$ uv add "psycpg[binary]"
$ uv add "psycpg[pool]"
$ uv run uvicorn app.main:app --reload
```

```
app = FastAPI(title="demo_7")
NOTE: this ^ is required for 'uv run' to work!
```

# Pipeline Flow of Data



*“A defined, repeatable process that moves data from a source to a destination, optionally transforming it along the way”*



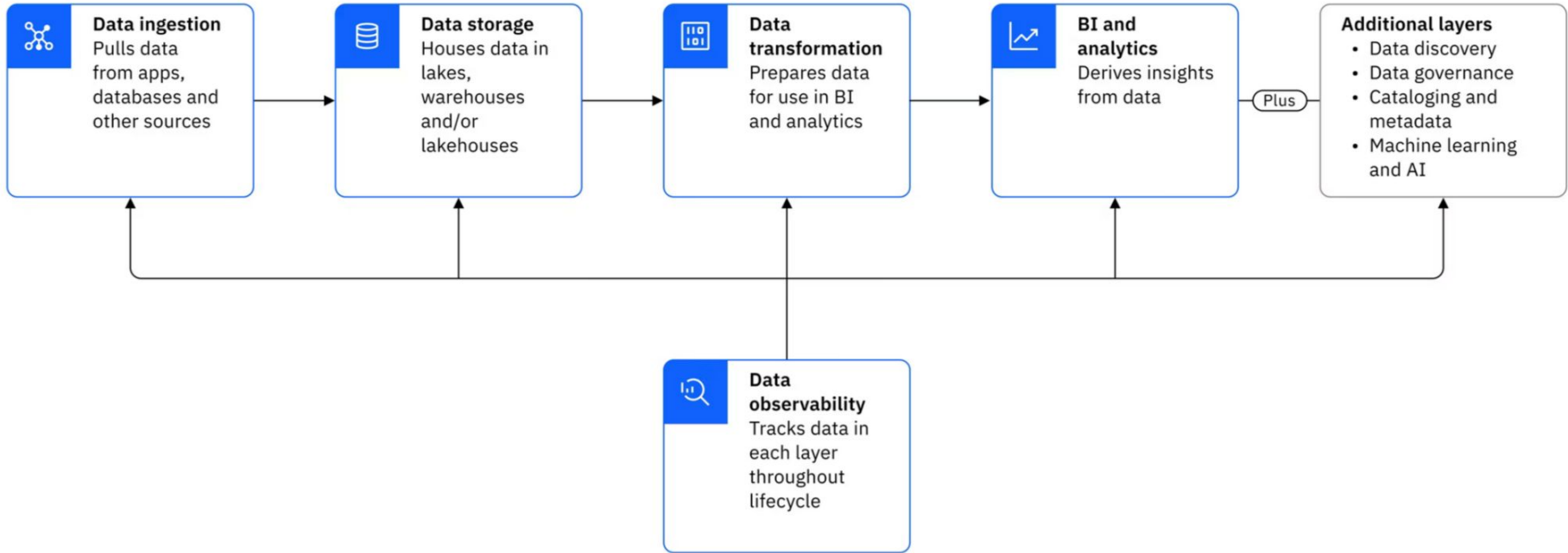
*“Pipelines gör dataflöden pålitliga, förutsägbara och lätta att förstå”*



Price = 999999

Price = none

Price = "294.0"



# Project Setup



# Git Clone (Repository)



**data-platform-lektion-6-preparation**

Public

<https://github.com/Krillinator/data-platform-lektion-6-preparation>



# Filetypes Structure





# Filetypes

## (Comma Separated Values)

> CSV

JSON

JSONL / NDJSON

Parquet

Avro

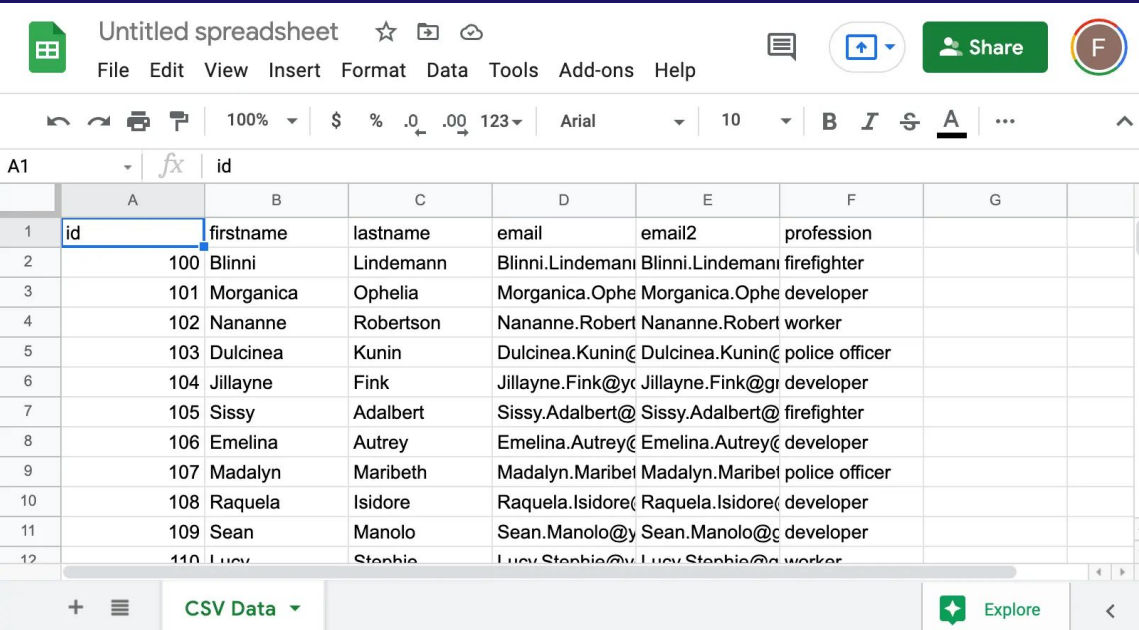
ORC

XML

TXT / LOG

Excel (XLSX)

GZIP / ZIP



Untitled spreadsheet

File Edit View Insert Format Data Tools Add-ons Help

100% 123 Arial 10 B I S A

	A	B	C	D	E	F	G
1	id	firstname	lastname	email	email2	profession	
2	100	Blinni	Lindemann	Blinni.Lindemann	Blinni.Lindemann	firefighter	
3	101	Morganica	Ophelia	Morganica.Ophe	Morganica.Ophe	developer	
4	102	Nananne	Robertson	Nananne.Robert	Nananne.Robert	worker	
5	103	Dulcinea	Kunin	Dulcinea.Kunin	Dulcinea.Kunin	police officer	
6	104	Jillayne	Fink	Jillayne.Fink@y	Jillayne.Fink@gr	developer	
7	105	Sissy	Adalbert	Sissy.Adalbert@	Sissy.Adalbert@	firefighter	
8	106	Emelina	Autrey	Emelina.Autrey	Emelina.Autrey	developer	
9	107	Madalyn	Maribeth	Madalyn.Maribet	Madalyn.Maribet	police officer	
10	108	Raquela	Isidore	Raquela.Isidore	Raquela.Isidore	developer	
11	109	Sean	Manolo	Sean.Manolo@y	Sean.Manolo@c	developer	
12	110	Lucy	Stephie	Lucy.Stephie@y	Lucy.Stephie@c	worker	

CSV Data Explore

# Filetypes

## (Javascript Object Notation)

CSV

> JSON

JSONL / NDJSON

Parquet

Avro

ORC

XML

TXT / LOG

Excel (XLSX)

GZIP / ZIP

