

SQL versus NoSQL

Guido Schmutz

Philipp Salvisberg

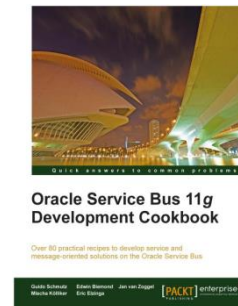
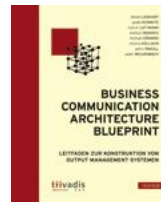


Trivadis
makes IT
easier.

BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA

■ Guido Schmutz

- Working for Trivadis for more than 17 years
- Oracle ACE Director for Fusion Middleware and SOA
- Co-Author of different books
- Consultant, Trainer Software Architect for Java, Oracle, SOA and Big Data / Fast Data
- Member of Trivadis Architecture Board
- Technology Manager @ Trivadis
- More than 25 years of software development experience
- Contact: guido.schmutz@trivadis.com
- Blog: <http://guidoschmutz.wordpress.com>
- Twitter: [gschmutz](https://twitter.com/gschmutz)



■ Philipp Salvisberg

- With Trivadis since April 2000
 - Senior Principal Consultant, Partner
 - Member of the Board of Directors
 - philipp.salvisberg@trivadis.com
 - www.salvis.com/blog
 - [@phsalvisberg](https://twitter.com/phsalvisberg)
- Main focus on database centric development with Oracle database
 - Application Development
 - Business Intelligence
 - Application Performance Management
- Over 20 years experience in using Oracle products



■ Agenda

1. Motivation
2. Overview of SQL and NoSQL Data Stores
3. Use Cases – Let's Get Ready to Rumble
4. Core Messages

■ Understanding the Merits

Why NoSQL?

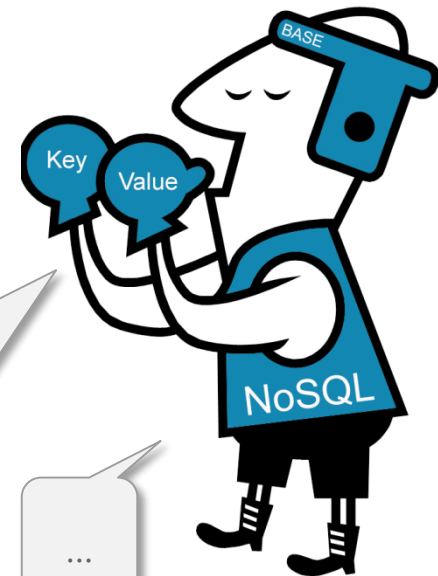
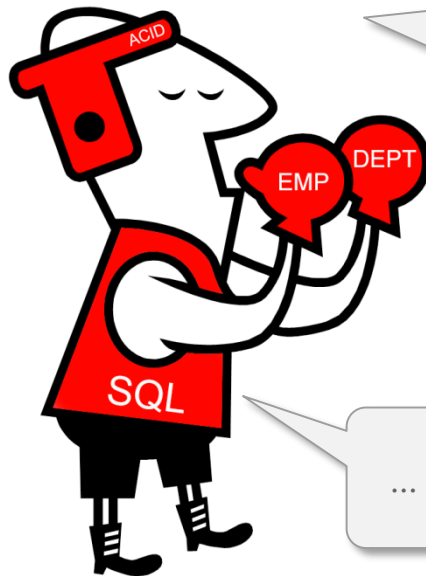
Can do everything with RDBMSs!
Having performance, scalability,
transactions and SQL.

RDBMSs do not scale as good as
NoSQL systems like Google's
BigTable.

A key-value store is adequate for key
lookups and easier to understand than
a RDBMS.

RDBMSs provide a common interface
with SQL, transactions, and relational
schema.

Some applications require a flexible
schema.
Adding new attributes at runtime in
RDBMSs is typically not possible.



■ Agenda

1. Motivation
2. Overview of SQL and NoSQL Data Stores
3. Use Cases – Let's Get Ready to Rumble
4. Conclusion

■ SQL Data Stores

- Relational Model
- Standardized, SQL:2011 is the 7th major revision since SQL-86
 - 9 parts, more than 4000 pages
 - But no single database implements all standards/features
- Rich set of features
 - Incl. SQL/PSM, SQL/MED, SQL/XML, SQL/RPR, Temporal Features
 - Incl. User-defined Types and Collection Types (since SQL:1999)
- ACID Transactions
 - **Atomicity**: all or nothing
 - **Consistency**: from valid state to valid state considering constraints, triggers, ...
 - **Isolation**: result is not affected through concurrent execution
 - **Durability**: committed data stays available after crash, power loss or errors
- Good support by different languages, frameworks and tools
- Good understanding of basic concepts by IT professionals

■ NoSQL Definition

- Next Generation Databases mostly addressing some of the points:
 - being non-relational,
 - distributed,
 - open-source and
 - horizontally scalable.
- Often more characteristics apply such as:
 - schema-free,
 - easy replication support,
 - simple API,
 - eventually consistent / BASE (not ACID),
 - a huge amount of data
 - and more.
- The misleading term "nosql" (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above

BASE

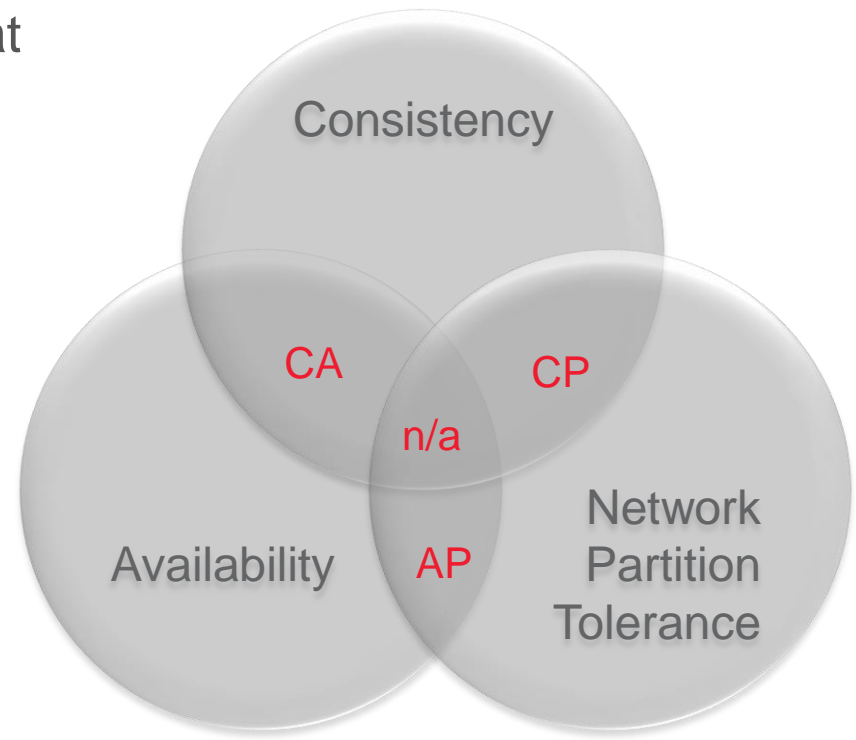
- **Basically Available:** Availability is more important than consistency
- **Soft State:** Higher availability results in an eventual consistent state
- **Eventually Consistent:** If no new updates are made to a given data item, eventually all accesses to that item will return the last updated value

