Bringing SQL to NoSQL:
Rich, Declarative Querying for NoSQL

Gerald Sangudi | @sangudi | Chief Architect | Couchbase
Keshav Murthy | @rkeshavmurthy | Director | Couchbase
Team @N1QL

Agenda

- SQL
- NoSQL
  - Motivation
  - Rich Data and JSON
  - Landscape
- SQL for NoSQL
  - Motivation
  - Considerations
- N1QL: Bringing SQL to NoSQL
  - Enterprises using N1QL
  - Features
  - Indexing
  - Architecture
  - Ecosystem
- Q & A

SQL

SQL

Structured Query Language is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS).

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and a data control language.

Source: https://en.wikipedia.org/wiki/SQL

SQL: Core Features

- Based on Relational Data Model
- Declarative Query Language
  - SELECT, UPDATE, DELETE, INSERT, MERGE
  - Arithmetic, Logical, Set operators
- Data types: Numerical, Character, Date, Time, etc.
- Schema design & management

Huge Impact of SQL on Business Applications

SQL: Database Features for Application Development

- Transactions
  - ACID
- Features that enrich SQL
  - Stored procedures, triggers, views, indexes, catalogs
  - Optimizers
- 4GL
- Extensions, integrations
  - Spatial, Text, Queues

NoSQL

NoSQL: Motivation

Why NoSQL?

- Agility
- Scalability
- Performance
- Availability

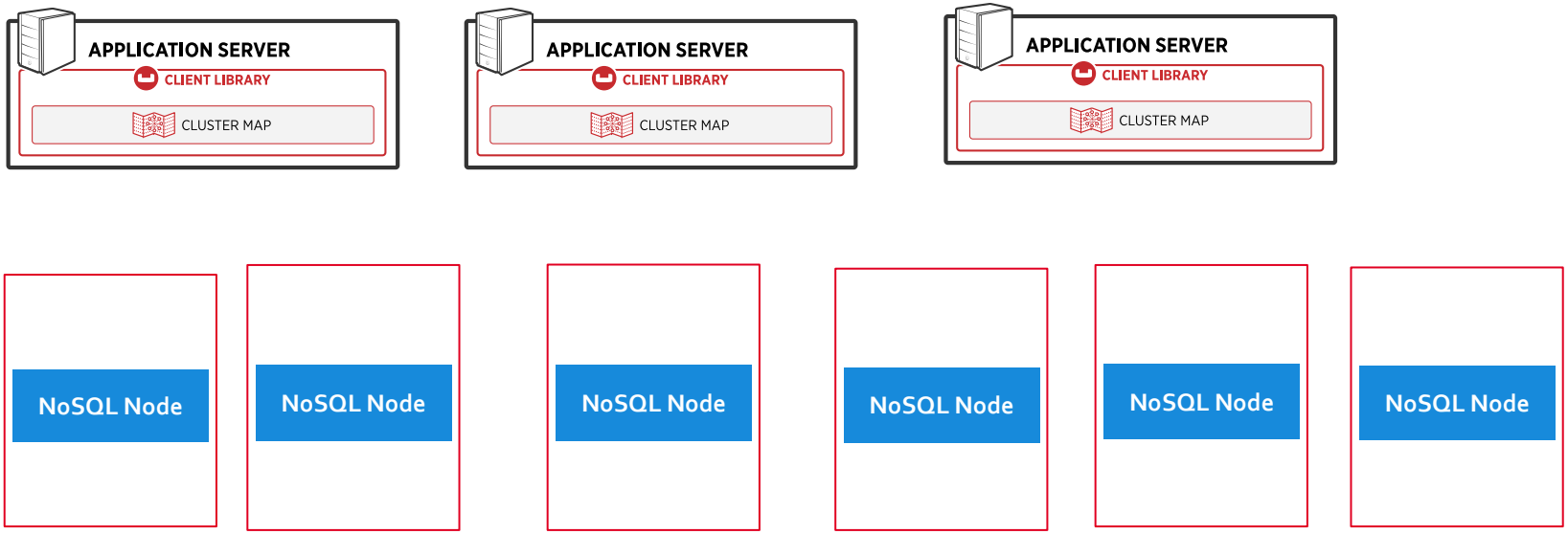
Why NoSQL? Agility

- Schema flexibility
- Easier management of change
  - In the business requirements
  - In the structure of the data
- Change is inevitable

Why NoSQL? Scalability

- Elastic scaling
  - Size your cluster for today
  - Scale out on demand
- Cost effective scaling
  - Commodity hardware
  - On premise or on cloud
  - Scale OUT instead of Scale UP

Why NoSQL? Scalability: Elastic Scaling



Why NoSQL? Performance

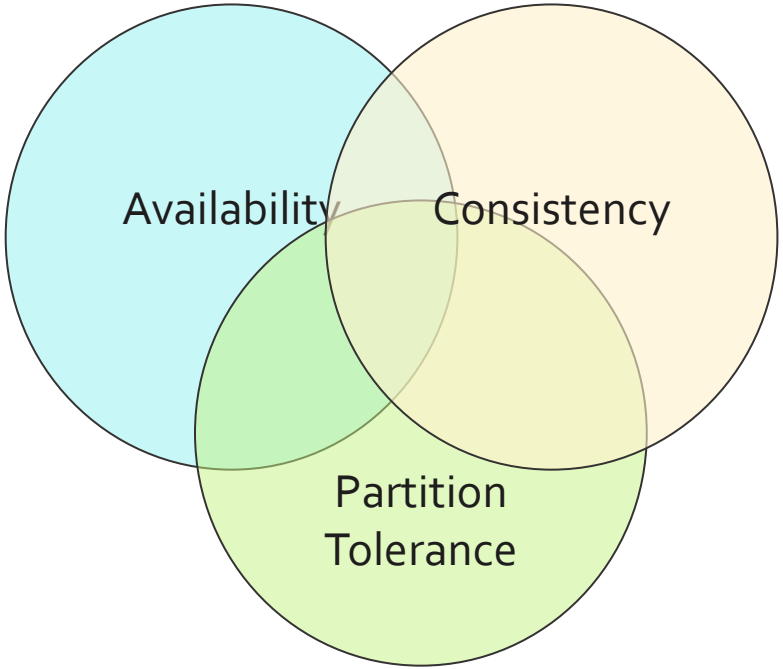
- NoSQL systems are optimized for specific access patterns
- Low response time for web & mobile user experience
  - Millisecond latency
- Consistently high throughput to handle growth

