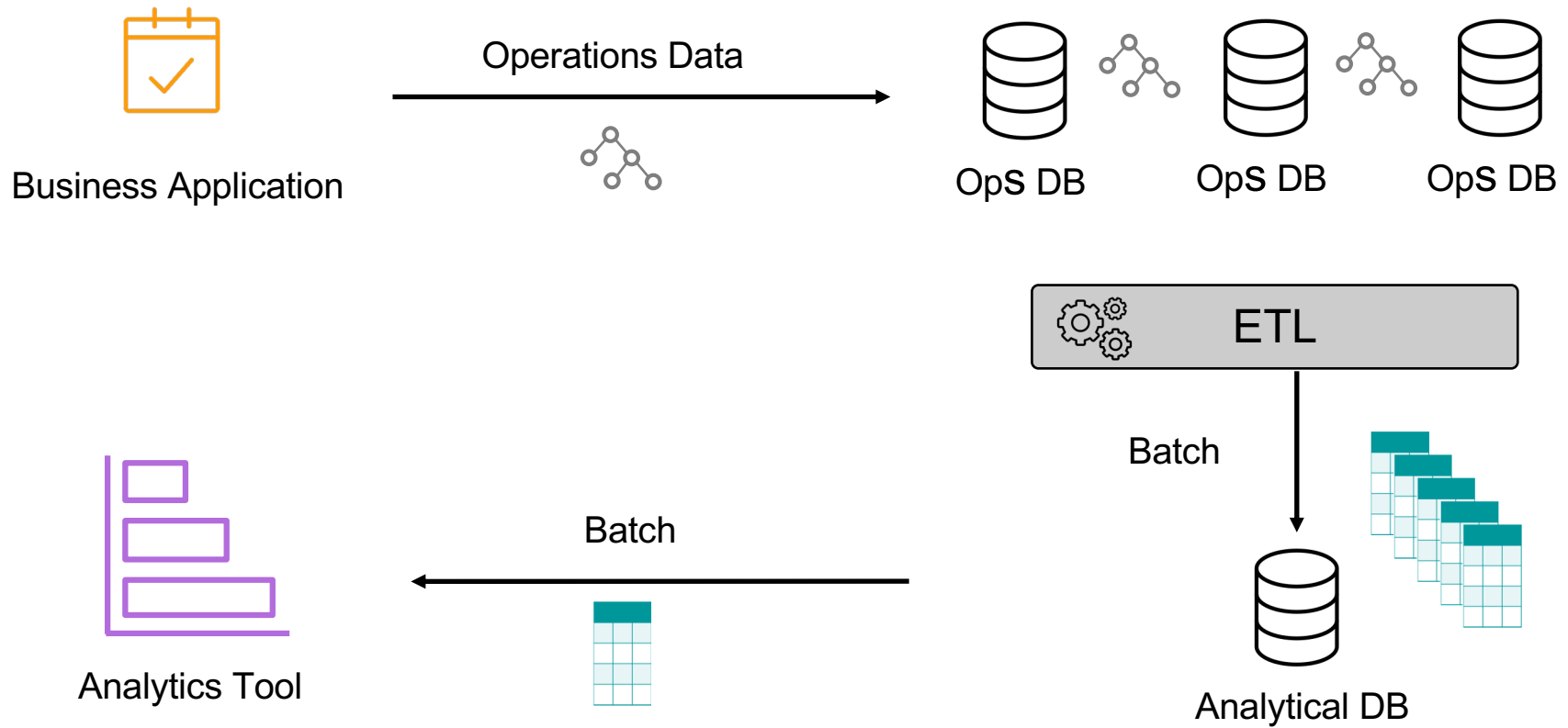




# Couchbase Analytics Service

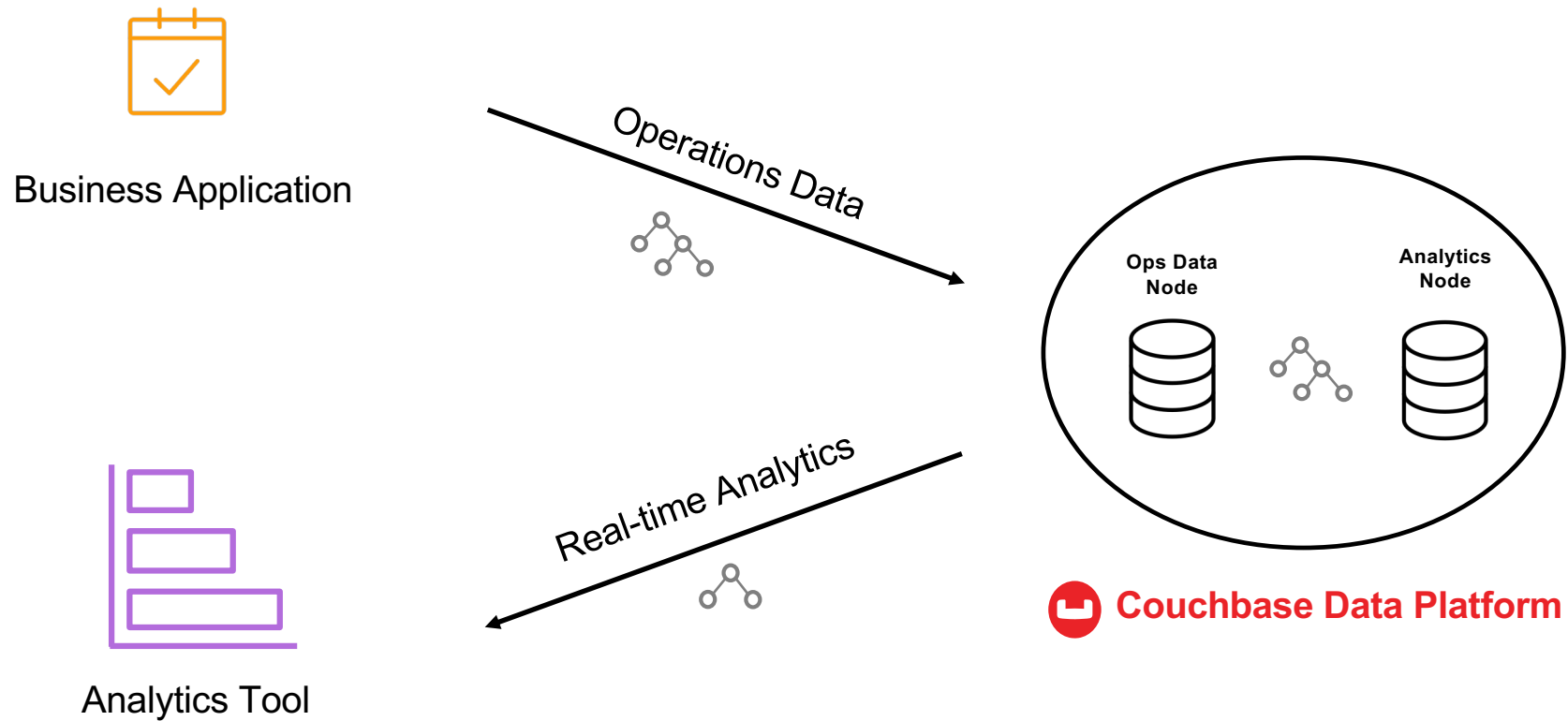


# Traditional Analytics Solutions





# Couchbase Analytics – Bringing NoETL to NoSQL

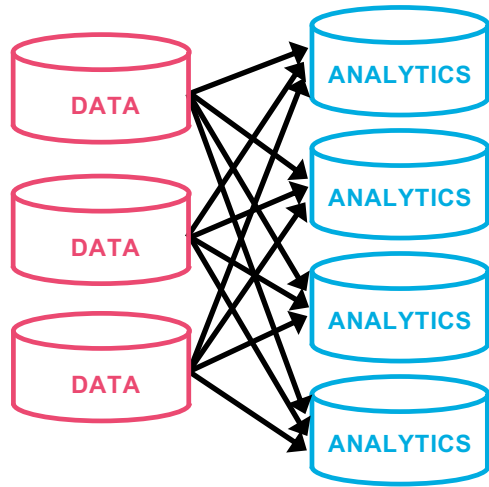


 Couchbase Data Platform



