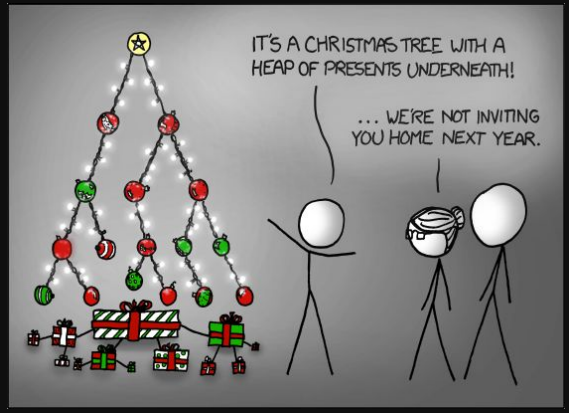
# Session 2

## Data structures for Business





NLAB: *Data at Scale*

Dr Evgeniya Lukinova

# Context is everything...

(so let's take a look)

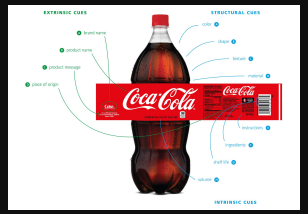
Transaction data

Google Analytics

mouseflow

Website tracking data

Security logs

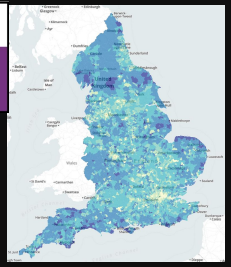



Product Detail Data

Survey Data

Office for National Statistics

census 2021



Census Data

Call Detail Record (CDR) Data

Social Media Data

(Formally)
Describing data
requires a paradigm

Intuitive paradigm?

Objects!

mongoDB.

**Objects** **vs** **Relational**


Lego slides from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

# Best paradigm?

Not always objects...

Alternatively I could store my lego in pieces...

Need to know what pieces (simple building blocks) are available and to have the assembly **instructions**

Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

Benefit: Flexibility

Otherwise to make cars:

→ 1st understand how it is constructed

→ Work out how to break it down

→ Create new cars

Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

# Best paradigm?

## But **why not just store all three** cars?

*What if I bought the wrong wheels to replicate my favorite car, and the hub caps are silver?*
(**data replication** )

- hard to update
- data often conflicts

Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

# Benefit: Flexibility

**Cars With the Most Safety Recalls - iSeeCars Study**

| Rank | Model | Expected 30-Year Lifetime Recalls |
|---|---|---|
| 1 | Porsche Taycan | 70.7 |
| 2 | Tesla Model Y | 66.9 |
| 3 | Tesla Model 3 | 60.7 |
| 4 | Porsche Panamera | 43.1 |
| 5 | Lucid Air | 40.1 |
| 6 | Tesla Model S | 38.5 |
| 7 | Tesla Model X | 37.6 |
| 8 | Lincoln Aviator | 26.2 |
| 9 | Genesis GV70 | 22.3 |
| 10 | Kia Telluride | 22.2 |


In 2023, the average car was projected to have 3.2 recalls throughout its 30-year lifespan.
The Porsche Taycan was projected to have 70.7, Tesla Model Y - 66.9, while the Mini Convertible was projected to have 0.2.

Images from:
https://imageio.forbes.com/specials-images/imageserve/668dcda8f89592a74415bc6f/2024-Most-Recalled-Cars/960x0.png?format=png&width=1440
https://imageio.forbes.com/specials-images/imageserve/668dcd1df6c9cef19e006df9/Tesla-Issues-Recall-On-2-Million-Of-Its-Vehicles-In-The-U-S--Due-To-Autopilot-Issue/960x0.jpg?format=jpg&width=1440

# Summary:

Storing one car and having to break it into pieces and rebuild things is hard.

*Better just to store the pieces and build instructions.*

Storing multiple copies so you don't have to break the original into pieces can duplicate data - introduces potential for error.

*Better just to store the pieces and build instructions.*

Storing pieces and build instructions can be slower if you just want to ask about a specific car.

*Are you sure you're only ever going to want to ask about a specific car?*

Is the speed gain worth it?

Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

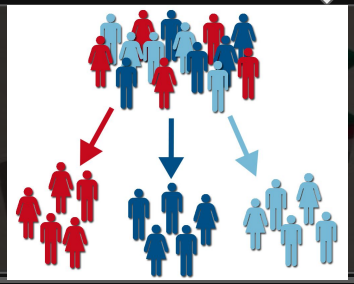

Why are we talking about lego anyway?
EMMET

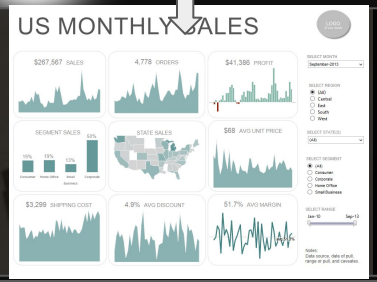
Best paradigm?

Why are we talking about lego anyway?

Let's consider transactional data, i.e. a receipt.

Original format







Of course data comes originally in one format.
Cost associated with:
→ Working out best "pieces"
→ Breaking it up

Typically not your job….

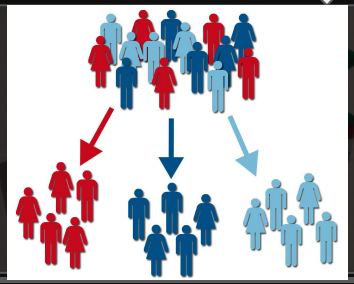
Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

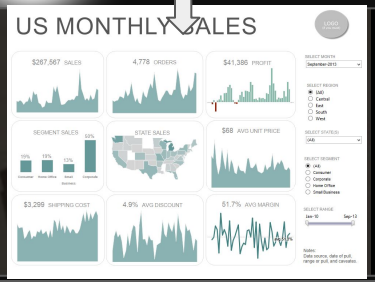
Best paradigm?

Why are we talking about lego anyway?

Let's consider transactional data, i.e. a receipt.

Original format







Of course data comes originally in one format.
Cost associated with:
→ Working out best "pieces"
→ Breaking it up

Typically not your job….

If it was your job, or if you had an **object based datastore**….

