Architecture and Administration Basics

Workshop **Day 2 – Data Modeling**

1 | Data Modeling

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

2 Data Modeling Considerations

Goals of Data Modeling for N1QL

1. Define entities
2. Define relationships
3. Define document boundaries
4. Express relationships

Defining Document Boundaries

Defining document boundaries entails

- Identifying parent and child objects
- Deciding whether to embed child objects

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

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)

Expressing Relationships

3 ways to express relationships in Couchbase

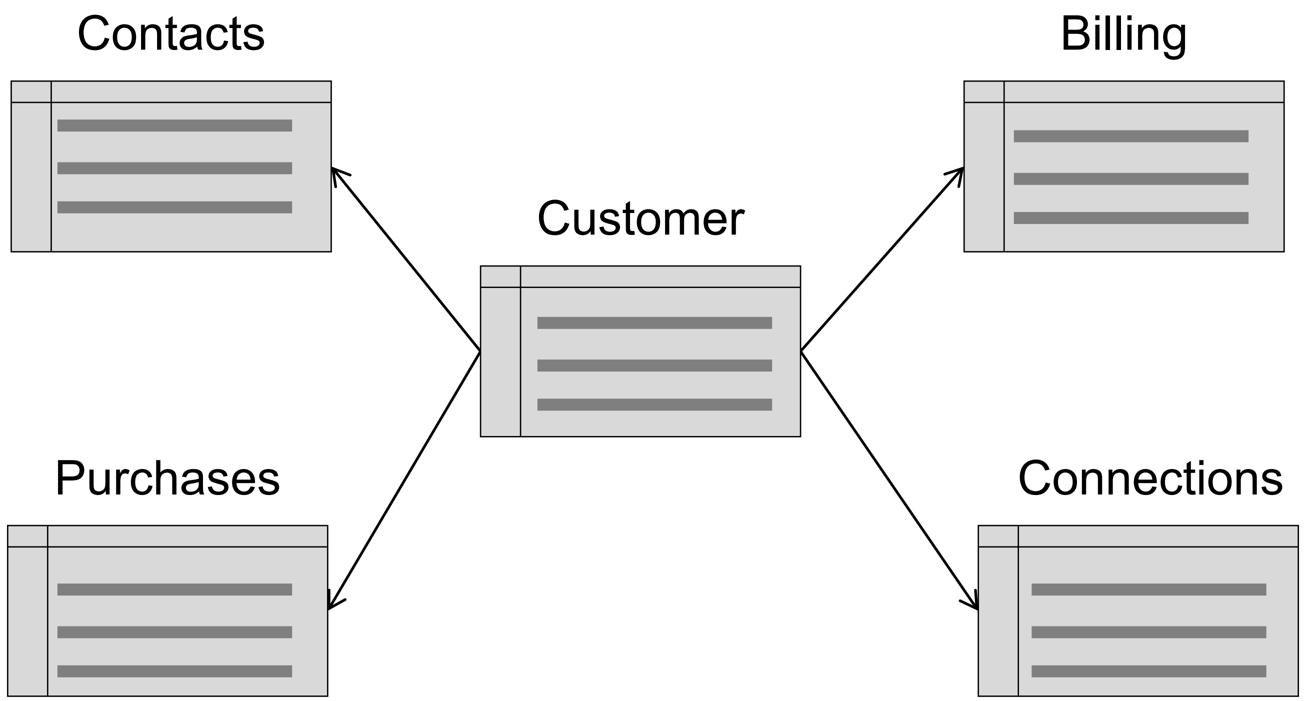
1. Parent contains keys of children (outbound)
2. Children contain key of parent (inbound)
3. Both of the above (dual)

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)

3 Data Modeling with JSON Objects

Modeling Data in Relational World



- Rich structure
  - Normalize & JOIN Queries
- Relationships
  - JOINS and Constraints
- Value evolution
  - INSERT, UPDATE, DELETE
- Structure evolution
  - ALTER TABLE
  - Application Downtime
  - Application Migration
  - Application Versioning

Using JSON For Real World Data

Table: Customer

CustomerID	Name	DOB
CBL2017	Jane Smith	1990-01-30

Customer DocumentKey: CBL2017

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

OR

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

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

Using JSON to Store Data

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.

