Couchbase Analytics Service

NOTES:
1. Many thanks to Couchbase for providing this material and granting us permission to use it!
2. These slides have been lightly modified for CS122D use.
Traditional Analytics Solutions

Business Application

Operations Data

OpS DB
OpS DB
OpS DB

ETL

Batch

Analytics Tool

Analytical DB
Couchbase Analytics – Bringing NoETL to NoSQL

Business Application

Operations Data

Real-time Analytics

Analytics Tool

Couchbase Data Platform

Ops Data Node

Analytics Node
What is Couchbase Analytics?

- Fast Ingest
- Complex Queries on Large Datasets
- Real-time Insights for Business Teams
# Couchbase Analytics Benefits

<table>
<thead>
<tr>
<th>Benefit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast Ingest</td>
<td>Operational data is available for analytical processing in near real-time by creating a <em>shadow dataset</em> via the Couchbase Database Change Protocol (DCP).</td>
</tr>
<tr>
<td>Real-time insights</td>
<td><em>MPP</em> query engine can run ad-hoc queries to perform complex joins, groupings, aggregations and count. <em>No impact on operational workflows due to workload isolation.</em></td>
</tr>
<tr>
<td>NoETL for NoSQL</td>
<td>No separate infrastructure or programming required to manage analytical workloads.</td>
</tr>
<tr>
<td>Reduced Complexity</td>
<td>Simplified operations and manageability with a single platform for running operational and analytical workloads.</td>
</tr>
<tr>
<td>Developer Productivity</td>
<td>Leverage existing skills - <em>SQL-like queries</em> to analyze data.</td>
</tr>
<tr>
<td>Secure Access</td>
<td>Access Control to analytical datasets can be managed independently.</td>
</tr>
<tr>
<td>Enterprise Ready</td>
<td>HA, scale, zero downtime upgrade are key tenets of the Couchbase platform.</td>
</tr>
</tbody>
</table>

- Yielding a NoSQL version of *HTAP* (Hybrid Transactional / Analytical Processing) or *HOAP* (Hybrid Operational / Analytical Processing)
Query and Analytics Uses

QUERY SERVICE

• Online search and booking, reviews and ratings
  • Property and room detail pages
  • Cross-sell links, up-sell links
  • Stars & likes & associated reviews
  • Their booking history

N1QL queries behind every page display and click/navigation

ANALYTICS SERVICE

• Reporting, Trend Analysis, Data Exploration
  • Daily discount availability report
  • Cities with highest room occupancy rates
  • Hotels with biggest single day drops
  • How many searches turn into bookings grouped by property rating? grouped by family size?

Business Analysts ask these questions without knowing in advance every aspect of the question
Architectural Info
Data Ingestion: Data Service → Analytics Service

- Separate services, separate nodes
  - Needed for performance isolation
  - Allows separate scaling based on needs

- Parallel shadowing of datasets (DCP)
  - Low impact on Data nodes
  - High data currency

- Other notes
  - M:N node connectivity
  - Not unlike GSI++
KV Buckets vs. Analytics Datasets

- Various shadowing patterns are possible (specified at CREATE DATASET time)

- Collections are coming in CB Server 7.0
LSM-Based Storage and Indexing

Log-Structured Merge Trees
- Support for fast ingestion
- B+ tree based components
- Bloom filters (search efficiency)

Sequential writes to disk

Periodically merge disk trees
An Indexed Analytics Dataset

Partitioned local storage and local indexing
- Hashed on primary key (PK)
- Primary index w/ PK + records in leaves
- Secondary index(es) with SK + PK
- Record updates are always local
N1QL for Analytics
1 Data Model (Review)
Data (JSON version)

Customers

{  
  "custid":"C37",
  "name":"T. Hanks",
  "address":{
    "street":"120 Harbor Blvd.",
    "city":"Boston, MA",
    "zipcode":"02115"
  },
  "rating":750
}

{  
  "custid":"C47",
  "name":"S. Lauren",
  "address":{
    "street":"17 Rue d'Antibes",
    "city":"Cannes, France"
  },
  "rating":625
}

Orders

{  
  "orderno":1004,
  "custid":"C35",
  "order_date":"2017-07-10",
  "ship_date":"2017-07-15",
  "items":[
    {
      "itemno":680,
      "qty":6,
      "price":9.99
    },
    {
      "itemno":195,
      "qty":4,
      "price":35.00
    }
  ]
}