Source: <http://nosql-database.org>

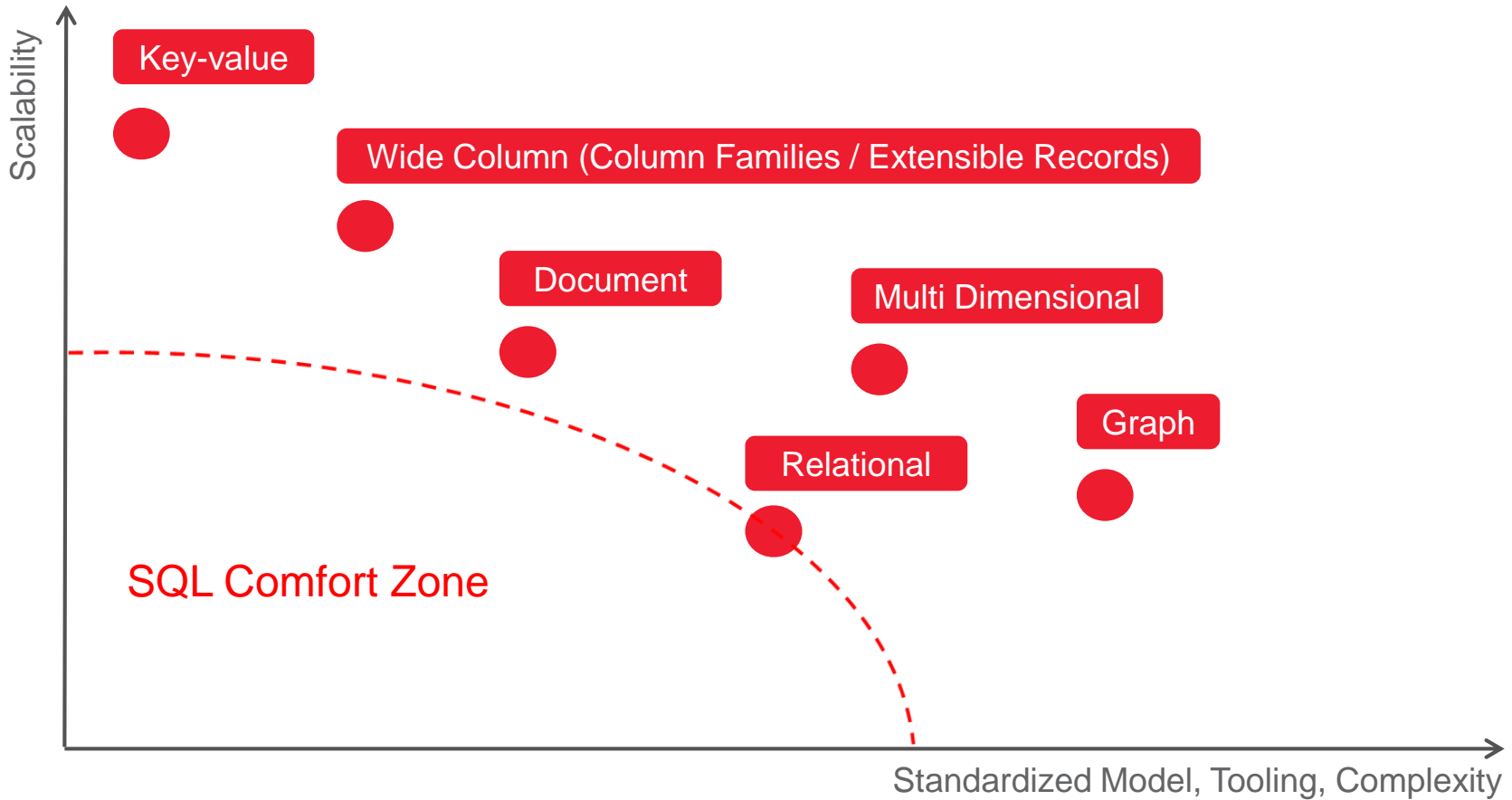
■ Brewer's CAP Theorem

Any networked shared-data system can have at most two of the three desirable properties:

- **C**onsistency
All of the nodes see the same data at the same time, regardless of where the data is stored
- **A**vailability
Node failures do not prevent survivors from continuing to operate
- Network **P**artition tolerance
The system continues to operate despite arbitrary message loss



