Couchbase

# Data Modelling

# 1 Data Access APIs

# Document Modeling Process

1. Understand your application requirements
2. Understand the Couchbase APIs
3. Map application access paths to best Couchbase API
4. Logical data modeling
5. Physical data modeling

# Understand your application requirements

Inputs / Outputs

Request rate, typical and peak

Latency

Consistency

Scope of Aggregations

Unique constraints

Data set size & growth rate

# Couchbase APIs

| Couchbase API | Latency | Throughput | Scalability | Applicability |
|---|---|---|---|---|
| Key/Value | 500us–10ms | > 1M ops/sec | Highest | General, best for high throughput, highly latency sensitive workloads |
| N1QL | 5ms+ | > 40K qps | High | General, best for secondary lookups, pushing complex logic to database |
| Views | 10–100ms | < 4K qps | Moderate | Aggregations, best for large scale aggregations (>1B docs) with low latency and moderate latency requirements |
| Full Text Search | 5ms+ | > 20K qps | High | Text search, best for natural language queries, relevance ranking |

# Mapping application requirements to APIs

| Application Requirements | Couchbase API |
|---|---|
| • Very high throughput<br>• Latency sensitive<br>• Needs strong consistency<br>• Large data set / high growth | Key/Value |
| • Secondary key lookups<br>• Operational aggregations<br>• Filtered queries<br>• Ad-hoc queries | N1QL |
| • Report on well defined metrics<br>• Large scale aggregations<br>• Latency sensitive | Views |
| • Find patterns within text fields<br>• Provide search relevancy rankings | Full text search |

# 2 Data Modelling

# Goals of Data Modeling for N1QL

1. Define entities
2. Define relationships
3. Define document boundaries
   - Identifying parent and child objects
   - Deciding whether to embed child objects
4. Express relationships

# Defining Relationships

- Parent-child relationships
  - If we model the child as a separate document and not embedded, we have defined a relationship (parent-child)
- Independent relationships
  - Relationships between two independent objects
  - E.g. a person and a company where they work; deleting one does not delete the other (hopefully)

# Identifying Parent and Child Objects

- A parent object has an independent lifecycle
  - It is not deleted as part of deleting any other object
  - E.g., a registered user of a site
- A child object has a dependent lifecycle; It has no meaningful existence without its parent
  - It must be deleted when its parent is deleted
  - E.g., an invoice line item (child of the invoice object)
  - E.g., a comment on a blog (child of the blog object)

# Deciding Whether to Embed Child Objects

- Couchbase provides per-document atomicity
  - If the child and parent must be atomically updated or deleted together, embedding the child facilitates this
  - E.g. if an order line subtotal and order total must be updated together atomically, embedding the order line item facilitates this