{  
  "orderno":1008,
  "custid":"C13",
  "order_date":"2017-10-13",
  "items":[
    {
      "itemno":460,
      "qty":20,
      "price":99.99
    }
  ]
}

Data from D. Chamberlin. SQL++ for SQL Users: A Tutorial
Data (Relational version)

Customers

```json
{  
  "custid":"C37",
  "name":"T. Hanks",
  "address_street":"120 Harbor Blvd.",
  "address_city":"Boston, MA",
  "address_zipcode":"02115"
  "rating":750
}

{  
  "custid":"C47",
  "name":"S. Lauren",
  "address_street":"17 Rue d'Antibes",
  "address_city":"Cannes, France"
  "address_zipcode":null

  "rating":625
}
```

Orders

```json
{  
  "orderno":1004,
  "custid":"C35",
  "order_date":"2017-07-10",
  "ship_date":"2017-07-15"
}

{  
  "orderno":1008,
  "custid":"C13",
  "order_date":"2017-10-13",
  "ship_date":null
}
```

Lineitems

```sql
CREATE TABLE lineitems(
  orderno INTEGER,
  itemno INTEGER,
  quantity INTEGER NOT NULL,
  price DECIMAL(8,2) NOT NULL,
  PRIMARY KEY(orderno, itemno)
)
```

```json
{  
  "orderno":1004,
  "itemno":680,
  "qty":6,
  "price":9.99
}

{  
  "orderno":1004,
  "itemno":195,
  "qty":4,
  "price":35.00
}

{  
  "orderno":1008,
  "itemno":460,
  "qty":20,
  "price":99.99
}
```
### Data (Relational version)

#### Customers

```json
{
    "custid": "C37",
    "name": "T. Hanks",
    "address_street": "120 Harbor Blvd.",
    "address_city": "Boston, MA",
    "address_zipcode": "02115"
    "rating": 750
}
{
    "custid": "C47",
    "name": "S. Lauren",
    "address_street": "17 Rue d'Antibes",
    "address_city": "Cannes, France"
    "address_zipcode": null
    "rating": 625
}
```

#### Orders

```json
{
    "orderno": 1004,
    "custid": "C35",
    "order_date": "2017-07-10",
    "ship_date": "2017-07-15"
}
{
    "orderno": 1008,
    "custid": "C13",
    "order_date": "2017-10-13",
    "ship_date": null
}
```

#### Lineitems

```sql
CREATE TABLE lineitems(
    orderno INTEGER,
    itemno INTEGER,
    quantity INTEGER NOT NULL,
    price DECIMAL(8,2) NOT NULL,
    PRIMARY KEY(orderno, itemno)
)
```

```json
{
    "orderno": 1004,
    "itemno": 680,
    "qty": 6,
    "price": 9.99,
    "currency": "USD"
}
{
    "orderno": 1008,
    "itemno": 195,
    "qty": 4,
    "price": 35.00,
    "currency": "USD"
}
{
    "orderno": 1008,
    "itemno": 460,
    "qty": 20,
    "price": 99.99,
    "currency": "EUR"
}
```
Sloppy Data

Customers

```
{
    "custid":"C37",
    "name":"T. Hanks",
    "address":{
        "street":"120 Harbor Blvd.",
        "city":"Boston, MA",
        "zipcode":"02115"
    },
    "rating":750
}
```

```
{
    "custid":"C47",
    "name":"S. Lauren",
    "address":{
        "street":"17 Rue d'Antibes",
        "city":"Cannes, France"
    },
    "rating":625
}
```  

Orders

```
{
    "orderno":1004,
    "custid":"C35",
    "order_date":"2017-07-10",
    "ship_date":"2017-07-15",
    "items": [
        {
            "itemno":680,
            "qty":6,
            "price":9.99
        },
        {
            "itemno":195,
            "qty":4,
            "price":"if you have to ask ..."
        }
    ]
}
```

```
{
    "orderno":1008,
    "custid":"C13",
    "order_date":"2017-10-13",
    "items": {
        "itemno":460,
        "qty":20,
        "price":99.99
    }
}
```
2

