Agenda

09:15 - 10:45 *Keynote: Understanding MongoDB and Document Modeling*

10:45 - 11:00 15 minute break

11:00 - 12:30 *Developer Data Platform (Atlas)*

12:30 - 13:00 *Lunch*

13:00 - 14:00 *MongoDB 201 - Beyond the Basics*

14:00 - 14:15 *15 minute break*

14:15 - 15:15 *Aggregation Framework - analyzing and summarizing data inside MongoDB*

15:15 - 15:30 *Wrap up // MongoDB University*

MongoDB at Morgan Stanley

Morgan Stanley

2012

- Start-up
- < $30m revenue
- $231m in funding ($150m just raised)
- Support and training services
- 1500 attendees at the inaugural MongoDB World

2016

- **Market capitalisation:** <$1b
- **Revenue:** $65.3m
- **Employees:** 620
- **Customers:** 2,800
- **Downloads:** 13m
- MongoDB Atlas launched

2025

- **Market capitalisation:** $21b
- **Revenue:** $2.5b
- **Employees:** 5,500+
- **Customers:** 53,000+
- **Downloads:** 500m+

Who Uses MongoDB at Morgan Stanley?

ESTAR: SEM

100+ shards, 5 nodes per shard, 2.5TB RAM per node
462TB data uncompressed
On-premises

SEM was conceived in 2012 and is Morgan Stanley's swap execution platform for Morgan Stanley's clients. Today, 100% of client swaps leverage the SEM platform, with the remainder due to be migrated to SEM.

FIDCOM: DAL

150TB
85TB per week copied to Reg environments
Hybrid Multi-Cloud Architecture
(AWS/Azure/On-premises)

Many teams across Morgan Stanley leverage the DAL for their use cases. Since 2014, DAL has leveraged MongoDB Enterprise Advanced to provide a centrally managed data framework with a consistent access pattern for OTC data to ISG users globally across the front, middle and back office.

ESTAR: ERS

Continuously upload data throughout the day
MongoDB Atlas on AWS - multi-region

Part of the RiskViewer application, MongoDB provides an inputs repository for position data to feed calculation engines for risk and scenarios.

Risk View Platform now moving over

IST: Risk Technology: MDX (Framehub)
IST: ESTAR: Datazone
IST: Advisory & Sales Distribution: Azure Research Platform
IST: ESTAR: Carbon
IST: TEDRA: SRD Data Fabric
FRPPE: CDRT: CRD
IST: TEDRA: HALO
IST: TEDRA: Vista

IST: CRM Apps: Rolodex
IST: Advisory & Sales Distribution: POET
IST: PWM: Helios
IST: PWM: Client Online
IMIT: Parametric: Reconciliations
WMT: E*Trade: Monitoring Dashboard
IST: ESTAR: Taurus
FRPPE: Risk Tech: RAPD Maiden

Welcome

Understanding MongoDB and Document Schema Design

Andrew Morgan
Senior Staff Developer Advocate
clusterdb.com

Agenda

What is MongoDB?

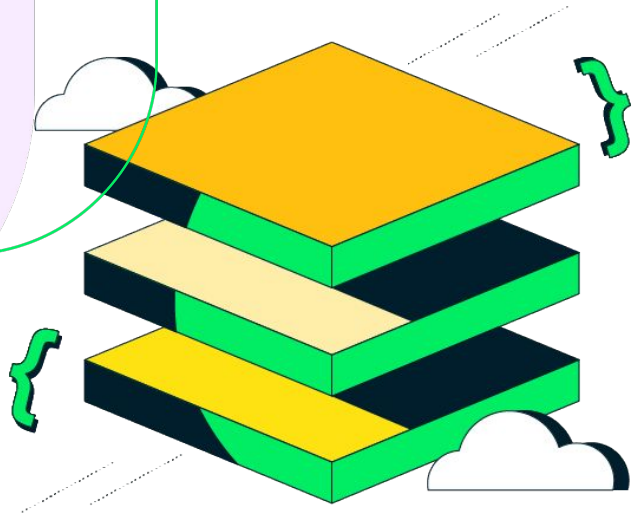
What makes a schema good?

RDBMS to MongoDB in 8 steps

Schema optimization techniques

Five common mistakes

What is MongoDB?



Modern Document Database

What is a database?

A structured form of data storage that abstracts away low level detail and allows us to efficiently retrieve and update data whilst avoiding data loss from race conditions or invalid modifications.

Traditionally databases are tabular and embrace 3rd normal form.

What is a database?

A structured form of data storage that abstracts away low level detail and allows us to efficiently retrieve and update data whilst avoiding data loss from race conditions or invalid modifications.

Traditionally databases are tabular and embrace 3rd normal form.

What is a database?

A structured form of data storage that **abstracts away low level detail** and allows us to efficiently retrieve and update data whilst avoiding data loss from race conditions or invalid modifications.

Traditionally databases are tabular and embrace 3rd normal form.

What is a database?

A structured form of data storage that abstracts away low level detail and allows us to **efficiently retrieve and update data** whilst avoiding data loss from race conditions or invalid modifications.

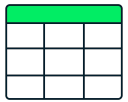
Traditionally databases are tabular and embrace 3rd normal form.

What is a database?

A structured form of data storage that abstracts away low level detail and allows us to efficiently retrieve and update data whilst **avoiding data loss from race conditions or invalid modifications**.

Traditionally databases are tabular and embrace 3rd normal form.

Why Normalise Data?

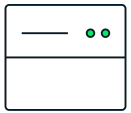


Single database

Ensure one data model meets all possible current and future use cases across your organisation

Implicit consistency

Ensure there is no possibility of consistency errors as nothing is ever duplicated



Minimal disk

Minimize disk storage

Abstract implementation

Limit what developers need to do if the storage changes

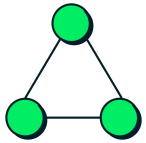
Why change after 50 years?

Expensive model

It's inefficient and slow - meaning expensive to run

Cloud pricing

We don't buy big up-front servers anymore



Enable distribution

It's a model that doesn't work well when distributed

Changes are slow

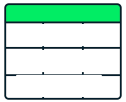
It can (ironically) be inflexible to change

> With MongoDB you are designing an optimal schema for your use cases - not a generic schema for every possible use case.

What does MongoDB do differently?

Sub documents

Fields can be grouped for retrieval and also query scope



Arrays

You can essentially co-locate rows from one table in another - join on write

Flexible schema

The ability to modify schema cheaply, without changing existing data

Object-based API

Interact via code not via text. Persist objects; query, update and aggregate through object-oriented APIs

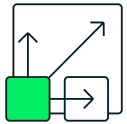
Why those changes matter

More efficient

Perform more transactions per second with less hardware

More productive

Developers find it easier and therefore can iterate faster using objects



More scalable

Easier to solve big data problems by adding hardware

$350k+ saved in year one

Transitioning from Oracle is already seeing a storage savings of $31k per month

15x faster performance

Capturing payment status in 5 milliseconds allows the bank to avoid fines & retain customers

99.999% uptime

MongoDB supports the most critical workloads (C1) at the bank

Elasticity scales with demand

10 million database transactions per day, and this gives confidence the bank is building on the right platform for the future

Documents

A row by any other name…

Data in an RDBMS is stored as rows and columns in tables

CustomerId	CustomerName	Address_Num	Address_St	Address_Town	Address_Area	Address_Code
1325142	Mark Bolan	15	Green road	London	Middlesex	E2 7OF

We could visualise that row as CSV

CustomerId	CustomerName	Address_Num	Address_St	Address_Town	Address_Area	Address_Code
1325142	Mark Bolan	15	Green road	London	Middlesex	E2 7OF

```
CustomerId,CustomerName,Address_Num,Address_St,Address_Town,Adress_Area,Address_Code
1325142,"Mark Bolan",15,"Green road","London","Middlesex","E2 7OF"
```

We could also visualise that row as JSON - that doesn't make it JSON

CustomerId	CustomerName	Address_Num	Address_St	Address_Town	Address_Area	Address_Code
1325142	Mark Bolan	15	Green road	London	Middlesex	E2 7OF

```
{
  "CustomerId": 1325142,
  "CustomerName": "Mark Bolan",
  "Address_Num": 15,
  "Address_St": "Green road",
  "Address_Town": "London",
  "Address_Area": "Middlesex",
  "Address_Code": "E2 7OF"
}
```

In MongoDB, Document is also a field type, like Date, Integer or Double

CustomerId	CustomerName	Address				
		Num	St	Town	Area	Code
1325142	Mark Bolan	15	Green road	London	Middlesex	E2 7OF

```
{
  "CustomerId": 1325142,
  "CustomerName": "Mark Bolan",
  "Address" : {
              "Num": 15,
              "St": "Green road",
              "Town": "London",
              "Area": "Middlesex",
              "Code": "E2 7OF"
             }
}
```

Where you have multiple values in an RDBMS it needs two tables

CustomerId	Name	CustomerSince
1325142	"Mark Bolan"	"1978-01-03"
1276191	"Sarah O'Brien"	"1996-05-08"
3416713	"Lee Wan"	"2023-03-16"

CustomerId	Phone Type	Phone Number
1325142	Cellphone	077652441661
3416713	Office	01698427887
1325142	Office	02034558982
1276191	Cellphone	07979554414
3416713	Cellphone	07623481631
1325142	Cellphone	07713165342

Using Arrays we co-locate rows from two tables in storage and cache

CustomerId	Name	CustomerSince	Phone (Array)	
			Type	Number
1325142	"Mark Bolan"	"1978-01-03"	Cellphone	077652441661
			Office	02034558982
			Cellphone	07713165342
1276191	"Sarah O'Brien"	"1996-05-08"	Cellphone	07979554414
3416713	"Lee Wan"	"2023-03-16"	Cellphone	07623481631
			Office	01698427887