→ Need to examine the structure of **each object type each time**
→ Need to work out what pieces you have **for each object**
(information loss?)
(may end up in complex structures...)

Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

Best paradigm?

Not always objects...

If this isn't your job, and has been done once by the company (or just stored correctly by design in the first place)...

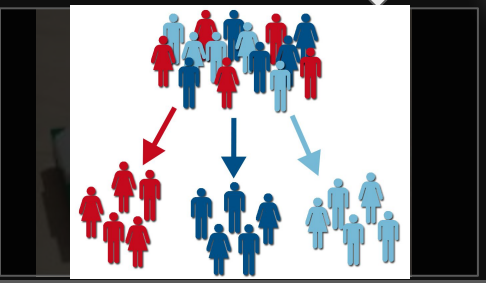
Still need to know what pieces (simple building blocks) are available and the assembly instructions

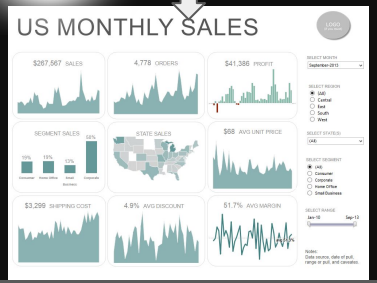
But we start from a better position.

Really want a well defined, standard set of base building blocks from which it is proven we can construct anything.

Plus a standard (easy) language for the instructions.

Rather than different styles of building blocks dreamt up by an individual or company.







Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

# Best paradigm?

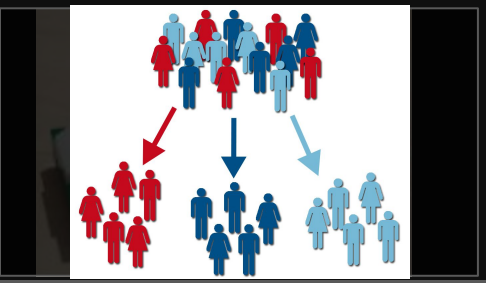
But **why not just store all three** cars?

*What if I want to make a new car?*
(**flexibility** )

*What if my data entry was wrong*
*customer was female instead of male?*

(**data duplication** )
- hard to update
- data often conflicts

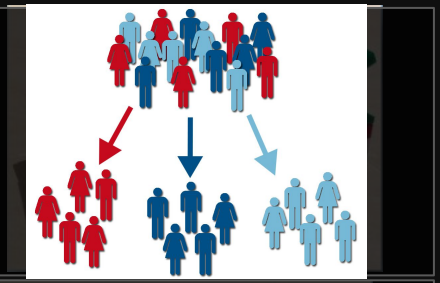
Same issue with "cars" as in the real-world, but potentially worse if incorrect data has then been used and **copied** into lots of different reports…




Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012

# Summary:

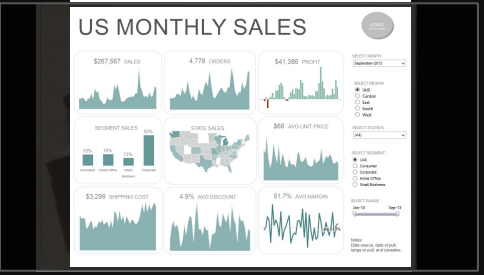
Storing complex structured data & having to break it into pieces & rebuild things is hard.



*Better just to store the pieces and build instructions.*

Storing multiple copies so you don't have to break the original into pieces can duplicate data - introduces potential for error.

*Better just to store the pieces and build instructions.*


Storing pieces and build instructions can be slower if you just want to ask about cars.

*Are you sure you're only ever going to want to show a single report?*
Is the speed gain worth it?

Images from:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012


So maybe we do want this lego style thing…

So what are these lego pieces anyway?

Data is Latin for “facts”

Turns out (rather than objects) **facts** make good basis (lego) pieces to construct things (i.e. cars, reports).

So what are these lego pieces anyway?

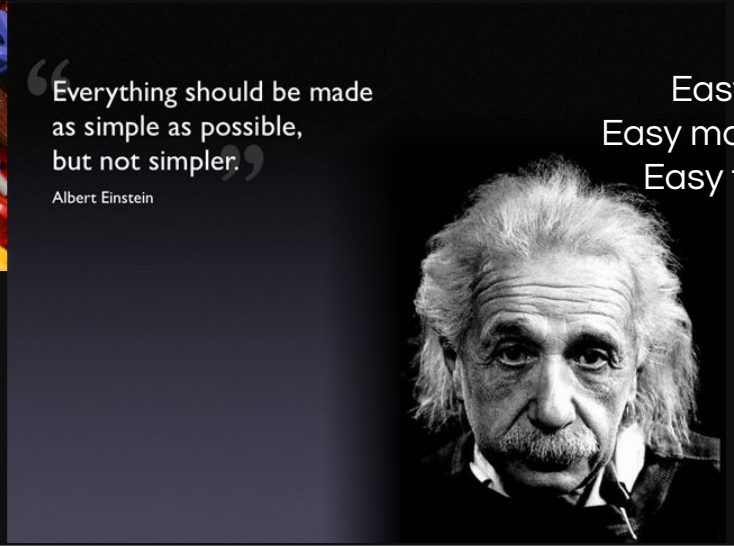
Why?

- Facts are either true or false.


"A Dairy Milk costs £2.50"
"Evgeniya lives in Nottingham"

Data is Latin for “facts”

Turns out (rather than objects) **facts** make good basis (lego) pieces to construct things (i.e. cars, reports).




Binary.
Easy for humans.
Easy mathematically.
Easy for machines.

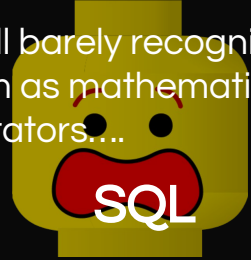
So what are these lego pieces anyway?

Data is Latin for “facts”

Turns out (rather than objects) **facts** make good basis (lego) pieces to construct things (i.e. cars, reports).

**~8 Mathematical operators** can be used to do almost all required data manipulation

You'll barely recognise them as mathematical operators....





Interested in only a few chocolate bars?

Filter based on fact x being true...

So what are these lego pieces anyway?