Why NoSQL? Availability

- Built-in replication and fail-over
  - No application downtime when hardware fails
- Online maintenance & upgrade
  - No application downtime

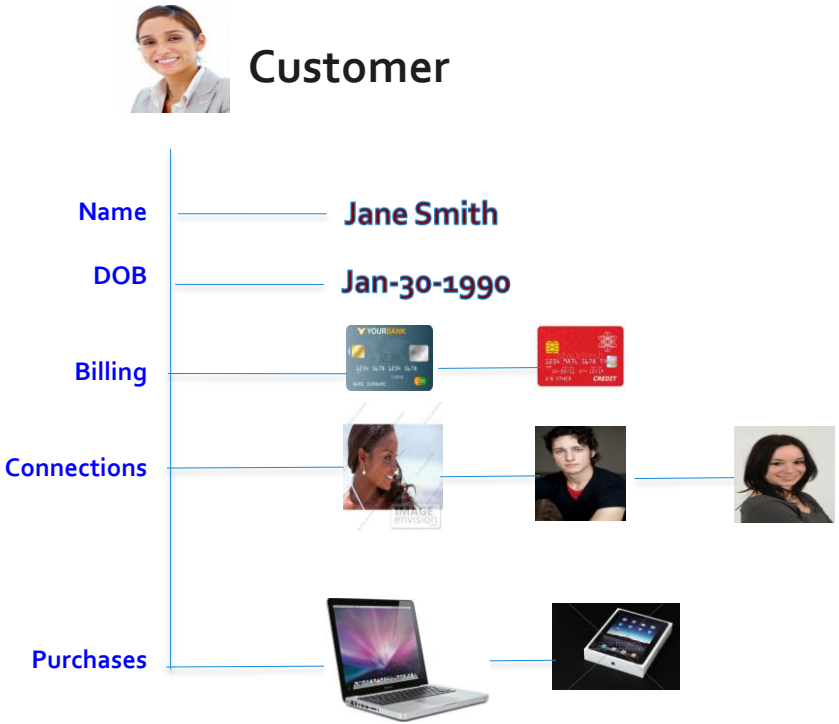
CAP Theorem - 101

- Each distributed system can only provide two properties simultaneously.
- Partition Tolerance is required in our systems
- Choice is between Availability and Consistency



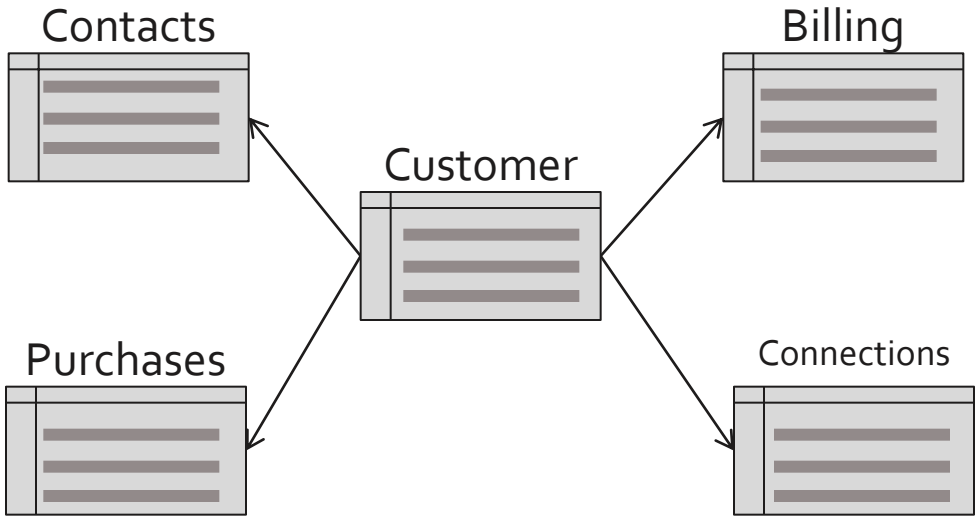
NoSQL: Rich Data and JSON

Properties of Real-World Data



- **Rich structure**
  - Attributes, Sub-structure
- **Relationships**
  - To other data
- **Value evolution**
  - Data is updated
- **Structure evolution**
  - Data is reshaped

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

JSON 101

- Used to represent object data in text
- Representation
  - `"Key":"Value"`
- Data Types:
  - Number, Strings, Boolean, objects, Arrays, NULL
- Hierarchical

```
{
    "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"
        }
    ],
    "address" :
        {
            "Street"   : "10, Downing Street",
            "City"     : "San Francico”,
            “State”    : “California”,
            “zip”      :94401
        }
}
```

Using JSON For Real World Data

Table: Customer

CustomerID	Name	DOB
CBL2015	Jane Smith	1990-01-30

Customer DocumentKey: CBL2015

```
{
  "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
CBL2015	Jane Smith	1990-01-30

Table: Billing

CustomerID	Type	Cardnum	Expiry
CBL2015	visa	5827…	2019-03

Customer DocumentKey: CBL2015