SQL Heritage
Just like SQL ...

```sql
SELECT name
FROM customers
WHERE rating > 650;
```

```json
[{
    "name": "M. Streep"
},
{
    "name": "T. Hanks"
},
{
    "name": "T. Cruise"
}
]
```
Just like SQL ...

```
SELECT name
FROM customers
WHERE rating > 650;

SELECT c.name, o.order_date
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.custid = "C41";
```

```
[
  {
    "name": "R. Duvall",
    "order_date": "2017-09-02"
  },
  {
    "name": "R. Duvall",
    "order_date": "2017-04-29"
  }
]
```
SELECT name
FROM customers
WHERE rating > 650;

SELECT c.name, o.order_date
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.custid = "C41";

SELECT c.name, o.order_date
FROM customers AS c JOIN orders AS o
  ON c.custid = o.custid
WHERE c.custid = "C41";
Just like SQL ...

```
SELECT name
FROM customers
WHERE rating > 650;

SELECT c.name, o.order_date
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.custid = "C41";

SELECT order_date, count(*) AS cnt
FROM orders
GROUP BY order_date
HAVING count(*) > 0
ORDER BY order_date DESC
LIMIT 3;
```

```
[
  {
    "cnt": 1,
    "order_date": "2017-10-13"
  },
  {
    "cnt": 1,
    "order_date": "2017-09-13"
  },
  {
    "cnt": 1,
    "order_date": "2017-09-02"
  }
]
```
... almost!

```sql
SELECT name, order_date
FROM customers, orders
WHERE customers.custid = orders.custid
  AND rating > 650;
```

Cannot resolve ambiguous alias reference for identifier rating (in line 4, at column 7)
... almost!

```sql
SELECT name, order_date
FROM customers, orders
WHERE customers.custid = orders.custid
  AND rating > 650;

SELECT c.name, o.order_date
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.rating > 650;

[{
  "name": "T. Hanks",
  "order_date": "2017-08-30"
},
{
  "name": "T. Cruise",
  "order_date": "2017-05-01"
},
{
  "name": "T. Cruise",
  "order_date": "2017-10-13"
},
{
  "name": "T. Cruise",
  "order_date": "2017-09-13"
}]
```
... almost!

SELECT name, order_date
FROM customers, orders
WHERE customers.custid = orders.custid
    AND rating > 650;

SELECT c.name, o.order_date
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
    AND c.rating > 650;

SELECT *
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
    AND c.rating > 650;

[    
  {    
    "c": {    
      "address": {    
        "city": "Boston, MA",
        "street": "120 Harbor Blvd.",
        "zipcode": "02115"
      },
      "custid": "C37",
      "name": "T. Hanks",
      "rating": 750
    },
    "o": {    
      "custid": "C37",
      "items": [
        {    
          "itemno": 460,
          "price": 99.98,
          "qty": 2
        }
      ]
    }
  }
...
3 SELECT VALUE
SELECT VALUE name FROM customers WHERE rating > 650;

[ "M. Streep",
  "T. Hanks",
  "T. Cruise"
]
SELECT VALUE name
FROM customers
WHERE rating > 650;

SELECT VALUE {
  "CustomerName": c.name,
  "OrderDate": o.order_date
}
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.rating > 650;

[
  {
    "CustomerName": "T. Hanks",
    "OrderDate": "2017-08-30"
  },
  {
    "CustomerName": "T. Cruise",
    "OrderDate": "2017-09-13"
  },
  {
    "CustomerName": "T. Cruise",
    "OrderDate": "2017-05-01"
  },
  {
    "CustomerName": "T. Cruise",
    "OrderDate": "2017-10-13"
  }
]
SELECT VALUE name
FROM customers
WHERE rating > 650;

SELECT VALUE {
  "CustomerName":c.name,
  "OrderDate":o.order_date
}
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.rating > 650;

SELECT c.name AS CustomerName,
       o.order_date AS OrderDate
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
  AND c.rating > 650;
SELECT VALUE name
FROM customers
WHERE rating > 650;

SELECT VALUE {
   "CustomerName":c.name,
   "OrderDate":o.order_date
}
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
   AND c.rating > 650;

SELECT VALUE {
   "CustomerName":c.name,
   "Orders":(SELECT VALUE o.orderno FROM orders AS o
               WHERE o.custid = c.custid)
}
FROM customers AS c
WHERE c.custid = "C41";
Quiz

Which query retrieves the names of the customers that have the highest rating?