Data is Latin for “facts”

Turns out (rather than objects) **facts** make good basis (lego) pieces to construct things (i.e. cars, reports).

**Simple set of operators, binary facts = well understood mathematical theory**

↓

**Translation from operators** (easy for humans) → **computer code is well understood.**

↓

Can directly use operators.

Describe what we want (ask questions), not how to do it (list steps).

Computer will (mostly) automatically optimise.

So what are these lego pieces anyway?

But wait, there's more...

Good base paradigm + development since the 70's mean....

Sets of facts can be designed to **prevent data duplication.**

Multiple people building reports from a single source of up-to-date data **ensures consistency as data changes.**

Data is Latin for “facts”

Turns out (rather than objects) **facts** make good basis (lego) pieces to construct things (i.e. cars, reports).

Sets of facts can have **some constraints** (basic business rules) **enforced on entry.**

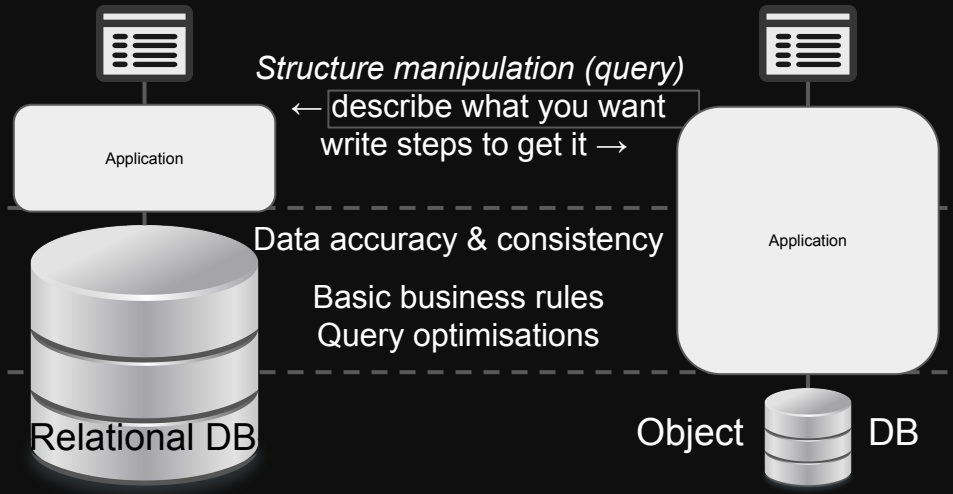
# What does that mean in practice?

Reduced development time.

Less errors.

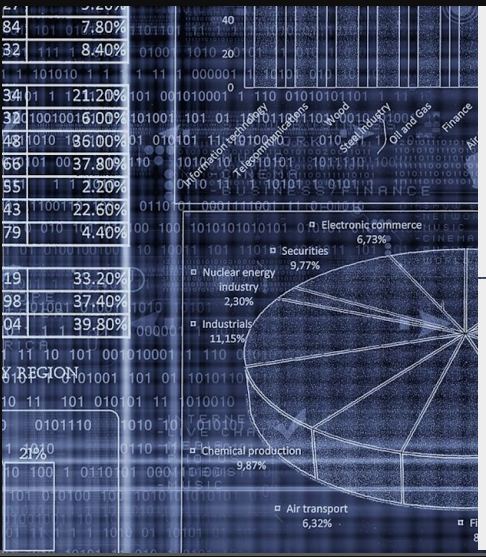
"... we find developers spend a significant fraction of their time building extremely complex and error-prone mechanisms to cope with eventual consistency and handle data that may be out of date. We think this is an unacceptable burden to place on developers and that consistency problems should be solved at the database level."

[Google] Shute, Jeff, et al. "F1: A distributed SQL database that scales." Proc. VLDB (2013): 1068-1079.



Sets of facts == relational databases!!

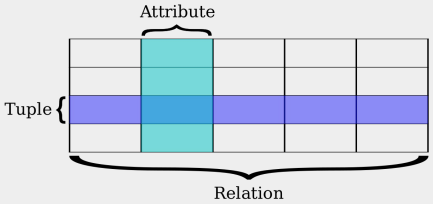
Lots more in a week…..




→ Replaced “Navigational Databases” in the late 70’s, because of E.F. Codd's work at IBM.

→ Based on **Tables**, **columns** and **rows**.

→ Or more accurately **relations**, **attributes** and **tuples**.

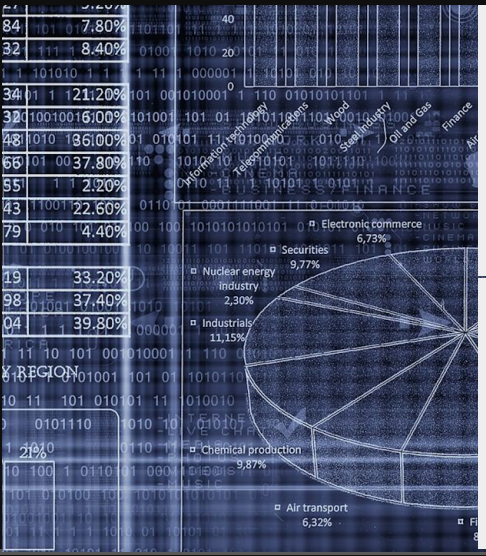




N LAB : *Data at Scale*

Dr Evgeniya Lukinova

# Data is Latin for “facts”




Kermit is a green Frog.
Miss Piggy is a pink pig.
Fozzy is an orange bear.

**Name** (Kermit) & **Colour** (Green) & **Animal** (Frog)
**Name** (Miss Piggy) & **Colour** (Pink) & **Animal** (Pig)
**Name** (Fozzy) & **Colour** (Orange) & **Animal** (Bear)

Relation header = (Name, Colour, Animal)
Relation content = { (Kermit, Green, Frog),
(Miss Piggy, Pink, Pig),
(Fozzy, Orange, Bear) }

| Name | Colour | Animal |
|---|---|---|
| Kermit | green | frog |
| Miss piggy | pink | pig |
| Fozzy | orange | bear |


N LAB: *Data at Scale*

Dr Evgeniya Lukinova

# Relations (Tables) are lists of facts about different <table name>

| Name | Colour | Animal |
|---|---|---|
| Kermit | green | frog |
| Miss piggy | pink | pig |
| Fozzy | orange | bear |

Each row (tuple) refers to facts about a single entity*

*need not be an object!