Using JSON to Store Data

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

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

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"
        }
    ]
}
```

Using JSON to Store Data

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"
        }
    ]
}
```

Using JSON to Store Data

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 }
    ]
}
```

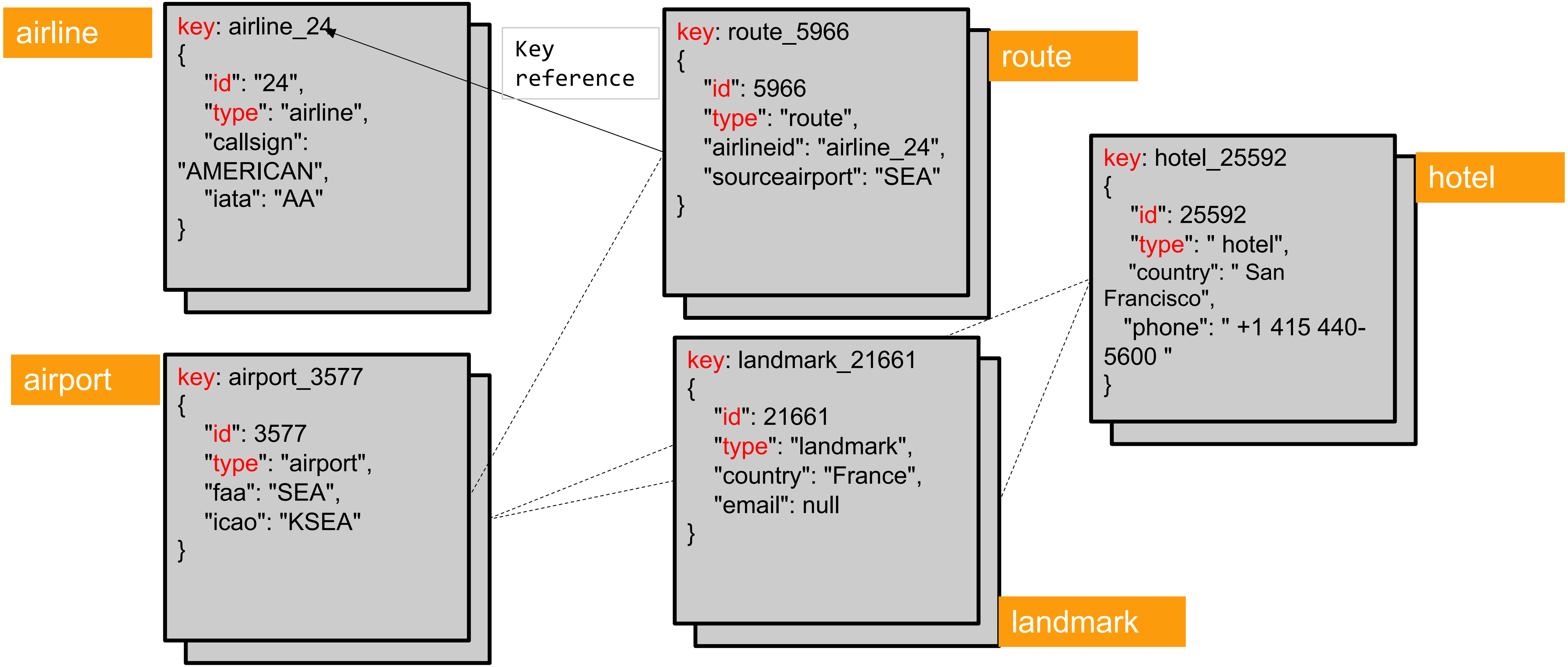
Models for Representing Data

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

4

Travel Sample

Travel-Sample



Travel-sample: Hotel Document