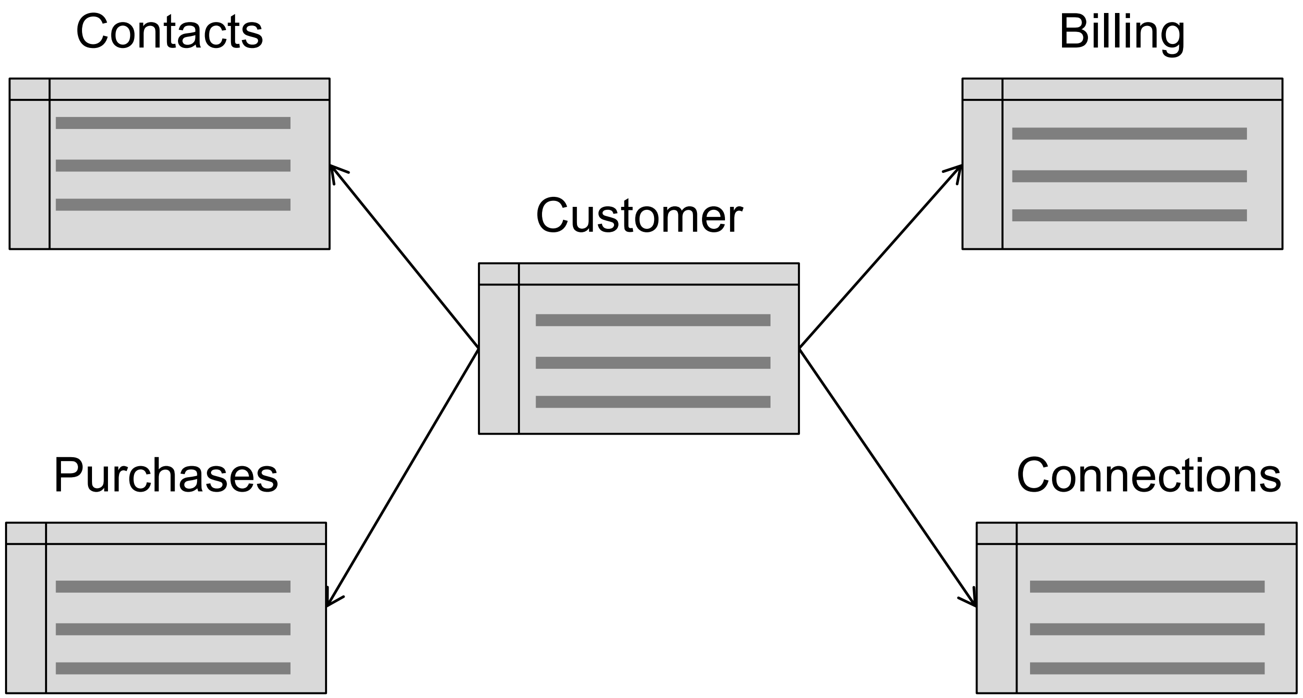
- There is a performance tradeoff
  - Embedding the child makes it faster to read the parent together with all its children (single document fetch)
  - If the child has high cardinality (its parent has many instances of the child), embedding the child makes the parent bigger and slower to store and fetch
  - If the child has high cardinality (its parent has many instances of the child), embedding the child makes it more expensive to update the parent or the child

# Expressing Relationships

- 3 ways to express relationships in Couchbase
  1. Parent contains keys of children (outbound)
  2. Children contain key of parent (inbound)
  3. Both of the above (dual)

- High cardinality affects outbound relationships
  - Makes parent document bigger and slower
  - Makes it expensive to load a subset of relationships (e.g. paging through blog comments)

# Data Modeling with JSON Example



# Data Modeling with JSON Example

Table: Customer

| CustomerID | Name | DOB |
|---|---|---|
| CBL2017 | Jane Smith | 1990-01-30 |

- The primary (CustomerID) becomes the DocumentKey
- Column name-Column value becomes KEY-VALUE pair.

Customer DocumentKey: CBL2017

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30”
}
```

OR

```
{
    "Name" : {
        "fname": "Jane",
        "lname": "Smith"
     }
    "DOB"  : "1990-01-30"
}
```

# Data Modeling with JSON Example

## Table: Customer

| CustomerID | Name | DOB |
|---|---|---|
| CBL2017 | Jane Smith | 1990-01-30 |

## Table: Billing

| CustomerID | Type | Cardnum | Expiry |
|---|---|---|---|
| CBL2017 | visa | 5827… | 2019-03 |

## Customer DocumentKey: CBL2017

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30",
    "Billing" : [
        {
            "type"    : "visa",
            "cardnum" : "5827-2842-2847-3909",
            "expiry"  : "2019-03"
        }
    ]
}
```

**Rich Structure & Relationships**

Billing information is stored as a sub-document

There could be more than a single credit card.  So, use an array.

# Data Modeling with JSON Example

Table: Customer

| CustomerID | Name | DOB |
|---|---|---|
| CBL2017 | Jane Smith | 1990-01-30 |

Table: Billing

| CustomerID | Type | Cardnum | Expiry |
|---|---|---|---|
| CBL2017 | visa | 5827… | 2019-03 |
| CBL2017 | master | 6274… | 2018-12 |