## Example: Shop receipts

Receipt Table (Relation)

| Receipt ID | Cust ID | Date | Shop | Total |
|---|---|---|---|---|
| 0001 | 0001 | 1/1/17 | NC1N 2HT | £1.70 |

Facts (about a receipt)

Receipt Line Table (Relation)

| Receipt ID | Line no. | Desc. | Qty | Amt |
|---|---|---|---|---|
| 0001 | 0001 | JS CARROTS 1.5KG | 1 | £0.70 |
| 0001 | 0002 | THE TIMES | 1 | £1.00 |

Facts (about a line on a receipt)

## The take-home message (long version)….

## +ve & -ve for objects as a storage paradigm
(vs. relational databases)

Data access is simple when processing objects as stored & if objects have identical/correct structure & data is error free (or you don't care)

Images:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012
Source:
https://softwareefficiency.wordpress.com/2015/03/14/big-data-technology-and-the-responsibility-shift/

# The take-home message (long version)....

## +ve & -ve for objects as a storage paradigm
(vs. relational databases)

Storing objects can be fast 1
dump objects (cars) one at a time as they turn up

Data access is simple when processing objects as stored 2 & if objects have identical/correct structure & data is error free (or you don't care) 3

**If true...**
- Fast, no need for "rebuilding".
- Potentially conceptually simpler

**If false...**
- Complex language and logical structure to navigate for rebuilding.
- Less automatic optimisations due to paradigm & reduced structure

**If true...**
- **no time** spent on business rule checks
- business rules can easily evolve
- good custom code c**an be faster**

**If false...**
- **re-inventing code** to check business rules in each application or not checking
- custom code is likely **slower / prone to errors**

**If true...**
- **no time** spent on error checks 4

**If false...**
- **re-inventing code** to check errors in each application or not checking
- custom code is likely **slower / has errors**

Images:
http://nosqlroadshow.com/nosql-london-2012/speaker/Akmal+B.+Chaudhri
https://www.slideshare.net/VeryFatBoy/nosql-roadshow-london-2012
Source:
https://softwareefficiency.wordpress.com/2015/03/14/big-data-technology-and-the-responsibility-shift/

**The take-home message (short version)….**

Normally ➡ Use relational databases

Unless….

Too much data makes checks / writes/access too slow.

Data is always in, and processed in, objects*.

Data structure changes constantly.

➡ May store as objects
(or another logical structure)

**The take-home message (short version)….**

Normally ➡ Use relational databases

Unless….

Too much data makes checks / writes/access too slow.

Data is always in, and processed in, objects*.

Data structure changes constantly.

➡ May store as objects

(more likely another logical structure or potentially in "best of both worlds" databases - covered later)

**The take-home message (short version)….**

# Normally

Use relational databases

Otherwise

Too much data makes checks / writes/access to slow.

Data is always in, and processed in, objects*.

Data structure changes constantly.

May store as objects

(more likely another logical structure or potentially in "best of both worlds" databases - covered later)

noSQL
newSQL
sparkSQL

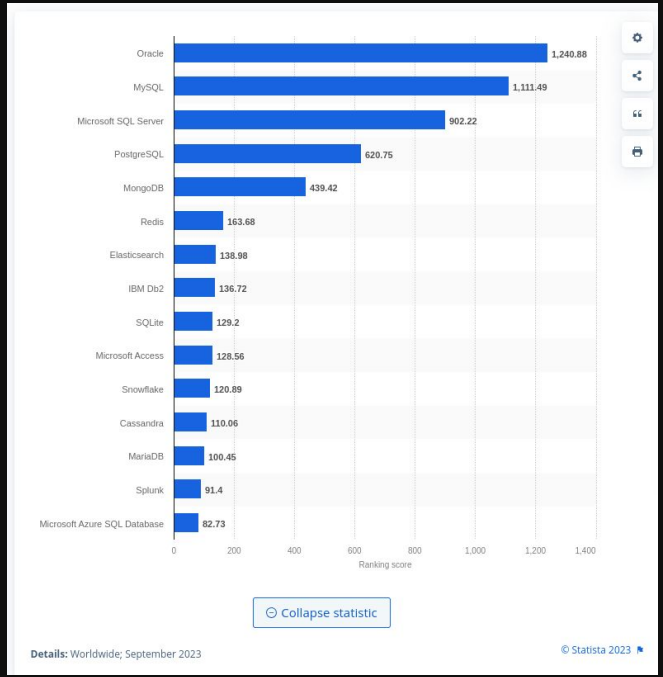



Image: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/

# Context is everything...

(so let's take a look)

Pick the processing & storage **paradigm*** for the job.

(* we now know two!)

Is traditional (relational) approach enough?

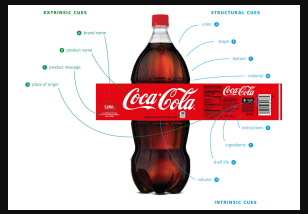
Transaction data

Google Analytics

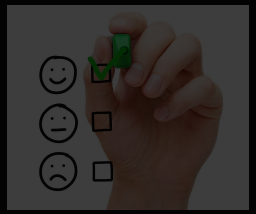
mouseflow

Website tracking data

Security logs




Product Detail Data

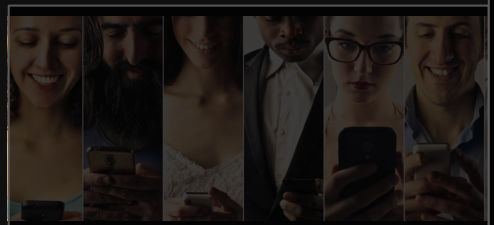


Survey Data

Office for National Statistics

census 2021

Census Data



Call Detail Record (CDR) Data

Social Media Data

# Transactional data

(Event data)

Pick the processing & storage **paradigm** for the job.

- Payment data
- Returns
- Loyalty card data

Is traditional approach enough?

- Signups
- Subscriptions
- Reservations
- Lending
- Trades

# Transactional data

(Event data)

**A case study:** Till transactions


Seem simple enough...