Using Arrays we co-locate in storage and cache

CustomerId	Name	CustomerSince	Phone (Array)	
			Type	Number
1325142	"Mark Bolan"	"1978-01-03"	Cellphone	077652441661
			Office	02034558982
			Cellphone	07713165342

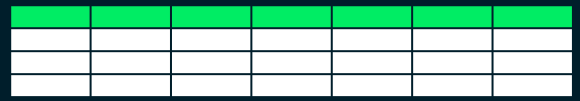
```
{ "CustomerId": 1325142,
  "Name": "Mark Bolan",
  "CustomerSince": "1978-01-03T00:00:00Z",
  "Phone": [
             { "Type": "Cellphone", "Number" :077652441661 },
             { "Type": "Office", "Number" :02034558982 },
             { "Type": "Cellphone", "Number" :07713165342 },
   ]
}
```

Prejoin Rows

Store an object as a single row

Access as Object OR Tables

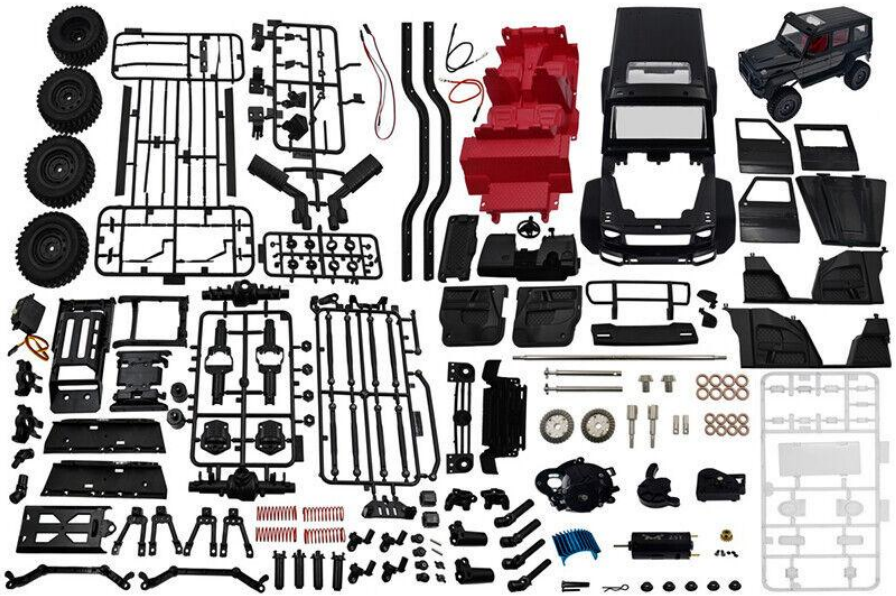

```
{
  "shipId": "CS12345",
  "shipName": "Ocean Explorer",
  "company": "Global Seas Charters",
  "contactEmail": "info@globalseacharters.com",
  "contactPhone": "+12345678901",
  "itinerary": "Caribbean Explorer - 7 Nights",
  "bookings": [
    {
      "fromDate": "2024-07-01",
      "toDate": "2024-07-08",
      "bookingId": "BKG123456",
      "customers": [
        {
          "name": "Alice Johnson",
          "passportNumber": "P12345678"
        },
        {
          "name": "John Johnson",
          "passportNumber": "P87654321"
        }
      ]
    },
    {
      "fromDate": "2024-08-15",
      "toDate": "2024-08-22",
      "bookingId": "BKG654321",
      "customers": [
        {
          "name": "Bob Smith",
          "passportNumber": "P23456789"
        }
      ]
    }
  ]
}
```

Why?

https://docs.google.com/document/d/1-wMxCkrWq6VOqrYQ6yocGr8347c78GRVT8w_4QlhR0o/edit?tab=t.0

My RC Car when it arrived



Just store the assembled car

What makes a Schema good?

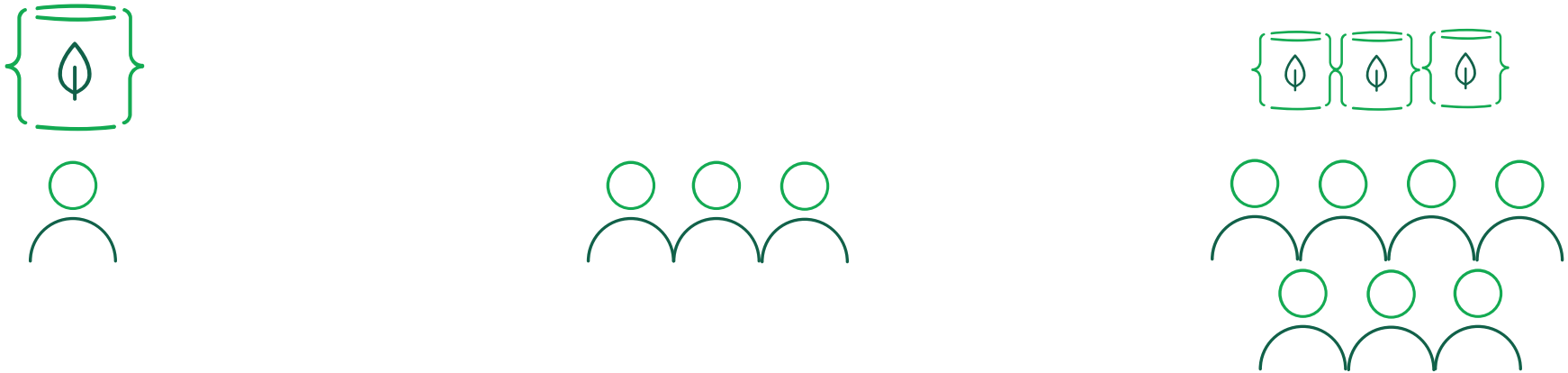
With a well designed schema...

- Data accessed frequently can be accessed efficiently
- Common operations touch as few documents as possible
- Documents are sized appropriately for the workload

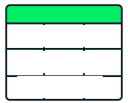
Schemas don't need to be complex

Simplicity

Performance



Three quantified rules



Sensible sizes

If your records are **tiny or > 200KB** then you should review your schema as it may be suboptimal

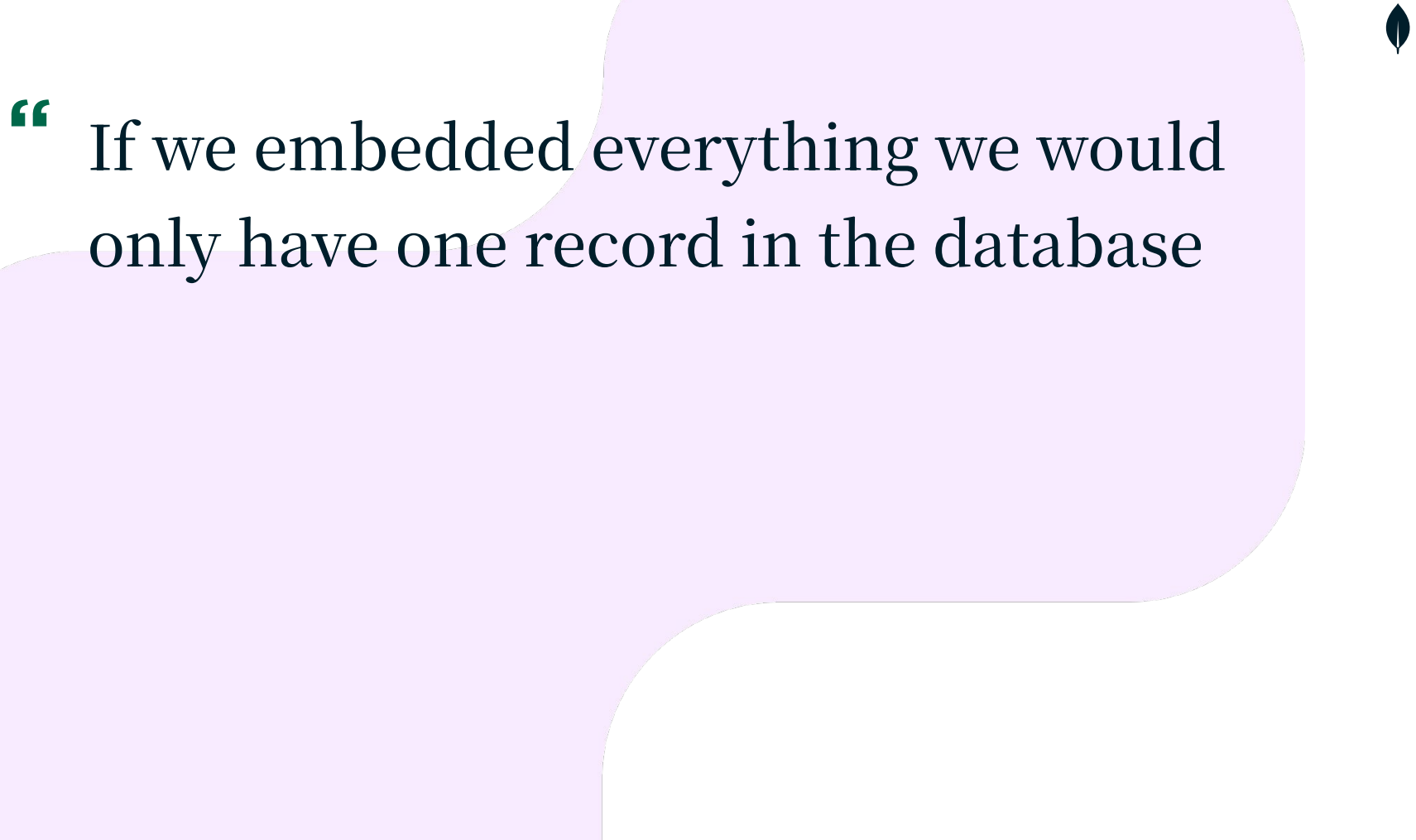
Avoid flatness, embrace depth

Do not have > 200 fields at the same level, **including arrays** with > 200 members

Duplicate values appropriately

If something is read **100s of times** more than it's changed it's OK to have multiple copies, but otherwise avoid duplicating mutable data

Modeling Relationships


“
If we embedded everything we would only have one record in the database

Embed

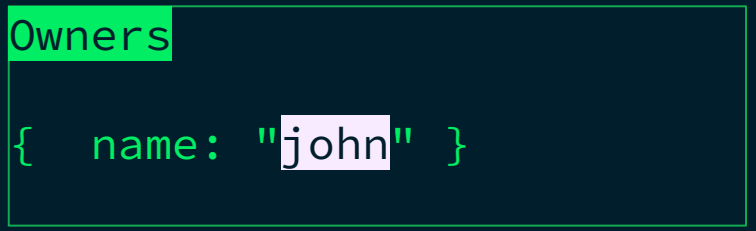
When the records won't get too large.