```
1 {  
2   "array": [  
3     1,  
4     2,  
5     3  
6   ],  
7   "boolean": true,  
8   "color": "gold",  
9   "null": null,  
10  "number": 123,  
11  "object": {  
12    "a": "b",  
13    "c": "d"  
14  },  
15  "string": "Hello World"  
16 }
```

# Filetypes

## (JSONL / NDJSON – Explained)

CSV

JSON

> JSONL / NDJSON

Parquet

Avro

ORC

XML

TXT / LOG

Excel (XLSX)

GZIP / ZIP

### Differences Between JSON and JSONL

Before diving deeper into how to work with **pandas jsonl**, let's consider the differences between JSON and JSONL.

- **JSON:** Typically stores data as one large object. It is suitable for smaller datasets or when you need to represent complex nested structures.
- **JSONL:** Each JSON object is line-separated. This format is more efficient when dealing with large datasets, as you can read or write one line at a time without loading the entire dataset into memory.

# Filetypes

## (JSONL / NDJSON – Showcase difference)

**Json:** Must be read as a whole object - the longer the object == longer time

```
[  
  { "name": "USB-C Cable", "price": 149.0, "quantity": 120 },  
  { "name": "Wireless Mouse", "price": 299.0, "quantity": 75 },  
  { "name": "Keyboard", "price": 1299.0, "quantity": 25 }  
]
```

**JsonL:** Read separately in chunks

```
{ "name": "USB-C Cable", "price": 149.0, "quantity": 120 }  
{ "name": "Wireless Mouse", "price": 299.0, "quantity": 75 }  
{ "name": "Keyboard", "price": 1299.0, "quantity": 25 }
```

# Filetypes (Parquet)

CSV  
JSON  
JSONL / NDJSON  
> Parquet  
Avro  
ORC  
XML  
TXT / LOG  
Excel (XLSX)  
GZIP / ZIP

**Row-based Storage**

ID	Name	Age	City
1	John	25	NYC
2	Jane	30	SF

**Columnar Storage (Parquet)**

ID	Name	Age	City
1	John	25	NYC
2	Jane	30	SF

**Benefits of Columnar Storage**

Better Compression

Efficient Analytical Queries

**Source:**

<https://motherduck.com/learn-more/why-choose-parquet-table-file-format/>

# Filetypes

## (Parquet – Showcase)

Parquet file