A

```sql
SELECT name
FROM customers
WHERE rating =
  (SELECT MAX(rating) FROM customers);
```

B

```sql
SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT VALUE MAX(c2.rating) FROM customers AS c2);
```

C

```sql
SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT MAX(c2.rating) FROM customers AS c2);
```

D

```sql
SELECT VALUE c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT VALUE MAX(c2.rating) FROM customers AS c2)[0];
```
SQL Pitfalls and the value of VALUE

SELECT name
FROM customers
WHERE rating =
 (SELECT MAX(rating) FROM customers);

SQL++ “best guesses” that customers is a field of customer

Type mismatch: expected value of type multiset or array, but got the value of type object (in line 4, at column 28)
SQL Pitfalls and the “value” of VALUE

SELECT name
FROM customers
WHERE rating =
  (SELECT MAX(rating) FROM customers);

SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT MAX(c2.rating) FROM customers AS c2);

Standard SQL would apply “flat world” row/column coercion magic
SQL Pitfalls and the value of VALUE

SELECT name
FROM customers
WHERE rating =
  (SELECT MAX(rating) FROM customers);

SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT MAX(c2.rating) FROM customers AS c2);

SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT VALUE MAX(c2.rating) FROM customers AS c2);

SQL++ SELECT statements always return collections (not scalars)
SQL Pitfalls and the value of VALUE

```
SELECT name
FROM customers
WHERE rating =
  (SELECT MAX(rating) FROM customers);

SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT MAX(c2.rating) FROM customers AS c2);

SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT VALUE MAX(c2.rating) FROM customers AS c2);

SELECT VALUE c1.name
FROM customers AS c1
WHERE c1.rating =
  (SELECT VALUE MAX(c2.rating) FROM customers AS c2)[0];
```

We know that the subquery returns only one value, so we extract it this way
Quiz Solution

Which query retrieves the names of the customers that have the highest rating?

A. 
```
SELECT name
FROM customers
WHERE rating =
    (SELECT MAX(rating) FROM customers);
```

B. 
```
SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
    (SELECT VALUE MAX(c2.rating) FROM customers AS c2);
```

C. 
```
SELECT c1.name
FROM customers AS c1
WHERE c1.rating =
    (SELECT MAX(c2.rating) FROM customers AS c2);
```

D. 
```
SELECT VALUE c1.name
FROM customers AS c1
WHERE c1.rating =
    (SELECT VALUE MAX(c2.rating) FROM customers AS c2)[0];
```
4

Nested Data
SELECT o.orderno,
o.order_date,
i.itemno AS item_number,
i.qty AS quantity
FROM orders AS o UNNEST o.items AS i
WHERE i.qty > 100
ORDER BY o.orderno, item_number;

[
{
   "orderno": 1002,
   "order_date": "2017-05-01",
   "item_number": 680,
   "quantity": 150
},
{
   "orderno": 1005,
   "order_date": "2017-08-30",
   "item_number": 347,
   "quantity": 120
},
{
   "orderno": 1006,
   "order_date": "2017-09-02",
   "item_number": 460,
   "quantity": 120
}]

Unnesting
Unnesting

SELECT o.orderno,
    o.order_date,
    i.itemno AS item_number,
    i.qty AS quantity
FROM orders AS o, o.items AS i
WHERE i.qty > 100
ORDER BY o.orderno, item_number;

SELECT o.orderno,
    o.order_date,
    i.itemno AS item_number,
    i.qty AS quantity
FROM orders AS o, o.items AS i
WHERE i.qty > 100
ORDER BY o.orderno, item_number;
Quantification

```
SELECT DISTINCT VALUE o.custid
FROM orders AS o
WHERE SOME i IN o.items SATISFIES i.price >= 25.00;

[ "C37",
  "C41",
  "C31",
  "C35",
  "C13"
]
```
Quantification

```sql
SELECT DISTINCT VALUE o.custid
FROM orders AS o
WHERE SOME i IN o.items SATISFIES i.price >= 25.00;

SELECT DISTINCT VALUE o.custid
FROM orders AS o
WHERE EVERY i IN o.items SATISFIES i.price >= 25.00;
```
Quantification

\[
\text{SELECT DISTINCT VALUE } o.\text{custid} \\
\text{FROM orders AS o} \\
\text{WHERE } \text{SOME } i \text{ IN o.items SATISFIES } i.\text{price } \geq 25.00; \\
\]

\[
\text{SELECT DISTINCT VALUE } o.\text{custid} \\
\text{FROM orders AS o} \\
\text{WHERE } \text{EVERY } i \text{ IN o.items SATISFIES } i.\text{price } \geq 25.00; \\
\]

\[
\text{SELECT DISTINCT VALUE } o.\text{custid} \\
\text{FROM orders AS o} \\
\text{WHERE } \text{array_count(o.items) } > 0 \\
\quad \text{AND } \text{EVERY } i \text{ IN o.items SATISFIES } i.\text{price } \geq 25.00; \\
\]
Quantification