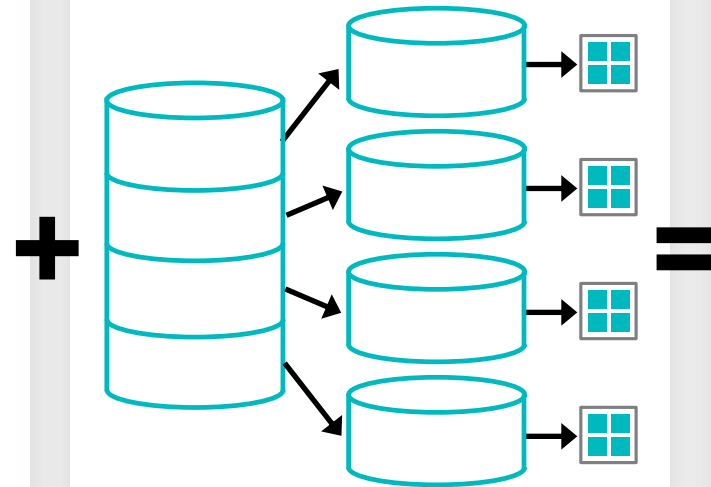
# What is Couchbase Analytics?

Shadow Dataset of a Couchbase Data node



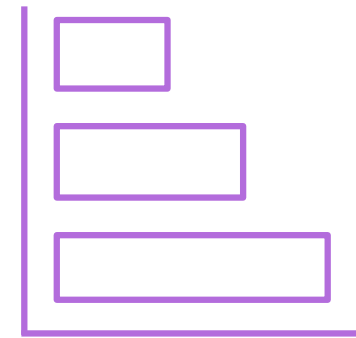
Fast Ingest

MPP architecture: parallelization among core and servers



Complex Queries on Large Datasets

Real-time Insights for Business Teams