```
{
  "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
CBL2015	Jane Smith	1990-01-30

Table: Billing

CustomerID	Type	Cardnum	Expiry
CBL2015	visa	5827…	2019-03
CBL2015	master	6274…	2018-12

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

Customer DocumentKey: CBL2015

```
{
  "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
CBL2015	XYZ987	Joe Smith
CBL2015	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: CBL2015

```
{
  "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

DocumentKey: **CBL2015**

Contacts

CustomerID	ConnId	Name
CBL2015	XYZ987	Joe Smith
CBL2015	SKR007	Sam Smith

Billing

Customer ID	Type	Cardnum	Expiry
CBL2015	visa	5827...	2019-03
CBL2015	master	6274...	2018-12

Customer

CustomerID	Name	DOB
CBL2015	Jane Smith	1990-01-30

Purchases

CustomerID	item	amt
CBL2015	mac	2823.52
CBL2015	ipad2	623.52

Connections

CustomerID	ConnId	Name
CBL2015	XYZ987	Joe Smith
CBL2015	SKR007	Sam Smith

```
{
  "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

Data Concern	Relational Model	JSON Document Model (NoSQL)
Rich Structure	▪ Multiple flat tables ▪ Constant assembly / disassembly	▪ Documents ✓ No assembly required!
Relationships	▪ Represented ✓ Queried (SQL)	▪ Represented ▪ **Queried? Not until now…**
Value Evolution	▪ Data can be updated	▪ Data can be updated
Structure Evolution	▪ Uniform and rigid ▪ Manual change (disruptive)	✓ Flexible ✓ Dynamic change

NoSQL: Landscape

NoSQL: Landscape

Document
- Couchbase
- MongoDB
- DynamoDB
- DocumentDB

Key-Value
- Riak
- BerkeleyDB
- Redis
- …

Graph
- OrientDB
- Neo4J
- DEX
- GraphBase

Wide Column
- Hbase
- Cassandra
- Hypertable

SQL for NoSQL

SQL for NoSQL: Motivation

Business Use Cases Drive Applications

Search stores for the shoe customer is looking for?

Process Order Checkout

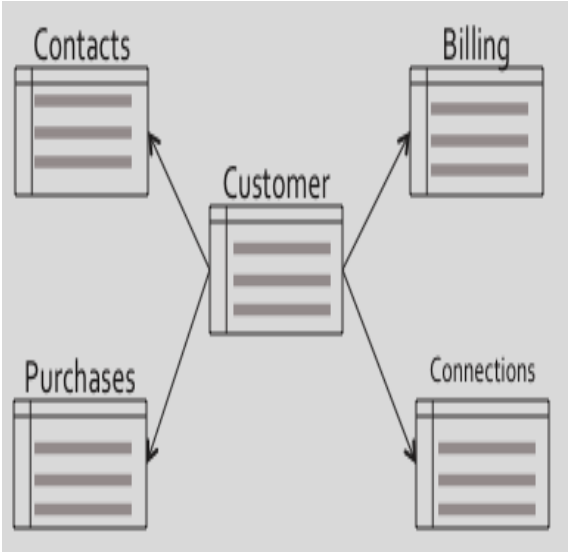
How many new customers we got last month?

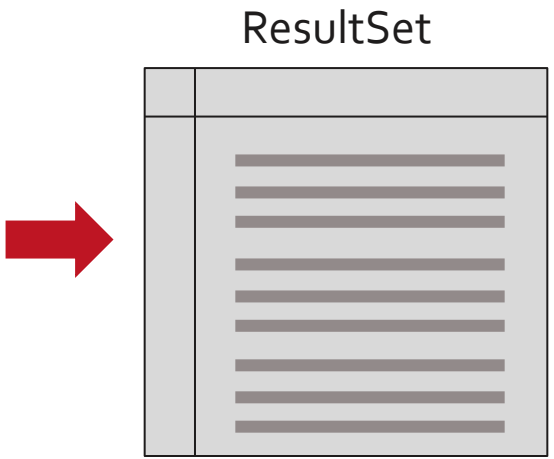
Generate a list of shipment due today

Load the new inventory data

Retrieve the customer order

Merge the customer lists





NoSQL Data Access

- NoSQL systems provide specialized APIs
  - Key-Value get and set
  - Script based query APIs
  - Limited declarative query
- Richer query is needed

LoyaltyInfo

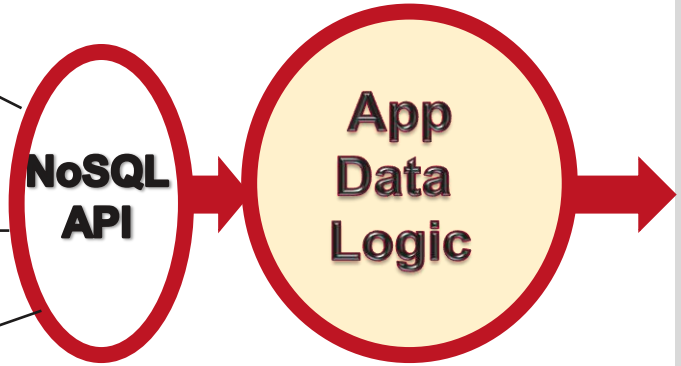

```
{
  "Name" : "Jane Smith",
  "DOB" : "1990-01-30",
  "Billing" : [
    {
      "type"   : "visa",
      "cardnum" : "5827-2842-2847-3909",
      "expiry" : "2019-03"
    },
    {
      "type"   : "master",
      "cardnum" : "6274-2842-2847-3909",
      "expiry" : "2019-03"
    }
  ],
  "Connections"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