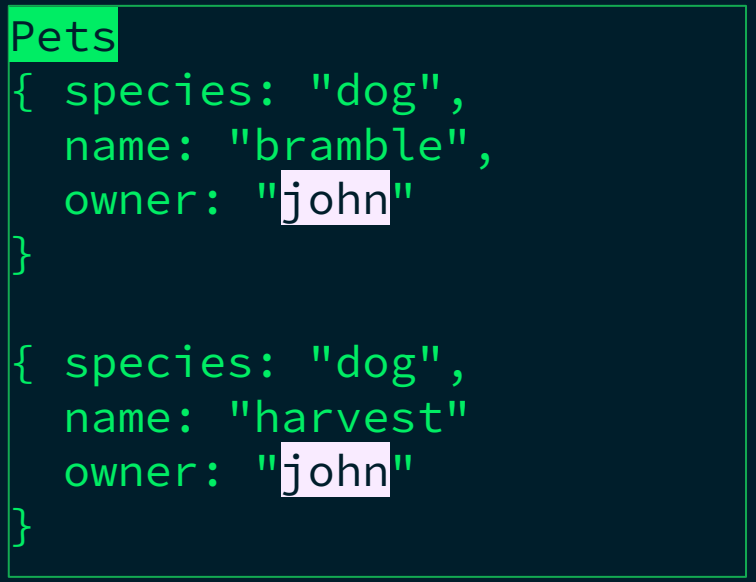
When embedded data is queried with the parent.

When most embedded data is often retrieved with the parent.

Where any data that is duplicated changes infrequently.

```
{
  owner: "john",
   pets: [
    { species: "dog",
         name: "bramble"},
    { species: "dog",
         name: "harvest"}
    ]
}
```





Link using keys

For 1 to Many relationships

If an embedded document would be overly large.

If the linked data is less frequently accessed.

Only link at ONE end

Owners

```
{  name: "john" }
{  name: "carol" }
```

Pets

```
{ species: "dog",
  name: "bramble",
  owner: "john"
}

{ species: "dog",
  name: "harvest"
  owner: ["john", "carol"]
}
```

Link using arrays

For Many to Many relationships.

Where data would be duplicated that changes frequently.

Represent links at one end only.

Link with arrays - which end?

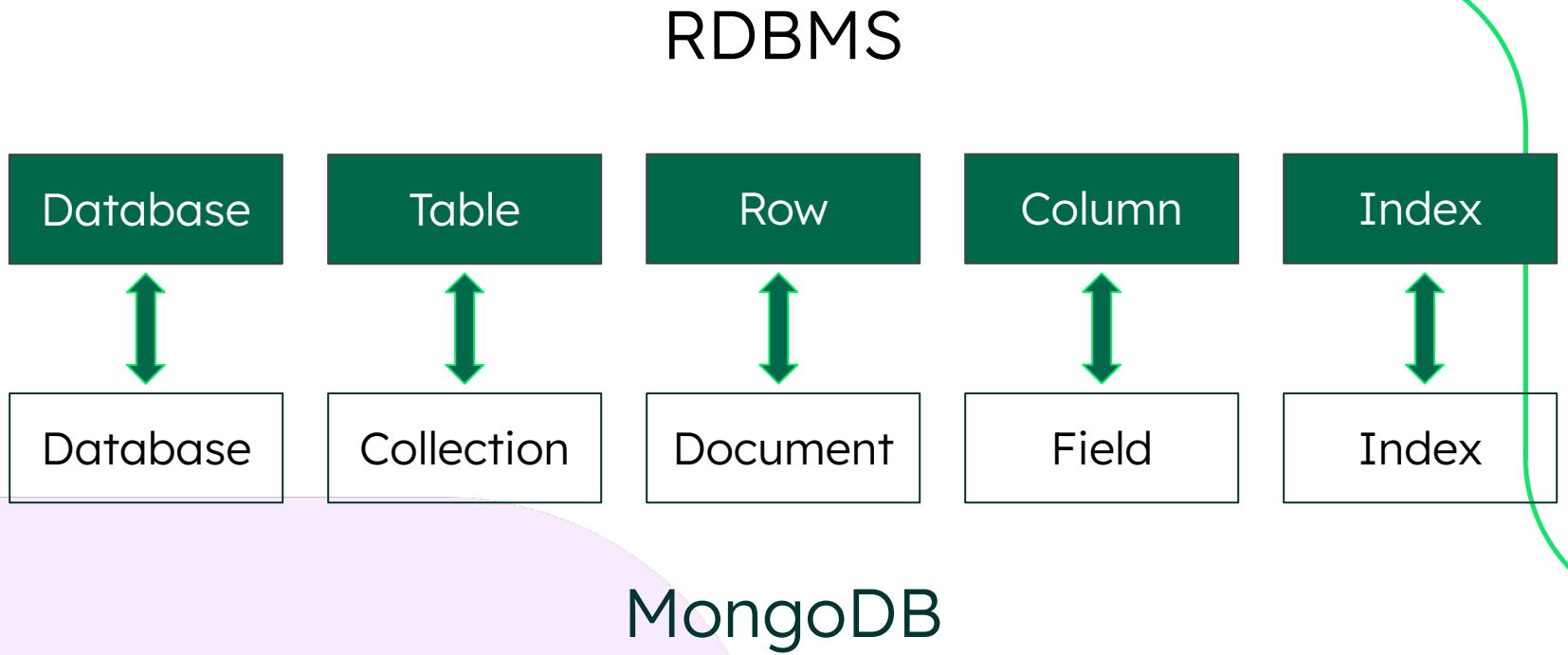
```
{
  user: "Elon",
  joined: "2013-08-09",
  follows: [ "John", "Jeff"
           … 200 items …,
           "Zuk" ]
}
```

Consider the extreme cases and high cardinalities

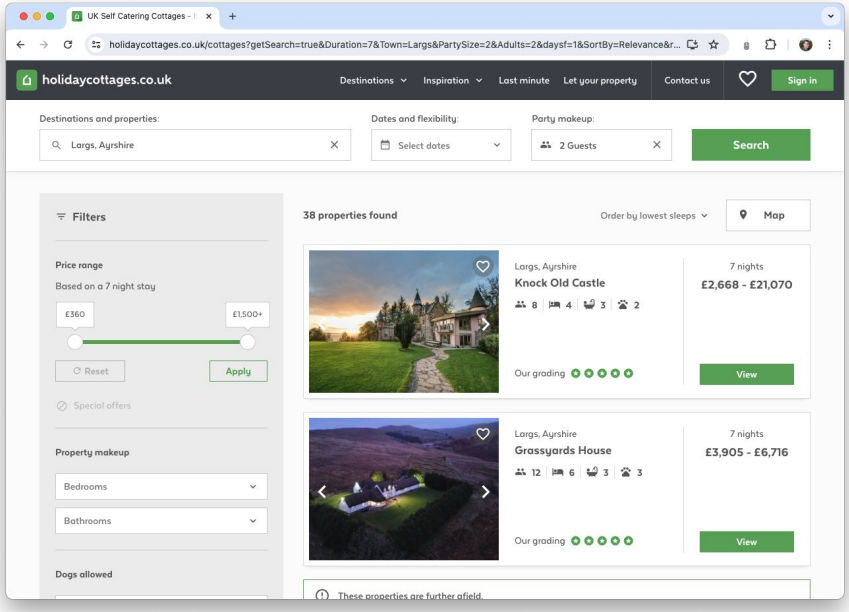
```
{
  user: "Elon",
  joined: "2013-08-09",
  followers: [ "John", "Jeff"
           … 30 million items …,
           "Zuk" ]
}
```