# Couchbase Analytics Benefits

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

\*Yielding a NoSQL version of **HTAP** (Hybrid Transactional / 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**

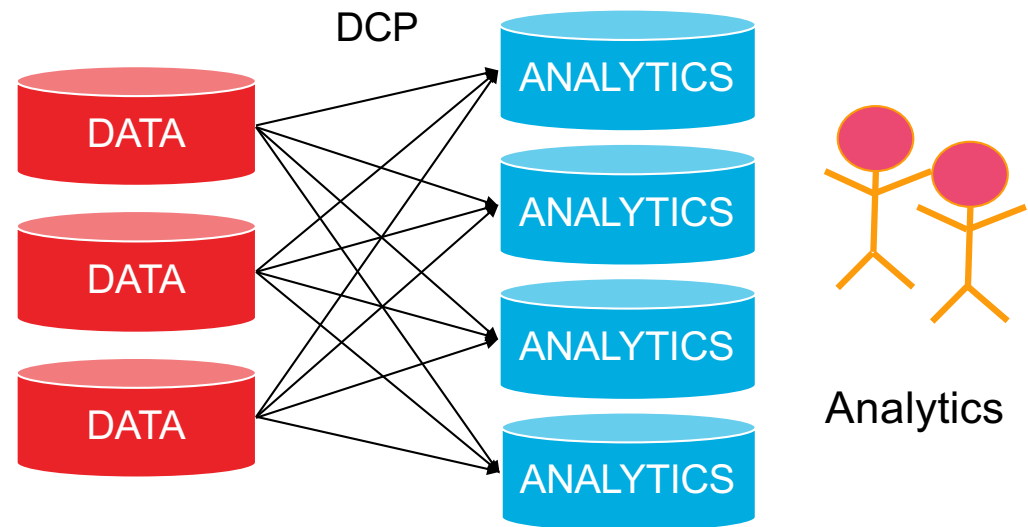
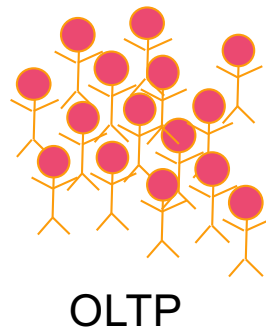


