N1QL

Business Application Questions & Tasks

Get the list of stores in customer region

Generate a list of shipment due today

Update the sale prices in outlet stores only

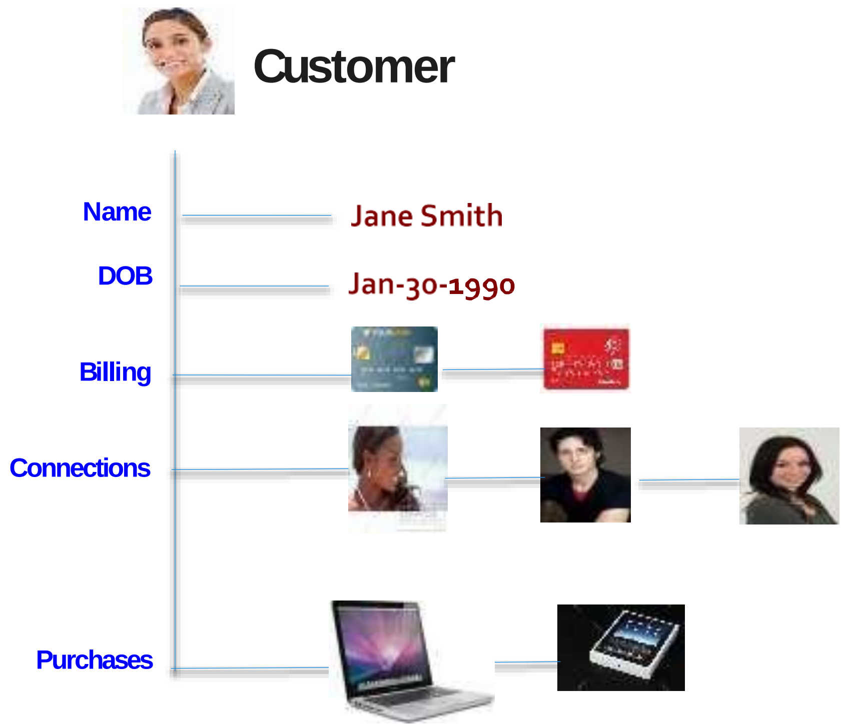
Search stores for the shoe customer is looking for?

How many new customers we got last month?

Load the new inventory data

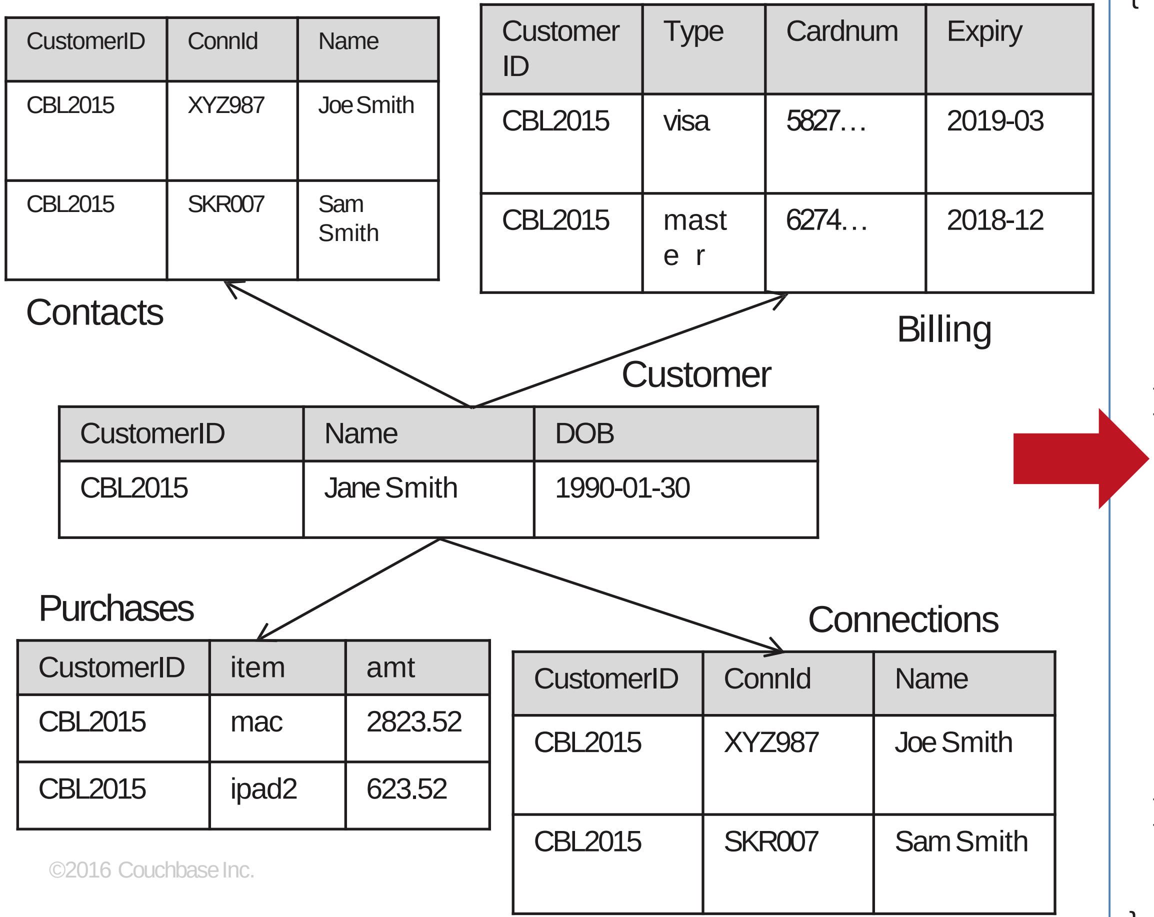
Merge the customer lists

Properties of Real-World Data



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

Transform: Relational to JSON



Contacts

CustomerID	ConnId	Name
CBL2015	XYZ987	Joe Smith
CBL2015	SKR007	Sam Smith

Billing

Customer ID	Type	Cardnum	Expiry
CBL2015	visa	5827…	2019-03
CBL2015	mast e r	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

DocumentKey: **CBL2015**

```
{
  "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 }
} ]
```