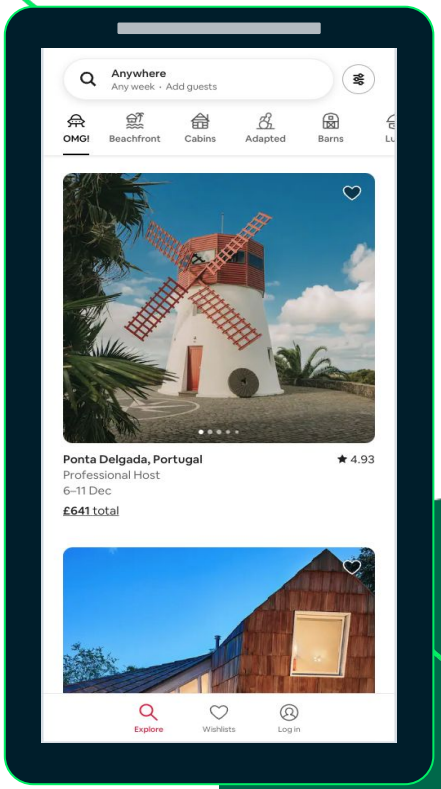
RDBMS to MongoDB step by step

Terminology

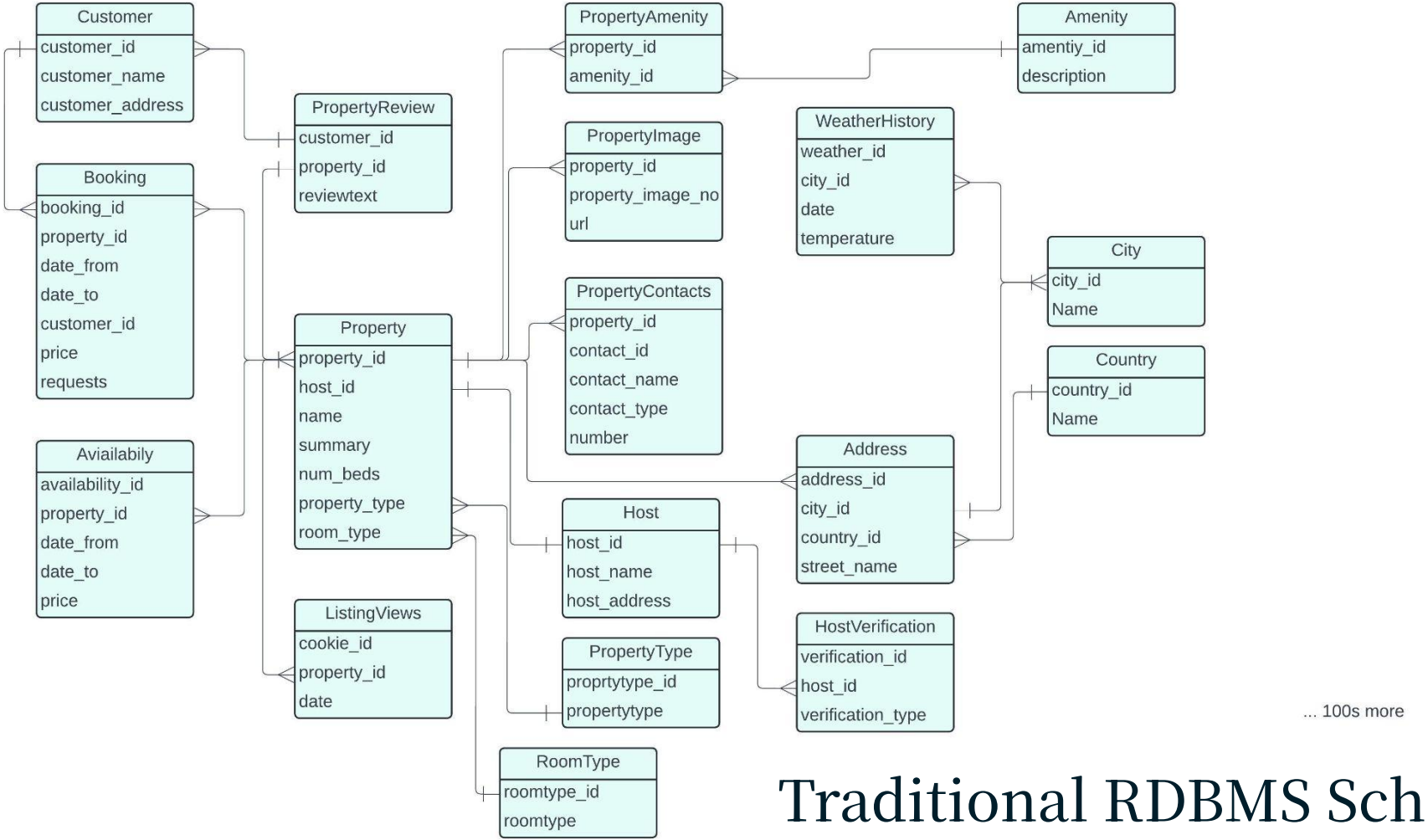


- Identify principal business entities
- Denormalize slowly changing domain values
- Retain domain tables (lookup lists)
- Embed weak tables where possible
- Replace associative tables with arrays