```
├─ Row Group 1
│   ├── Column: id
│   ├── Column: name
│   ├── Column: age
│   └─ Column: city
├─ Row Group 2
│   ├── Column: id
│   ├── Column: name
│   ├── Column: age
│   └─ Column: city
```

# Filetypes

## (Parquet – Showcase)

CSV  
JSON  
JSONL / NDJSON  
Parquet  
> Avro  
ORC  
XML  
TXT / LOG  
Excel (XLSX)  
GZIP / ZIP

- Large data volumes
- Strict schemas
- Schema changes over time
- Streaming systems (Kafka)



**Avro**

# Filetypes

## (Avro – Showcase (JSON))

This is normal **JSON**, easy to read, stored as text (*ASCII with UTF-8 as possible extensions*)

```
{  
  "name": "USB-C Cable",  
  "price": 149.0,  
  "quantity": 120  
}
```



# Filetypes

## (Avro – Showcase (AVRO))

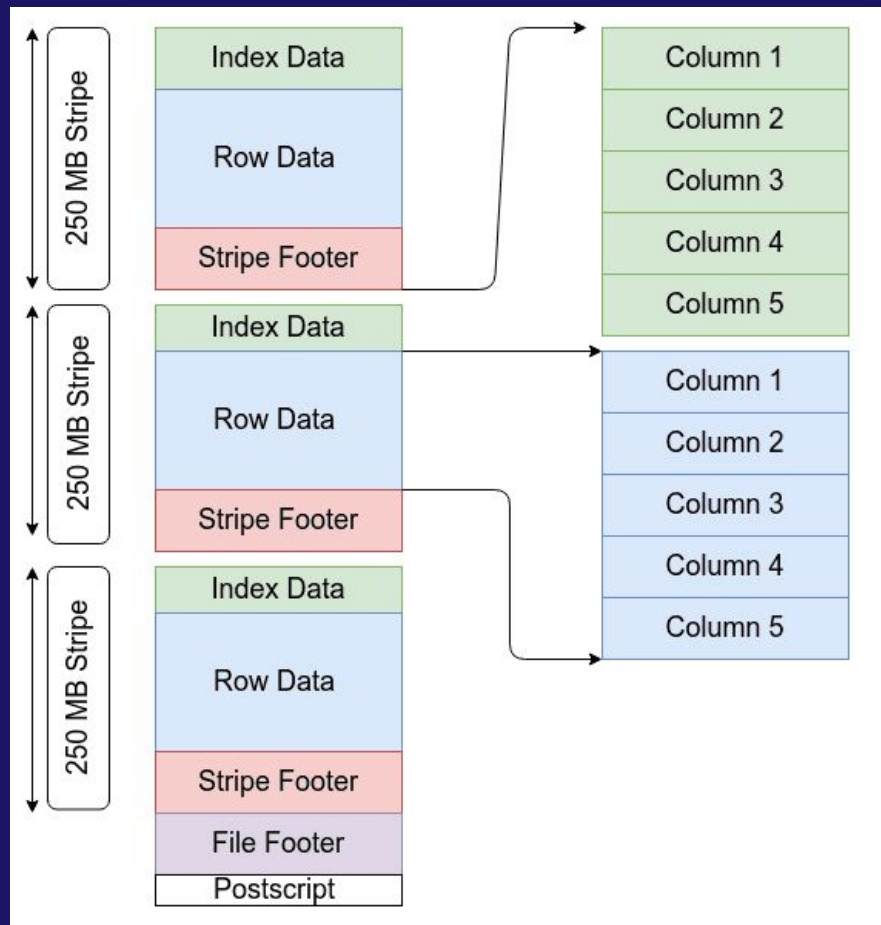
This is AVRO, stored in binary, not to be read by humans, but understood by machines