SELECT DISTINCT VALUE o.custid
FROM orders AS o
WHERE SOME i IN o.items SATISFIES i.price >= 25.00;

SELECT DISTINCT VALUE o.custid
FROM orders AS o
WHERE EVERY i IN o.items SATISFIES i.price >= 25.00;

SELECT DISTINCT VALUE o.custid
FROM orders AS o
WHERE array_count(o.items) > 0
    AND EVERY i IN o.items SATISFIES i.price >= 25.00;

SELECT VALUE c
FROM customers AS c
WHERE c.custid IN (  
    SELECT DISTINCT VALUE o.custid
    FROM orders AS o
    WHERE SOME i IN o.items SATISFIES i.price >= 25.00
  )

[
  {  
    "address": {  
      "city": "Boston, MA",  
      "street": "120 Harbor Blvd.",  
      "zipcode": "02115"
    },  
    "custid": "C37",  
    "name": "T. Hanks",  
    "rating": 750
  },
  {  
    "address": {  
      "city": "St. Louis, MO",  
      "street": "150 Market St.",  
      "zipcode": "63101"
    },  
    "custid": "C41",  
    "name": "R. Duvall",
    "..."}
5

Grouping and Aggregation
SQL Grouping and Aggregation

```
SELECT c.address.city, count(*) AS cnt
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
GROUP BY c.address.city
```

```json
[
  {
    "cnt": 2,
    "city": "Boston, MA"
  },
  {
    "cnt": 6,
    "city": "St. Louis, MO"
  }
]
```
SQL Grouping and Aggregation

```
SELECT c.address.city, count(*) AS cnt
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
GROUP BY c.address.city
```

<table>
<thead>
<tr>
<th>c.address.city</th>
<th>c</th>
<th>o</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston, MA</td>
<td>C37</td>
<td>O1005</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>C35</td>
<td>O1004</td>
<td></td>
</tr>
<tr>
<td>St. Louis, MO</td>
<td>C41</td>
<td>O1006</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>C41</td>
<td>O1001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C31</td>
<td>O1003</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C13</td>
<td>O1007</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C13</td>
<td>O1002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C13</td>
<td>O1008</td>
<td></td>
</tr>
</tbody>
</table>
SQL++ Aggregation (only)

```
SELECT c.name, array_count(o.items) AS order_size
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
ORDER BY order_size DESC
LIMIT 3
```

```json
[
    {
        "order_size": 4,
        "name": "T. Hanks"
    },
    {
        "order_size": 3,
        "name": "R. Duvall"
    },
    {
        "order_size": 2,
        "name": "R. Duvall"
    }
]
```
SELECT c.name, array_count(o.items) AS order_size
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
ORDER BY order_size DESC
LIMIT 3

SELECT VALUE max(rating) FROM customers
WHERE LENGTH(name) > 3
SQL++ Aggregation (only)

```
SELECT c.name, array_count(o.items) AS order_size
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
ORDER BY order_size DESC
LIMIT 3

SELECT VALUE max(rating) FROM customers
WHERE LENGTH(name) > 3

ARRAY_MAX((SELECT VALUE rating FROM customers c
            WHERE LENGTH(name) > 3))
```
SQL++ Grouping (only)

```
SELECT c.address.city, g
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
GROUP BY c.address.city GROUP AS g;

[  
  {  
      "city": "Boston, MA",  
      "g": [  
        {  
            "c": {  
                "address": { "city": "Boston, MA", ... },  
                "custid": "C35", "name": "J. Roberts",  
                "rating": 565  
            },  
            "o": {  
                "custid": "C35",  
                "items": [  
                    { "itemno": 680, "price": 9.99, "qty": 6 },  
                    { "itemno": 195, "price": 35, "qty": 4 }],  
                "order_date": "2017-07-10", "orderno": 1004,  
                "ship_date": "2017-07-15"  
            }  
        },  
        ...  
      }  
  }  
]
```
SQL Grouping and Aggregation Explained

```sql
SELECT c.address.city, count(*) AS cnt
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
GROUP BY c.address.city
```

```
[{
  "cnt": 2,
  "city": "Boston, MA"
},
{
  "cnt": 6,
  "city": "St. Louis, MO"
}
]
```
**SQL Grouping and Aggregation Explained**

```sql
SELECT c.address.city, count(*) AS cnt
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
GROUP BY c.address.city

SELECT c.address.city, array_count(g) AS cnt
FROM customers AS c, orders AS o
WHERE c.custid = o.custid
GROUP BY c.address.city GROUP AS g;
```

```
[
  {
    "cnt": 2,
    "city": "Boston, MA"
  },
  {
    "cnt": 6,
    "city": "St. Louis, MO"
  }
]
```
6 Missing Information
## Remember the Data