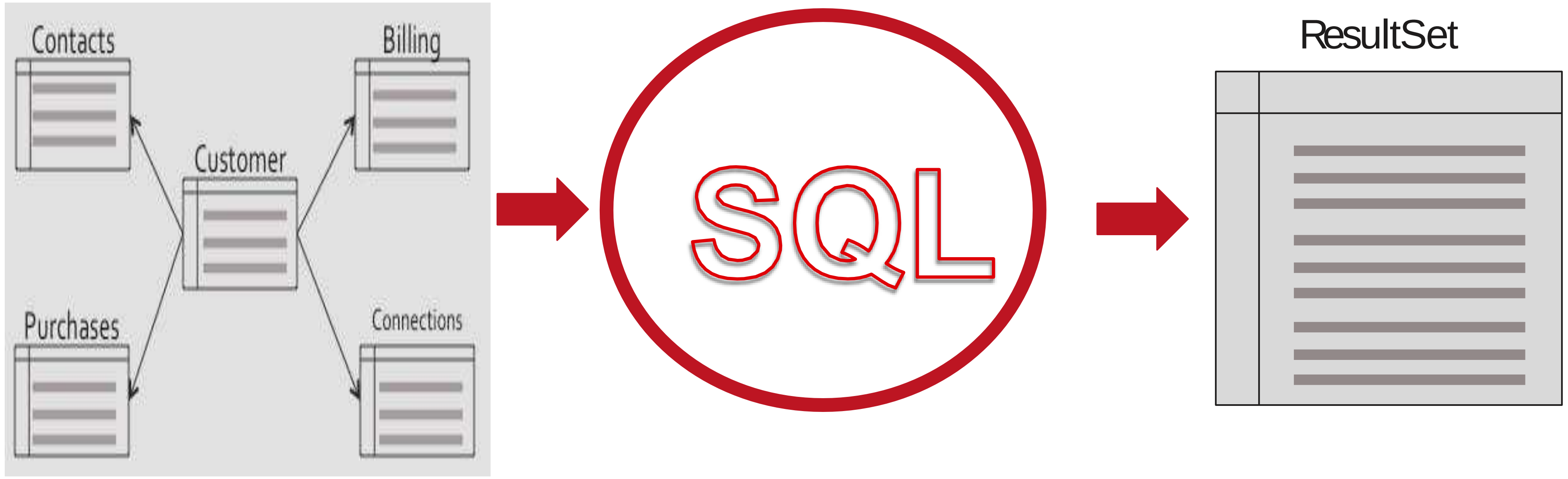
JSON

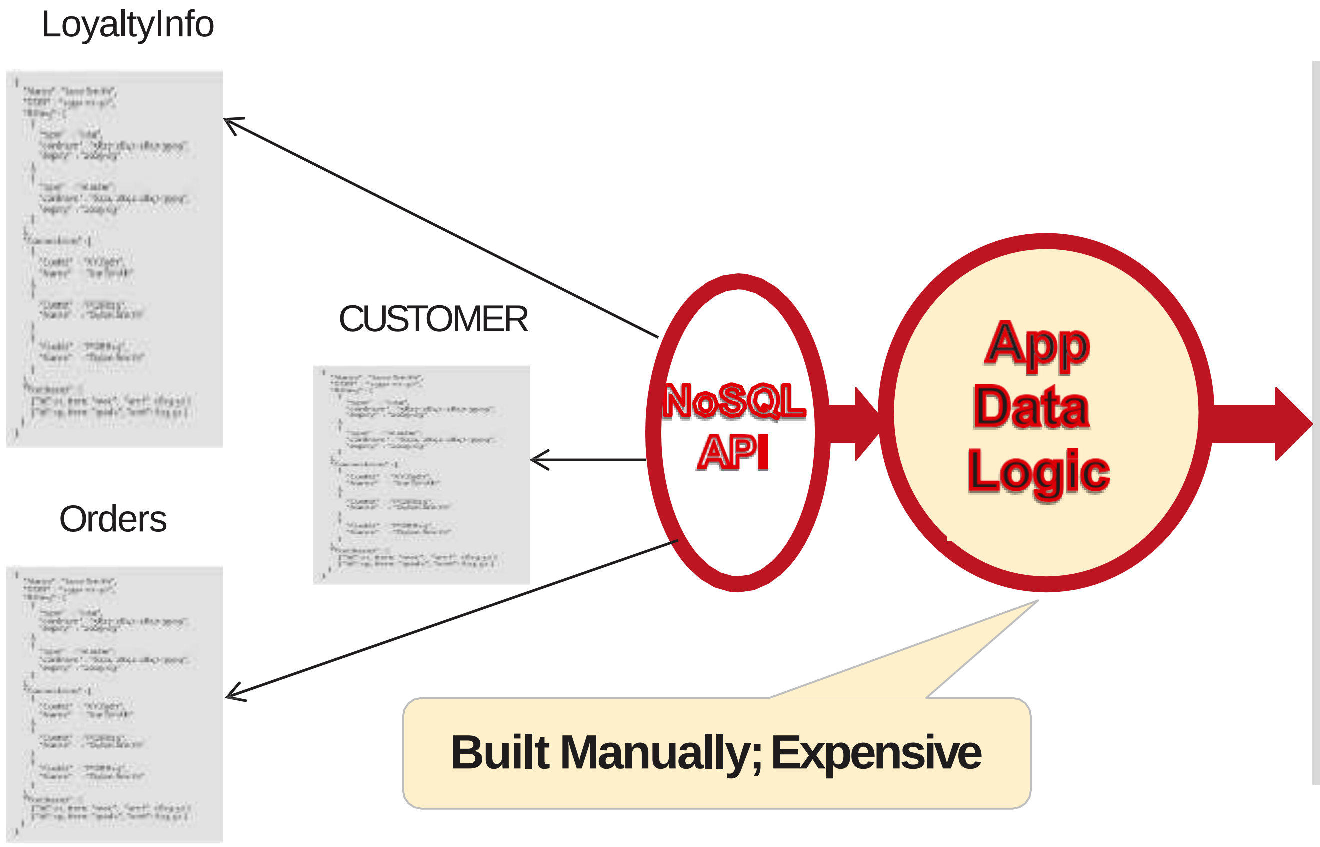
- JSON is a means to the end and not the end itself
  - JSON is the representation of the enterprise data model for applications
  - JSON flexibility translates to application flexibility
    - Simple flattened data can be represented
    - Entities with complex data, always accessed analyzed together should be living together
  - Applications are designed to handle the flexible data model.

```
{ "type" : "visa",
  "cardnum" : "5827-2842-2847-3909",
},
{
  "type" : "master",
  "cardnum" : "6274-2842-2847-3909"
}
],
"Connections" : [
}
```