CUSTOMER


```
{
  "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 }
  ]
}
```


Orders


```
{
  "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 }
  ]
}
```

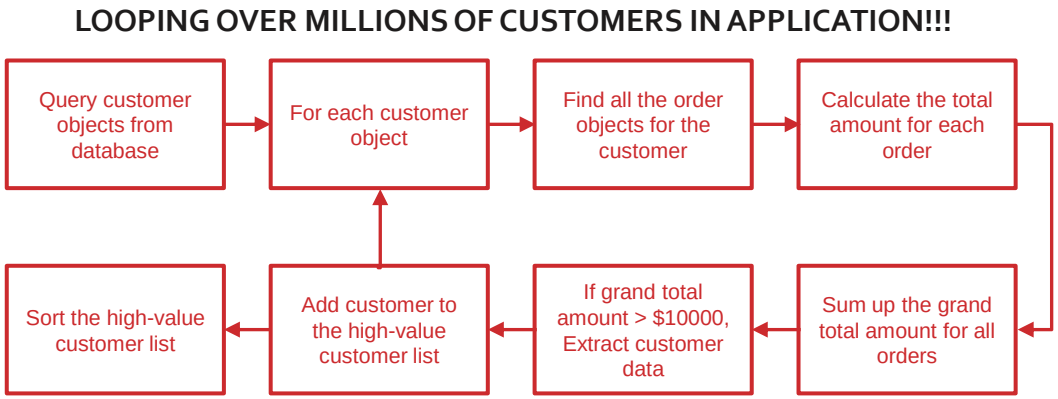
ResultDocuments


```
{
  "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 }
  ]
}
```


- NoSQL systems provide specialized APIs
- Key-Value get and set
- Each task requires custom built program
- Should test & maintain it

Client Side Implementation of Query is Inadequate

Find High-Value Customers with Orders > $10000



- Complex codes and logic
- Inefficient processing on client side

Why SQL for NoSQL?

SQL for NoSQL: Considerations

Considerations

- Flexible Schema
  - Cannot rely on predefined schema
  - columns, data types, data comparison
- Nested Objects
  - support scalars, objects and array
  - SQL operators on nested objects
- Work with distributed data store
- Query Performance Optimization
  - Exploit data store performance
  - Design right kinds of indices
  - Optimizer

```
{
    "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"
        }
    ],
    "address" :
        {
            "Street"   : "10, Downing Street",
            "City"     : "San Francico”,
            “State”    : “California”,
            “zip”      :94401
        }
}
```

Industry Is Extending SQL to Query JSON

- JSON support in SQL
  - IBM Informix
  - Oracle
  - PostgreSQL
  - MySQL
- Microsoft: SQL for DocumentDB
- UC San Diego: SQL++
- **Couchbase: N1QL**

N1QL: Bringing SQL to NoSQL

N1QL: Agenda

- Enterprises Using N1QL
- Approach
- Features
- Indexing
- Architecture
- Ecosystem

N1QL: Enterprises Using N1QL

Enterprises using Couchbase

Technology

Retail & Apparel

Communications

E-Commerce & Digital Advertising

Finance & Business Services

Travel & Hospitality

Games & Gaming

Media & Entertainment

N1QL: Approach

LoyaltyInfo

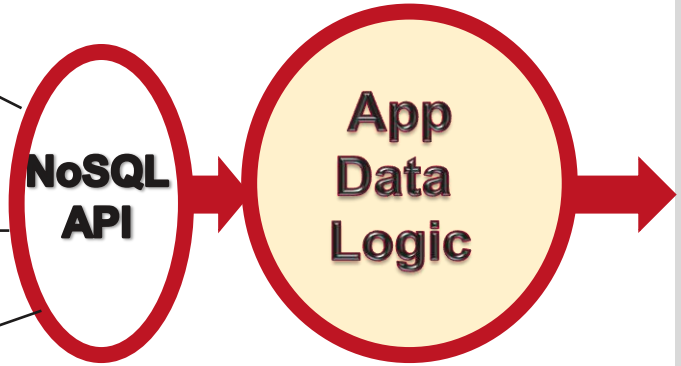

```
{
  "Name" : "Jane Smith",
  "DOB" : "1990-01-30",
  "Billing" : [
    {
      "type"   : "visa",
      "cardnum" : "5827-2842-2847-3909",
      "expiry" : "2019-03"
    },
    {
      "type"   : "master",
      "cardnum" : "6274-2842-2847-3909",
      "expiry" : "2019-03"
    }
  ],
  "Connections"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


CUSTOMER


```
{
  "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 }
  ]
}
```


Orders


```
{
  "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 }
  ]
}
```

ResultDocuments


```
{
  "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 }
  ]
}
```


- NoSQL systems provide specialized APIs
- Key-Value get and set
- Each task requires custom built program
- Should test & maintain it

LoyaltyInfo

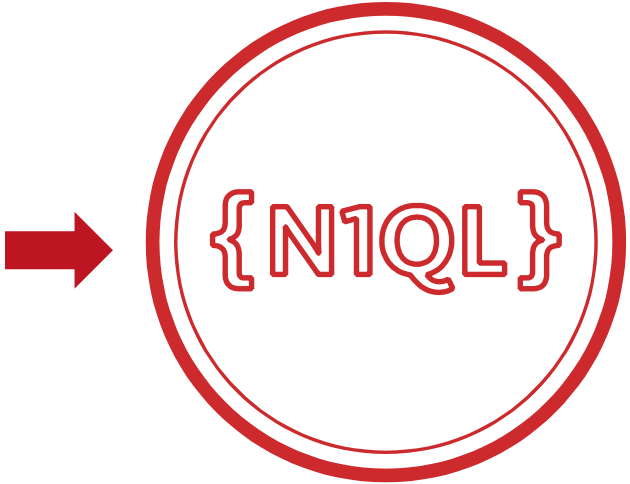
```
{
  "Name" : "Jane Smith",
  "DOB" : "1990-01-30",
  "Billing" : [
    {
      "type"   : "visa",
      "cardnum" : "5827-2842-2847-3909",
      "expiry" : "2019-03"
    },
    {
      "type"   : "master",
      "cardnum" : "6274-2842-2847-3909",
      "expiry" : "2019-03"
    }
  ],
  "Connections"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