```
Obj{"name": "USB-C Cable",  
  "price": 100,  
  "quantity": 10}
```

# Filetypes

## (Optimized Row Columnar)

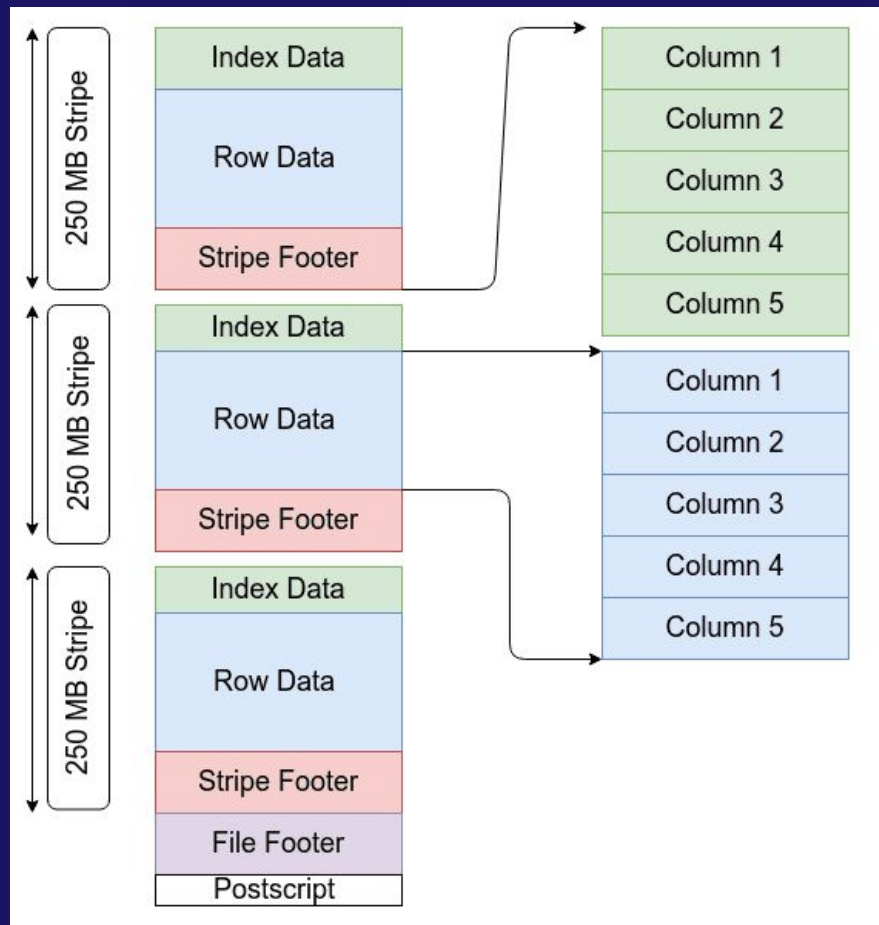
CSV  
JSON  
JSONL / NDJSON  
Parquet  
Avro  
> ORC  
XML  
TXT / LOG  
Excel (XLSX)  
GZIP / ZIP



# Filetypes

## (Optimized Row Columnar)

- Data is stored by **column**
- Grouped into **row groups** (*stripes*)
- Strong typing
- Heavy compression
- Less common today, Parquet is more preferred. Legacy systems still use this



# Filetypes

## (eXtensible Markup Language)

CSV  
JSON  
JSONL / NDJSON  
Parquet  
Avro  
ORC  
> XML  
TXT / LOG  
Excel (XLSX)  
GZIP / ZIP

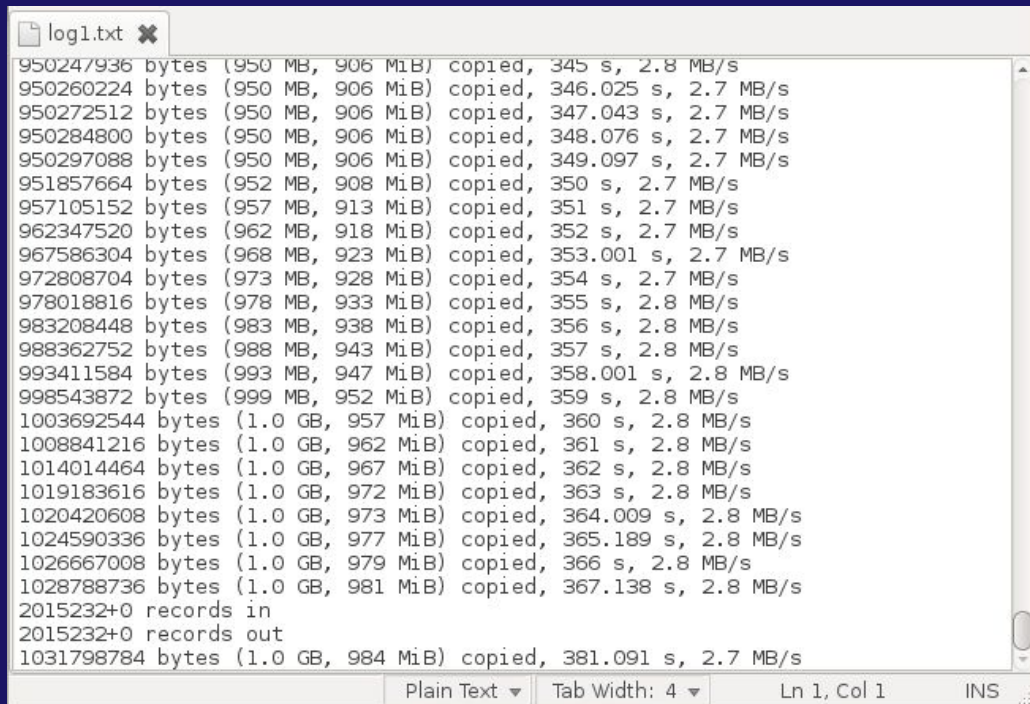
```
-<CATALOG>
  -<CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  -<CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
```

[https://www.w3schools.com/xml/cd\\_catalog.xml](https://www.w3schools.com/xml/cd_catalog.xml)

# Filetypes

## (TXT / LOG)

CSV  
JSON  
JSONL / NDJSON  
Parquet  
Avro  
ORC  
XML  
> TXT / LOG  
Excel (XLSX)  
GZIP / ZIP