■ Data Store Positioning



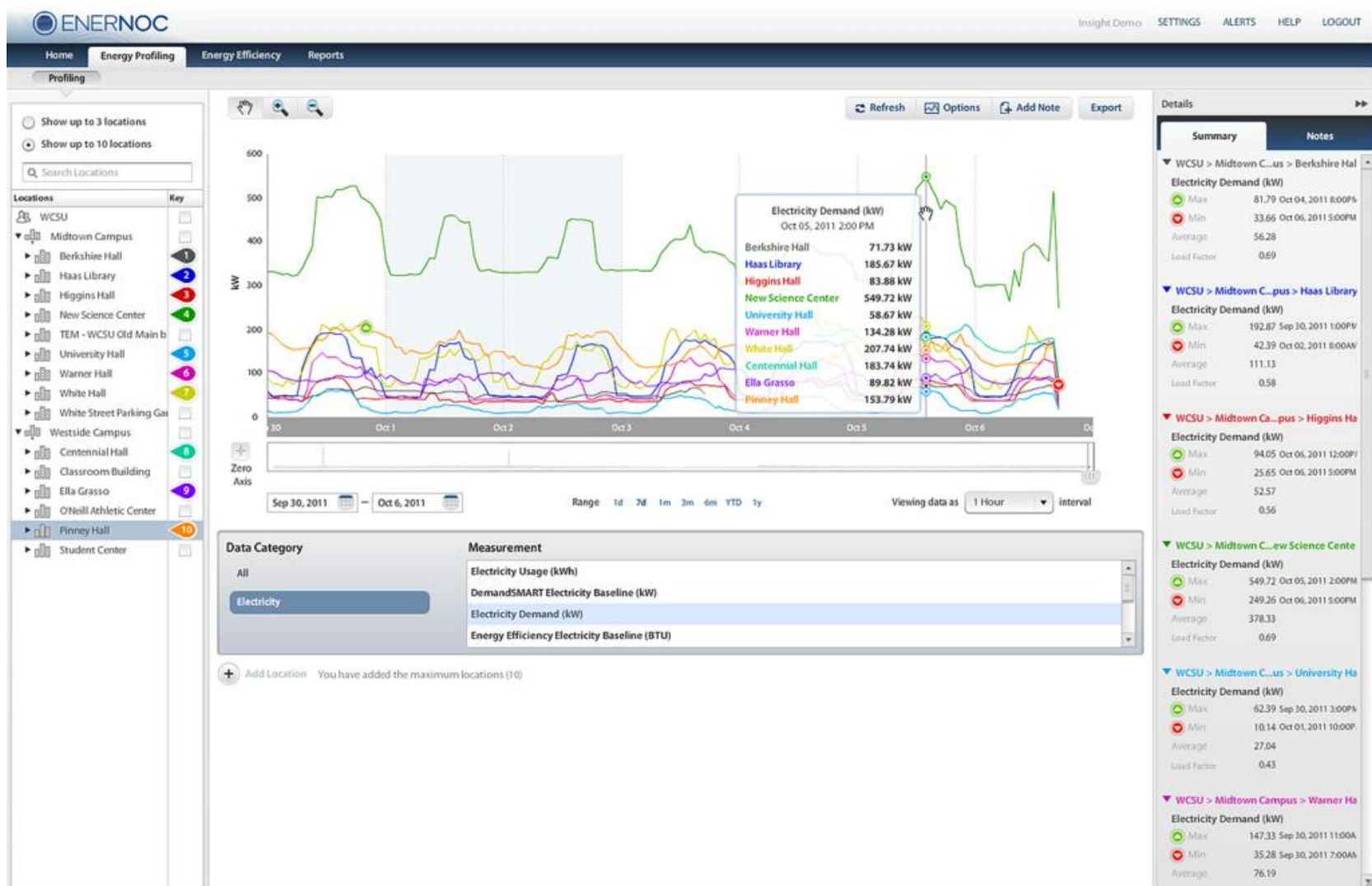
■ Agenda

1. Motivation
2. Overview of SQL and NoSQL Data Stores
3. Use Cases – Let's Get Ready to Rumble
4. Core Messages

Round 1

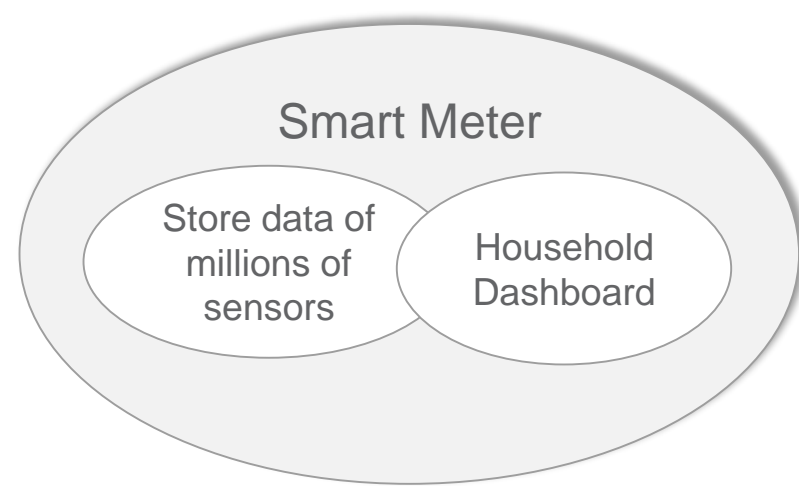
Smart Meter

Smart Meter – Customer Dashboard



■ Smart Meter – Use Cases

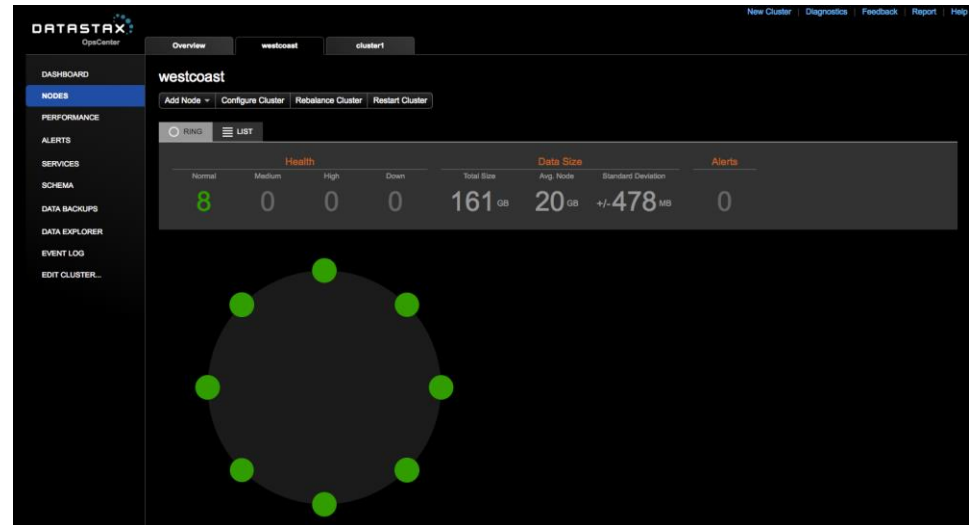
- Store sensor and its sub-sensor values
 - 2 Mio sensors, up to 10 sub-sensors
 - Energy consumption per second per sensor (kWh)
 - Delivery interval between 1 second and 5 minutes
- Query usage per sensor and its sub-sensors to visualize a time series on a customer dashboard
 - Available in different granularities, values are aggregated in
 - Minute
 - Quarter of hour (15-minutes)
 - Hour
 - Day
 - Responsive UI



■ Cassandra NoSQL Datastore



- Wide-Column Store
- Developed at Facebook
- Professional grade support from DataStax
- Main Features
 - Real-Time
 - Highly Distributed
 - Support for Multiple Data Center
 - Highly Scalable
 - No Single Point of Failure
 - Fault Tolerant
 - Tunable Consistency
 - CQL – Cassandra Query Language



■ The Wide Column Store Way (Cassandra)

Household	Bucket								
AFG10	MINUTE-2014/03/5	sensor	1	1	1	...	2	2	...
		at
		kwh	7.05	7.10	8.11	...	6.95	7.04	...
AFG10	QHOURL-2014/03	sensor	1	1	1	...	2	2	...
		at
		kwh	105.78	104.73	102.29	...	102.78	101.61	...
AFG10	HOURL-2014/03	sensor	1	1	1	...	2	2	...
		at	5T11	5T10	5T09	...	5T11	5T10	...
		kwh	423.00	410.33	395.99	...	598.32	522.12	...
AFG10	DAY-2014	sensor	1	1	1	...	2	2	...
		at	5T	3T	2T	...	5T	4T	...
		kwh	10100.2	9892.2	8987.4	...	879.8	912,3	...
GXK11	MINUTE-2014/03/5	sensor	1	1	1	...	2	2	...
		at	11:59	11:03	11:04	...	11:01	11:02	...
		kwh	100.10	90.88	95.00	...	92.50	88.50	...