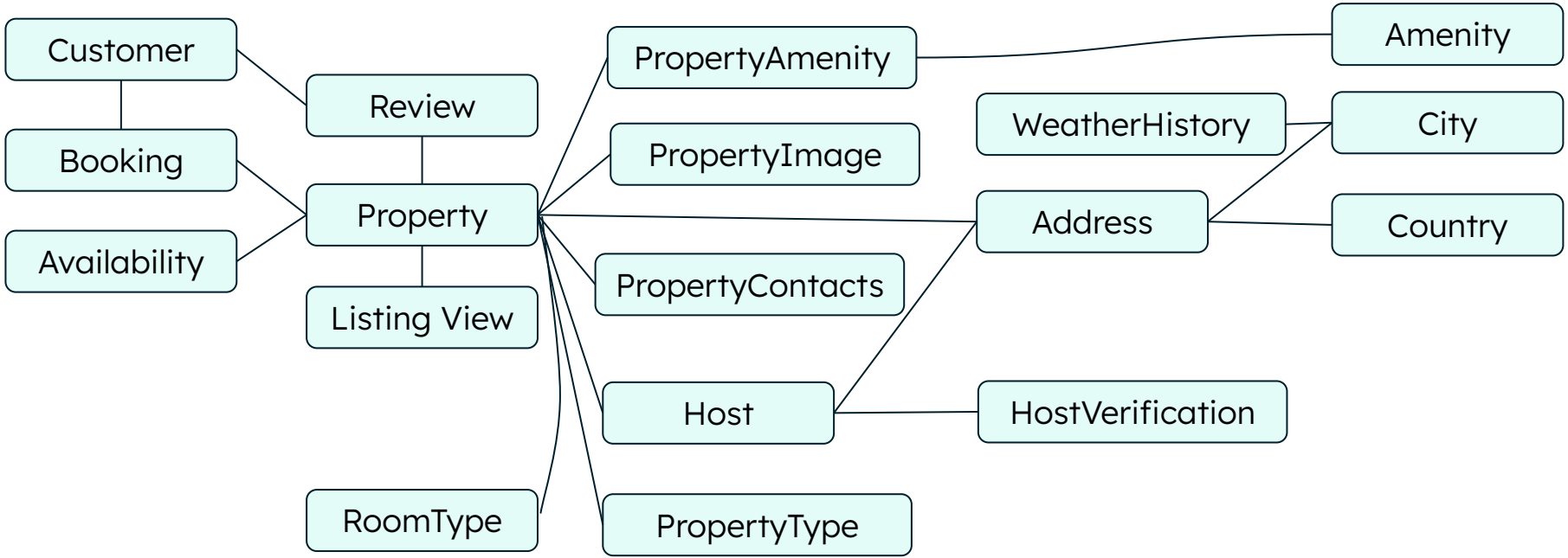


Let's book a vacation

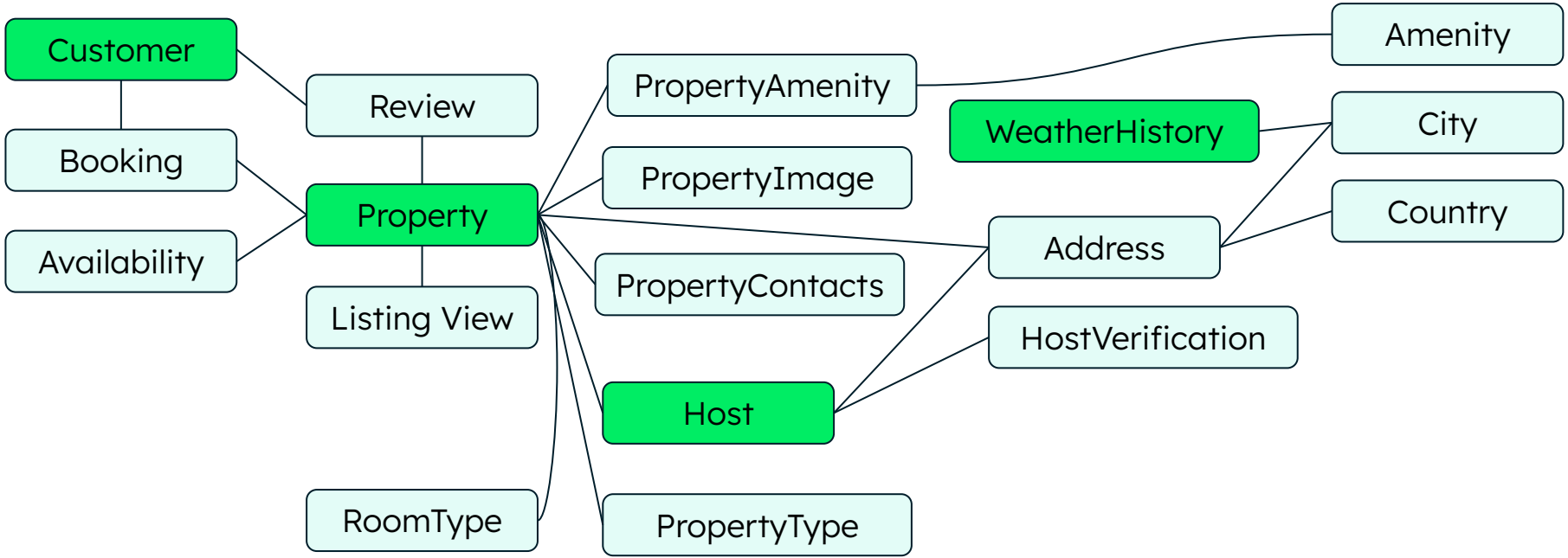


Traditional RDBMS Schema

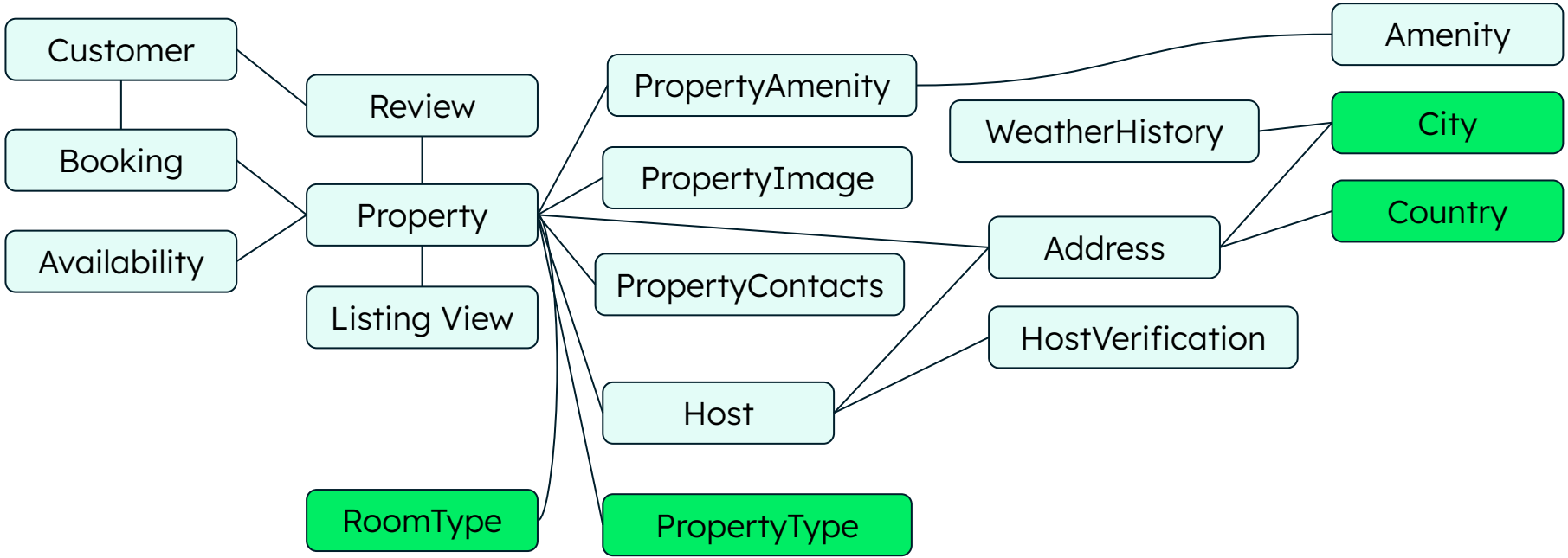
Traditional RDBMS schema



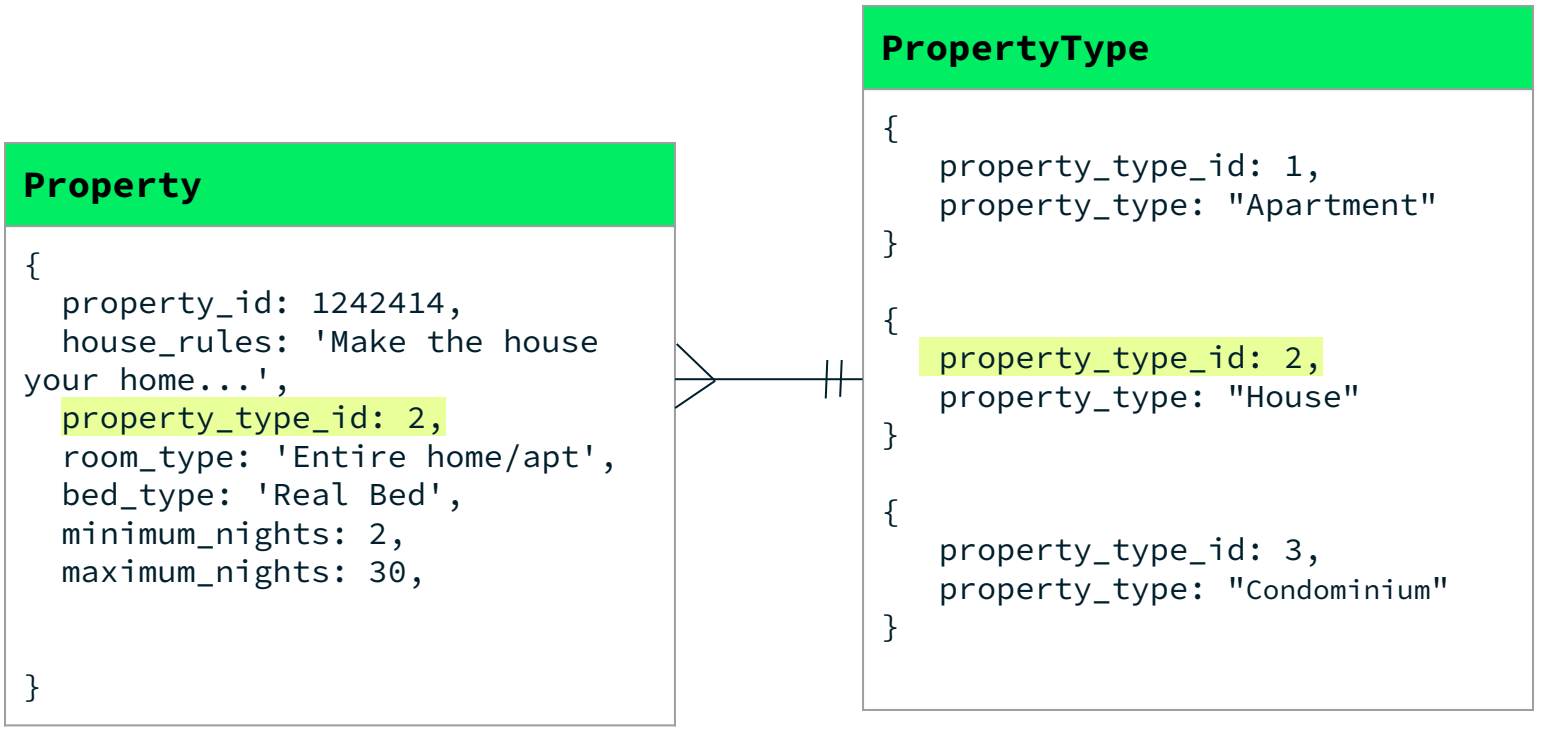
Identify principal business entities



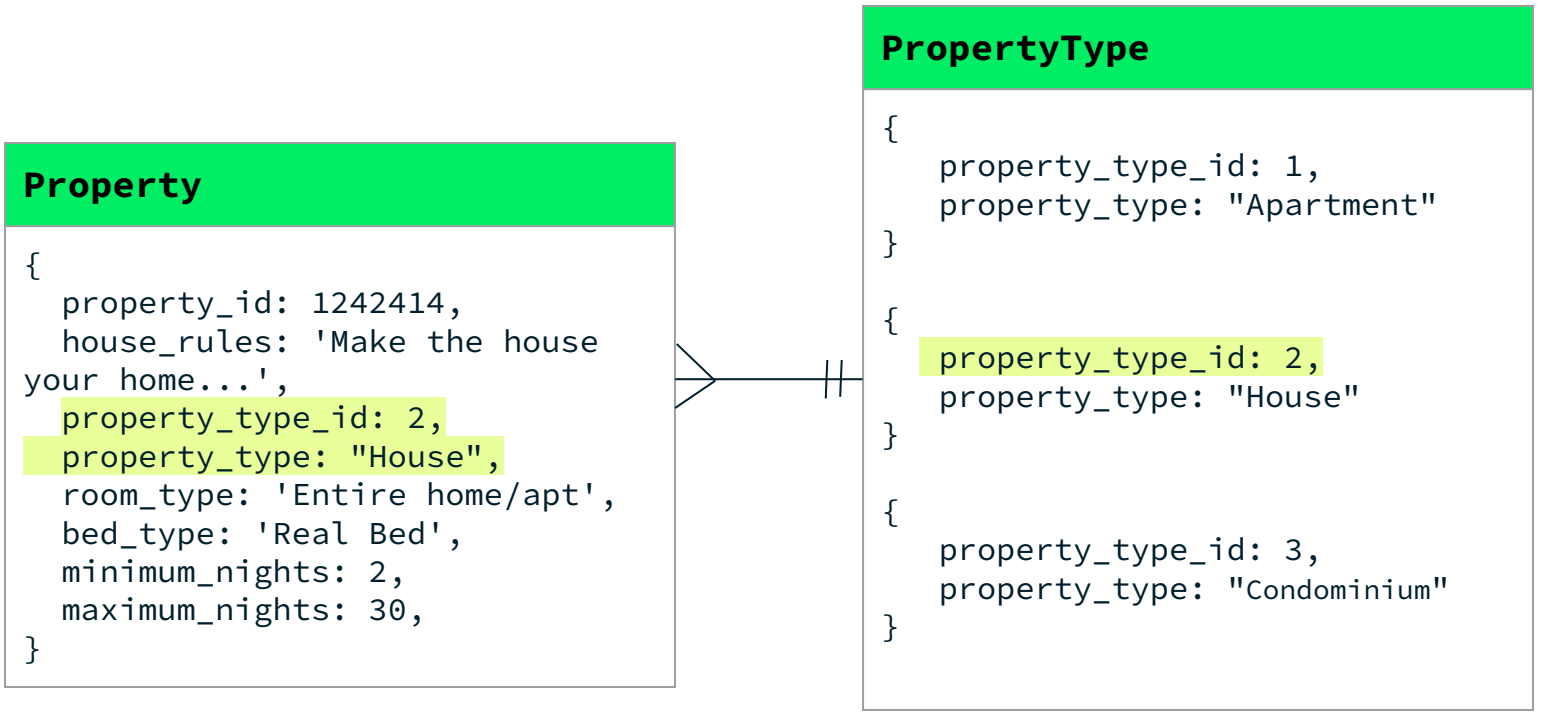
Denormalise domain tables (lookups)