### Customers

```json
{
  "custid":"C37",
  "name":"T. Hanks",
  "address":{
    "street":"120 Harbor Blvd.",
    "city":"Boston, MA",
    "zipcode":"02115"
  },
  "rating":750
}

{
  "custid":"C47",
  "name":"S. Lauren",
  "address":{
    "street":"17 Rue d'Antibes",
    "city":"Cannes, France"
  },
  "rating":625
}
```

### Orders

```json
{
  "orderno":1004,
  "custid":"C35",
  "order_date":"2017-07-10",
  "ship_date":"2017-07-15",
  "items":[
    {
      "itemno":680,
      "qty":6,
      "price":9.99
    },
    {
      "itemno":195,
      "qty":4,
      "price":35.00
    }
  ]
}

{  
  "orderno":1008,
  "custid":"C13",
  "order_date":"2017-10-13",
  "items":[
    {
      "itemno":460,
      "qty":20,
      "price":99.99
    }
  ]
}
```

Data from D. Chamberlin. *SQL++ for SQL Users: A Tutorial*
Have I "missed" anything?

```
SELECT o.orderno, o.order_date, o.ship_date, o.custid
FROM orders o
WHERE o.ship_date IS MISSING

[{
  "orderno": 1005,
  "order_date": "2017-08-30",
  "custid": "C37"
},
{
  "orderno": 1008,
  "order_date": "2017-10-13",
  "custid": "C13"
}]
```
SELECT o.orderno, o.order_date, o.ship_date, o.custid
FROM orders o
WHERE o.ship_date IS MISSING

SELECT VALUE {
    "orderno": o.orderno,
    "order_date": o.order_date,
    "ship_date": o.ship_date,
    "custid": o.custid
}
FROM orders o
WHERE o.ship_date IS MISSING

[{
    "orderno": 1005,
    "order_date": "2017-08-30",
    "custid": "C37"
},
{
    "orderno": 1008,
    "order_date": "2017-10-13",
    "custid": "C13"
}]

Have I "missed" anything?
Have I "missed" anything?

```sql
SELECT o.orderno, o.order_date, o.ship_date, o.custid
FROM orders o
WHERE o.ship_date IS MISSING

SELECT VALUE {
  "orderno": o.orderno,
  "order_date": o.order_date,
  "ship_date": o.ship_date,
  "custid": o.custid
}
FROM orders o
WHERE o.ship_date IS MISSING

... WHERE o.ship_date IS NOT MISSING
... WHERE o.ship_date IS UNKNOWN
... WHERE o.ship_date IS NULL
...```
Dealing with different "cases"

```
SELECT VALUE {
  "orderno": o.orderno,
  "order_date": o.order_date,
  "ship_date":
    CASE
      WHEN o.ship_date IS MISSING THEN "TBD"
    ELSE o.ship_date
    END,
  "custid": o.custid
} FROM orders o
ORDER BY ship_date DESC
```

```
[{
  "orderno": 1005,
  "order_date": "2017-08-30",
  "ship_date": "TBD",
  "custid": "C37"
}, {
  "orderno": 1008,
  "order_date": "2017-10-13",
  "ship_date": "TBD",
  "custid": "C13"
}, {
  "orderno": 1007,
  "order_date": "2017-09-13",
  "ship_date": "2017-09-20",
  "custid": "C13"
}, ...
```
Further N1QL/SQL++ Info
Read The SQL++ Book! (Or take the online tutorial)

D. Chamberlin
SQL++ for SQL Users: A Tutorial.

https://sqlplusplus-tutorial.couchbase.com/tutorial/

https://www.couchbase.com/sql-plus-plus-for-sql-users

This link will show you how to set up the SQL++ Tutorial data on your own Couchbase Server instance!
N1QL or Couchbase Server Questions?