24h * 60m * 11 sensor = 15'840 cols

30d * 24h * 4q * 11 sensor = 31'680 cols

30d * 24h * 11 sensor = 7'920 cols

365d * 11 sensor = 4'011 cols

Growth

■ The Cassandra Way

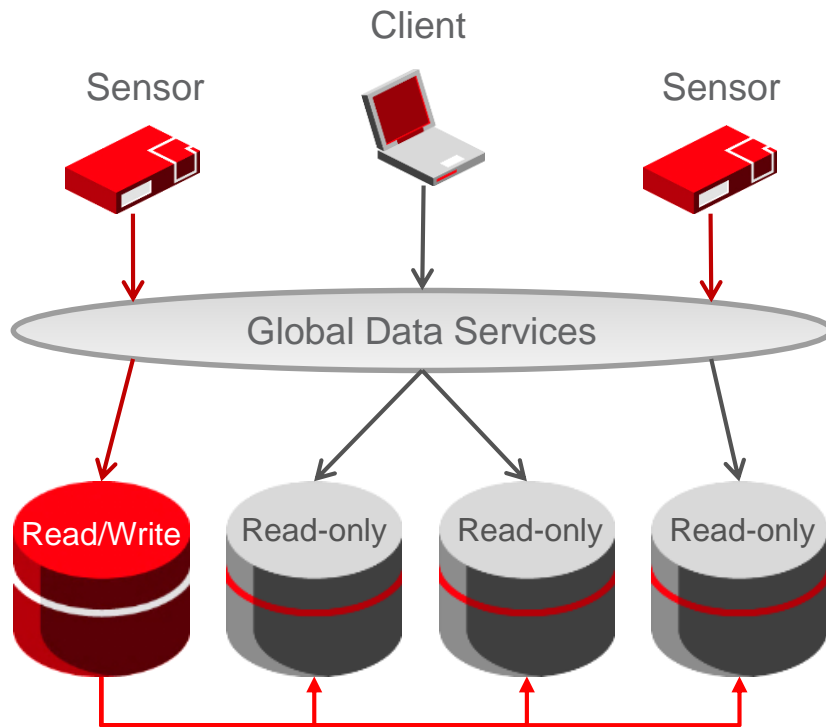
- 288 nodes on EC2
- Over 1 Mio writes/sec => 60Mio writes/min
- Rolling counters, always up to date

Household	Bucket								
AFG10	MINUTE-2014/03/5	sensor	1	1	1	...	2	2	...
		at	11:59	11:58	11:57	...	11:59	11:58	...
		kwh	7.05	7.10	8.11	...	6.95	7.04	...
AFG10	QHOURL-2014/03	sensor	1	1	1	...	2	2	...
		at	5T11:45	5T11:30	5T11:15	...	5T11:45	5T11:30	...
		kwh	105.38	104.33	103.38	...	103.38	101.61	...

```
UPDATE meter_reading_timeunit
SET kwh_consumed = kwh_consumed + 10010
WHERE household_id = 2dc487f0-b271-11e3-a5e2-0800200c9a66
AND sensor_id = 1
AND bucket_id = 'MINUTE-2014/03/23/11'
AND at_timestamp = '2014-03-23T11:01:00';
```

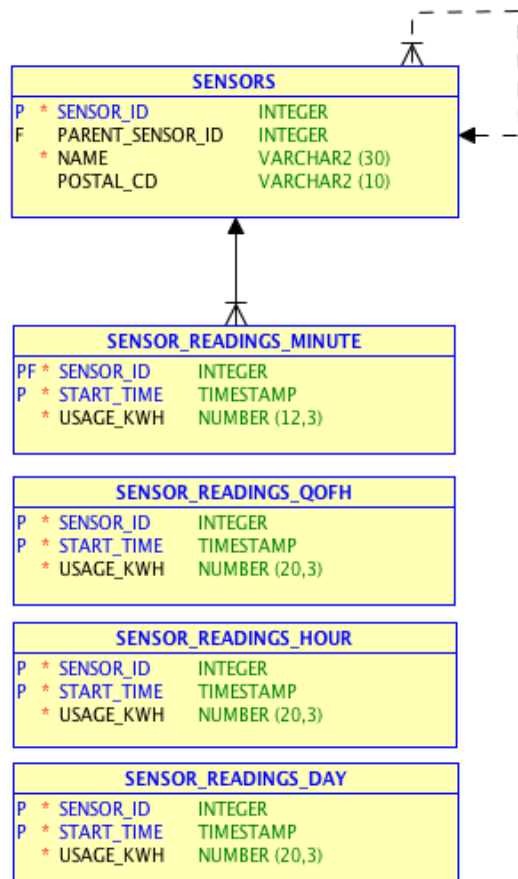
```
select household_id, bucket_id, at_timestamp, sensor_id, kwh_consumed
from meter_reading_timeunit
where household_id = 2dc487f0-b271-11e3-a5e2-0800200c9a66
and bucket_id = 'MINUTE-2014/03/23/11'
and sensor_id = 1
and at_timestamp > '2014-03-23T11:00:00'
order by sensor_id, at_timestamp DESC;
```

■ Relational Architecture



- Active Data Guard Configuration
- Global Data Services redirects requests based on
 - Server loads
 - Request type (read/write)
- Reader farm is geographically spread
- Failover/switchover to any node in the reader farm is possible
 - Read services are not affected
 - Write services are unavailable for a short period of time
- Scalability of the write services is the bottleneck of the system

■ Relational Data Model



- **SENSOR_READINGS_...**
 - Index-organized tables
 - Daily partitions
- **JDBC Batch Merges**
 - A transaction per sensor delivery
 - A single network roundtrip to merge 55 readings of a sensor delivery
 - Average between
 - 0.4 Mio tpm (delivery per 5 minutes)
 - 120 Mio tpm (delivery per second)
 - Top TPC-C Benchmark: 8.5 Mio tpm
- Batch job to aggregate readings every 15 minutes, avoiding intermediate results (updates)
 - Quarter of hour (5760 times a day)
 - Hour (24 times a day)
 - Day (once a day)