Customer DocumentKey: CBL2017

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30",
    "Billing" : [
        {
            "type"    : "visa",
            "cardnum" : "5827-2842-2847-3909",
            "expiry"  : "2019-03"
        },
        {
            "type"    : "master",
            "cardnum" : "6274-2542-5847-3949",
            "expiry"  : "2018-12"
        }
    ]
}
```

- **Value evolution**
  - Simply add additional array element or update a value.

# Data Modeling with JSON Example

Table: Connections

| CustomerID | ConnId | Name |
|---|---|---|
| CBL2017 | XYZ987 | Joe Smith |
| CBL2017 | SKR007 | Sam Smith |

- **Structure evolution**
  - Simply add new key-value pairs
  - No downtime to add new KV pairs
  - Applications can validate data
  - Structure evolution over time.
- **Relations via Reference**

Customer DocumentKey: CBL2017

```
{
    "Name" : "Jane Smith",
    "DOB"  : "1990-01-30",
    "Billing" : [
        {
            "type"    : "visa",
            "cardnum" : "5827-2842-2847-3909",
            "expiry"  : "2019-03"
        },
        {
            "type"    : "master",
            "cardnum" : "6274-2542-5847-3949",
            "expiry"  : "2018-12"
        }
    ],
    "Connections" : [
        {
            "ConnId"   : "XYZ987",
            "Name"     : "Joe Smith"
        },
        {
            "ConnId"   : ”SKR007",
            "Name"     : ”Sam Smith"
        }
    ]
}
```

# Data Modeling with JSON Example

Contacts

| CustomerID | ConnId | Name |
|---|---|---|
| CBL2017 | XYZ987 | Joe Smith |
| CBL2017 | SKR007 | Sam Smith |

Billing

| CustomerID | Type | Cardnum | Expiry |
|---|---|---|---|
| CBL2017 | visa | 5827… | 2019-03 |
| CBL2017 | master | 6274… | 2018-12 |

Customer

| CustomerID | Name | DOB |
|---|---|---|
| CBL2017 | Jane Smith | 1990-01-30 |

Purchases

| CustomerID | item | amt |
|---|---|---|
| CBL2017 | mac | 2823.52 |
| CBL2017 | ipad2 | 623.52 |

Connections

| CustomerID | ConnId | Name |
|---|---|---|
| CBL2017 | XYZ987 | Joe Smith |
| CBL2017 | SKR007 | Sam Smith |

DocumentKey: **CBL2017**

```
{
   "Name" : "Jane Smith",
   "DOB"  : "1990-01-30",
   "Billing" : [
      {
         "type"    : "visa",
         "cardnum" : "5827-2842-2847-3909",
         "expiry"  : "2019-03"
      },
      {
         "type"    : "master",
         "cardnum" : "6274-2842-2847-3909",
         "expiry"  : "2019-03"
      }
   ],
   "Connections" : [
      {
         "CustId"   : "XYZ987",
         "Name"     : "Joe Smith"
      },
      {
         "CustId"   : "PQR823",
         "Name"     : "Dylan Smith"
      }
      {
         "CustId"   : "PQR823",
         "Name"     : "Dylan Smith"
      }
   ],
   "Purchases" : [
      { "id":12, item: "mac",   "amt": 2823.52 }
      { "id":19, item: "ipad2", "amt": 623.52 }
   ]
}
```

# Models for Representing Data

| Model | Relational Model | JSON Document Model (NoSQL) |
|---|---|---|
| **1:1** | ▪ Foreign Key<br>▪ Denormalize | ▪ Embedded Object (implicit)<br>▪ Document Key Reference |
| **1:N** | ▪ Foreign Key | ▪ Embedded Array of Objects<br>▪ Document key Reference |
| **N:M** | ▪ Foreign Key | ▪ Embedded Array of Objects, arrays of arrays with references |

# 3 JSON Design Choices

# JSON Design Choices

Single Root Attributes vs. "type"/"class" parameter

Objects vs. Arrays

Array Element Types

Timestamp Formats

Property Names

Empty and Null Property Values

JSON Schema

# Single Root Attributes