```
log1.txt
950247936 bytes (950 MB, 906 MiB) copied, 345 s, 2.8 MB/s
950260224 bytes (950 MB, 906 MiB) copied, 346.025 s, 2.7 MB/s
950272512 bytes (950 MB, 906 MiB) copied, 347.043 s, 2.7 MB/s
950284800 bytes (950 MB, 906 MiB) copied, 348.076 s, 2.7 MB/s
950297088 bytes (950 MB, 906 MiB) copied, 349.097 s, 2.7 MB/s
951857664 bytes (952 MB, 908 MiB) copied, 350 s, 2.7 MB/s
957105152 bytes (957 MB, 913 MiB) copied, 351 s, 2.7 MB/s
962347520 bytes (962 MB, 918 MiB) copied, 352 s, 2.7 MB/s
967586304 bytes (968 MB, 923 MiB) copied, 353.001 s, 2.7 MB/s
972808704 bytes (973 MB, 928 MiB) copied, 354 s, 2.7 MB/s
978018816 bytes (978 MB, 933 MiB) copied, 355 s, 2.8 MB/s
983208448 bytes (983 MB, 938 MiB) copied, 356 s, 2.8 MB/s
988362752 bytes (988 MB, 943 MiB) copied, 357 s, 2.8 MB/s
993411584 bytes (993 MB, 947 MiB) copied, 358.001 s, 2.8 MB/s
998543872 bytes (999 MB, 952 MiB) copied, 359 s, 2.8 MB/s
1003692544 bytes (1.0 GB, 957 MiB) copied, 360 s, 2.8 MB/s
1008841216 bytes (1.0 GB, 962 MiB) copied, 361 s, 2.8 MB/s
1014014464 bytes (1.0 GB, 967 MiB) copied, 362 s, 2.8 MB/s
1019183616 bytes (1.0 GB, 972 MiB) copied, 363 s, 2.8 MB/s
1020420608 bytes (1.0 GB, 973 MiB) copied, 364.009 s, 2.8 MB/s
1024590336 bytes (1.0 GB, 977 MiB) copied, 365.189 s, 2.8 MB/s
1026667008 bytes (1.0 GB, 979 MiB) copied, 366 s, 2.8 MB/s
1028788736 bytes (1.0 GB, 981 MiB) copied, 367.138 s, 2.8 MB/s
2015232+0 records in
2015232+0 records out
1031798784 bytes (1.0 GB, 984 MiB) copied, 381.091 s, 2.7 MB/s
```

Plain Text ▾ Tab Width: 4 ▾ Ln 1, Col 1 INS

# Filetypes

## (Excel Spreadsheet (*xml-based*))

CSV  
JSON  
JSONL / NDJSON  
Parquet  
Avro  
ORC  
XML  
TXT / LOG  
> Excel (XLSX)  
GZIP / ZIP

*XLSX is a binary, structured file format for spreadsheets (xml based)*

- Is a **ZIP archive**
- Contains **XML files**
  - With sheets,
  - styles,
  - formulas,
  - metadata

# Filetypes

## (GZIP / ZIP)

CSV  
JSON  
JSONL / NDJSON  
Parquet  
Avro  
ORC  
XML  
TXT / LOG  
Excel (XLSX)  
> GZIP / ZIP

*“How are files packaged and compressed”*



```
curl_easy_setopt(comm, CURLOPT_URL, url);  
if (curl_easy_perform(comm) != CURLE_OK) {  
    fprintf(stderr, "Failed to set URL [%s]\n", errorbuf);  
    return 1;  
}  
curl_easy_setopt(comm, CURLOPT_FOLLOWLOCATION, 1);  
if (curl_easy_perform(comm) != CURLE_OK) {  
    fprintf(stderr, "Failed to set redirect option [%s]\n", errorbuf);  
    return 1;  
}  
curl_easy_setopt(comm, CURLOPT_WRITEFUNCTION, write_callback);  
if (curl_easy_perform(comm) != CURLE_OK) {  
    fprintf(stderr, "Failed to set writer [%s]\n", errorbuf);  
    return 1;  
}
```

# ETL

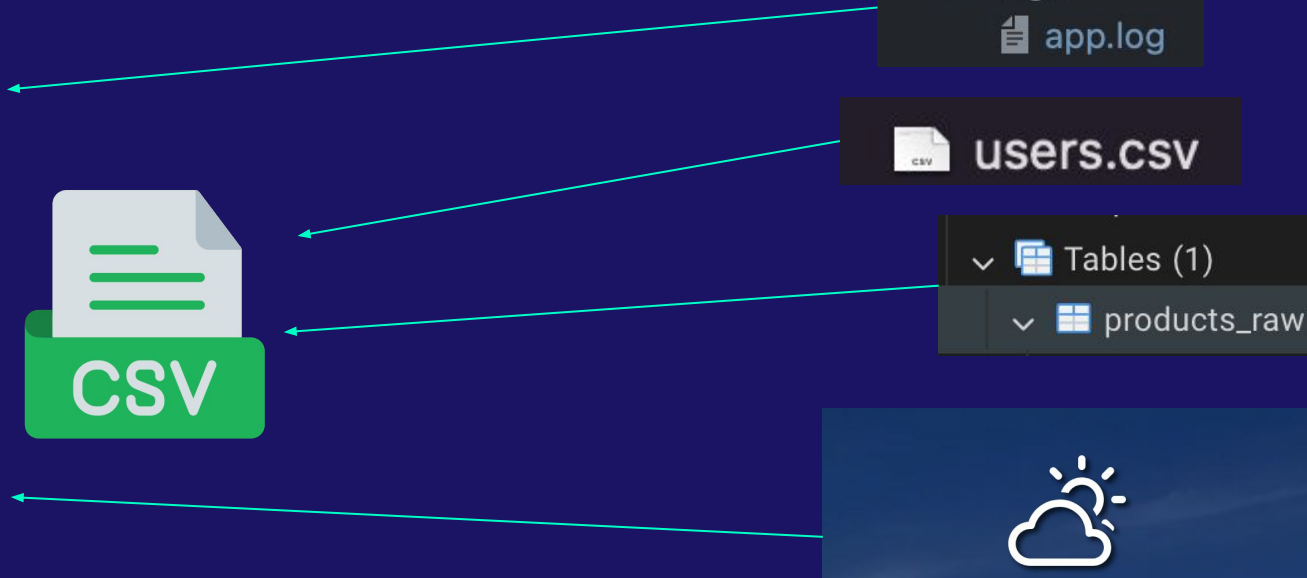
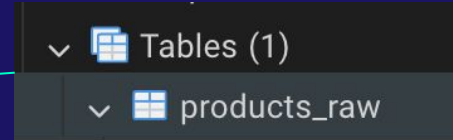
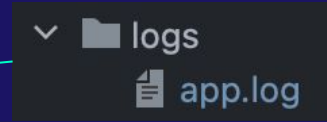
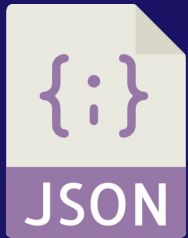
## Extract, Transform, Load





# ETL

## (Extract)



# ETL

(Is this ready to be stored?)