■ Query Sensor Data – The SQL Way

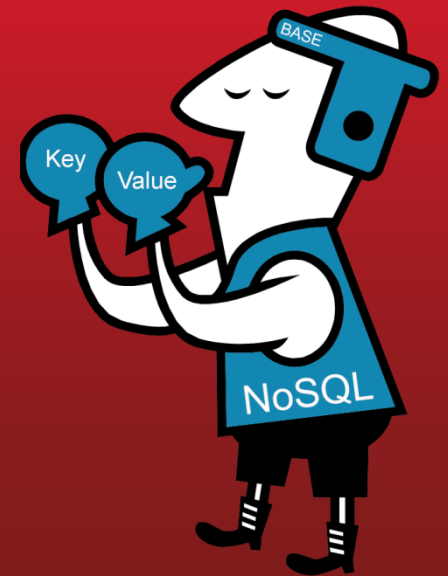
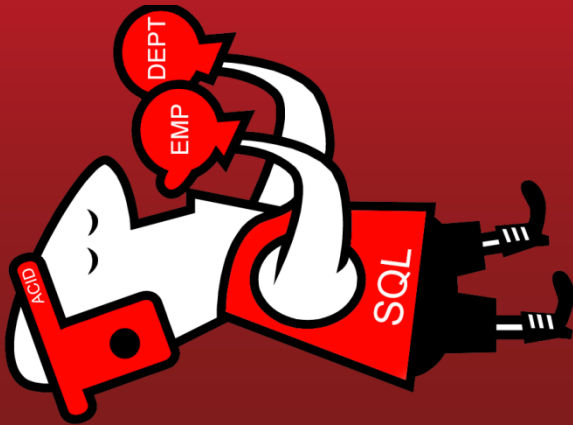
```
MERGE INTO sensor_readings_minute t
USING (SELECT sensor_id, TRUNC(start_time, 'MI') AS start_time,
        usage_kwh FROM TABLE(:p_sensor_data_list)) s
  ON (t.sensor_id = s.sensor_id AND t.start_time = s.start_time)
WHEN NOT MATCHED THEN
  INSERT (t.sensor_id, t.start_time, t.usage_kwh)
  VALUES (s.sensor_id, s.start_time, s.usage_kwh)
WHEN MATCHED THEN
  UPDATE SET t.usage_kwh = t.usage_kwh + s.usage_kwh;
```

Use aggregate tables to change granularity (quarter of hours, hours, days)

```
SELECT sensor_id, start_time, usage_kwh
  FROM sensor_readings_minute
 WHERE sensor_id = :p_sensor_id
    AND start_time BETWEEN :p_from AND :p_to
 ORDER BY sensor_id, start_time;
```

Smart Meter

0 – 1



Round 2

Order Entry

Order Entry – Example

The screenshot displays the Sage Accpac O/E Order Entry window. The main form contains the following fields:

- Order No.: QT0000000000003
- Customer No.: 1200
- Mr. Ronald Black
- Order Date: 05/31/2010
- Order Type: Quote
- Ship-To Location: 1
- Description: Quote 3 with header comments
- Price List: USA
- Location: 1
- Exp. Date: 05/31/2010
- Order Subtotal: 225.54

The inventory summary table shows the following data:

Item	Type	Item No. / Misc. Charge	Kit/BOM	Description	Price List	Location	Exp.
1	Item	A1-105/0		13W Mini Fluore...	USA	1	5/31
2	Item	A1-700/0		Calculator	USA	1	5/31

The inventory summary table also includes the following data:

	Qty. on Hand	Qty. on Sales Order	Qty. on Purchase Order	Qty. Committed	Qty. Available
Location 1 (Ea.)	70	21	105	0	70
All Locations (Ea.)	378	29	500,535	0	378

The Aged Receivables chart shows the following data:

Aging Periods (Days)	Amount
Current	20,000
1-30	5,000
31-60	2,000
61-90	1,000
Over 90	1,000

Total: 28,826

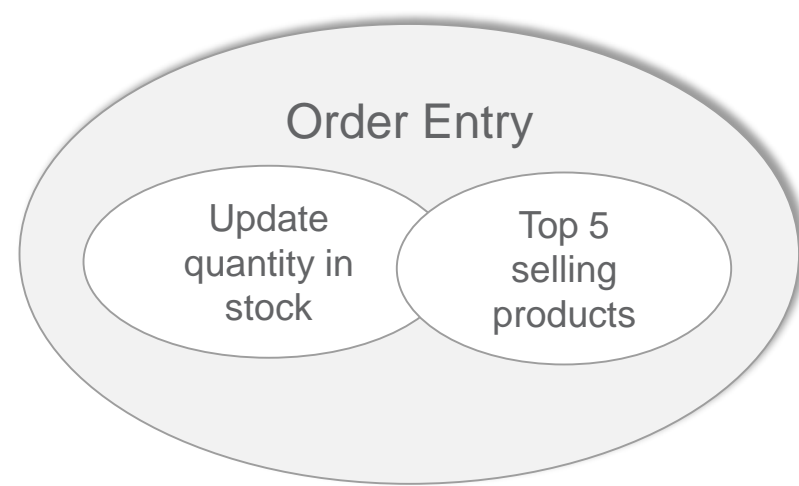
The Payables Outstanding chart shows the following data:

Payables Outstanding (Days)	Amount
Current	400
1-30	300
31-60	200
61-90	100
Over 90	100