Models for Representing Data

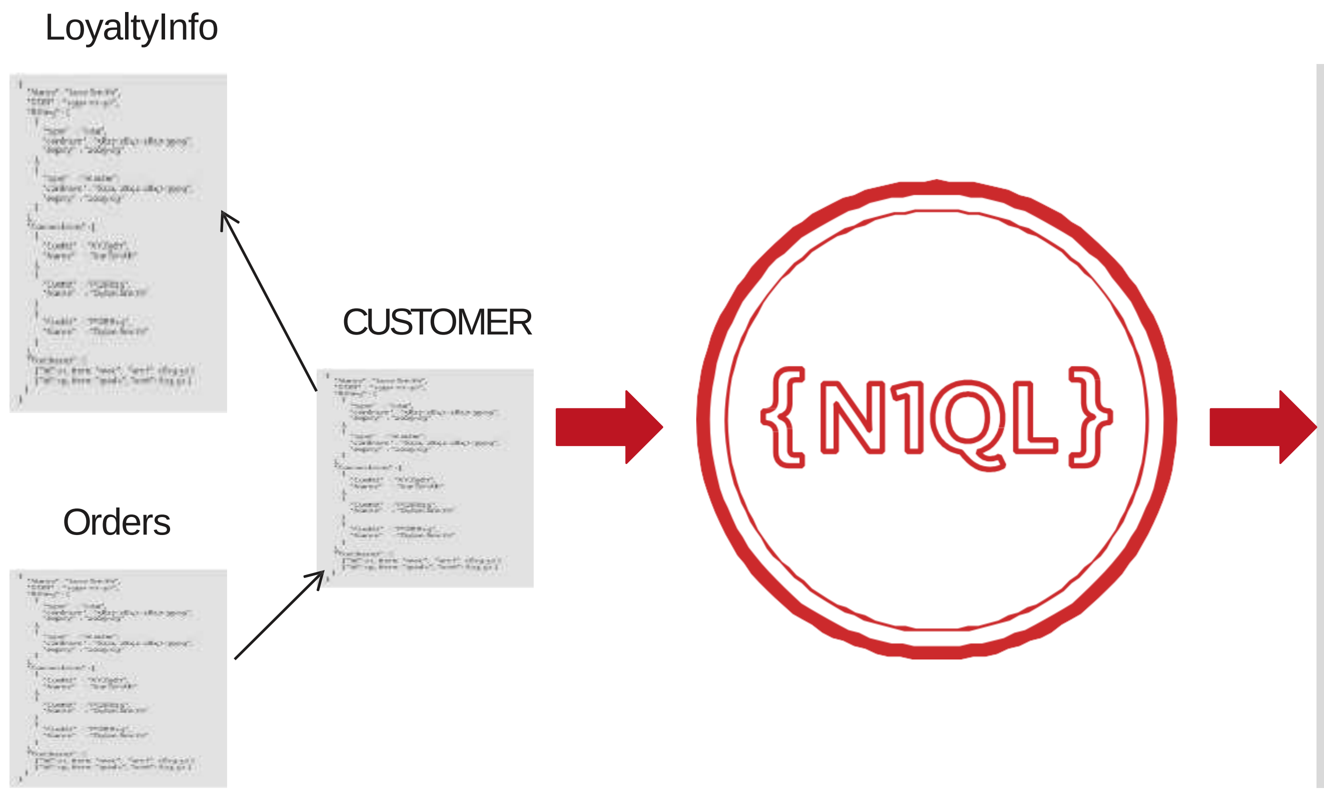
Data Concern	Relational Model	JSON Document Model (NoSQL)
Rich Structure	▪ Multiple flat tables ▪ Constant assembly / disassembly	▪ Documents ✓ No assembly required!
Relationships	▪ Represented ✓ Queried (SQL)	▪ Represented ▪ Queried? Not until now…
Value Evolution	▪ Data can be updated	▪ Data can be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


Contacts
Billing
Customer
Purchases
Connections
SQL
ResultSet



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 }
   ]
}
```



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 }
  ]
}
```

Goal of N1QL: SQL for JSON

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

N1QL: Developers & Enterprises

- Application Developers in all languages
  - Couchbase SDK Support for N1QL
  - Open REST API
- Exchanges data with other databases using Standard Tools
- Simba provides ODBC, JDBC drivers

N1QL: expressive

- Access to every part of JSON document
- Scalar & Aggregate functions
- Issue subquery in any expressions
- Subqueries
- Subqueries in the FROM clause

N1QL: powerful

- Access to every part of JSON document
- JOINS, Aggregations, standard scalar functions
- Aggregation on arrays
- NEST & UNNEST operations
- Covering Index

N1QL: querying

- INSERT
- UPDATE
- DELETE
- MERGE
- SELECT
- EXPLAIN

INSERT

Use the INSERT statement to insert one or more new documents into an existing keyspace.

```
INSERT INTO `travel-sample` ( KEY, VALUE )
  VALUES
  (
    "k001",
    { "id": "01", "type": "airline"}
  )
RETURNING META().id as docid, *;
```

Results

```
{
  "requestID": "06c5acc1-69d3-4aad-9c11-b90a9bc895d8",
  "signature": {
    "*": "*",
    "id": "json"
  },
  "results": [
    {
      "docid": "k001",
      "travel-sample": {
        "id": "01",
        "type": "airline"
      }
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "5.033416ms",
    "executionTime": "5.011203ms",
    "resultCount": 1,
    "resultSize": 151,
    "mutationCount": 1
  }
}
```

UPDATE

UPDATE replaces a document that already exists with updated values.

```
UPDATE `travel-sample`
SET city = "San Francisco"
WHERE lower(city) = "sanfrancisco"
RETURNING *
```

```
UPDATE product USE KEYS "odwalla-juice1" SET type = "product-juice" RETURNING product.type

"results": [
        {
            "type": "product-juice"
        }
    ]
```

UPSERT

Used to insert a new record or update an existing one. If the document doesn't exist it will be created. UPSERT is a combination of INSERT and UPDATE.

```
UPSERT INTO product (KEY, VALUE) VALUES ("odwalla-juice1", { "productId": "odwalla-juice1",
      "unitPrice": 5.40, "type": "product", "color":"red"}) RETURNING * ;

"results": [
        {
            "color": "red",
            "productId": "odwalla-juice1",
            "type": "product",
            "unitPrice": 5.4
        }
    ]
```