```
1  {
2    "userId": "u-42",
3    "name": "Anna Svensson",
4    "age": "29",
5    "signupDate": "2024/01/15",
6    "purchases": [
7      { "item": "Book", "price": "199", "currency": "SEK" },
8      { "item": "Pen", "price": "19", "currency": "sek" }
9    ]
10 }
11
```

# ETL

## (Transform)

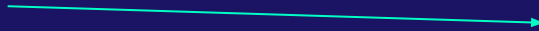
*“Data transformation is everything you do to clean, reshape, and standardize data so it can be trusted and used”*



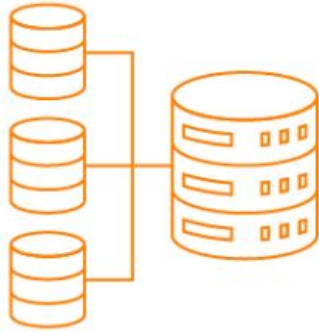
# ETL

## (Load)

Transformed data



# The ETL Process Explained



## Extract

Retrieves and verifies data  
from various sources



## Transform

Processes and organizes  
extracted data so it is usable



## Load

Moves transformed data  
to a data repository

# ELT

## Extract, Load, Transform



# ELT

## (Extract, Load, Transform)



### Extract

Retrieves and verifies data  
from various sources



### Load

Moves transformed data  
to a data repository



### Transform

Processes and organizes  
extracted data so it is usable

## ETL vs ELT: What's the difference?

	ETL	ELT
Extraction	Raw data is extracted from disparate sources	Raw data is extracted from disparate sources
Transformation	Raw data is transformed on a secondary server or staging area	Data is transformed within the data warehouse or system
Loading	Data is loaded into the warehouse or system after transformation	Raw data is loaded directly into the warehouse or system





03

Workshop

# Data Insertion Creating Problems



# Database Inserts

## Work Lab



*Dela upp er 2-2 eller 3-3*

***Krav innan ni kör igång:***

- *GitHub Projekt (slide #15)*
- *Projekt kopplad till databas*
- *Postman/thunderclient  
öppen*

# Database Inserts

## Work Lab

Skapa produkter som går in mot databasen,  
men som inte riktigt är helt rätt

Exempelvis:

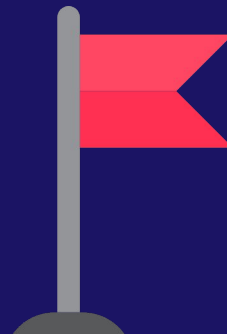
```
1  {  
2  |    "name": "Wireless Mouse",  
3  |    "price": 249.0,  
4  |    "quantity": "2",  
5  |    "currency": "SEK"  
6  }
```

# Data Quality Checks



## Vad gör vi när vi saknar fält värden?

order_id	customer_email	amount	quality_status	is_valid
1001	<u>alice@example.com</u>	250.00	OK	TRUE
1002		180.50	MISSING_EMAIL	FALSE
1003	<u>bob@example.com</u>		-MISSING_AMOUN T	FALSE



# Tänk om datan är omöjlig?

raw_row	reason	rejected_at
{"order_id":null,"customer_email":"a@b.com","amount":120}	MISSING_ID	2026-02-09T00:42:00Z
{"order_id":"ABC","customer_email":"x@y.com","amount":-50}	INVALID_AMOUNT	2026-02-09T00:42:01Z
{"order_id":1003,"customer_email":"bad@@mail","amount":"NaN"}	BROKEN_DATATYPE	2026-02-09T00:42:02Z



# Regler för datahantering i ETL

*(Transformera - Flagga - Avvisa)*

*(Transform - flag - reject)*

## Grundprincip

- Ingen gissning. Inga antaganden. (never guess)
- Vi ändrar inte betydelse (the purpose of data remains unchanged)
- Vi gör data användbar eller synlig som problem (data is usable or visible as problem)
- Vi korrigerar format.. inte innehåll. (*correct format, not content*)

*I kommande uppgifter, förlita dig på att vi arbetar mot PostgreSQL som databas*