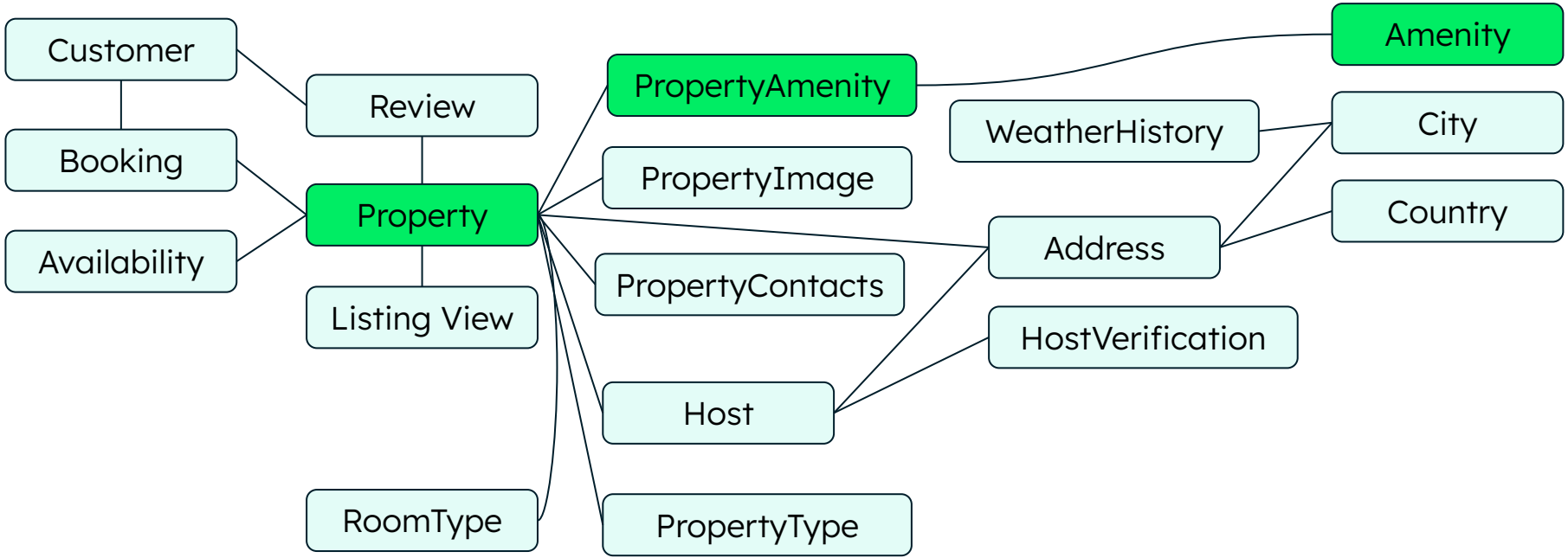
Embed domain (lookup) values



Embed domain (lookup) values



Denormalise multi-attribute values



Associative domains

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your
home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30
}
```

PropertyAmenity

```
{
    property_id: 1242414,
    amenity_id: 3
}

{
    property_id: 1242414,
    amenity_id: 5
}

{
    property_id: 1242414,
    amenity_id: 6
}
```

Amenity

```
{
    amenity_id: 3,
    name: "Pool"
}
{
    amenity_id: 4,
    name: "Boat"
}
{
    amenity_id: 5,
    name: "Wifi"
}
{
    amenity_id: 6,
    name: "Iron"
}
```

Associative domains

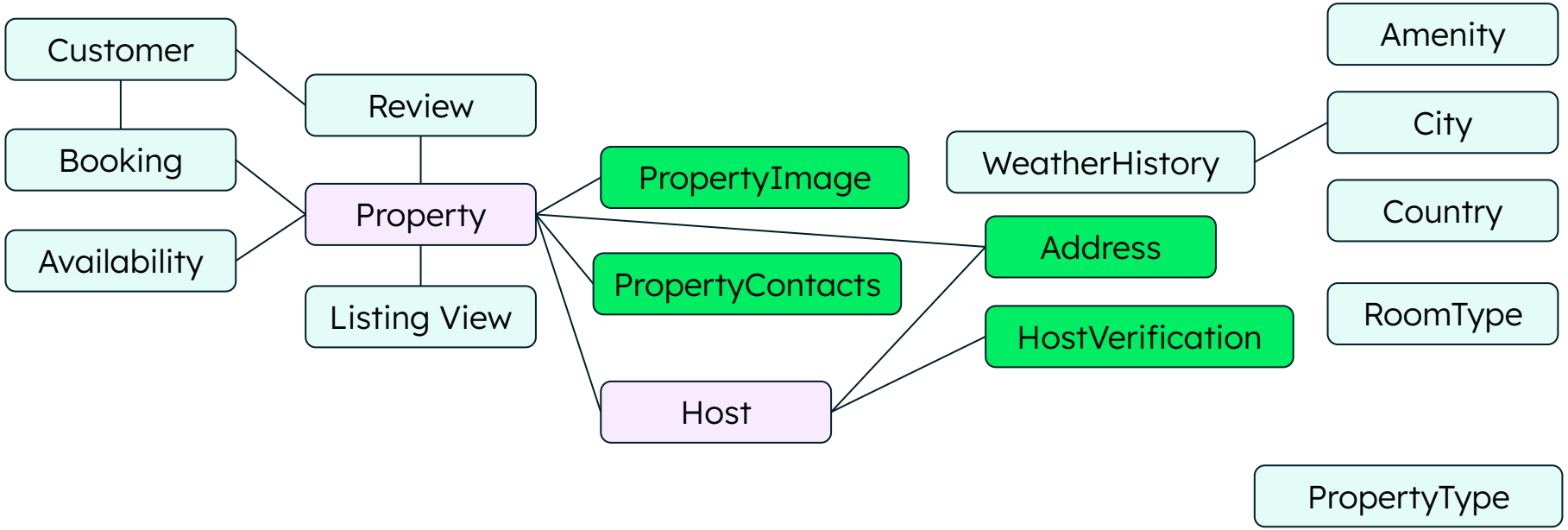
Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your
home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities :[ "Pool, "Wifi", "Iron"]
}
```

Amenity

```
{
   amenity_id: 3,
   name: "Pool"
}
{
   amenity_id: 4,
   name: "Boat"
}
{
   amenity_id: 5,
   name: "Wifi"
}
{
   amenity_id: 6,
   name: "Iron"
}
```

Child tables - low cardinality



Child tables - low cardinality

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your
home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities :[ "Pool, "Wifi", "Iron" ]
}
```

PropertyImages

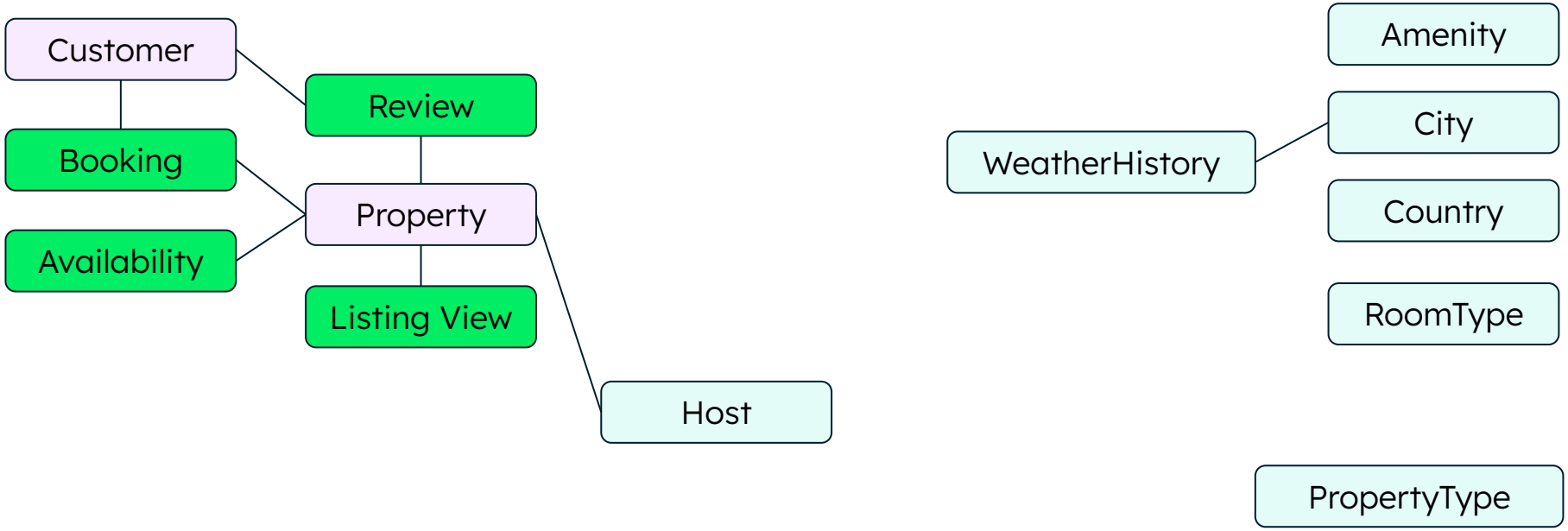
```
{
   property_id: 1242414,
   url: "http://...",
   description: "Outside"
}
{
   property_id: 1242414,
   url: "http://...",
   description: "Sea View"
}
{
   property_id: 1242414,
   url: "http://...",
   description: "Pool"
}
```

Child tables - low cardinality

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities :[ "Pool, "Wifi", "Iron" ],
  images: [
    { url: "http://images.net/12431.jpg", description: "Outside"},
    { url: "http://images.net/12437.jpg", description: "Sea View"},
    { url: "http://images.net/12439.jpg", description: "Pool"}
 ]
}
```