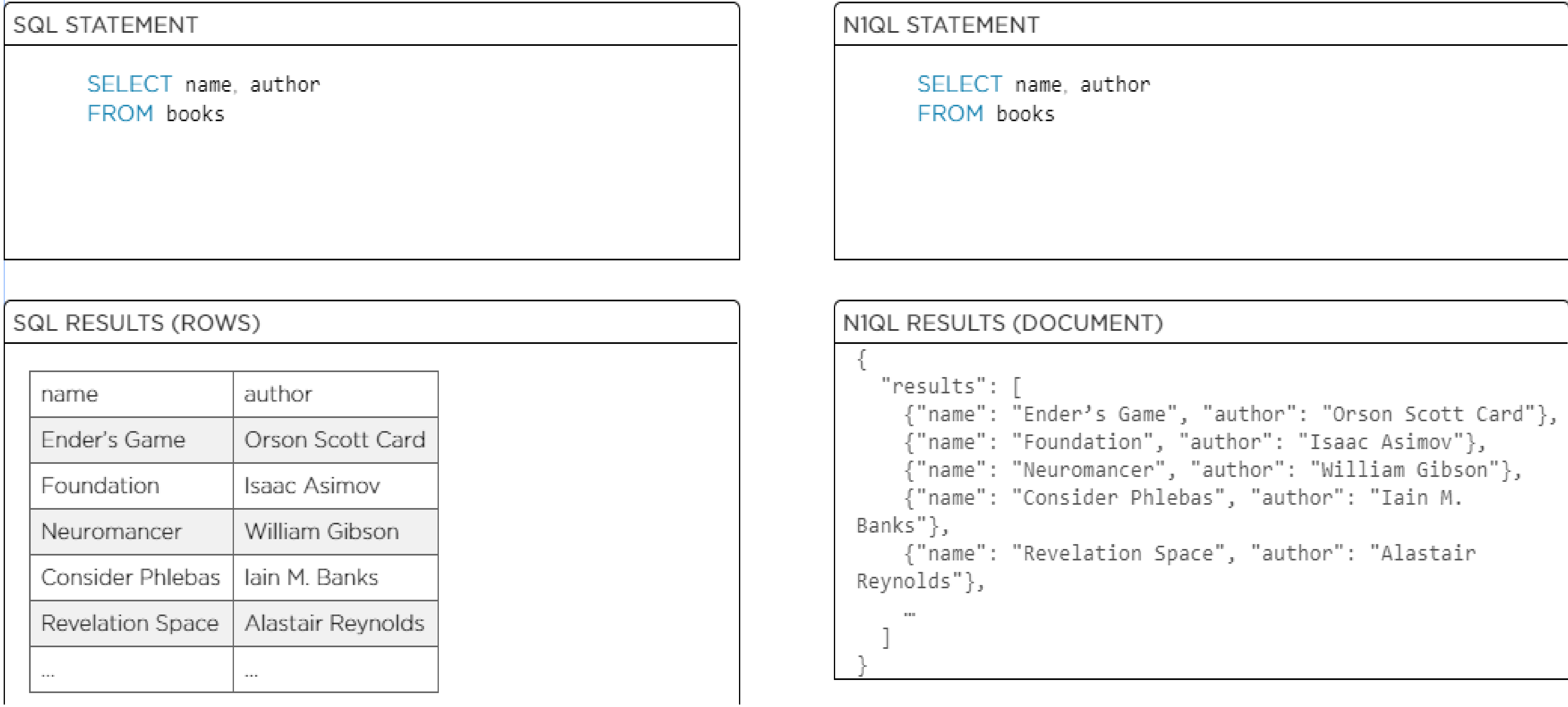
DELETE

DELETE immediately removes the specified document from your keyspace

```
DELETE FROM product p
WHERE p.unitPrice = 5.25
RETURNING p.productId

"results": [
        {
            "productId": "product99"
        }
    ]
```

Select



SQL STATEMENT

```sql
SELECT name, author
FROM books
```

N1QL STATEMENT

```sql
SELECT name, author
FROM books
```

SQL RESULTS (ROWS)

name	author
Ender's Game	Orson Scott Card
Foundation	Isaac Asimov
Neuromancer	William Gibson
Consider Phlebas	Iain M. Banks
Revelation Space	Alastair Reynolds
...	...

N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"name": "Ender's Game", "author": "Orson Scott Card"},
    {"name": "Foundation", "author": "Isaac Asimov"},
    {"name": "Neuromancer", "author": "William Gibson"},
    {"name": "Consider Phlebas", "author": "Iain M. 
Banks"},
    {"name": "Revelation Space", "author": "Alastair 
Reynolds"},
    …
  ]
}
```

WHERE

SQL STATEMENT

```
SELECT name, author
FROM books
WHERE YEAR(published) >= 2014
```

N1QL STATEMENT

```
SELECT name, author
FROM books
WHERE DATE_PART_STR(published, "year") >= 2014
```

SQL RESULTS (ROWS)

name	author
Slow Bullets	Alastair Reynolds
Dark Lightning	John Varley
Coming Home	Jack McDevitt
The Peripheral	William Gibson
Armada	Ernest Cline
...	...

N1QL RESULTS (DOCUMENTS)

```
{
  "results": [
    {"name": "Slow Bullets", "author": "Alastair
Reynolds"},
    {"name": "Dark Lightning", "author": "John Varley"},
    {"name": "Coming Home", "author": "Jack McDevitt"},
    {"name": "The Peripheral", "author": "William
Gibson"},
    {"name": "Armada", "author": "Ernest Cline"},
    …
  ]
}
```

Order By

SQL STATEMENT

```
SELECT name, YEAR(published) AS published
FROM  books
WHERE author = "Alastair Reynolds"
ORDER BY published
```

N1QL STATEMENT

```
SELECT name, DATE_PART_STR(published, "year") as
published
FROM  books
WHERE author = "Alastair Reynolds"
ORDER BY published
```

SQL RESULTS (ROWS)

name	date
Revelation Space	2000
Chasm City	2001
Redemption Ark	2002
Absolution Gap	2003
Century Rain	2004
...	...

N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"name": "Revelation Space", "published": "2000"},
    {"name": "Chasm City", "published": "2001"},
    {"name": "Redemption Ark", "published": "2002"},
    {"name": "Absolution Gap", "published": "2003"},
    {"name": "Century Rain", "published": "2004"},
    …
  ]
}
```

Distinct

SQL STATEMENT

```
SELECT DISTINCT(series), author
FROM books
WHERE series IS NOT NULL
ORDER BY series
```

N1QL STATEMENT

```
SELECT DISTINCT(series), author
FROM books
WHERE series IS NOT MISSING
ORDER BY series
```

SQL RESULTS (ROWS)

series	author
Commonwealth	Peter F. Hamilton
Culture	Iain M. Banks
Ender's Game	Orson Scott Card
Foundation	Isaac Asimov
Revelation Space	Alastair Reynolds
...	...