CUSTOMER

```
{
  "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 }
  ]
}
```

Orders

```
{
  "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 }
  ]
}
```



ResultDocuments

```
{
  "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 }
  ]
}
```

Considerations: SQL for NoSQL

- Flexible Schema
  - Cannot rely on predefined schema
  - columns, data types, data comparison
- Nested Objects
  - support scalars, objects and array
  - SQL operators on nested objects
- Work with distributed data store
- Query Performance Optimization
  - Exploit data store performance
  - Design right kinds of indices
  - Optimizer

```
{
    "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"
        }
    ],
    "address" :
        {
            "Street"   : "10, Downing Street",
            "City"     : "San Francico”,
            “State”    : “California”,
            “zip”      :94401
        }
}
```

N1QL: Approach

- Flexible Schema
  - Rely on JSON data interpretation
  - Define 4-value predicate logic
  - True, False, NULL, MISSING
- Nested Objects
  - Key name becomes column reference
  - Use dot-notation and array[] reference
- SQL operators on nested objects
  - select, join, project operators
  - nest and unnest for arrays & objects
- Query Performance Optimization

```
SELECT c.name,
       c.address.zip,
       c.phone[0]
FROM  customer c
WHERE c.address.zip = 94587
AND   ANY s IN c.status
          SATISFIES
          s = 'Premium'
      END
AND purchases IS NOT MISSING;
```

N1QL: Goal

Give developers and enterprises an expressive, powerful, and complete language for querying, transforming, and manipulating JSON data.

N1QL: Features

N1QL: SELECT Statement

```
SELECT    customers.id,
          customers.NAME.lastname,
          customers.NAME.firstname
          Sum(orderline.amount)
FROM      orders UNNEST orders.lineitems AS orderline
          JOIN customers ON KEYS orders.custid
WHERE customers.state = 'NY'
GROUP BY customers.id,
          customers.NAME.lastname
HAVING    sum(orderline.amount) > 10000
ORDER BY sum(orderline.amount) DESC
```

- Dotted sub-document reference
- Names are CASE-SENSITIVE

- UNNEST to flatten the arrays

JOINS with Document KEY of customers

N1QL: Examples

```
SELECT d.C_ZIP, SUM(ORDLINE.OL_QUANTITY) AS  TOTALQTY
FROM CUSTOMER d
         UNNEST ORDERS as CUSTORDERS
             UNNEST CUSTORDERS.ORDER_LINE AS ORDLINE
WHERE d.C_STATE = ”NY”
GROUP BY d.C_ZIP
ORDER BY TOTALQTY DESC;

INSERT INTO CUSTOMER("PQR847", {"C_ID":4723, "Name":"Joe"});

UPDATE CUSTOMER c SET c.STATE=“CA”, c.C_ZIP = 94501
                  WHERE c.ID = 4723;
```

N1QL: Composable SELECT Statement

```
SELECT *
FROM
      (        SELECT   a, b, c
               FROM     us_cust
               WHERE    x = 1
               ORDER BY x LIMIT 100 OFFSET 0
            UNION ALL
               SELECT   a, b, c
               FROM     canada_cust
               WHERE    y = 2
               ORDER BY x LIMIT 100 OFFSET 0) AS newtab
LEFT OUTER JOIN contacts
     ON KEYS newtab.c.contactid
ORDER BY a, b, c
LIMIT 10 OFFSET 0
```

N1QL: SELECT Statement Highlights

- **Querying across relationships**
  - JOINs
  - Subqueries
- **Aggregation**
  - MIN, MAX
  - SUM, COUNT, AVG, ARRAY_AGG [DISTINCT]
- **Combining result sets using set operators**
  - UNION, UNION ALL, INTERSECT, EXCEPT

N1QL : Query Operators [1 of 2]

- **USE KEYS …**
  - Direct primary key lookup bypassing index scans
  - Ideal for hash-distributed datastore
  - Available in SELECT, UPDATE, DELETE
- **JOIN … ON KEYS …**
  - Nested loop JOIN using key relationships
  - Ideal for hash-distributed datastore
  - Current implementation supports INNER and LEFT OUTER joins

N1QL : Query Operators [2 of 2]

- **NEST**
  - Special JOIN that embeds external child documents under their parent
  - Ideal for JSON encapsulation
- **UNNEST**
  - Flattening JOIN that surfaces nested objects as top-level documents
  - Ideal for decomposing JSON hierarchies

JOIN, NEST, and UNNEST can be chained in any combination

N1QL : Expressions for JSON

Ranging over collections	• WHERE **ANY** c IN children SATISFIES c.age > 10 END • WHERE **EVERY** r IN ratings SATISFIES r > 3 END
Mapping with filtering	• **ARRAY** c.name FOR c IN children **WHEN** c.age > 10 END
Deep traversal, SET, and UNSET	• WHERE ANY node **WITHIN** request SATISFIES node.type = "xyz" END • UPDATE doc **UNSET** c.field1 FOR c **WITHIN** doc END
Dynamic Construction	• SELECT **{ "a": expr1, "b": expr2 }** AS obj1, name FROM … // Dynamic object • SELECT **[a, b]** FROM … // Dynamic array
Nested traversal	• SELECT **x.y.z, a[0]** FROM **a.b.c** …
IS [NOT] MISSING	• WHERE name **IS MISSING**