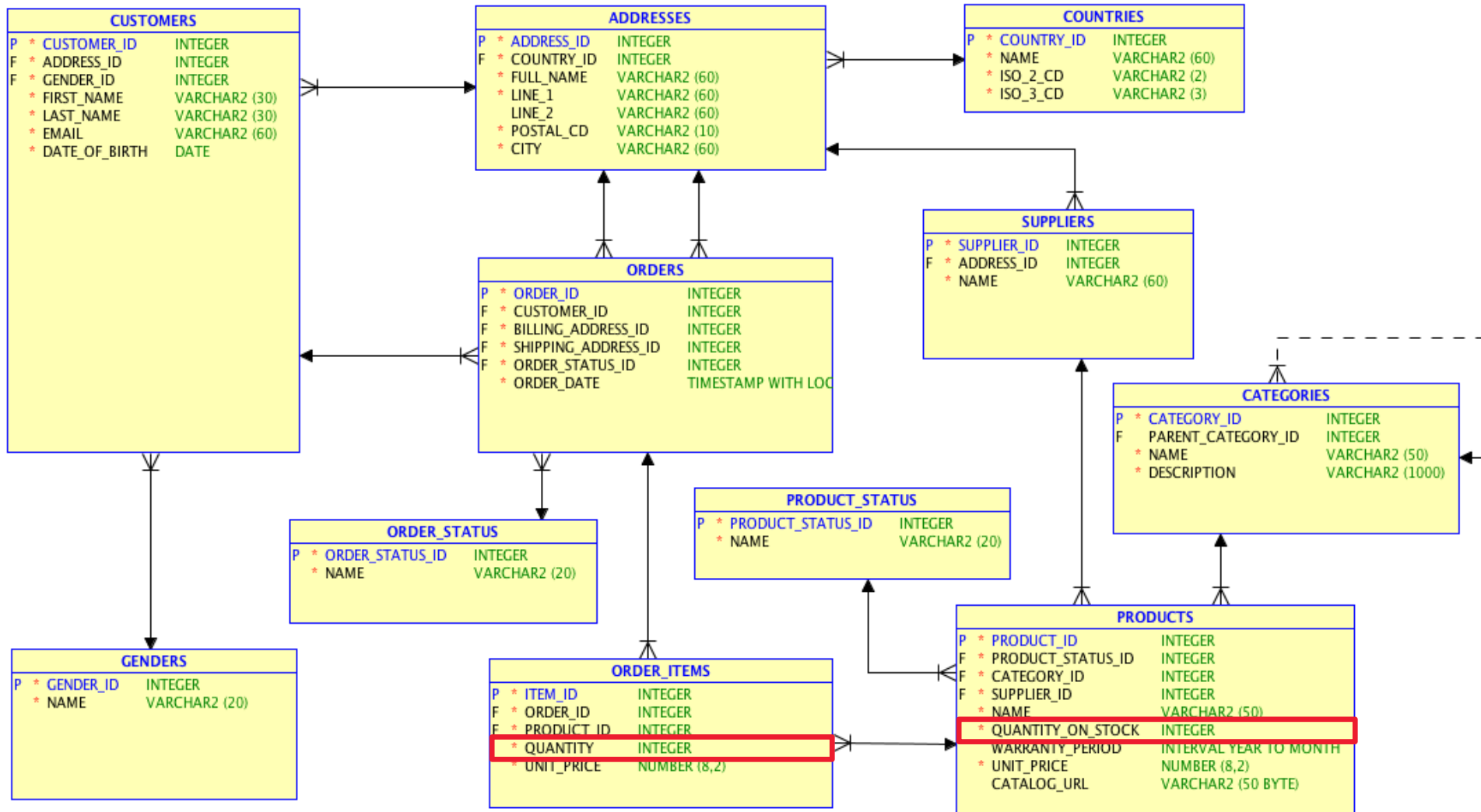
Annual Purchases: 24,815
Ending A/P Balance: 183

■ Order Entry – Use Cases

- Update the quantity in stock of all ordered products
 - When order status changes from "incomplete" to "complete"
 - When order status changes from "complete" to "cancelled"
 - Ensure that the quantity in stock is always correct (no lost updates or similar)
- Create a report for the 5 top-selling products for a year



Relational Model



■ Change Quantity in Stock – The SQL Way

Single Transaction

```
UPDATE ORDERS
  SET order_status = :p_value_for_complete
 WHERE order_id = :p_order_id;

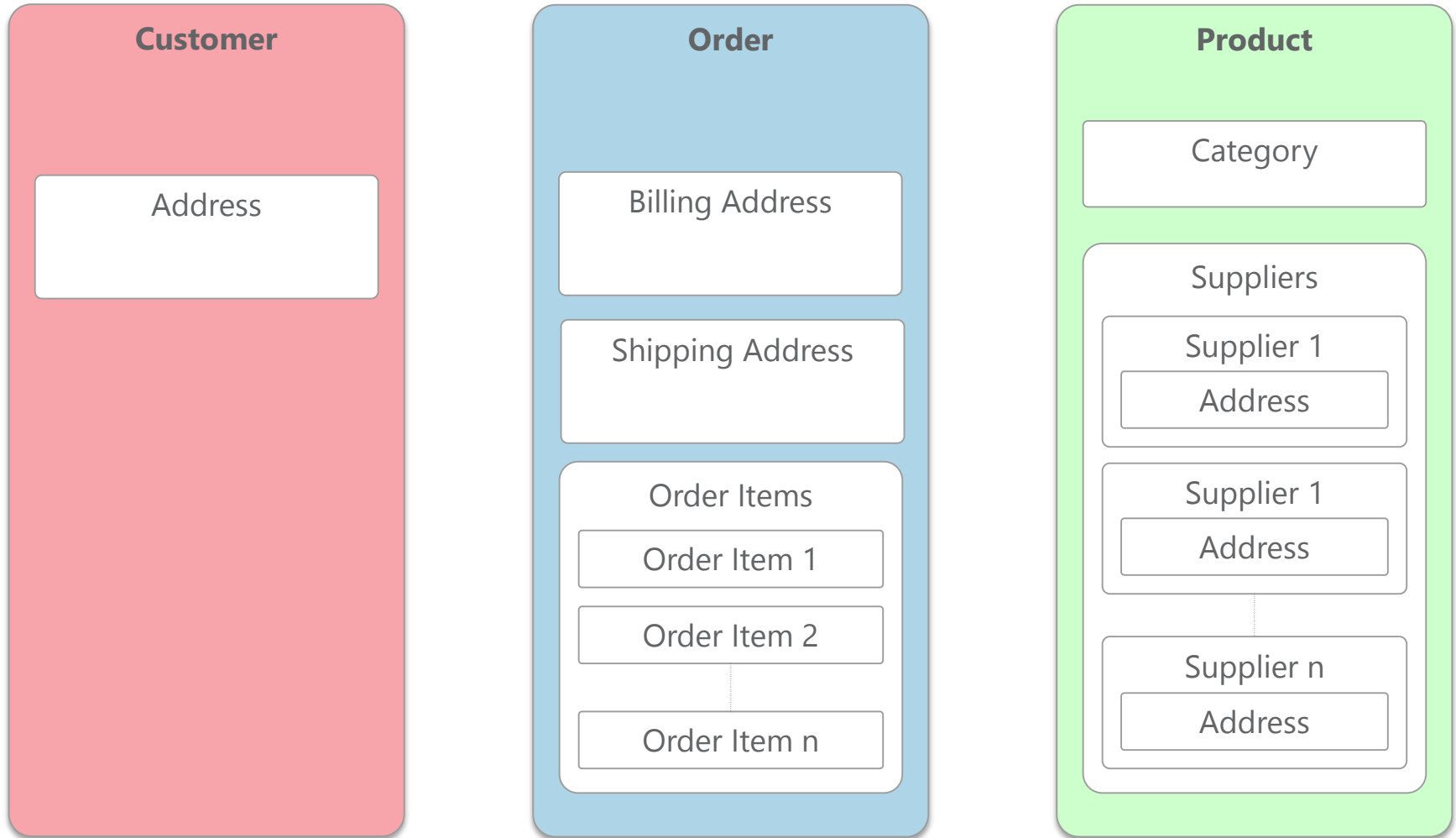
MERGE INTO PRODUCTS t
USING (SELECT product_id,
              SUM(quantity) AS quantity
       FROM order_items
       WHERE order_id = :p_order_id
       GROUP BY product_id) s
ON (t.product_id = s.product_id)
WHEN MATCHED THEN
  UPDATE SET t.quantity_on_stock =
            t.quantity_on_stock - s.quantity;

COMMIT;
```