N1QL RESULTS (DOCUMENT)

```
{
   "results": [
      {"series": "Commonwealth", "author": "Peter F.
Hamilton"},
      {"series": "Culture", "author": "Iain M. Banks"},
      {"series": "Ender's Game", "author": "Orson Scott
Card"},
      {"series": "Foundation", "author": "Isaac Asimov"},
      {"series": "Revelation Space", "author": "Alastair
Reynolds"},
      …
   ]
}
```

Order By

SQL STATEMENT

```
SELECT name, YEAR(published) AS year
FROM books
WHERE series = “Foundation”
ORDER BY year
```

N1QL STATEMENT

```
SELECT name, DATE_PART_STR(published, "year") AS year
FROM books
WHERE series = "Foundation"
ORDER BY year
```

SQL RESULTS (ROWS)

name	year
Foundation	1951
Foundation and Empire	1952
Second Foundation	1953
Foundation's Edge	1982
Foundation and Earth	1986
...	...

N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"name": "Foundation", "year": "1951"},
    {"name": "Foundation and Empire", "year": "1952"},
    {"name": "Second Foundation", "year": "1953"},
    {"name": "Foundation’s Edge", "year": "1982"},
    {"name": "Foundation and Earth", "year": "1986"},
    …
  ]
}
```

Group By

SQL STATEMENT

```
SELECT book, AVG(rating) AS average
FROM reviews
GROUP BY book
HAVING COUNT(*) > 100000
ORDER BY average DESC
```

N1QL STATEMENT

```
SELECT book, AVG(rating) AS average
FROM reviews
GROUP BY book
HAVING COUNT(*) > 100000
ORDER BY average DESC
```

SQL RESULTS (ROWS)

book	average
Ready Player One	4.31
Ender's Game	4.28
Foundation	4.07
Speaker for the Dead	4.01
Neuromancer	3.85
...	...

N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"book": "Ready Player One", "average": "4.31"},
    {"book": "Ender's Game", "average": "4.28"},
    {"book": "Foundation", "average": "4.07"},
    {"book": "Speaker for the Dead", "average": "4.01"},
    {"book": "Neuromancer", "average": "3.85"},
    …
  ]
}
```

Join

SQL STATEMENT

```
SELECT b.name, YEAR(a.year) AS year, a.name AS award
FROM awards a INNER JOIN books b
ON a.book_id = b. id
WHERE a.year > 1969
ORDER BY name, year, award
```

N1QL STATEMENT

```
SELECT b.name, DATE_PART_STR(a.year, "year") as year,
a.name as award
FROM awards a INNER JOIN books b
ON KEYS a.book_id
ORDER BY b.name, year, award
```

SQL RESULTS (ROWS)

name	year	award
Gateway	1978	Hugo
Gateway	1978	Nebula
Neuromancer	1984	Philip
Neuromancer	1985	Hugo
Neuromancer	1985	Nebula
...	...	...

N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"name": "Gateway", "year": "1978", "award": "Hugo"},
    {"name": "Gateway", "year": "1978", "award":
"Nebula"},
    {"name": "Neuromancer", "year": "1984", "award":
"Philip"},
    {"name": "Neuromancer", "year": "1985", "award":
"Hugo"},
    {"name": "Neuromancer", "year": "1985", "award":
"Nebula"},
    …
  ]
}
```

Subquery

SQL STATEMENT

```
SELECT name, author
FROM books
WHERE author_id IN (
  SELECT id
  FROM authors
  WHERE country = “UK”)
```

SQL RESULTS (ROWS)

name	author
Terminal World	Alastair Reynolds
2001: A Space Odyssey	Arthur C. Clarke
The Algebraist	Iain M. Banks
Glasshouse	Charles Stross
Great North Road	Peter F. Hamilton
...	...

N1QL STATEMENT

```
SELECT b.name, b.author
FROM books b
WHERE EXISTS (
  SELECT id
  FROM authors
  USE KEYS b.author_id
  WHERE country = "UK")
```

N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"name": "Terminal World", "author": "Alastair
Reynolds"},
    {"name": "2001: A Space Odyssey", "author": "Arthur C.
Clarke"},
    {"name": "The Algebraist", "author": "Iain M. Banks"},
    {"name": "Glasshouse", "author": "Charles Stross"},
    {"name": "Great North Road", "author", "Peter F.
Hamilton"},
    ...
  ]
}
```

Union

SQL STATEMENT

```
SELECT name, "Book" as type
FROM books
WHERE favorite = "TRUE"
UNION ALL (
  SELECT name, "Movie" as type
  FROM movies
  WHERE favorite = "TRUE")
ORDER BY name
```

N1QL STATEMENT

```
SELECT name, "Book" as type
FROM books
WHERE favorite = TRUE
UNION ALL (
  SELECT name, "Movie" as type
  FROM movies
  WHERE favorite = TRUE)
ORDER BY name
```