Child tables - high cardinality



Child tables - high cardinality

Property

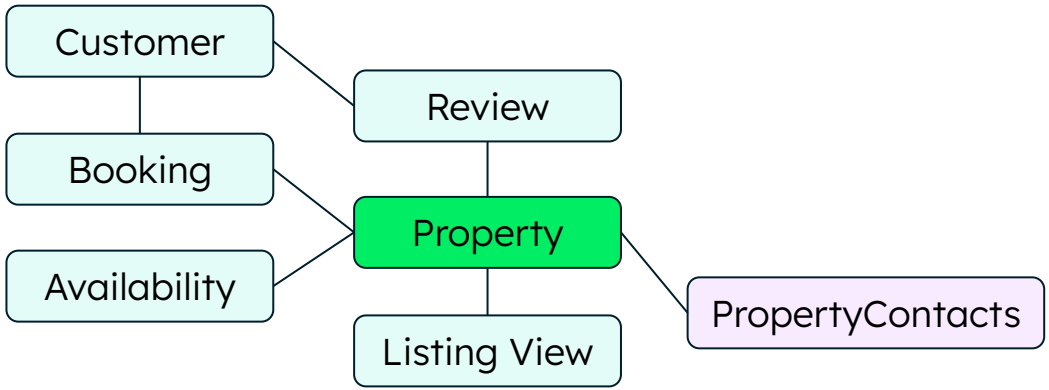
```
{
  property_id: 1242414,
  house_rules: 'Make the house your
home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities: [ "Pool, "Wifi", "Iron" ]
 }
```

Booking

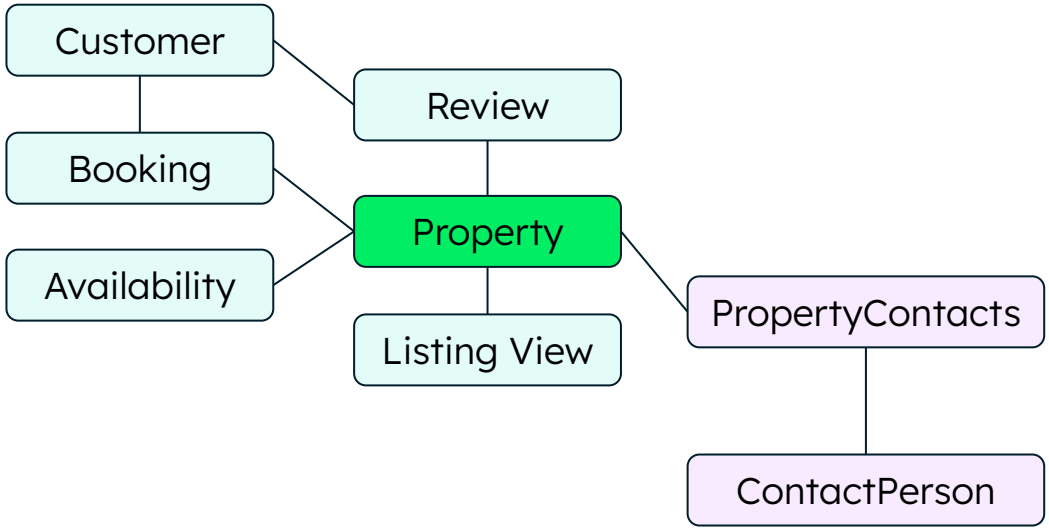
```
{
 booking_id: 1231,
 property_id: 1242414,
 customer_id: 8872,
 from_date: 2025-07-03,
 to_date: 2025-07-18,
 guest_names: [
 ]
}
```

… Many more bookings for this property …

Many to many relationships



Many to many relationships



Many to many

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your
home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities: [ "Pool, "Wifi",
"Iron"]
}
```

PropertyContact

```
{
   property_id: 1242414,
   contact_id: 1233
}

{
   property_id: 1242414,
   contact_id: 1344
}

{
   property_id: 1242414,
   contact_id: 766
}
```

Contact

```
{
   contact_id: 1233,
   name: "John",
   tel: "0775245152"
}
{
   contact_id: 1344,
   name: "Andrew",
   tel: "0775345952"

}
{
    contact_id: 2135,
   name: "Daniel",
   tel: "071235151"

}
{
   contact_id: 766,
   name: "Rick",
   tel: "077095142"

}
```

Many to many

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your
home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities: [ "Pool, "Wifi", "Iron"],
  property_contacts: [ 1233, 1433, 766 ]
}
```

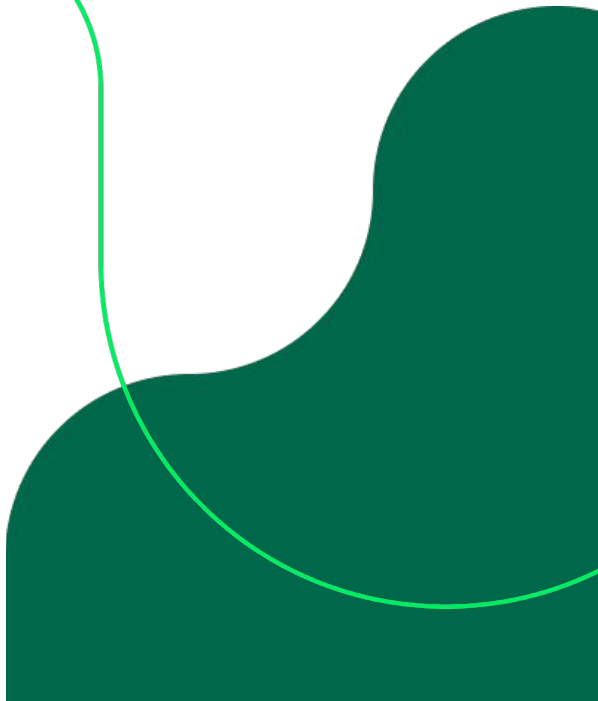
Contact

```
{
   contact_id: 1233,
   name: "John",
   tel: "0775245152"
}
{
   contact_id: 1344,
   name: "Andrew",
   tel: "0775345952"

}
{
    contact_id: 2135,
   name: "Daniel",
   tel: "071235151"

}
{
   contact_id: 766,
   name: "Rick",
   tel: "077095142"

}
```

With a well designed schema...

- Data accessed frequently can be accessed efficiently
- Common operations touch as few documents as possible
- Documents are sized appropriately for the workload

- Identify principal business entities
- Denormalize slowly changing domain values
- Retain domain tables (lookup lists)
- Embed weak tables where possible
- Replace associative tables with arrays
- Duplicate data to cache-in-record
- Add computed fields
- Right-size your documents

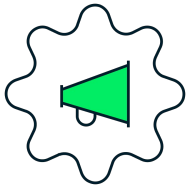


Schema optimization techniques

Understanding your workload

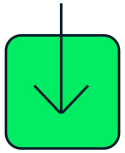
List your operations

Write down what operations you know need to take place along with any SLAs



Optimise for what needs to be fast

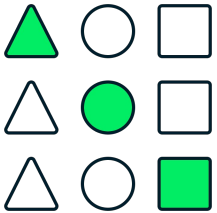
Design your schema to make your most frequent operations fast or those that matter



Recognise payload versus process fields

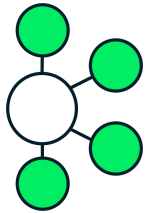
Identify what data you are just saving and retrieving versus what fields you need to process