# Architectural Info



# 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++

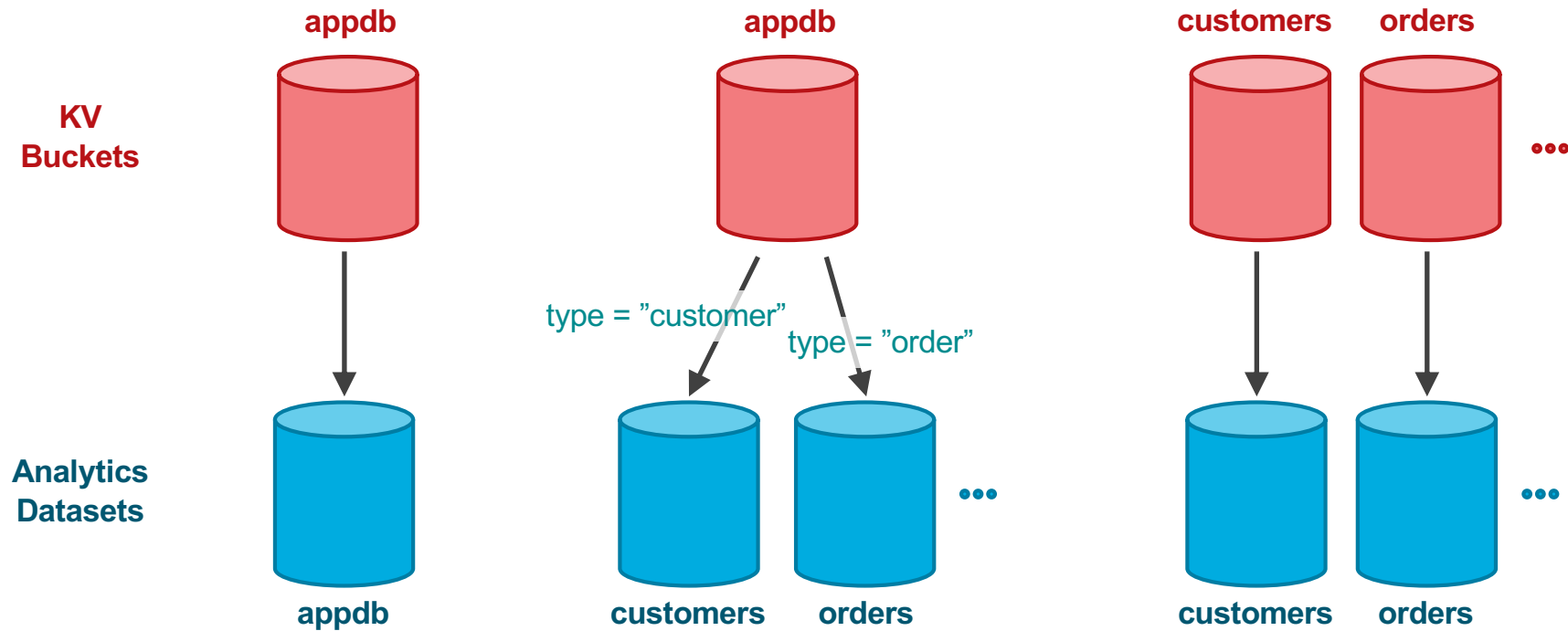




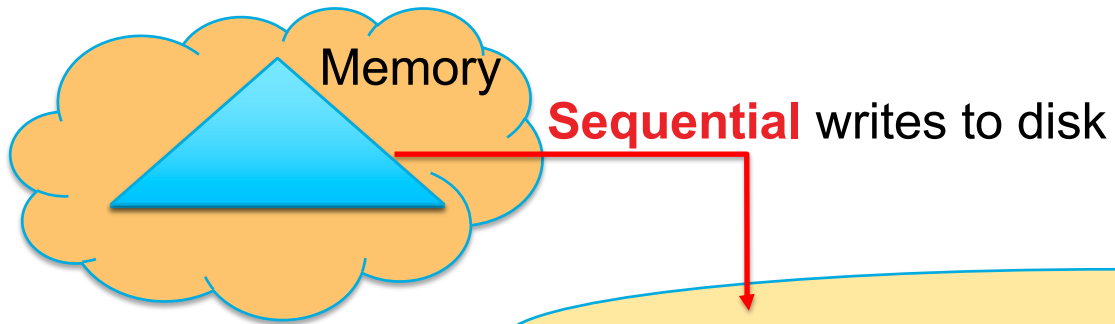


# KV Buckets vs. Analytics Datasets

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

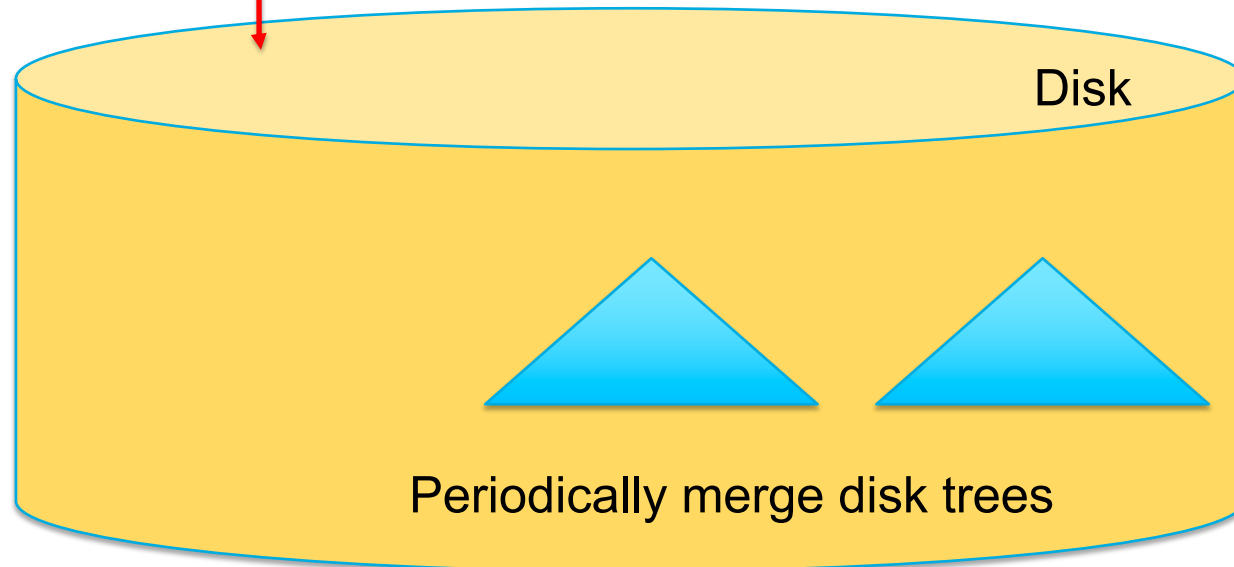


# LSM-Based Storage and Indexing



## Log-Structured Merge Trees

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

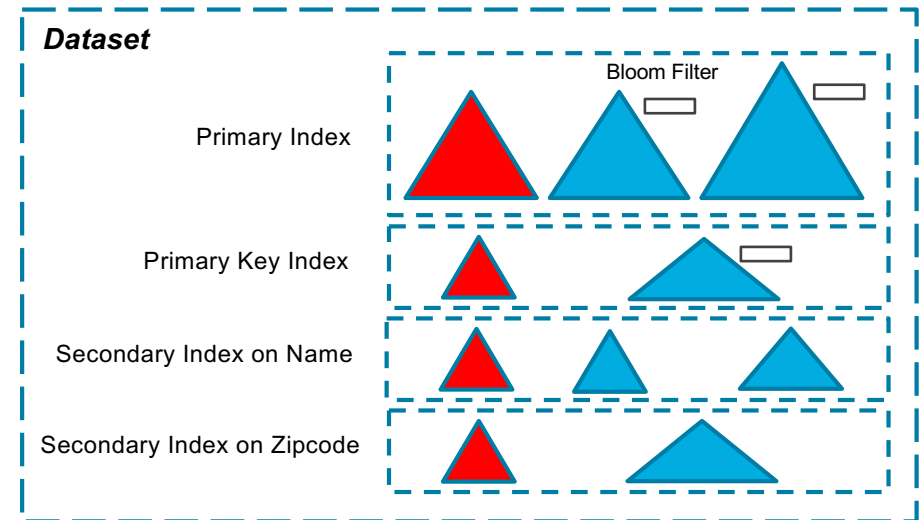
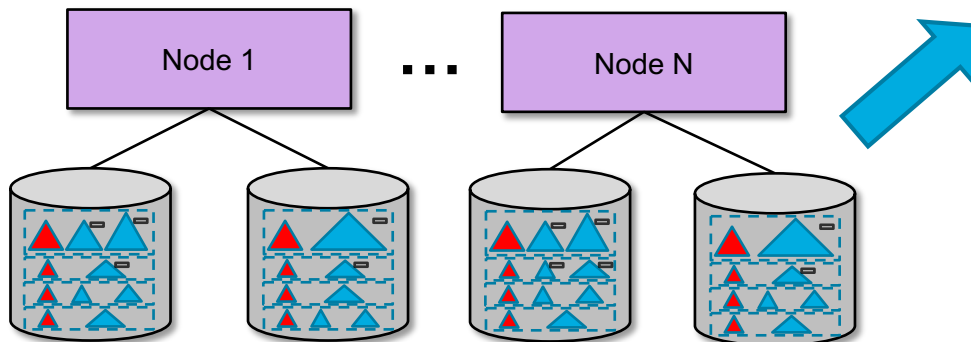




# An Indexed Analytics Dataset

## 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

```
{
  "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

```
{
  "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
}
```

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

## Lineitems

```
{
  "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

```
{
  "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

```
{
  "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
}
```

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

## Lineitems

```
{
  "orderno": 1004,
  "itemno": 680,
  "qty": 6,
  "price": 9.99,
  "currency": "USD"
}
{
  "orderno": 1004,
  "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 ...

---



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

```
[
  {
    "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"
  }
]
```

# 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 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!

---



```
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!

```
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

# Added "VALUE"

---



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

```
[  
  "M. Streeper",  
  "T. Hanks",  
  "T. Cruise"  
]
```



## Added "VALUE"

```
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"
  }
]
```



## Added "VALUE"

---

```
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;
```



## Added "VALUE"

```
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";
```

```
[
  {
    "Orders": [
      1006,
      1001
    ],
    "CustomerName": "R. Duvall"
  }
]
```

# Quiz



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];
```



# 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);
```


```
[  
  "T. Cruise",  
  "T. Hanks"  
]
```

```
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



# Unnesting

```
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



```
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;
```

```
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



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

```
[
  "C41",
  "C31",
  "C13"
```

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

```
]
```



# Quantification

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