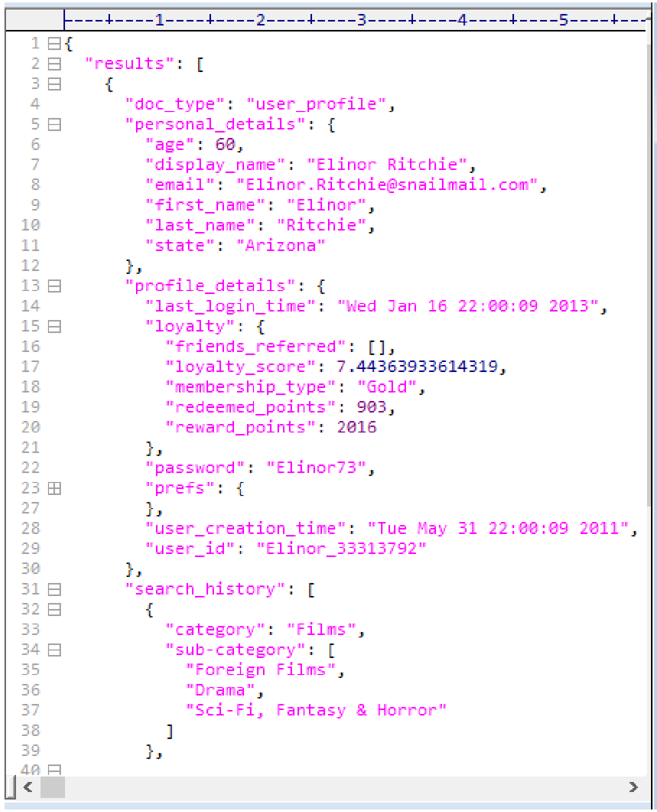
SQL RESULTS (ROWS)

name	type
Aliens	Movie
Blade Runner	Movie
Chasm City	Book
Chasm City	Movie
Ender's Game	Book
...	...

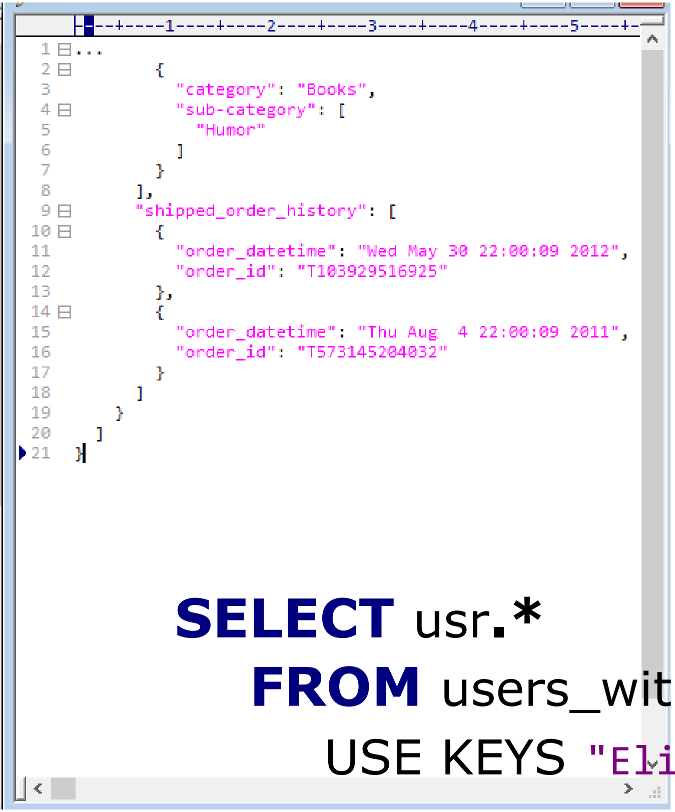
N1QL RESULTS (DOCUMENT)

```
{
  "results": [
    {"name": "Aliens",  "type": "Movie"},
    {"name": "Blade Runner", "Type": "Movie"},
    {"name": "Chasm City", "average": "Book"},
    {"name": "Dark City", "average": "Movie"},
    {"name": "Ender's Game", "average": "Book"},
    …
  ]
}
```

NEST



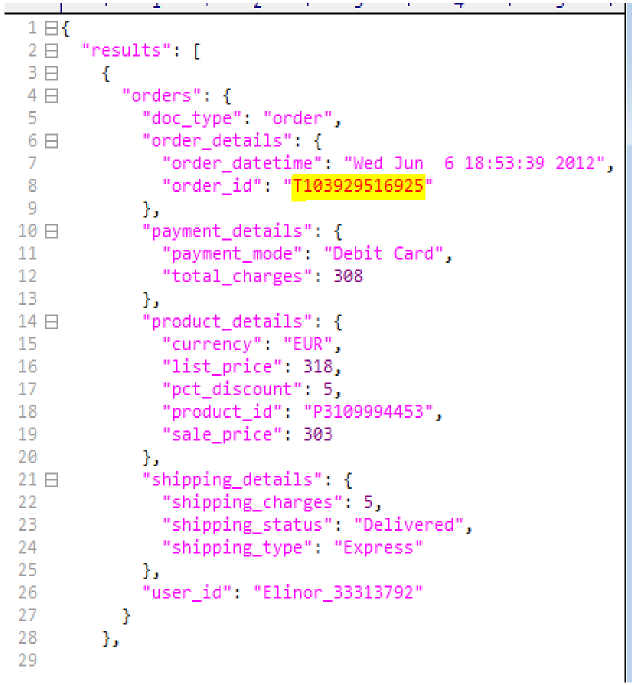
```
{
  "results": [
    {
      "doc_type": "user_profile",
      "personal_details": {
        "age": 60,
        "display_name": "Elinor Ritchie",
        "email": "Elinor.Ritchie@snailmail.com",
        "first_name": "Elinor",
        "last_name": "Ritchie",
        "state": "Arizona"
      },
      "profile_details": {
        "last_login_time": "Wed Jan 16 22:00:09 2013",
        "loyalty": {
          "friends_referred": [],
          "loyalty_score": 7.44363933614319,
          "membership_type": "Gold",
          "redeemed_points": 903,
          "reward_points": 2016
        },
        "password": "Elinor73",
        "prefs": {
        },
        "user_creation_time": "Tue May 31 22:00:09 2011",
        "user_id": "Elinor_33313792"
      },
      "search_history": [
        {
          "category": "Films",
          "sub-category": [
            "Foreign Films",
            "Drama",
            "Sci-Fi, Fantasy & Horror"
          ]
        },
```