```
{
 ▾ track: {
      artist: "Paul Lekakis",
      created: "2015-08-18T19:57:07",
      genre: "Hi-NRG",
      id: "3305311F4A0FAAFEABD001D324906748B18FB24A",
      mp3: https://goo.gl/KgKoR7,
    ▾ ratings: [
      ▾ {
            created: "2015-08-20T12:24:44",
            rating: 4,
            username: "sublimatingraga37014"
        },
      ▾ {
            created: "2015-08-21T09:23:57",
            rating: 4,
            username: "untillableshowings34122"
        },
      ▾ {
            created: "2015-08-21T13:53:34",
            rating: 3,
            username: "megacephalousfusty75226"
        }
      ],
      title: "My House",
      updated: "2015-08-18T19:57:07"
   }
}
```



```
{
    artist: "Paul Lekakis",
    created: "2015-08-18T19:57:07",
    genre: "Hi-NRG",
    id: "3305311F4A0FAAFEABD001D324906748B18FB24A",
    mp3: https://goo.gl/KgKoR7,
  ▾ ratings: [
    ▾ {
          created: "2015-08-20T12:24:44",
          rating: 4,
          username: "sublimatingraga37014"
      },
    ▾ {
          created: "2015-08-21T09:23:57",
          rating: 4,
          username: "untillableshowings34122"
      },
    ▾ {
          created: "2015-08-21T13:53:34",
          rating: 3,
          username: "megacephalousfusty75226"
      }
    ],
    title: "My House",
    type: "track",
    updated: "2015-08-18T19:57:07"
}
```


22

# Objects vs. Arrays

- Two different ways to represent attributes:


```
{
 ▾ userprofile: {
    ▸ address: {…},
      created: "2015-01-28T13:50:56",
      dateOfBirth: "1986-06-09",
      email: "andy.bowman@games.com",
    ▸ favoriteGenres: [ … ],
      firstName: "Andy",
      gender: "male",
      lastName: "Bowman",
    ▾ phones: {
          cell: "212-771-1834"
      },
    ▸ picture: {…},
      pwd: "636f6c6f7261646f",
      status: "active",
      title: "Mr",
      updated: "2015-08-25T10:29:16",
      username: "copilotmarks61569"
   }
}
```

```
{
 ▾ userprofile: {
    ▸ address: {…},
      created: "2015-01-28T13:50:56",
      dateOfBirth: "1986-06-09",
      email: "andy.bowman@games.com",
    ▸ favoriteGenres: [ … ],
      firstName: "Andy",
      gender: "male",
      lastName: "Bowman",
    ▾ phones: [
        ▾ {
              number: "212-771-1834",
              type: "cell"
          }
      ],
    ▸ picture: {…},
      pwd: "636f6c6f7261646f",
      status: "active",
      title: "Mr",
      updated: "2015-08-25T10:29:16",
      username: "copilotmarks61569"
   }
}
```

# Array Element Types

- Array elements can be simple types, objects or arrays:

```
{
 ▾ playlist: {
        created: "2014-12-04T03:36:18",
        id: "003c6f65-641a-4c9a-8e5e-41c947086cae",
        name: "Eclectic Summer Mix",
        owner: "copilotmarks61569",
    ▾ tracks: [
            "9FFAF88C1C3550245A19CE3BD91D3DC0BE616778",
            "3305311F4A0FAAFEABD001D324906748B18FB24A",
            "0EB4939F29669774A19B276E60F0E7B47E7EAF58"
        ],
        updated: "2015-09-11T10:39:40",
        visibility: "PUBLIC"
    }
}
```

Array of strings

```
{
 ▾ playlist: {
        created: "2014-12-04T03:36:18",
        id: "003c6f65-641a-4c9a-8e5e-41c947086cae",
        name: "Eclectic Summer Mix",
        owner: "copilotmarks61569",
    ▾ tracks: [
         ▾ {
                id: "9FFAF88C1C3550245A19CE3BD91D3DC0BE616778"
            },
         ▾ {
                id: "3305311F4A0FAAFEABD001D324906748B18FB24A"
            },
         ▾ {
                id: "0EB4939F29669774A19B276E60F0E7B47E7EAF58"
            }
        ],
        updated: "2015-09-11T10:39:40",
        visibility: "PUBLIC"
    }
}
```

Array of objects

# Timestamp Formats

- When storing timestamps, you have at least 3 options:

```
{
  ▾ country: {
        countryCode: "US",
        gdp: 53548,
        name: "United States of America",
        population: 325296592,
        region: "Americas",
        region-number: 21,
        sub-region: "Northern America",
        updated: "2010-07-15T15:34:27"
    }
}
```

String (ISO 8601)

```
{
  ▾ country: {
        countryCode: "US",
        gdp: 53548,
        name: "United States of America",
        population: 325296592,
        region: "Americas",
        region-number: 21,
        sub-region: "Northern America",
        updated: 1279208067000
    }
}
```

Number (Unix style)

```
{
  ▾ country: {
        countryCode: "US",
        gdp: 53548,
        name: "United States of America",
        population: 325296592,
        region: "Americas",
        region-number: 21,
        sub-region: "Northern America",
      ▾ updated: [
            2010,
            7,
            15,
            15,
            34,
            27
        ]
    }
}
```

Array of time components

- Make timestamp values relative to UTC





25

# Property Names

Choose meaningful property names

Be consistent in naming properties

e.g. country_code vs. countryCode (preferred)

Array types should have plural property names

All other property names should be singular

Avoid (if possible) reserved words in your database system and programming language(s)

e.g. `user`  // Reserved word in Couchbase Server

Avoid (if possible) special characters such as hypens

e.g. `region-number`  // Contains a hyphen

# Empty and Null Property Values

Keep in mind that JSON supports optional properties

If a property has a null value, consider dropping it from the JSON, unless there's a good reason not to

N1QL makes it easy to test for missing or null property values

```
SELECT * FROM couchmusic1 WHERE userprofile.address IS NULL;

SELECT * FROM couchmusic1 WHERE userprofile.gender IS MISSING;
```

Be sure your application code handles the case where a property value is missing





27

# JSON Schema

Couchbase Server pays absolutely no attention to the shape of your JSON documents so long as they are well-formed

There are times when it is useful to validate that a JSON document conforms to some expected shape

JSON Schema is a JSON-based format for defining the structure of JSON data

There are implementations for most popular programming languages

Learn more here: [http://json-schema.org](http://json-schema.org)

```
{
  id: "http::/couchmusic.org/schema/couchmusic2-country.json",
  $schema: http://json-schema.org/draft-04/schema#,
  type: "object",
  properties: {
    countryCode: {
      type: "string",
      minLength: 2,
      maxLength: 2
    },
    gdp: {
      type: "integer",
      minimum: 0
    },
    name: {
      type: "string"
    },
    population: {
      type: "number",
      minimum: 0
    },
    region-number: {
      type: "integer",
      minimum: 0
    },
    type: {
      enum: [
        "country"
      ]
    },
    updated: {
      type: "string",
      format: "date-time"
    }
  },
  required: [
    "countryCode",
    "gdp",
    "name",
    "population",
    "region-number",
    "updated"
  ],
  additionalProperties: false
}
```

# 4 Data Nesting

# Data Nesting (aka Denormalization)

As you know, relational database design promotes separating data using normalization, which doesn't scale

For NoSQL systems, we often avoid normalization so that we can scale

Nesting allows related objects to be organized into a hierarchical tree structure where you can have multiple levels of grouping

Rule of thumb is to nest no more than 3 levels deep unless there is a very good reason to do so

You will often want to include a timestamp in the nested data

# Example #1 of Data Nesting

Playlist with owner attribute containing username of corresponding userprofile

```
{
  ▾ playlist: {
        created: "2014-12-04T03:36:18",
        id: "003c6f65-641a-4c9a-8e5e-41c947086cae",
        name: "Eclectic Summer Mix",
        owner: "copilotmarks61569",
      ▸ tracks: [ … ],
        updated: "2015-09-11T10:39:40",
        visibility: "PUBLIC"
    }
}
```

```
{
  ▾ userprofile: {
      ▸ address: { … },
        created: "2015-01-28T13:50:56",
        dateOfBirth: "1986-06-09",
        email: "andy.bowman@games.com",
      ▸ favoriteGenres: [ … ],
        firstName: "Andy",
        gender: "male",
        lastName: "Bowman",
      ▸ phones: { … },
      ▸ picture: { … },
        pwd: "636f6c6f7261646f",
        status: "active",
        title: "Mr",
        updated: "2015-08-25T10:29:16",
        username: "copilotmarks61569"
    }
}
```

31

# Example #1 of Data Nesting

Playlist with owner attribute containing a subset of the corresponding userprofile