```
[
  "C41",
  "C31",
  "C13"
]
```

```
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;
```



# 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
```

```
[
  {
    "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
```

c.address.city	c	o	
Boston, MA	C <sub>C37</sub>	O <sub>1005</sub>	} 2
	C <sub>C35</sub>	O <sub>1004</sub>	
St. Louis, MO	C <sub>C41</sub>	O <sub>1006</sub>	} 6
	C <sub>C41</sub>	O <sub>1001</sub>	
	C <sub>C31</sub>	O <sub>1003</sub>	
	C <sub>C13</sub>	O <sub>1007</sub>	
	C <sub>C13</sub>	O <sub>1002</sub>	
	C <sub>C13</sub>	O <sub>1008</sub>	



## 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
```

```
[
  {
    "order_size": 4,
    "name": "T. Hanks"
  },
  {
    "order_size": 3,
    "name": "R. Duvall"
  },
  {
    "order_size": 2,
    "name": "R. Duvall"
  }
]
```

## SQL++ Aggregation (only) – *PICK UP HERE*

---



```
SELECT c.name, array_count(o.items) AS order_size    [
FROM customers AS c, orders AS o                    750
WHERE c.custid = o.custid                            ]
ORDER BY order_size DESC
LIMIT 3
```

```
SELECT VALUE max(rating) FROM customers
```





## SQL++ Aggregation (only)

---

```
SELECT c.name, array_count(o.items) AS order_size [
FROM customers AS c, orders AS o                750
WHERE c.custid = o.custid                       ]
ORDER BY order_size DESC
LIMIT 3
```

```
SELECT VALUE max(rating) FROM customers
```

```
array_max((SELECT VALUE rating FROM customers))
```



# 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"
      }
    }
  },
  ...
]
```

```
{
  "c": {
    "address": { "city": "Boston, MA", ... },
    "custid": "C37", "name": "T. Hanks",
    "rating": 750
  },
  "o": {
    "custid": "C37",
    "items": [
      { "itemno": 460, "price": 99.98, "qty": 2 },
      { "itemno": 347, "price": 22, "qty": 120 },
      { "itemno": 780, "price": 1500, "qty": 1 },
      { "itemno": 375, "price": 149.98, "qty": 2 }
    ],
    "order_date": "2017-08-30", "orderno": 1005
  }
}
...
]
```



# SQL Grouping and Aggregation Explained

---

```
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

---

```
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

```
{
  "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*



# 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"  
  }  
]
```



# Have I "missed" anything?

```
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?

```
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
...
```

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



## 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"
  },
  ...
]
```



# 7

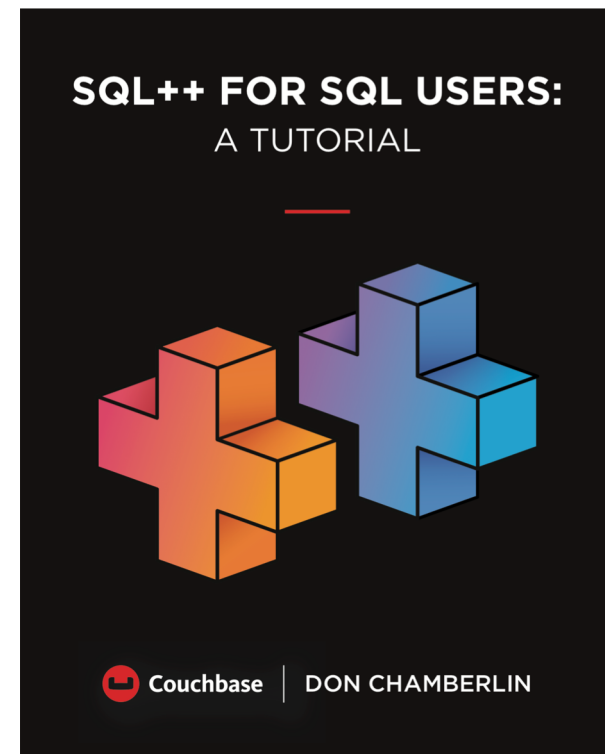
## Futher N1QL/SQL++ Info

# Read The SQL++ Book! (Or take the online tutorial)



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

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



# N1QL or Couchbase Server Questions?

---