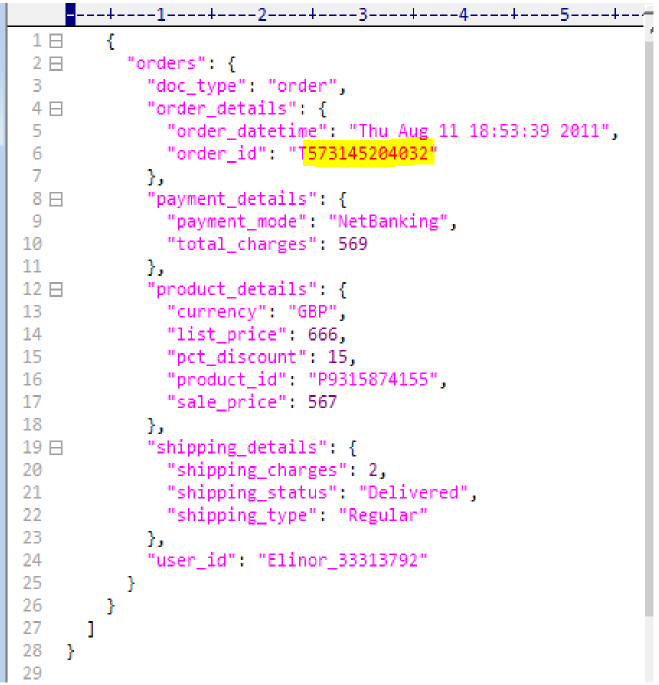
```
...
        {
          "category": "Books",
          "sub-category": [
            "Humor"
          ]
        }
      ],
      "shipped_order_history": [
        {
          "order_datetime": "Wed May 30 22:00:09 2012",
          "order_id": "T103929516925"
        },
        {
          "order_datetime": "Thu Aug  4 22:00:09 2011",
          "order_id": "T573145204032"
        }
      ]
    }
  ]
}
```

```
SELECT usr.*
   FROM users_with_orders usr
      USE KEYS "Elinor_33313792"
```

NEST



```
{
  "results": [
    {
      "orders": {
        "doc_type": "order",
        "order_details": {
          "order_datetime": "Wed Jun  6 18:53:39 2012",
          "order_id": "T103929516925"
        },
        "payment_details": {
          "payment_mode": "Debit Card",
          "total_charges": 308
        },
        "product_details": {
          "currency": "EUR",
          "list_price": 318,
          "pct_discount": 5,
          "product_id": "P3109994453",
          "sale_price": 303
        },
        "shipping_details": {
          "shipping_charges": 5,
          "shipping_status": "Delivered",
          "shipping_type": "Express"
        },
        "user_id": "Elinor_33313792"
      }
    },
```



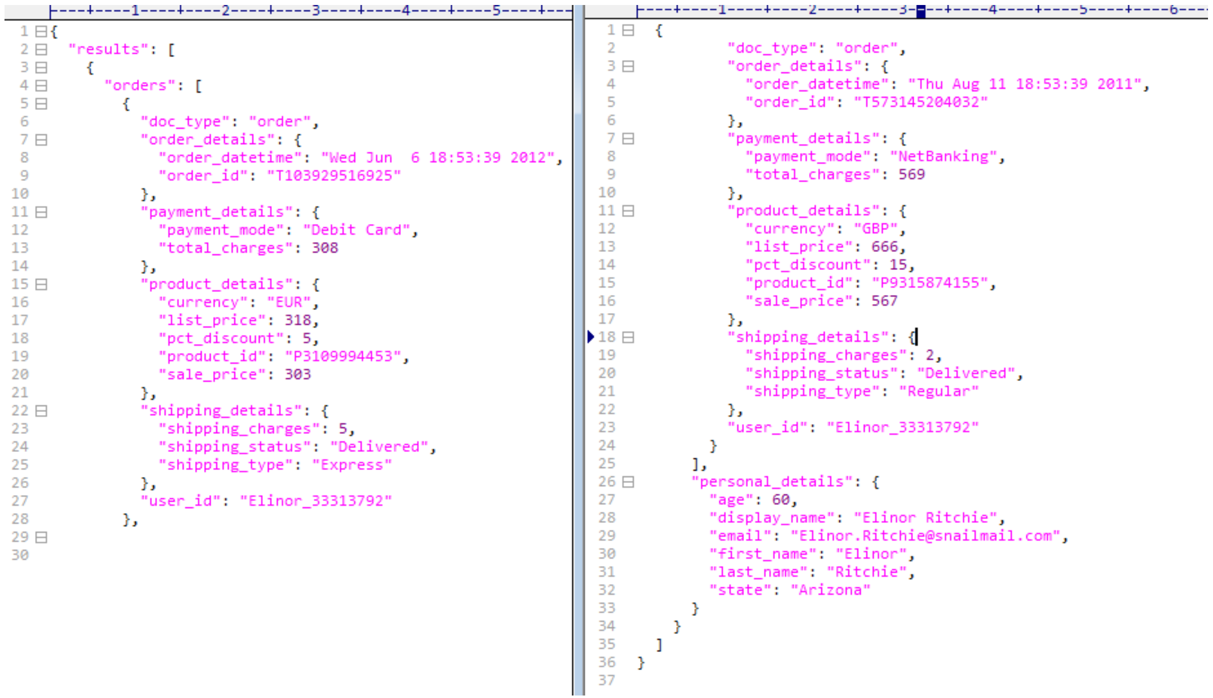
```
    {
      "orders": {
        "doc_type": "order",
        "order_details": {
          "order_datetime": "Thu Aug 11 18:53:39 2011",
          "order_id": "T573145204032"
        },
        "payment_details": {
          "payment_mode": "NetBanking",
          "total_charges": 569
        },
        "product_details": {
          "currency": "GBP",
          "list_price": 666,
          "pct_discount": 15,
          "product_id": "P9315874155",
          "sale_price": 567
        },
        "shipping_details": {
          "shipping_charges": 2,
          "shipping_status": "Delivered",
          "shipping_type": "Regular"
        },
        "user_id": "Elinor_33313792"
      }
    }
  ]
}
```

Select * FROM orders_with_users orders

Use Keys ["T103929516925","T573145204032"]