```
 "docid": "hotel_25390"
{
      "address": "321 Castro St",
…
      "city": "San Francisco",
      "country": "United States",
      "description": "An upscale bed and breakfast in a restored house.",
      "directions": "at 16th",
      "geo": {
        "accuracy": "ROOFTOP",
        "lat": 37.7634,
        "lon": -122.435
      },
      "id": 25390,
      "name": "Inn on Castro",
      "phone": "+1 415 861-0321",
      "price": "$95–$190",
      "public_likes": ["John Smith", "Joe Carl", "Jane Smith", "Kate Smith"],
      "reviews": [
        {
          "author": "Mason Koepp",
          "content": ”blah-blah",
          "date": "2012-08-23 16:57:56 +0300",
          "ratings": {
            "Check in / front desk": 3,
            "Cleanliness": 3,
            "Location": 4,
            "Overall": 2,
            "Rooms": 2,
            "Service": -1,
            "Value": 2
          }
        }
      ],
      "state": "California",
```

Document Key

city: Attributes (key-value pairs)

geo: Object. **1:1** relationship

public_likes: Array of strings: Embedded 1:many relationship

reviews: Array of objects: Embedded **1:N** relationship

ratings: object within an array

Querying Objects

```
> select h.geo from `travel-sample` h
where type = 'hotel' and city = 'San
Francisco' and meta().id = "hotel_25390”;
[
  {
    "geo": {
      "accuracy": "ROOFTOP",
      "lat": 37.7634,
      "lon": -122.435
    }
  }
]
```

```
> select h.geo.lat, h.geo.lon from
`travel-sample` h where type = 'hotel' and
city = 'San Francisco' and meta().id =
"hotel_25390"
[
  {
    "lat": 37.7634,
    "lon": -122.435
  }
]
```

```
> select reviews[*].ratings from `travel-
sample` h where type = 'hotel' and city =
'San Francisco' and meta().id =
"hotel_25390" ;
[
  {
    "ratings": [
      {
        "Business service": -1,
        "Check in / front desk": 3,
        "Cleanliness": 3,
        "Location": 4,
        "Overall": 2,
        "Rooms": 2,
        "Service": -1,
        "Value": 2
      }
    ]
  }
]
```

Querying Objects: Accessing data within Objects

```
>select name, city from `travel-sample` h
where geo = {
        "accuracy": "ROOFTOP",
        "lat": 37.7634,
        "lon": -122.435
     };

[
  {
    "city": "San Francisco",
    "name": "Inn on Castro"
  }
]
```

```
> select h.geo.lat, h.geo.lon from
`travel-sample` h where type = 'hotel' and
city = 'San Francisco' and meta().id =
"hotel_25390" [
  {
    "lat": 37.7634,
    "lon": -122.435
  }
]
```

```
select name, city from `travel-sample` h
where geo.accuracy = "ROOFTOP" and geo.lat
between 37.7 and 37.8
and geo.lon between -122.4 and -122.3;

[
  {
    "city": "San Francisco",
    "name": "Courtyard San Francisco Downtown"
  },
  {
    "city": "San Francisco",
    "name": "Hotel Vitale"
  },
  {
    "city": "San Francisco",
    "name": "South Park"
  },
  {
    "city": "San Francisco",
    "name": "City Kayak"
  },
```

Querying Objects: Search WITHIN

```
select COUNT(1)
FROM system:dual
WHERE ANY v WITHIN {"a":1, "b": "Hello"}
           SATISFIES v = "Hello"
      END;

[  {
      "$1": 1
   }
]
```

```
select COUNT(1)
FROM system:dual
WHERE ANY v WITHIN {"a":1, "b": "World"}
           SATISFIES v = "Hello"
      END;

[  {
      "$1": 0
   }
]
```

```
SELECT COUNT(1)
FROM system:dual
WHERE ANY v WITHIN
  {  "a":1,
     "b": {
          "x": "Mercury",
          "y": "Venus",
          "z": "Earth”
          }
  }
           SATISFIES v = "Earth" END;
[  {
      "$1": 1
   }
]
```

Thank you

Couchbase