```
1  {
2    .... "product_id": "USP239",
3    .... "name": "Wireless Mouse",
4    .... "price": 249.0,
5    .... "currency": "SEK",
6    .... "category": "Electronics",
7    .... "brand": null
8  }
```

Ser datan korrekt ut, **acceptera**



```
1 {  
2   "product": "mouse",  
3   "price": 199,  
4   "currency": "SEK"  
5 }
```



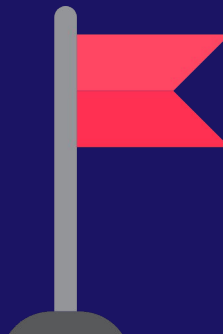
När datan **inte** är **korrekt** (*datatyp, mellanslag, fel ISO standard*)

Då behövs datan **städas upp** och transformeras till något mer användbart.

*UTAN ATT ÄNDRA DATANS INNEBÖRD*

```
1  {  
2  |  "product": "HDMI cable",  
3  |  "price": 1,  
4  |  "currency": "SEK"  
5  }
```

- tekniskt giltig
- misstänkt lågt värde
- kan vara korrekt (*kampanj, fel, bulk*)



## Åtgärd

- behåll posten
- märk den som flaggad
- exkludera från beräkningar som standard

```
1  {  
2    "product": "Laptop",  
3    "price": 12000  
4  }
```



## Problemtyp

- obligatoriskt fält saknas (currency)
- datan kan inte användas alls
- Omöjliga värden
- Ingen identitet

```
1  {  
2    "original_data": {  
3      "product": "Laptop",  
4      "price": 12000  
5    },  
6    "reject_reason": "missing_required_field: currency"  
7  }
```





Accept



Clean/Transform



Flag



Reject

OK or ✓

>> or ~>

! or ?

X or //

# Decide Quality JSON Examples [Easy]



# Example 1

```
{  
  "id": "sku-29",  
  "name": "  Milk  ",  
  "price": 19,  
  "currency": "sek"  
}
```

## Example 2

```
{  
  "id": "sku-30",  
  "name": "Bread",  
  "price": "25",  
  "currency": "SEK"  
}
```



## Example 3

```
{  
  "id": "sku-31",  
  "name": "Shoes",  
  "price": -799,  
  "currency": "SEK"  
}
```

# Example 4

```
{  
  "id": "sku-32",  
  "name": "Socks",  
  "price": 99,  
  "quantity": "two",  
  "currency": "SEK"  
}
```

# Example 5

```
{  
  "id": "sku-33",  
  "name": "Headphones",  
  "price": 199,  
  "currency": "USD",  
  "country": "SE"  
}
```

# Example 6

```
1  {
2    "order_id": "A-1001",
3    "customer": { "id": "c42", "name": "anna" },
4    "items": [
5      { "sku": "KB-1", "unit_price": "199", "quantity": "2", "currency": "sek" },
6      { "sku": "MS-9", "unit_price": "99", "quantity": "1", "currency": "SEK" }
7    ]
8  }
```

# Decide Quality JSON Examples [Easy] – FACIT



# Example 1

## (FACIT)

```
{  
  "id": "sku-29",  
  "name": "  Milk  ",  
  "price": 19,  
  "currency": "sek"  
}
```

**Transform >>**

Trimming and uppercasing does not change meaning.

## Example 2

### (FACIT)

```
{  
  "id": "sku-30",  
  "name": "Bread",  
  "price": "25",  
  "currency": "SEK"  
}
```

**Transform >>**

**Clearly numeric, only representation is wrong.**

## Example 3

### (FACIT)

```
{  
  "id": "sku-31",  
  "name": "Shoes",  
  "price": -799,  
  "currency": "SEK"  
}
```

**Flag !**

**Could be refund, error, or discount.. intent unclear.**



# Example 4

## (FACIT)

```
{  
  "id": "sku-32",  
  "name": "Socks",  
  "price": 99,  
  "quantity": "two",  
  "currency": "SEK"  
}
```

**Reject X**

Breaking type rules for 'two' aka 2

## Example 5

### (FACIT)

```
{  
  "id": "sku-33",  
  "name": "Headphones",  
  "price": 199,  
  "currency": "USD",  
  "country": "SE",  
  "pricing_currency": "SEK"  
}
```

**Reject X**

Two currencies, no precedence rule

## Example 6

### (FACIT)