N1QL : Data Types from JSON

N1QL supports all JSON data types

- Numbers
- Strings
- Booleans
- Null
- Arrays
- Objects

N1QL : Data Type Handling

Non-JSON data types

- MISSING
- Binary

Data type handling

- Date functions for string and numeric encodings
- Total ordering across all data types
  - Well defined semantics for ORDER BY and comparison operators
- Defined expression semantics for all input data types
  - No type mismatch errors

N1QL: Data Modification Statements

- UPDATE … SET … WHERE …
- DELETE FROM … WHERE …
- INSERT INTO … (KEY, VALUE) VALUES …
- INSERT INTO … (KEY …, VALUE …) SELECT …
- MERGE INTO … USING … ON …

  WHEN [NOT] MATCHED THEN …

Note: Couchbase provides per-document atomicity.

N1QL: Data Modification Statements

JSON literals can be used in any expression

```
INSERT INTO ORDERS (KEY, VALUE)
VALUES ("1.ABC.X382", {"O_ID":482, "O_D_ID":3, "O_W_ID":4});

UPDATE ORDERS
    SET O_CARRIER_ID = ”ABC987”
WHERE O_ID = 482 AND O_D_ID = 3 AND O_W_ID = 4

DELETE FROM NEW_ORDER
WHERE NO_D_ID = 291 AND
      NO_W_ID = 3482 AND
      NO_O_ID = 2483
```

N1QL: Indexing

Index Statements

- CREATE INDEX ON …
- DROP INDEX …
- EXPLAIN …

Highlights

- Composite Index
- Array Index
- Functional index
- Partial Index

Index Overview: Secondary Index

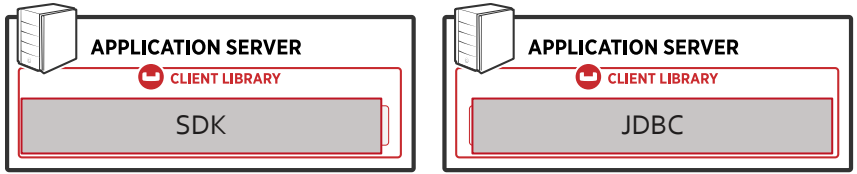
Document key: "customer534"

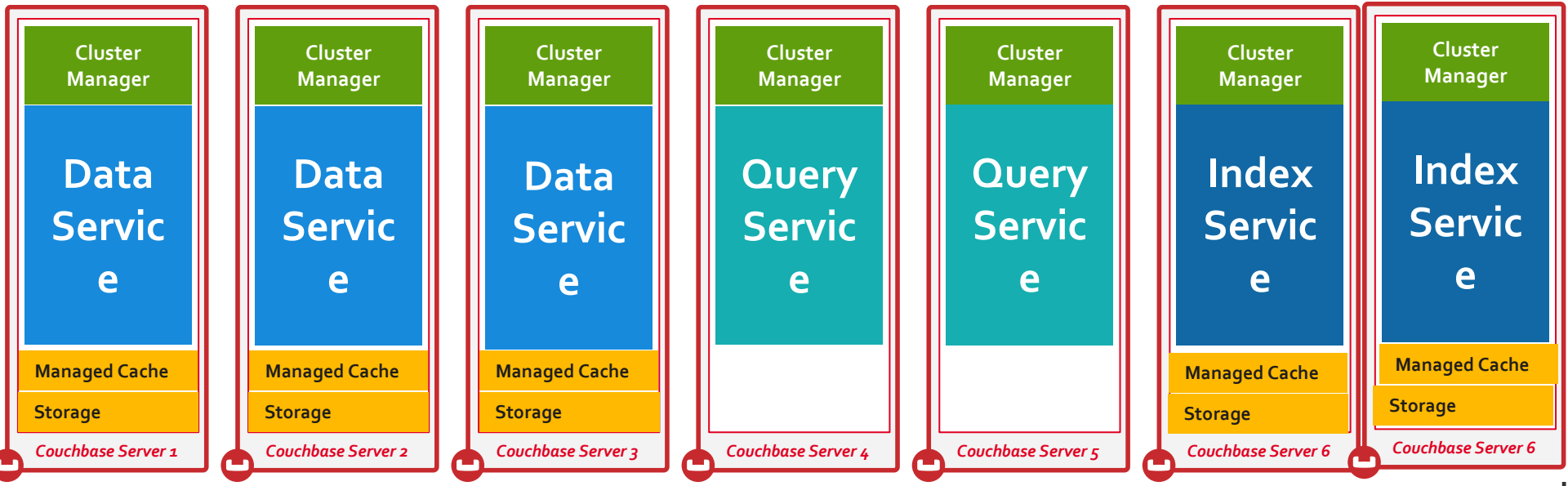
```
"customer": {
"ccInfo": {
    "cardExpiry": "2015-11-11",
    "cardNumber":"1212-232-1234",
    "cardType": "americanexpress"
    },
  "customerId": "customer534",
  "dateAdded": "2014-04-06",
  "dateLastActive":"2014-05-02",
 "emailAddress":"iles@kertz.name",
  "firstName": "Mckayla",
  "lastName": "Brown",
  "phoneNumber": "1-533-290-6403",
  "postalCode": "92341",
  "state": "VT",
  "type": "customer"
            }
```