```
{
  ▾ playlist: {
        created: "2014-12-04T03:36:18",
        id: "003c6f65-641a-4c9a-8e5e-41c947086cae",
        name: "Eclectic Summer Mix",
      ▾ owner: {
            created: "2015-01-28T13:50:56",
            firstName: "Andy",
            lastName: "Bowman",
            title: "Mr",
            updated: "2015-08-25T10:29:16",
            username: "copilotmarks61569"
        },
      ▸ tracks: [ … ],
        updated: "2015-09-11T10:39:40",
        visibility: "PUBLIC"
    }
}
```


* Note the inclusion of the **updated** attribute


```
{
  ▾ userprofile: {
      ▸ address: { … },
        created: "2015-01-28T13:50:56",
        dateOfBirth: "1986-06-09",
        email: "andy.bowman@games.com",
      ▸ favoriteGenres: [ … ],
        firstName: "Andy",
        gender: "male",
        lastName: "Bowman",
      ▸ phones: { … },
      ▸ picture: { … },
        pwd: "636f6c6f7261646f",
        status: "active",
        title: "Mr",
        updated: "2015-08-25T10:29:16",
        username: "copilotmarks61569"
    }
}
```

# Example #2 of Data Nesting

Playlist with tracks attribute containing an array of track IDs

```
{
  ▾ playlist: {
        created: 1417685778000,
        id: "003c6f65-641a-4c9a-8e5e-41c947086cae",
        name: "Eclectic Summer Mix",
      ▸ owner: {…},
      ▾ tracks: [
            "9FFAF88C1C3550245A19CE3BD91D3DC0BE616778",
            "3305311F4A0FAAFEABD001D324906748B18FB24A",
            "0EB4939F29669774A19B276E60F0E7B47E7EAF58"
        ],
        updated: 1441985980000,
        visibility: "PUBLIC"
    }
}
```

# Example #2 of Data Nesting

Playlist with tracks attribute containing an array of track IDs


```
{
  ▾ playlist: {
        created: 1417685778000,
        id: "003c6f65-641a-4c9a-8e5e-41c947086cae",
        name: "Eclectic Summer Mix",
      ▸ owner: {…},
      ▾ tracks: [
          ▾ {
                artist: "Gene Harris",
                genre: "Jazz Blues",
                id: "9FFAF88C1C3550245A19CE3BD91D3DC0BE616778",
                mp3: https://goo.gl/DEYx4X,
                title: "Battle Hymn of the Republic",
                updated: 1445167377000
            },
          ▸ {…},
          ▸ {…}
        ],
        updated: 1441985980000,
        visibility: "PUBLIC"
    }
}
```


* Note the inclusion of the **updated** attribute





34

# 5 Key Design

# Natural Keys

A key formed of attributes that exist in the real world:

- Phone numbers
- Usernames
- Social security numbers
- Account numbers
- SKU, UPC or QR codes
- Device IDs

Often the first choice for document keys

Be careful when working with any personally identifiable information (PII), sensitive personal information (SPI) or protected health information (PHI)





36

# Surrogate Keys

We often use surrogate keys when no obvious natural key exist

They are not derived from application data

They can be generated values

- 3305311F4A0FAAFEABD001D324906748B18FB24A (SHA-1)
- 003C6F65-641A-4CGA-8E5E-41C947086CAE (UUID)
- They can be sequential numbers (often implemented using the Counter feature of Couchbase Server)
- 456789, 456790, 456791, …





37

# Key Value Patterns

Common practice for users of Couchbase Server to follow patterns for formatting key values by using symbols such as single or double colons

DocType::ID

- userprofile::fredsmith79
- playlist::003c6f65-641a-4c9a-8e5e-41c947086cae

AppName::DocType::ID

- couchmusic::userprofile::fredsmith79

DocType::ParentID::ChildID

- playlist::fredsmith79::003c6f65-641a-4c9a-8e5e-41c947086cae

Supports easy document viewing in the Couchbase web console

# Lookup Key Pattern

The purpose of the Lookup Key Pattern is to allow multiple ways to reach the same data, essentially a secondary index

- For example, we want to lookup a Userprofile by their email address instead of their ID

To accomplish this, we create another small document that refers to the Userprofile document we are interested in

Implementing this pattern is straightforward, just create an additional document containing a single property that stores the key to the primary document

With the introduction of N1QL, this pattern will be less commonly used

# Example of Lookup Key Pattern

Lookup document can be JsonDocument or StringDocument

`userprofile::copilotmarks61569`