```
1  {
2    "order_id": "A-1001",
3    "customer": { "id": "c42", "name": "anna" },
4    "items": [
5      { "sku": "KB-1", "unit_price": "199", "quantity": "2", "currency": "sek" },
6      { "sku": "MS-9", "unit_price": "99", "quantity": "1", "currency": "SEK" }
7    ]
8  }
```

#### Transform >>

Trim name, convert price/quantity to numbers, uppercase currency. No meaning change.

# Decide Quality JSON Examples [Level: Medium]



# Example 1

```
1  {
2    "order_id": "A-1002",
3    "customer": { "id": "c43", "name": "Erik" },
4    "items": [
5      { "sku": "KB-1", "unit_price": 199, "quantity": 2, "currency": "SEK" },
6      { "unit_price": 99, "quantity": 1, "currency": "SEK" }
7    ]
8  }
```

## Example 2

```
1  {
2    "order_id": "A-1003",
3    "customer": { "id": "c44", "name": "Sara" },
4    "items": [
5      { "sku": "HDMI-2", "unit_price": 1, "quantity": 3, "currency": "SEK" },
6      { "sku": "SSD-1T", "unit_price": 9999, "quantity": 1, "currency": "SEK" }
7    ],
8    "totals": { "declared_total": 4, "currency": "SEK" }
9  }
```

# Example 3

```
1  {  
2  |  .."order_id": "A-2001",  
3  |  .."customer": {  
4  |  |  ...."id": "c51",  
5  |  |  ...."name": "Maria"  
6  |  |  ..},  
7  |  .."items": [  
8  |  |  ....{  
9  |  |  |  ...."sku": "KB-1",  
10 |  |  |  ...."unit_price": 199,  
11 |  |  |  ...."quantity": "two",  
12 |  |  |  ...."currency": "SEK"  
13 |  |  |  ....}  
14 |  |  ..]  
15 }
```

## Example 4

```
1  {  
2  |  · "subscription_id": · "S-1001",  
3  |  · "start_date": · "2024-06-01",  
4  |  · "end_date": · "2024-05-01"  
5  }
```



# THE FINALE

```
1  {
2    "order_id": "1005",
3    "customer": {
4      "id": "c55",
5      "is_vip": "true"
6    },
7    "shipment": {
8      "shipped_at": "2024-13-13",
9      "delivered_at": "2024-13-14",
10     "tracking": { "code": 123456, "carrier": "DHL" }
11   },
12   "items": [
13     { "sku": "KB-1", "unit_price": "199.00", "quantity": 2, "currency": "SEK" }
14   ],
15   "paid": "yes"
16 }
```

# Decide Quality JSON Examples [Level: Medium] Facit



# Example 1

## (FACIT)

```
1  {
2    "order_id": "A-1002",
3    "customer": { "id": "c43", "name": "Erik" },
4    "items": [
5      { "sku": "KB-1", "unit_price": 199, "quantity": 2, "currency": "SEK" },
6      { "unit_price": 99, "quantity": 1, "currency": "SEK" }
7    ]
8  }
```

### Reject X

Second item missing required sku, invalid record.

If you decide to keep it, only reject that nested item

## Example 2

### (FACIT)

```
1  {
2    "order_id": "A-1003",
3    "customer": { "id": "c44", "name": "Sara" },
4    "items": [
5      { "sku": "HDMI-2", "unit_price": 1, "quantity": 3, "currency": "SEK" },
6      { "sku": "SSD-1T", "unit_price": 9999, "quantity": 1, "currency": "SEK" }
7    ],
8    "totals": { "declared_total": 4, "currency": "SEK" }
9  }
```

**Accept OK**

All looks great!

## Example 3 (FACIT)

```
1  {  
2  .. "order_id": "A-2001",  
3  .. "customer": {  
4  .... "id": "c51",  
5  .... "name": "Maria"  
6  .. },  
7  .. "items": [  
8  .... {  
9  ..... "sku": "KB-1",  
10 ..... "unit_price": 199,  
11 ..... "quantity": "two",  
12 ..... "currency": "SEK"  
13 ..... }  
14 .. ]  
15 }
```

### Reject X

Cannot safely transform without guessing.  
Hard datatype rule violated

## Example 4

### (FACIT)

```
1  {
2  |  · "subscription_id": · "S-1001",
3  |  · "start_date": · "2024-06-01",
4  |  · "end_date": · "2024-05-01"
5  }
```

#### Reject X

Dates are valid ISO strings but the logic is impossible. End cannot be before start date.

# FACIT – REJECT

```
1  {
2    "order_id": "1005",
3    "customer": {
4      "id": "c55",
5      "is_vip": "true"
6    },
7    "shipment": {
8      "shipped_at": "2024-13-13", ←
9      "delivered_at": "2024-13-14", ←
10     "tracking": { "code": 123456, "carrier": "DHL" }
11   },
12   "items": [
13     { "sku": "KB-1", "unit_price": "199.00", "quantity": 2, "currency": "SEK" }
14   ],
15   "paid": "yes" ←
16 }
```

# THANKS!

Do you have any questions?  
[kristoffer.johansson@sti.se](mailto:kristoffer.johansson@sti.se)

CREDITS: This presentation template was created by  
Slidesgo, including icons by Flaticon, and  
infographics & images by Freepik.