NEST

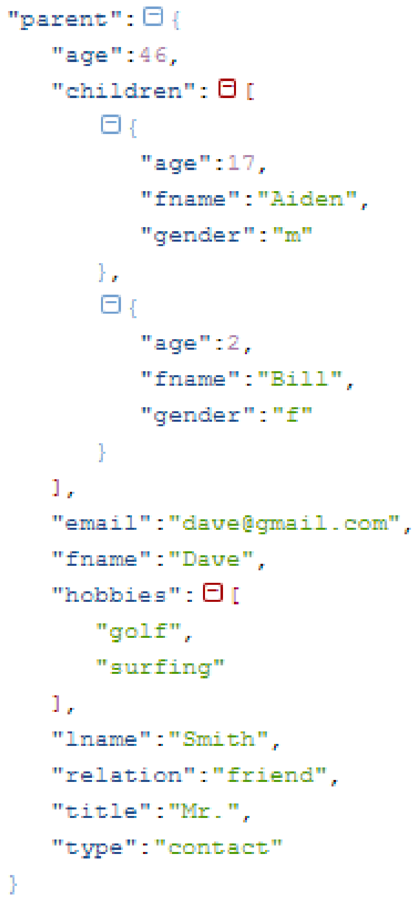
```
SELECT usr.personal_details, orders
    FROM users_with_orders usr
        USE KEYS "Elinor_33313792"
            NEST orders_with_users orders
               ON KEYS ARRAY s.order_id FOR s IN usr.shipped_order_history END
```



```
{
  "results": [
    {
      "orders": [
        {
          "doc_type": "order",
          "order_details": {
            "order_datetime": "Wed Jun  6 18:53:39 2012",
            "order_id": "T103929516925"
          },
          "payment_details": {
            "payment_mode": "Debit Card",
            "total_charges": 308
          },
          "product_details": {
            "currency": "EUR",
            "list_price": 318,
            "pct_discount": 5,
            "product_id": "P3109994453",
            "sale_price": 303
          },
          "shipping_details": {
            "shipping_charges": 5,
            "shipping_status": "Delivered",
            "shipping_type": "Express"
          },
          "user_id": "Elinor_33313792"
        },
```

```
        {
          "doc_type": "order",
          "order_details": {
            "order_datetime": "Thu Aug 11 18:53:39 2011",
            "order_id": "T573145204032"
          },
          "payment_details": {
            "payment_mode": "NetBanking",
            "total_charges": 569
          },
          "product_details": {
            "currency": "GBP",
            "list_price": 666,
            "pct_discount": 15,
            "product_id": "P9315874155",
            "sale_price": 567
          },
          "shipping_details": {
            "shipping_charges": 2,
            "shipping_status": "Delivered",
            "shipping_type": "Regular"
          },
          "user_id": "Elinor_33313792"
        }
      ],
      "personal_details": {
        "age": 60,
        "display_name": "Elinor Ritchie",
        "email": "Elinor.Ritchie@snailmail.com",
        "first_name": "Elinor",
        "last_name": "Ritchie",
        "state": "Arizona"
      }
    }
  ]
}
```

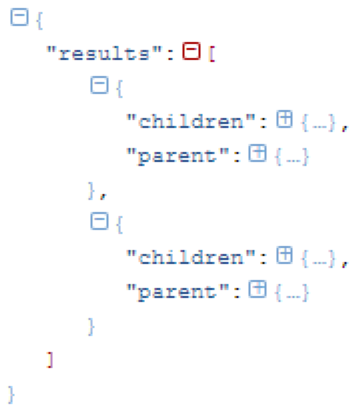
UNNEST



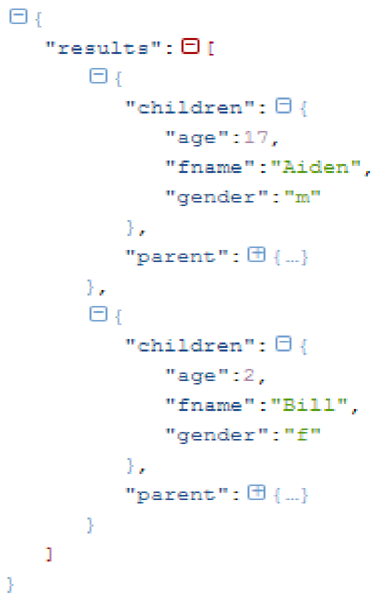
```
"parent":{
    "age":46,
    "children":[
        {
            "age":17,
            "fname":"Aiden",
            "gender":"m"
        },
        {
            "age":2,
            "fname":"Bill",
            "gender":"f"
        }
    ],
    "email":"dave@gmail.com",
    "fname":"Dave",
    "hobbies":[
        "golf",
        "surfing"
    ],
    "lname":"Smith",
    "relation":"friend",
    "title":"Mr.",
    "type":"contact"
}
```

A single document with two children; Aiden and Bill as a first name

UNNEST



```
{
  "results": [
    {
      "children": {…},
      "parent": {…}
    },
    {
      "children": {…},
      "parent": {…}
    }
  ]
}
```

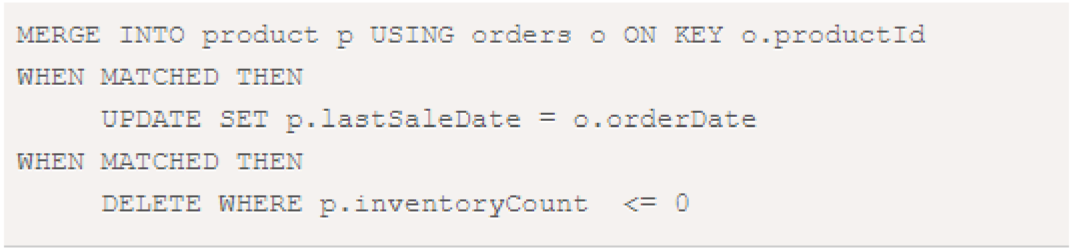


```
{
  "results": [
    {
      "children": {
        "age":17,
        "fname":"Aiden",
        "gender":"m"
      },
      "parent": {…}
    },
    {
      "children": {
        "age":2,
        "fname":"Bill",
        "gender":"f"
      },
      "parent": {…}
    }
  ]
}
```

```
SELECT *
  FROM tutorial AS parent
    UNNEST parent.children
      WHERE parent.fname = 'Dave'
```

MERGE

A MERGE statement provides the ability to update, insert into, or delete from a keyspace based on the results of a join with another keyspace or subquery



```
MERGE INTO product p USING orders o ON KEY o.productId
WHEN MATCHED THEN
     UPDATE SET p.lastSaleDate = o.orderDate
WHEN MATCHED THEN
     DELETE WHERE p.inventoryCount  <= 0
```

updates product based on orders

Lab :

- Create bucket, load data, create primary index, and install tools

  Selecting documents and limiting results in *Query Workbench* and the *cbq* command line tool

  Selecting nested attributes, aliasing, concatenating, and accessing documents by key

  Manipulating data using N1QL DML