Description qty amount
Description qty amount
Description qty amount
....
Total, cash taken, change

## What about credit cards?

Seem a little more complex...

Description qty amount
Description qty amount
Description qty amount
....
Total, payment type, cash taken, card type, card details (number?)

## What about promotions?

Description qty amount
item saving qty -amount
Description qty amount
Description qty amount
....
Total, payment type, cash taken, card type, card details (number?)

Description qty amount
Disc. Item qty disc_amt
Description qty amount
....
Total, payment type, cash taken, card type, card details (number?)

## What about multi-buys/meal deals?

Description qty amount
Description qty amount
Description qty amount
Disc. Item qty disc_amt
item saving qty -amount
multi buy disc. qty -amount
....
Total, payment type, cash taken, card type, card details (number?)

## What about prepaid items?

Description qty amount
Description qty amount
Description qty amount
Disc. Item qty disc_amt
item saving qty -amount
multi buy disc. qty -amount
Description qty 0.00
....
Total, payment type, cash taken, card type, card details (number?)

## What about refunds?

Description qty amount
Description qty amount
Description qty amount
Disc. Item qty disc_amt
item saving qty -amount
multi buy disc. qty -amount
Description qty 0.00
Description qty -amount
....
Total, payment type, cash taken, card type, card details (number?)




# Transactional data

(Event data)

**A case study:** Till transactions

Description qty amount
Description qty amount
Description qty amount
Disc. Item qty disc_amt
item saving qty -amount
multi buy disc. qty -amount
Description qty 0.00
Description qty -amount
....
Total, payment type, cash taken, card type, card details (number?)

**Actually looks like**

Description qty amount
Description qty amount
Description qty amount
Description qty amount
Description qty -amount
Description qty -amount
Description qty 0.00
Description qty -amount
....
Total, payment type, cash taken, card type, card details (number?)

**Take home message:** In real world analytics understanding the business practice and process is **exceptionally important.**

**Standard potentially incorrect assumption:**
negative values are refunds.

**Garbage in, garbage out.**

# Transactional data

(Event data)

**A case study:** Till transactions

**Refunds, multibuys, promos, prepaid**

Description qty amount
Description qty amount
Description qty amount
Disc. Item qty disc_amt
item saving qty -amount
multi buy disc. qty -amount
Description qty 0.00
Description qty -amount
....

Total, payment type, cash taken, card type, card details (number?)

**Actually looks like**

Description qty amount
Description qty amount
Description qty amount
Description qty amount
Description qty -amount
Description qty -amount
Description qty 0.00
Description qty -amount
....

Total, payment type, cash taken, card type, card details (number?)

**So more info is recorded**

Description qty amount refund_flag collection_flag
Description qty amount refund_flag collection_flag
Description qty amount refund_flag collection_flag
Description qty amount refund_flag collection_flag
Description qty -amount refund_flag collection_flag
Description qty -amount refund_flag collection_flag
Description qty 0.00 refund_flag collection_flag
Description qty -amount refund_flag collection_flag
....

Total, payment type, cash taken, card type, card details (number?)

**But wait, there's more...**

What about loyalty points?

Payment by part card, part cash?

Payment part by points?

CLUBCARD STATEMENT

CLUBCARD NUMBER [redacted]
POINTS THIS VISIT 21
INCLUDES :
DOUBLE POINTS 10
GREEN CLUBCARD BAG RE-USE 1
TOTAL UP TO 10/05/11 6/8
TOTAL INCLUDES :
GREEN CLUBCARD POINTS 8

What about discounts from loyalty card coupons?

What about recording an identifier for the promotion?

***In real world analytics understanding the business practice and process is exceptionally important.***

Let's take a look at some real world examples of what is recorded....

**and more...**

# Transactional data

(Event data)

**Back to receipts!**

We really saw a simplistic version.

**More realistic**
(just Receipt Table)

**Refunds, multibuys, promos, prepaid**

Description qty amount
Description qty amount
Description qty amount
Disc. Item qty disc_amt
item saving qty -amount
multi buy disc. qty -amount
Description qty 0.00
Description qty -amount
....
Total, payment type, cash taken, card type, card details (number?)

## Example: Shop receipts

Receipt Table (Relation)

| Receipt ID | Cust ID | Date | Shop | Total |
|---|---|---|---|---|
| 0001 | 0001 | 1/1/17 | NC1N 2HT | £1.70 |

Facts (about a receipt)

Receipt Line Table (Relation)

| Receipt ID | Line no. | Desc. | Qty | Amt |
|---|---|---|---|---|
| 0001 | 0001 | JS CARROTS 1.5KG | 1 | £0.70 |
| 0001 | 0002 | THE TIMES | 1 | £1.00 |

Facts (about a line on a receipt)

| Field Name | Description | Example |
|---|---|---|
| loyalty_number | Customer number (may not exist) | *783252* |
| epos_transaction_id | Unique transaction ID? | *43586329* |
| till_transaction_type_code | Sale (0), Refund (1), Cancellation (2)…. | *1* |
| reciept_date | Date of transaction | *2017-04-17* |
| reciept_time | Time of transaction | *14:46:00* |
| staff_id | Identifier of the staff member that served the | *87* |
| discount_card_number | card number where applicable - otherwise 0. | *0* |
| store_number | Identifier of the store | *313* |
| points_change | Loyalty card points earned or spent | *28* |
| number_of_deals | Number of deals in basket | *0* |
| total_inc_vat | Total spend including VAT | *2.99* |
| total_exc_vat | Total spend before VAT | *2.4916* |
| sales_units | Number of items in basket. This can be negative (due to refunds). | *1* |
| deal_savings_local | Total savings due to store level deals | *0.00* |
| deal_savings_global_promo | Total savings due to company wide promotions | *0.00* |
| savings_coupons | Total savings due to redeemed coupons | *0.00* |
| register_number | The register that the transaction was done on | *32* |
| payment_card | Payment amount done on card | *2* |
| payment_cash | Payment amount taken in cash | *5* |
| bar_code | Bar code printed on receipt | *6686586581583500135* |

# Transactional data

(Event data)