- Secondary Index can be created on any combination of attribute names.
  - `CREATE INDEX idx_cust_cardnum customer(ccInfo.cardNumber, postalcode)`
- Useful in speeding up the queries.
- Need to have matching indices with right key-ordering
  - `(ccInfo.cardExpiry, postalCode)`
  - `(type, state, lastName, firstName)`

N1QL: Architecture

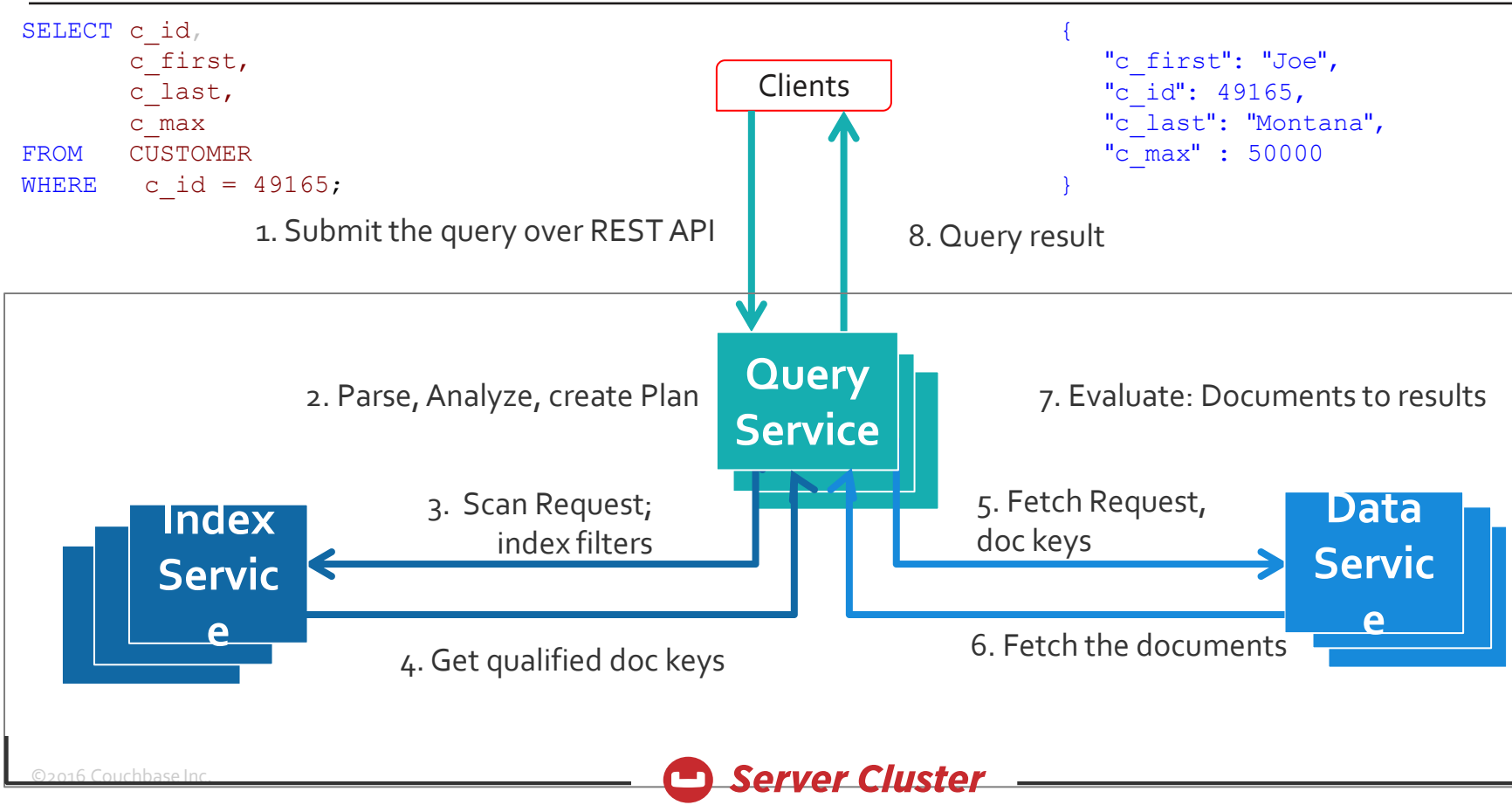
Couchbase Server Cluster Service Deployment