Optimisation techniques



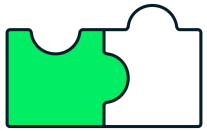
Caching

Duplicate data to reduce the resource requirements for reading data



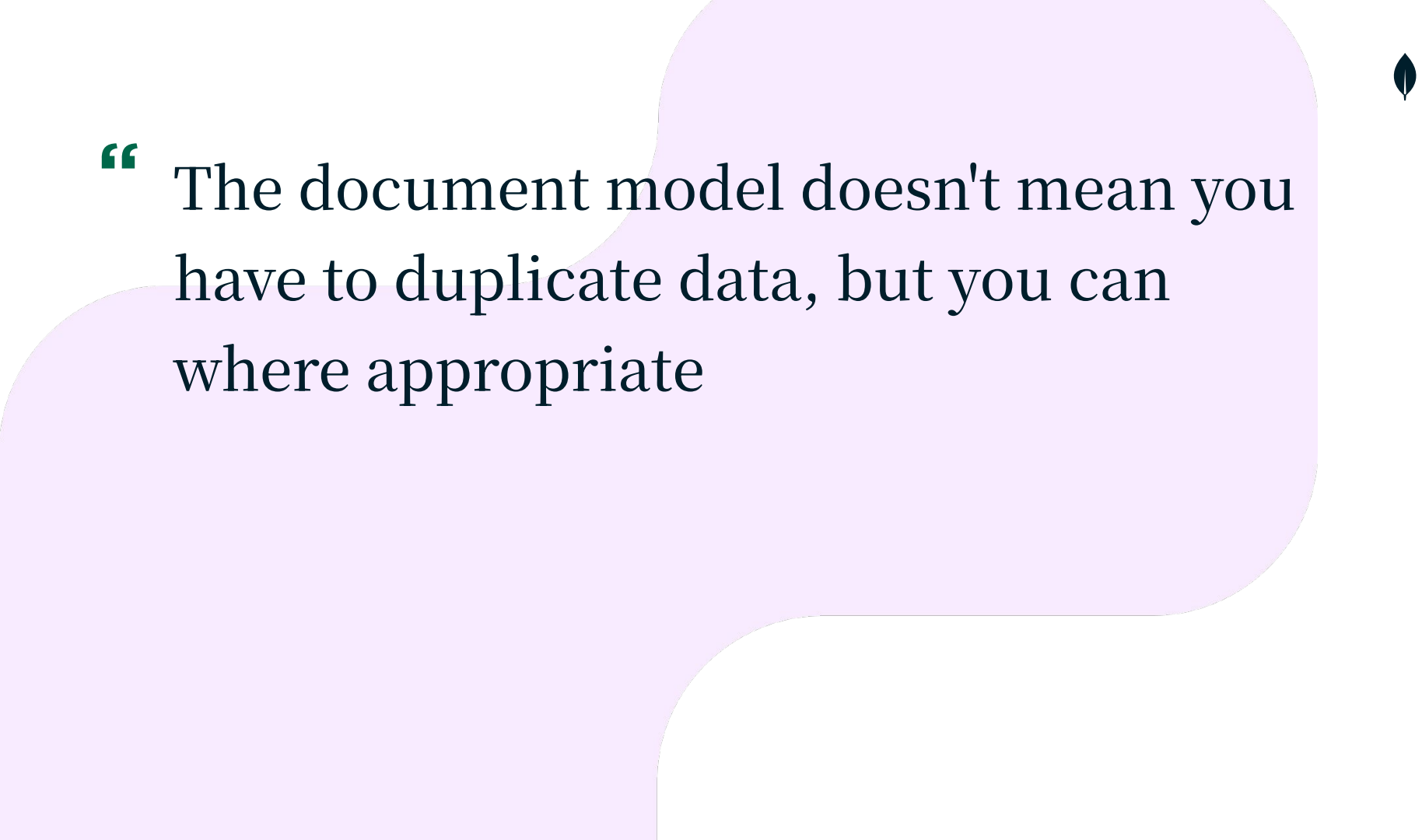
Decomposition

Break larger single records into multiple documents to improve cache or write efficiency



Grouping

Store multiple small related records together to improve cache and read efficiency


“
The document model doesn't mean you have to duplicate data, but you can where appropriate

Subset Pattern (caching)

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities :[ "Pool, "Wifi", "Iron" ],
  recent_reviews: [
    { review_id: 1231,
      reviewer: "sarah",
      date: "2024-04-12",
      comment: "Epic Place",
      score: 5},

    … LATEST reviews as array …

  ]
}
```

Review

```
{
 property_id: 1242414,
 review_id: 1231,
 customer_id: 8872,
 reviewer: "sarah",
 date: "2024-04-12",
 comment: "Epic place",
 score: 5
}
```

… Hundreds more reviews for this property …

Extended Reference Pattern (caching)

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your home...',
  property_type_id: 2,
  property_type: "House",
  room_type: 'Entire home/apt',
  bed_type: 'Real Bed',
  minimum_nights: 2,
  maximum_nights: 30,
  amenities: [ "Pool, "Wifi", "Iron" ],
  future_bookings: [
   { booking_id: 1231, from: 2025-07-03, to: 2025-07-18 },
   { booking_id: 5503, from: 2025-08-09, to: 2025-08-17 },
   … All future bookings for the property …
 ]

 }
```

Booking

```
{
 property_id: 1242414,
 booking_id: 1231,
 customer_id: 8872,
 from_date: 2025-07-03,
 to_date: 2025-07-18,
 guest_names: [
 ]
}
```

… All bookings for this property …

Computed Pattern (caching)

Property

```
{
  property_id: 1242414,
  house_rules: 'Make the house your home...',
  property_type_id: 2,
  property_type: "House",
  amenities: [ "Pool, "Wifi", "Iron" ],
  num_reviews: 241,
  total_score: 1049,
  score_histogram: [ 2, 6, 0, 130, 103 ],
  recent_reviews: [
   { review_id: 1231, reviewer: "sarah"
     date: "2024-04-12"
     comment: "Epic Pla
     … LATEST 10 reviews
  ], }
```

4.49 / 5 (241 reviews)

⭐⭐⭐⭐⭐ (103)
⭐⭐⭐⭐ (130)
⭐⭐⭐ (0)
⭐⭐ (6)
⭐ (2)

Review

```
{
 property_id: 1242414,
 review_id: 1231,
 customer_id: 8872,
 reviewer: "sarah",
 date: "2024-04-12",
 comment: "Epic place",
 score: 5
}
```

… Many more reviews for this property …

Sibling Pattern (decomposition)

Booking

```
{
 bookingId: "HLB123456789",
 customerId: 124312,
 property: {
   propertyId: "PROP789",
   propertyName: "Disney Dream",
   },
 bookingDates: {
   checkIn: "2024-07-07",
   checkOut: "2024-06-22"
 },
 pricing: {
   nightlyRate: 250.00,
   totalNights: 7,
   totalCost: 1925.00,
   currency: "USD"
 },
 specialRequests: "Late check-out requested,
baby cot needed",
 bookingStatus: "Confirmed"
}
```

BookingExternal

```
{
  bookingId: "HLB123456789",
  HTTPResponse: {
      status: 201,
      body: "vPNR: ABC123
1.1SMITH/JOHN MR
2 AC 123 Y 07JUL MIAMIA HK1 0900 1730  /DCAC
/E
3 DISNEY DREAM /ARR-07JUL /N-1NT
/C-2125551234 /R-CONFIRMED
4 OSI AC CTCM 2125556789
5 TK:OK0715/AC/1234567890
6 SSR DOCS AC HK1
P/US/123456789/US/01JAN70/M/07JUL24/SMITH/JO
HN
7 SSR RQST AC VGML
8 RM*THANK YOU FOR CHOOSING DISNEY CRUISES"
   }
}
```

Bucket Pattern (grouping)

ListingViews

```
{
  _id: "b23efda45da6500",
  date:  "2022-06-07:15:44:03",
  listingid: 155241,
  viewerid: 1886451
}
{
  _id: "b23efdc45da6512",
  date:  "2022-06-07:15:44:33",
  listingid: 155267,
  viewerid: 1886451
}
{
  _id: "b23efdc45da6512",
  date:  "2022-06-07:15:45:02",
  listingid: 155267,
  viewerid: 1886105
}
```

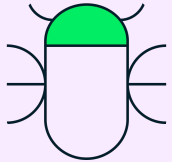
ListingViews(Bucketed)

```
{
 _id: "b23efda45da6500",
 viewerid: 1886451,
 nViews: 3,
 views: [{
    date:  "2022-06-07:15:44:03",
    listingid: 155241
  }, {
    date:  "2022-06-07:15:44:33",
    listingid: 155267
  }, {
    date:  "2022-06-07:15:45:02",
    listingid: 155267
  }
 ]
}
```

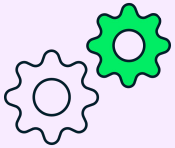
Five common mistakes

Relational data modeling

`JOINS are expensive`



- As many collections as tables

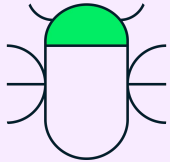


- The Document Model provides better code and performance
- MongoDB is not optimized for joins

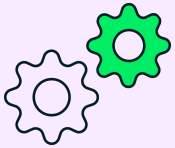
- Assess your important queries
- Follow the rules we talked about

Poor index design

Indexes are a key part of schema design



- Indexes are critical

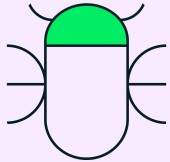


- You should use compound indexes in MongoDB when querying multiple fields

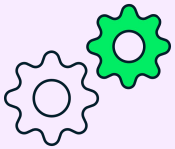
- Index queries where possible
- Unused indexes waste CPU
- Use the _id index

Arbitrary search

MongoDB is not a search engine (but…)



- Users want to query any set of fields and sort

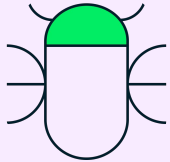


- A databases is the wrong tool for this
- Use a search engine (Atlas Search)

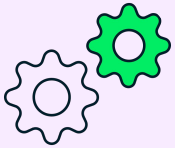
- Indexes do not intersect
- Wide compound indexes are limited
- Wildcard indexes are not the answer

Not learning update()

Apply complex business changes server side



- updateOne() is very powerful in MongoDB

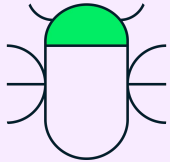


- It can do a lot of checking, computing and atomic manipulation

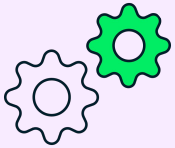
- Failure to learn it leads to correctness issues and poor code

Ignoring schema validation

Schemaless is great for PoC



- MongoDB has powerful *optional* schema validation



- When you create a rule about what should be in a document - apply it

- This makes sure all future code or manual edits work correctly

TLDR

Conclusion

MongoDB is a fully featured transactional, relational database

MongoDB adds Arrays, objects and all the API's to work with them to the traditional table model

Schema design is about co-locating data that's used together to improve performance