## Example: Shop receipts

Receipt Table (Relation)

| Receipt ID | Cust ID | Date | Shop | Total |
|---|---|---|---|---|
| 0001 | 0001 | 1/1/17 | NC1N 2HT | £1.70 |

Facts (about a receipt)

| Field Name | Description | Example |
|---|---|---|
| **loyalty_number** | Customer number (may not exist) | *783252* |
| **epos_transaction_id** | Unique transaction ID? | *43586329* |
| **till_transaction_type_code** | Sale (0), Refund (1), Cancellation (2)…. | *1* |
| **reciept_date** | Date of transaction | *2017-04-17* |
| **reciept_time** | Time of transaction | *14:46:00* |
| **staff_id** | Identifier of the staff member that served the | *87* |
| **discount_card_number** | card number where applicable - otherwise 0. | *0* |
| **store_number** | Identifier of the store | *313* |
| **points_change** | Loyalty card points earned or spent | *28* |
| **number_of_deals** | Number of deals in basket | *0* |
| **total_inc_vat** | Total spend including VAT | *2.99* |
| **total_exc_vat** | Total spend before VAT | *2.4916* |
| **sales_units** | Number of items in basket. This can be negative (due to refunds). | *1* |
| **deal_savings_local** | Total savings due to store level deals | *0.00* |
| **deal_savings_global_promo** | Total savings due to company wide promotions | *0.00* |
| **savings_coupons** | Total savings due to redeemed coupons | *0.00* |
| **register_number** | The register that the transaction was done on | *32* |
| **payment_card** | Payment amount done on card | *2* |
| **payment_cash** | Payment amount taken in cash | *5* |
| **bar_code** | Bar code printed on receipt | *6686586581583500135* |

Receipt Line Table (Relation)

| Receipt ID | Line no. | Desc. | Qty | Amt |
|---|---|---|---|---|
| 0001 | 0001 | JS CARROTS 1.5KG | 1 | £0.70 |
| 0001 | 0002 | THE TIMES | 1 | £1.00 |

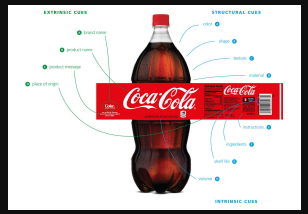
Facts (about a line on a receipt)

| Field Name | Description | Example |
|---|---|---|
| **loyalty_number** | Customer number (may not exist) | *783252* |
| **transaction_id** | Unique transaction ID | *43586329* |
| **item_id** | Unique item identifier | *741321* |
| **store_number** | Unique store ID | *96* |
| **reciept_date** | Date of transaction | *2017-04-17* |
| **reciept_time** | Time of transaction | *14:46:00* |
| **transaction_type** | Sale (0), Refund (1), Cancellation (2)…. | *1* |
| **qty** | Quantity of this item purchased | *2* |
| **total_inc_vat** | Receipt line total including VAT | *4.29* |
| **total_exc_vat** | Receipt line total excluding VAT | *3.6508* |
| **discount** | Amount the item was discounted | *0.0* |
| **refund_flag** | Is this being refunded Y/N | *Y* |
| **price_overridden** | Was the price overridden by the register staff? Y or N | *N* |
| **total** | Total amount for the line on the receipt | *0.0000* |
| **loyalty_points** | Total loyalty points (may be negative) for this receipt line | *0.0000* |
| **sap_sales_at_tisp** | | *0.0000* |

# Transactional data

(Event data)

Large retailer
~ 250 million transactions per year within the UK




Product Detail Data

## Example: Shop receipts

Receipt Table (Relation)

| Receipt ID | Cust ID | Date | Shop | Total |
|---|---|---|---|---|
| 0001 | 0001 | 1/1/17 | NC1N 2HT | £1.70 |

Facts (about a receipt)

Receipt Line Table (Relation)

| Receipt ID | Line no. | Desc. | Qty | Amt |
|---|---|---|---|---|
| 0001 | 0001 | JS CARROTS 1.5KG | 1 | £0.70 |
| 0001 | 0002 | THE TIMES | 1 | £1.00 |

Facts (about a line on a receipt)


Other directly related tables:

- Customer
- Item
- Store
- Payment information

Other indirectly related tables:

- Brand
- Sub Brand
- Category

Other complications:

- information change over time (changes in facts)
  hard for (any) paradigm to handle efficiently
  (correction vs updates?)

# Context is everything...

(so let's take a look)

Pick the processing & storage **paradigm*** for the job.

(* we now know two!)

Is a traditional (relational) approach enough?

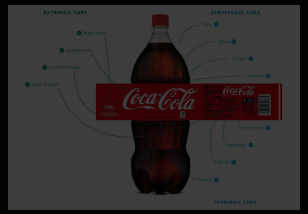
Transaction data

Google Analytics

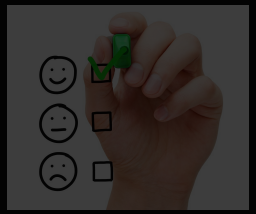
mouseflow

Website tracking data

Security logs



Product Detail Data

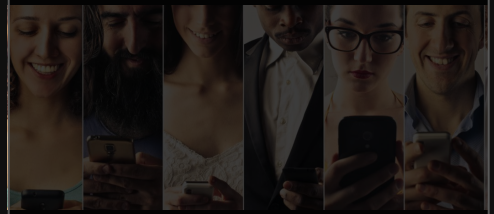


Survey Data

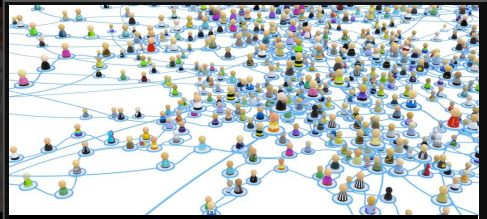
Office for National Statistics

census 2021

Census Data

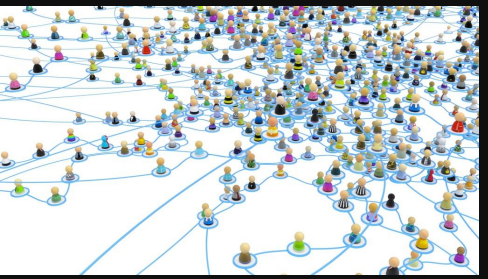


Call Detail Record (CDR) Data



Social Media Data

# Twitter/X data



## Standard Tweets

coordinates
favorited
truncated
created_at
id_str
entities
- urls
- Hashtags
- user_mentions

in_reply_to_user_id_str
contributors
text
retweet_count
place
user [....]
source

Total Number of Tweets sent per Day:

500 million

Last updated: 03/06/23

## Extra information from IBM Watson Analytics

| | |
|---|---|
| Sentiment | +ve,-vd, neutral, ambivalent, unknown |
| Sentiment positive signals | ["great", "awesome"] |
| Sentiment negative signals | ["poor", "terrible"] |

IBM Watson

## Direct messages

id
created_at
recipient_id
recipient_screen_name
sender_id
sender_screen_name
text
entities [hashtags, urls, user_mentions]

recipient
- id
- created_at
- description
- followers_count
- profile_image_url
- geo_enabled
- verified
- … (38 total)

sender
- id
- created_at
- description
- followers_count
- profile_image_url
- geo_enabled
- verified
- … (38 total)

# Context is everything...

(so let's take a look)

Pick the processing & storage **paradigm*** for the job.

(* we now know two!)

Is a traditional (relational) approach enough?

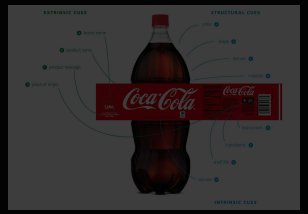
Transaction data

Google Analytics

mouseflow

Website tracking data

Security logs



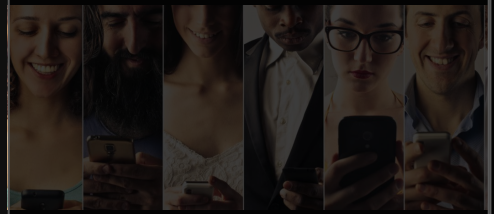
Product Detail Data

Survey Data

Office for National Statistics

census 2021

Census Data



Call Detail Record (CDR) Data

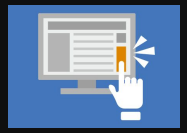
Social Media Data

N/LAB: *Data at Scale*

Dr Evgeniya Lukinova

# Web tracking data

(Event data)



Website tracking data

**Server generating the page can track what it sends.**

**Code running in browsers can send logs back.**

referrer_url
page_url
location
IP
Browser
Operating System
Language settings
Screen resolution

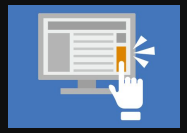
GUID (via cookie, javascript etc)

**Optional, if event:**
click information (what clicked, etc)
form submission (event, success code)
mouse movement (I.e. mouseflow)

**Pretty well everything can be recorded**

# Web tracking data

(Event data)



Website tracking data

**Server generating the page can track what it sends.**

**Code running in browsers can send logs back.**

referrer_url
page_url
location
IP
Browser
Operating System
Language settings
Screen resolution

GUID (via cookie, javascript etc)

**Optional, if event:**
click information (what clicked, etc)
form submission (event, success code)
mouse movement (I.e. mouseflow)

**Pretty well everything can be recorded**

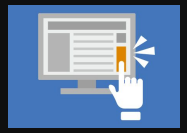
Track sessions

- GUID
- User login

**Challenge:**
Multiple tabs, non-linear navigation flow

# Web tracking data

(Event data)



Website tracking data

**Server generating the page can track what it sends.**

**Code running in browsers can send logs back.**

referrer_url
page_url
location
IP
Browser
Operating System
Language settings
Screen resolution

GUID (via cookie, javascript etc)

**Optional, if event:**
click information (what clicked, etc)
form submission (event, success code)
mouse movement (I.e. mouseflow)

**Pretty well everything can be recorded**

Google Analytics

Track sessions
- GUID
- User login

**Challenge:**
Multiple tabs, non-linear navigation flow

Dwell time

Bounce rate (visit 1 page only)

Conversion rate (all vs. geo vs. …)

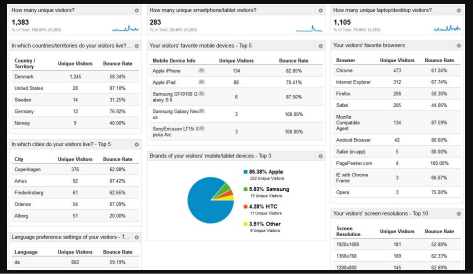
Traffic source statistics

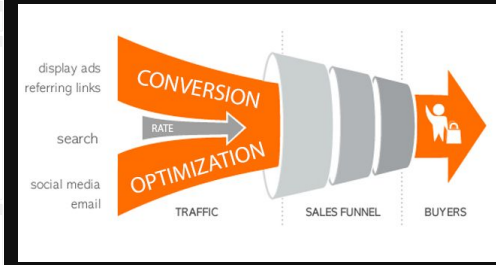
Unique vs return visitors

Traffic flow (cart abandonment)

Search terms on site

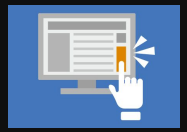
Percentage of desktop/mobile use





# Web tracking data

(Event data)



Website tracking data

**Server generating the page can track what it sends.**

**Code running in browsers can send logs back.**

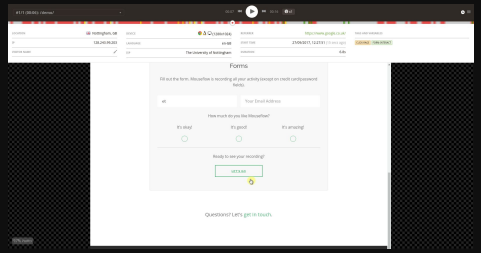
referrer_url
page_url
location
IP
Browser
Operating System
Language settings
Screen resolution

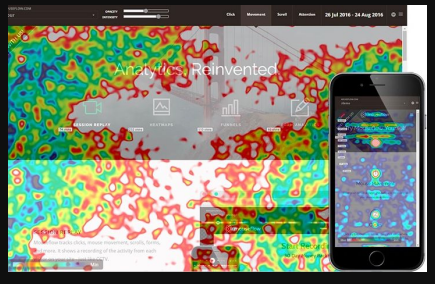
GUID (via cookie, javascript etc)