■ 5 Top-Selling Products – The SQL Way

```
SELECT p.name AS product_name,  
       SUM(i.quantity * i.unit_price) AS sales_volume  
FROM   order_items i  
INNER JOIN orders o  
       ON o.order_id = i.order_id  
INNER JOIN products p  
       ON p.product_id = i.product_id  
WHERE  o.order_date > DATE '2013-01-01'  
       AND o.order_date <= DATE '2014-01-01'  
       AND o.order_status = :p_value_for_complete  
GROUP BY p.name  
ORDER BY 2 DESC  
FETCH FIRST 5 ROWS WITH TIES;
```

■ MongoDB Document Data Model (Aggregate Pattern)



■ Update Quantity in Stock – The MongoDB Way

Transaction 1

```
db.orders.update( { orderId: 1},  
                  { $set : { orderStatus: "COMPLETE" } },  
                  { multi: false } );
```

Read Operation

```
db.orders.find ( { orderId: 1} );
```

Transaction 2 .. n

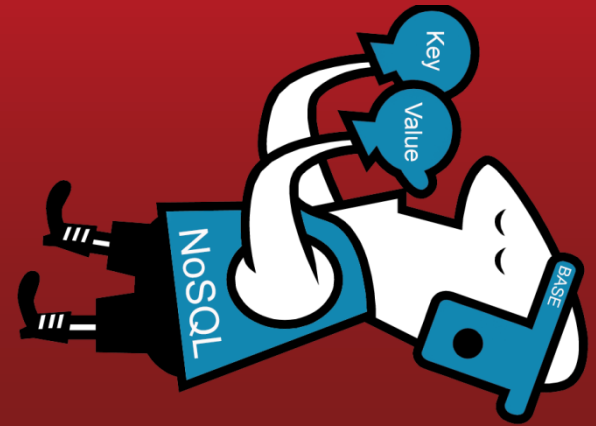
```
ForEach orderItems.item {  
    db.products.update( { productId : 101 },  
                        { $inc : { quantity: -10 } },  
                        { multi: false }  
    );  
}
```

■ 5 Top-Selling Products – The MongoDB Way

```
db.orders.aggregate([
  { $match : {
    orderStatus: "COMPLETE",
    orderDate: { $gt: ISODate("2014-01-01"),
                  $lt: ISODate("2014-04-01") }
  } },
  { $unwind : "$orderItems" },
  { $project : { _id: 0,
    productId: "$orderItems.productId",
    total : { $multiply : ["$orderItems.quantity",
                          "$orderItems.unitPrice"] }
  } },
  { $group : { _id: "$productId",
    total : { $sum : "$total" } } },
  { $sort : { total: -1 } },
  { $limit : 5 }
])
```

Order Entry

1 – 1



Round 3

Spotify

■ Spotify – Example

The screenshot shows the Spotify Premium desktop application interface. The top bar includes the Spotify Premium logo, a search bar, and the user's name 'Guido Schmutz' with a notification badge showing '4'.

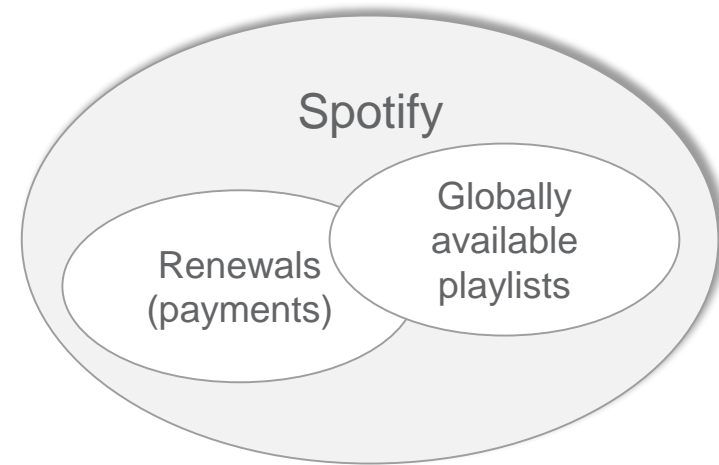
The main content area displays the 'Coldplay – Live 2012' playlist. The playlist cover art is a colorful, abstract design. Below the cover, the text 'by you' and 'Including artists: Coldplay' is visible. There are buttons for 'Share...', 'Start Radio', and 'Available Offline' (which is currently disabled). The playlist contains 15 tracks with a total duration of 1h 6min.

	Track	Artist	Time	Album
★	Mylo Xyloto – Live	Coldplay	0:58	Live 2012
★	Hurts Like Heaven – Live	Coldplay	4:16	Live 2012
★	In My Place – Live	Coldplay	3:55	Live 2012
★	Major Minus – Live	Coldplay	3:40	Live 2012
★	Yellow – Live	Coldplay	6:52	Live 2012
★	God Put a Smile Upon Your...	Coldplay	5:22	Live 2012
★	Princess of China – Live	Coldplay	3:49	Live 2012
★	Up in Flames – Live	Coldplay	3:18	Live 2012
★	Viva La Vida – Live	Coldplay	4:58	Live 2012
★	Charlie Brown – Live	Coldplay	5:01	Live 2012
★	Paradise – Live	Coldplay	5:32	Live 2012
★	Us Against the World – Live	Coldplay	3:52	Live 2012
★	Clocks – Live	Coldplay	4:45	Live 2012
★	Fix You – Live	Coldplay	5:01	Live 2012
★	Every Teardrop Is a Waterfall – Live	Coldplay	5:24	Live 2012