```
{
  ▾ userprofile: {
      ▸ address: {…},
        created: "2015-01-28T13:50:56",
        dateOfBirth: "1986-06-09",
        email: "andy.bowman@games.com",
      ▸ favoriteGenres: […],
        firstName: "Andy",
        gender: "male",
        lastName: "Bowman",
      ▸ phones: {…},
      ▸ picture: {…},
        pwd: "636f6c6f7261646f",
        status: "active",
        title: "Mr",
        updated: "2015-08-25T10:29:16",
        username: "copilotmarks61569"
    }
}
```

`andy.bowman@games.com`

```
{
    username: "copilotmarks61569"
}
```

JSON

`andy.bowman@games.com`

```
copilotmarks61569
```

String

# 6 Making Tradeoffs

# Making Tough Choices

Eric Brewer is famous for showing the trade-offs that are necessary when dealing with distributed systems

- Consistency, availability and partition tolerance are all desirable properties but we must choose the ones that are most important for our use cases

We must also make trade-offs in data modeling:

- Document size
- Complexity
- Speed
- Atomicity

# Document Size

Couchbase Server supports documents up to 20 Mb

Larger documents take more disk space, more time to transfer across the network and more time to serialize/deserialize

If you are dealing with documents that are potentially large (greater than 1 Mb), you must test thoroughly to find out if speed of access is adequate as you scale. If not, you will need to break up the document into smaller ones.

You may need to limit the number of dependent child objects you embed





43

# Complexity

Complexity affects every area of software systems including data modeling

Space saving sometimes comes at the complexity of data processing

E.g., consider the complexity of queries (N1QL)

```
SELECT c.name, COUNT(*) AS playlist_count
FROM couchmusic1.playlist p JOIN couchmusic1.userprofile u
   ON KEYS 'userprofile::' || p.owner JOIN couchmusic1.country c
      ON KEYS 'country::' || u.address.countryCode
WHERE c.`region-number` = 154
   AND p.visibility = 'PUBLIC'
   AND u.status = 'active'
GROUP BY c.name
ORDER BY c.name;
```

# Speed

As it relates to data modeling, speed of access is critical

When using N1QL to access data, keep in mind that query by document key is fastest and query by secondary index is usually much slower

If implementing an interactive use case, you will want to avoid using JOINs

You can use data duplication to improve the speed of accessing related data and thus trade improved speed for greater complexity and larger document size

Keep in mind that Couchbase Views can be used when up to the second accuracy is not required







45

# Atomicity

Atomicity in Couchbase Server is at the document level

Couchbase Server does not support transactions

- They can be simulated if you are willing to write and maintain additional code to implement them (generally not recommended)

If you absolutely need changes to be atomic, they will have to be part of the same document

The maximum document size for Couchbase Server may limit how much data you can store in a single document

Multi-document transactions will be introduced in Couchbase 6.5



46

# Embed vs. Refer

## When to embed:

- Reads greatly outnumber writes
- You're comfortable with the slim risk of inconsistent data across the multiple copies
- You're optimizing for speed of access

## When to refer:

- Consistency of the data is a priority
- You want to ensure your cache is used efficiently
- The embedded version would be too large or complex





47

# Summary

In this module, you have learned to:

- Make full use of JSON capabilities
- Use data nesting to minimize the need for JOINs
- Establish key value patterns and use them consistently
- Be clear about the trade-offs you are making, document your decisions and the assumptions they are based on

Couchbase

# Thank You