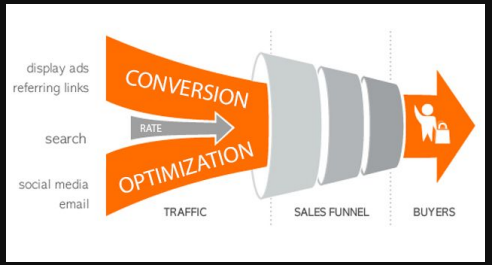
**Optional, if event:**
click information (what clicked, etc)
form submission (event, success code)
mouse movement (I.e. mouseflow)

**Pretty well everything can be recorded**

mouseflow






Site design
- engagement
- promotion placement
- page simplification

Retention analysis & redesign

Conversion analysis & redesign

Dwell time++

Ad campaign evaluation

N/LAB: *Data at Scale*

Dr Evgeniya Lukinova

# Context is everything...

(so let's take a look)

Pick the processing & storage **paradigm*** for the job.

(* we now know two!)

Is a traditional (relational) approach enough?

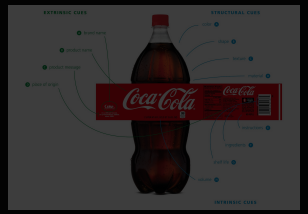
Transaction data

Google Analytics

mouseflow

Website tracking data

Security logs



Product Detail Data

Survey Data

Office for National Statistics

census 2021

Census Data

Call Detail Record (CDR) Data

Social Media Data

# Call Detail Record (CDR) Data (More event data...)

| Initiating subscriber ID | Unique BTS ID | Timestamp |
|---|---|---|
| *jggsmi13227abc* | *12038097523* | *14-03-2014 00:01:12* |

Calls

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

duration
charge
serviceflow
roaming
result code
**incoming**
...

# Call Detail Record (CDR) Data (More event data...)

| Initiating subscriber ID | Unique BTS ID | Timestamp |
|---|---|---|
| *jggsmi13227abc* | *12038097523* | *14-03-2014 00:01:12* |

Calls

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

duration
charge
serviceflow
roaming
result code
**incoming**
...

SMS

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

charge
serviceflow
roaming
result code
incoming
**sms length**
...

# Call Detail Record (CDR) Data (More event data...)

| Initiating subscriber ID | Unique BTS ID | Timestamp |
|---|---|---|
| *jggsmi13227abc* | *12038097523* | *14-03-2014 00:01:12* |

## Calls

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

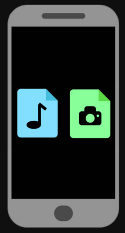
duration
charge
serviceflow
roaming
result code
**incoming**
...

## SMS

start time
tower id
caller id
called id
tariff type
prepaid balance
product id
...

charge
serviceflow
roaming
result code
incoming
**sms length**
...

## Data



start time
tower id
caller id
tariff type
prepaid balance
charge

product id
duration
**data up**
**data down**
...

# Context is everything...

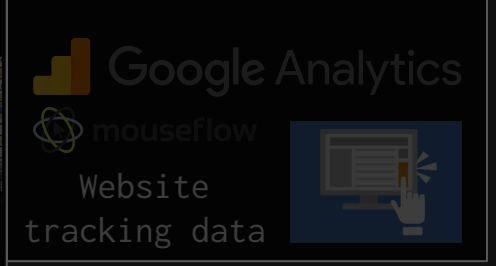
(so let's take a look)

Pick the processing & storage **paradigm*** for the job.

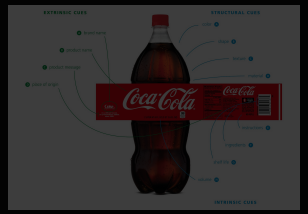
(* we now know two!)

Is a traditional (relational) approach enough?

Transaction data

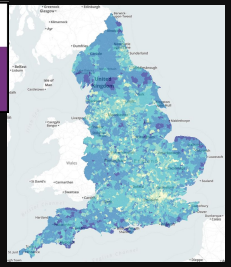



Security logs

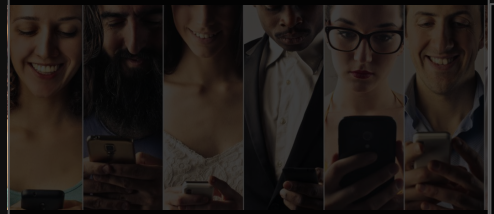


Product Detail Data

Survey Data

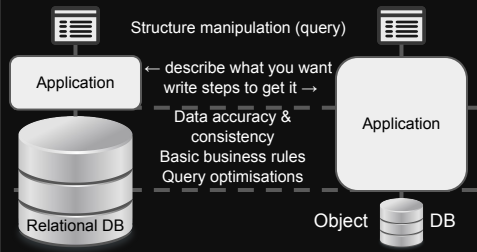



Census Data



Call Detail Record (CDR) Data

Social Media Data




N/LAB: *Data at Scale*

Dr Evgeniya Lukinova

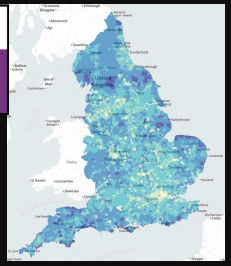
# To summarize...

Pick the processing & **storage paradigm*** for the job.
(* we now know two!)

Survey Data

Security logs

Office for National Statistics
census 2021



Census Data

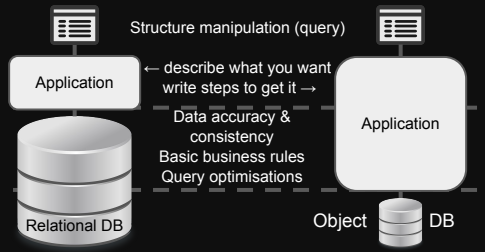
Facebook

Credit Card Transactions

Pratt & Whitney
A United Technologies Company

Plane Engine Maintenance

Some hints:
(from prior slides)




Are we going to process objects as stored & if objects have identical/correct structure & data is error free (or we do not care)?

Use relational DBs unless...

- Too much data makes checks /access too slow.
- Data is always in, and processed in, objects*.
- Data structure changes constantly.