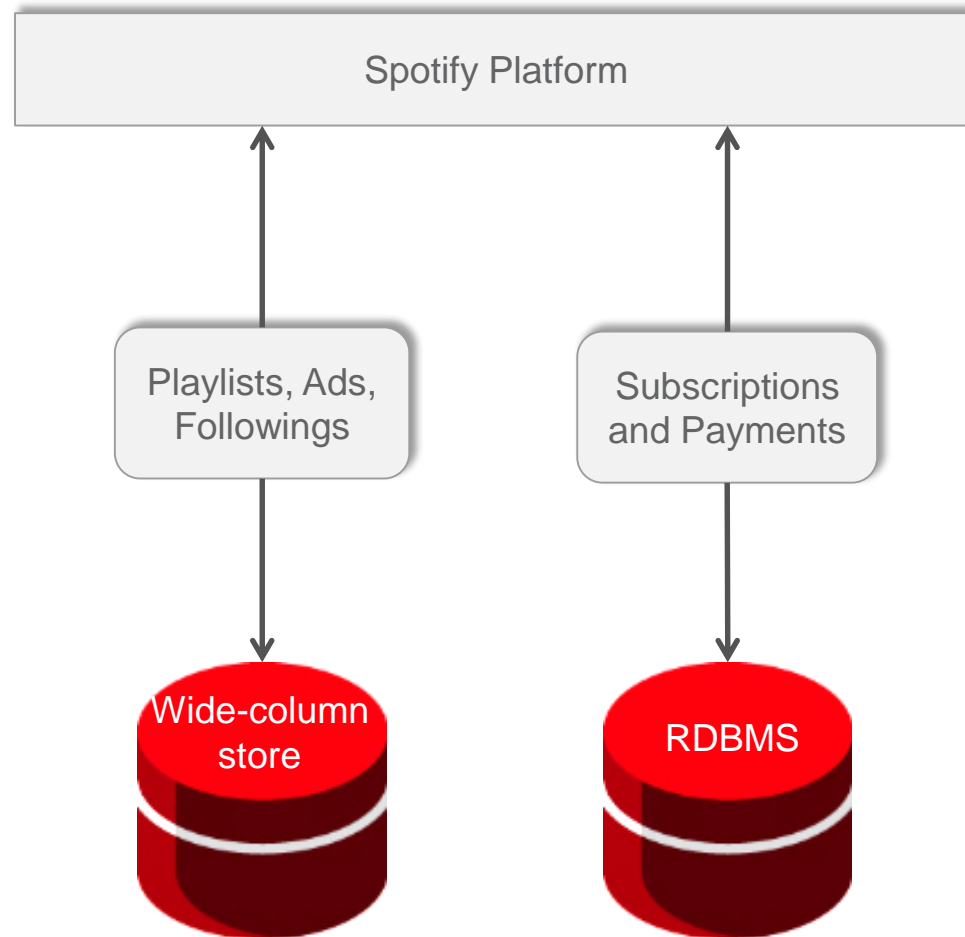
The bottom of the interface shows a playback bar with a progress slider at 0:04, a play button, and a volume control icon. The right sidebar displays the 'Activity' section, showing recent listening activity by users like Greg Stipkovich and Milow.

■ Spotify – Use Cases

- Playlist, Showing Ads, Following Artists ... are all uses cases which have to be highly available, and accessible worldwide
 - Needs to be distributed to be fast
 - Service should be available even if a partition happen (due to network failure/machine failure)
- First time subscription and subscription renewal must be absolutely consistent
 - Customer should only pay once!

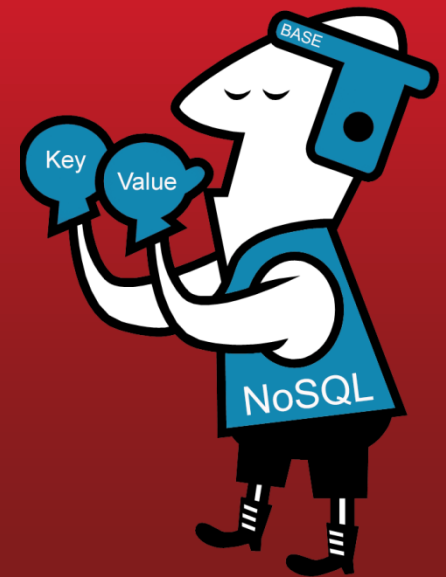


■ Polyglot Persistence – SQL And NoSQL





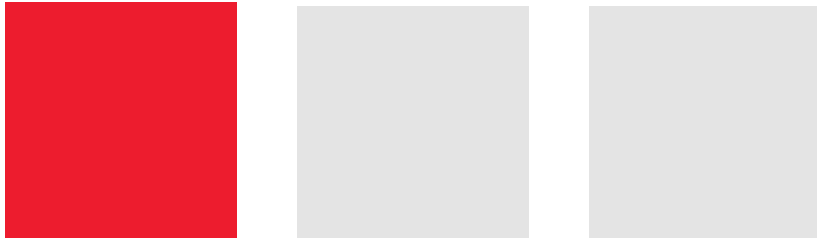
Spotify
2 – 2
Draw!



■ Agenda

1. Motivation
2. Overview of SQL and NoSQL Data Stores
3. Use Cases – Let's Get Ready to Rumble
4. Core Messages

■ Core Messages



- We will see a major consolidation in the NoSQL area
- SQL is and stays important
- Polyglot persistence will be part of every solution design in the near future
- Enterprise capabilities are required
 - Tooling (monitoring, backup & recovery, data security, ...)
 - Organization, skills
 - Opportunity for cloud based solutions

Questions and answers ...

Guido Schmutz
Technology Manager

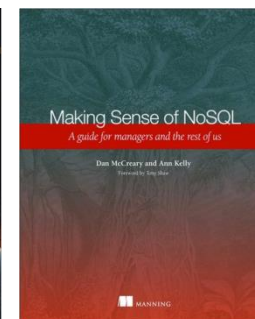
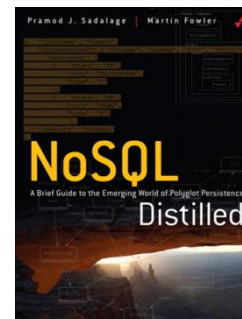
Philipp Salvisberg
Senior Principal Consultant



BASEL BERN BRUGES LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA

Further information ...

- <http://martinfowler.com/books/nosql.html>
- <http://www.manning.com/mccreary/>
- <http://highlyscalable.wordpress.com>
- <http://nosql-database.org>
- <http://db-engines.com/>



Highly Scalable Blog

Articles on Big Data, NoSQL, and Highly Scalable Software Engineering

NoSQL Data Modeling Techniques

Posted on March 1, 2012

NoSQL databases are often compared to various non-functional attributes, such as scalability, performance, and consistency. This aspect of NoSQL is well studied both in practice and theory because specific non-functional properties are often the main justification for NoSQL usage and fundamental results on distributed systems like the CAP theorem apply well to NoSQL systems. At the same time, NoSQL data modeling is not as well studied and lacks the systematic theory found in relational databases. In this article I provide a short comparison of NoSQL system families from the data modeling point of view and digest several common modeling techniques.

I would like to thank Daniel Kishinevsky who reviewed the article and cleaned up the grammar.

To explore data modeling techniques, we have to start with a more or less systematic view of NoSQL data models that preferably reveals trends and interconnections. The following figure depicts imaginary "evolution" of the major NoSQL system families, namely, Key-Value stores, MapReduce-style databases, Document databases, Full Text Search Engines, and Graph databases.