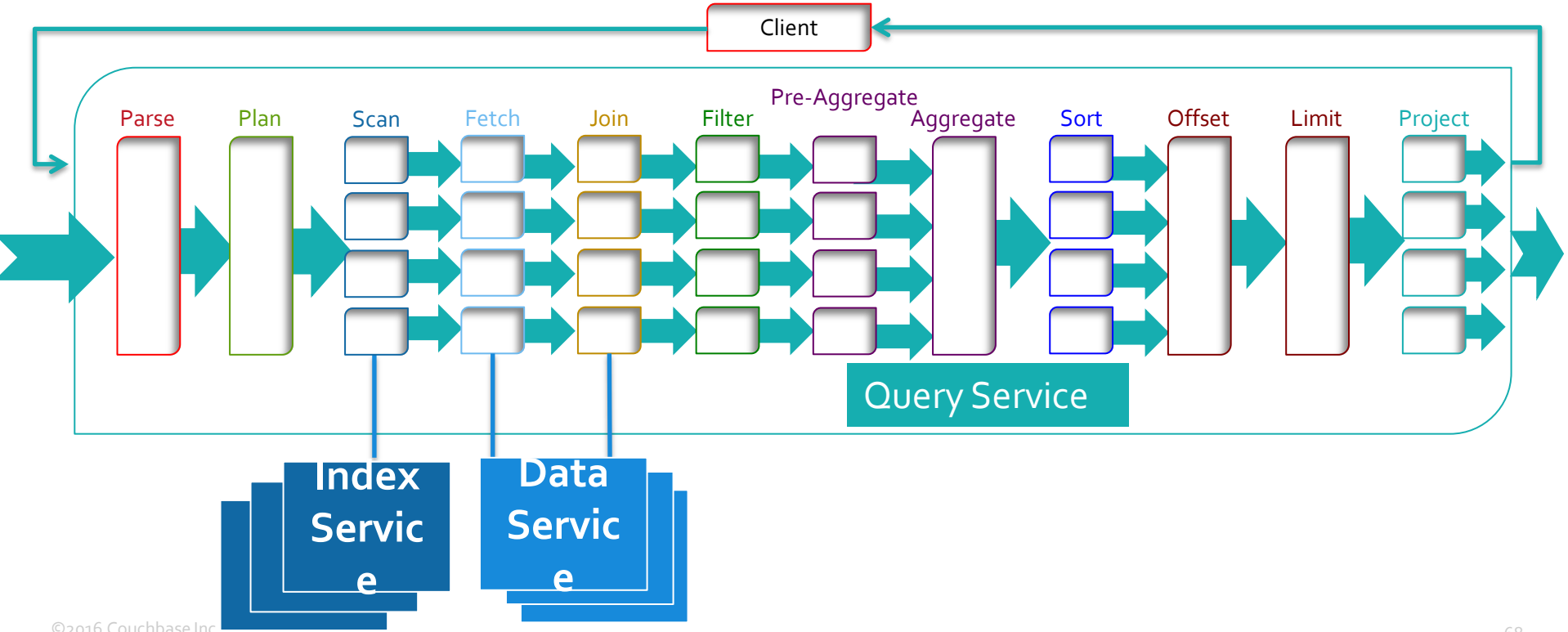


Server Cluster

N1QL: Query Execution Flow



N1QL: Inside the Query Service



N1QL: Ecosystem

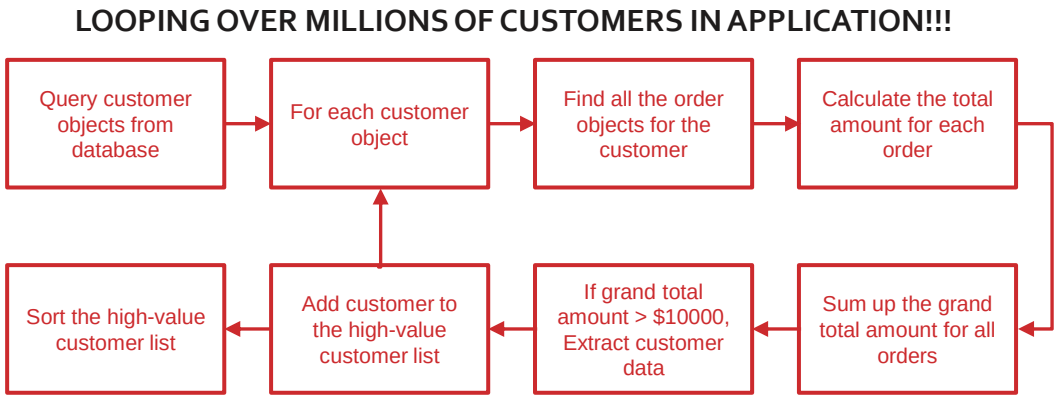
Enterprise Ecosystem via JDBC/ODBC

JDBC ODBC SDK

Bringing SQL to NoSQL: Summary

Find High-Value Customers with Orders > $10000

Without SQL



- Complex codes and logic
- Inefficient processing on client side

With SQL

```
SELECT     Customers.ID, Customers.Name, SUM(OrderLine.Amount)
FROM       Orders UNNEST Orders.LineItems AS OrderLine
           JOIN Customers ON KEYS Orders.CustID
GROUP BY   Customers.ID, Customers.Name
HAVING     SUM(OrderLine.Amount) > 10000
ORDER BY   SUM(OrderLine.Amount) DESC
```

- Proven and expressive query language
- Leverage SQL skills and ecosystem
- Extended for JSON

Bringing SQL to NoSQL

Query Features	SQL on RDBMS	SQL on NoSQL
Statements	▪ SELECT, INSERT, UPDATE, DELETE, MERGE	▪ SELECT, INSERT, UPDATE, DELETE, MERGE
Query Operations	▪ Select, Join, Project, Subqueries ▪ Strict Schema ▪ Strict Type checking	▪ Select, Join, Project, Subqueries ✓ Nest & Unnest ✓ Look Ma! No Type Mismatch Errors! ▪ JSON keys act as columns
Schema	▪ Predetermined Columns	✓ Fully addressable JSON ✓ Flexible document structure
Data Types	▪ SQL Data types ▪ Conversion Functions	▪ JSON Data types ▪ Conversion Functions
Query Processing	▪ INPUT: Sets of Tuples ▪ OUPUT: Set of Tuples	▪ INPUT: Sets of JSON ▪ OUTPUT: Set of JSON

Try SQL for JSON

- **N1QL is available Couchbase 4.x and above**
  - http://query.couchbase.com
  - **Online Interactive Tutorial**
- Ask us questions
  - Couchbase **forums, Stack Overflow, @N1QL**

Thank you
Q & A

Gerald Sangudi | @sangudi | Chief Architect | Couchbase

Keshav Murthy| @rkeshavmurthy | Director | Couchbase

@N1QL | query.couchbase.com