Hitachi Consulting
We make it happen. Better.

NoSQL Data Stores

with Microsoft Azure

Khalid M. Salama, Ph.D.
Business Insights & Analytics
Hitachi Consulting UK

 |

Outline

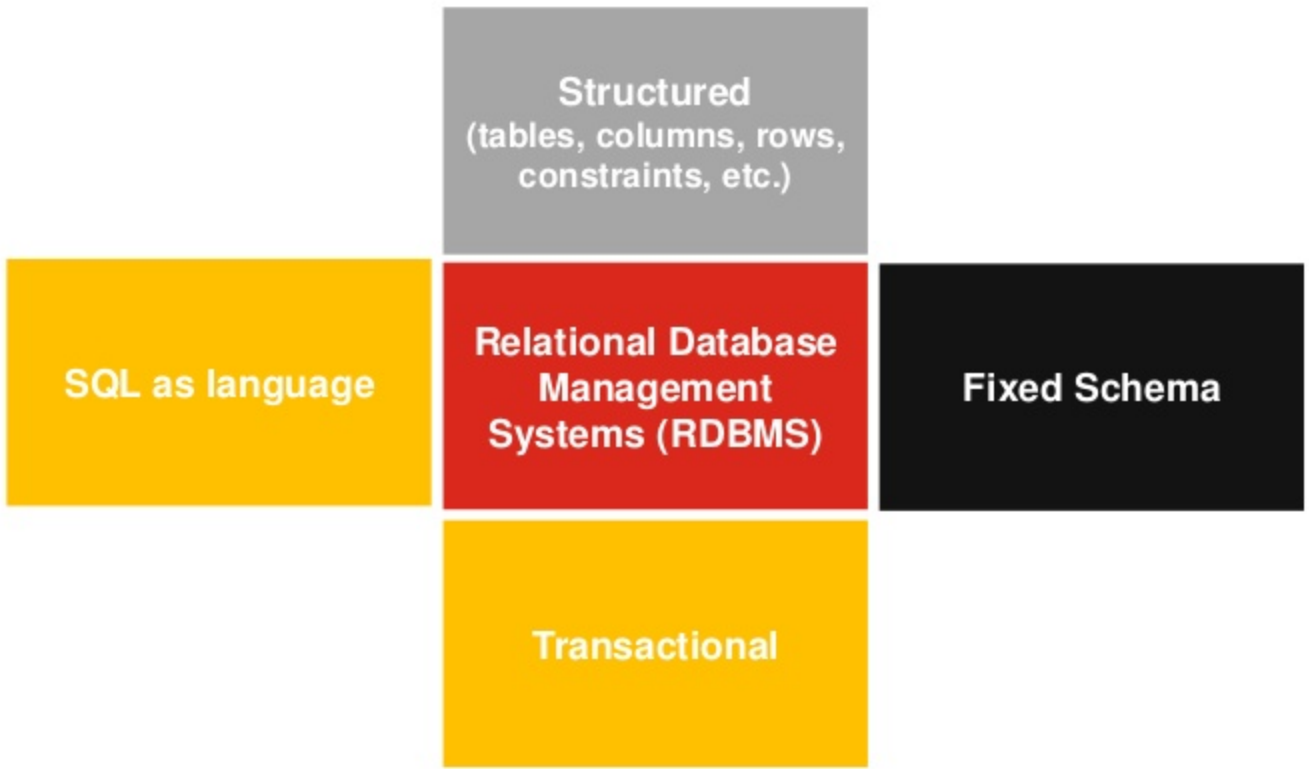
Hitachi Consulting
We make it happen. Better.

- What is NoSQL, and why?
- NoSQL Data Stores, tools & Technologies
- Solution Architecture Patterns
- Introducing Azure Redis Cache
- Introducing Azure Table Storage
- Introducing Azure DocumentDB
- Introducing HBase on Azure HDInsight
- How to Get Started with NoSQL
- Appendix: how to play with neo4j

Fundamentals

What is SQL?

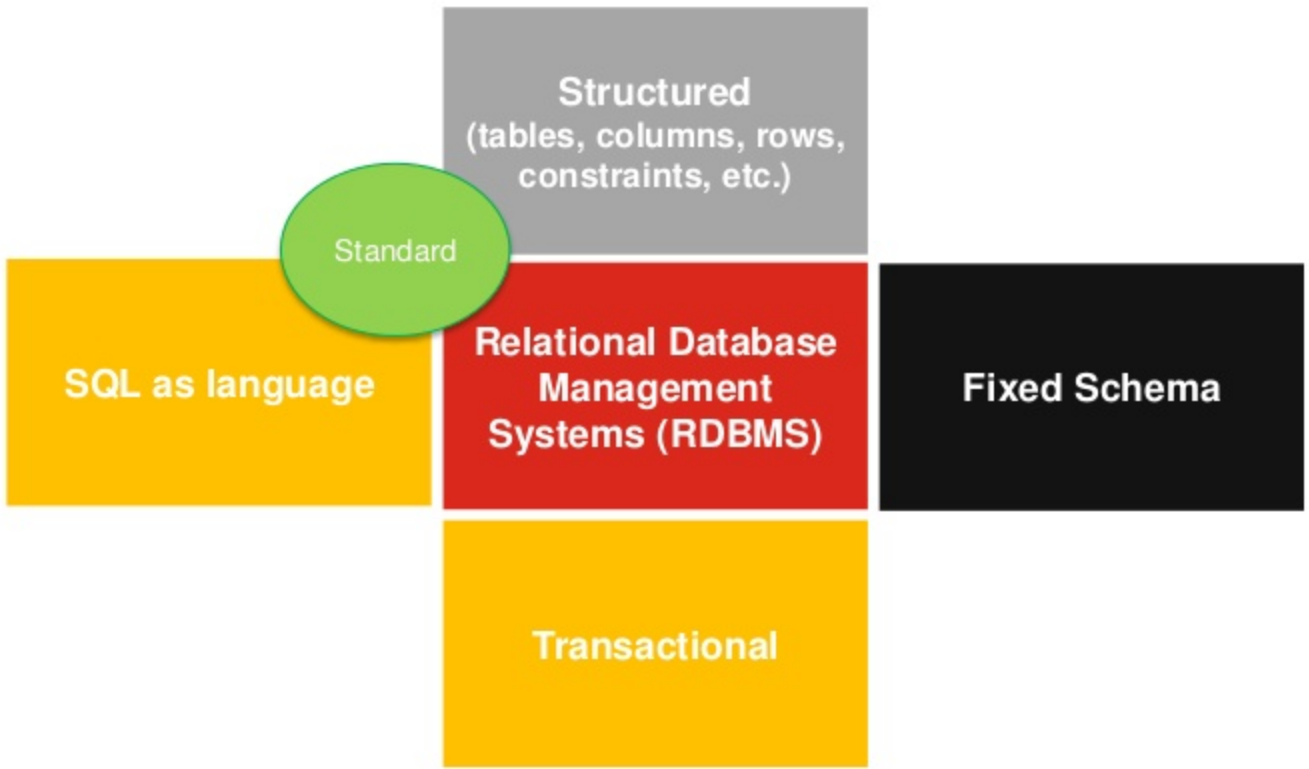
Hitachi Consulting
We make it happen. Better.



 |

What is SQL?

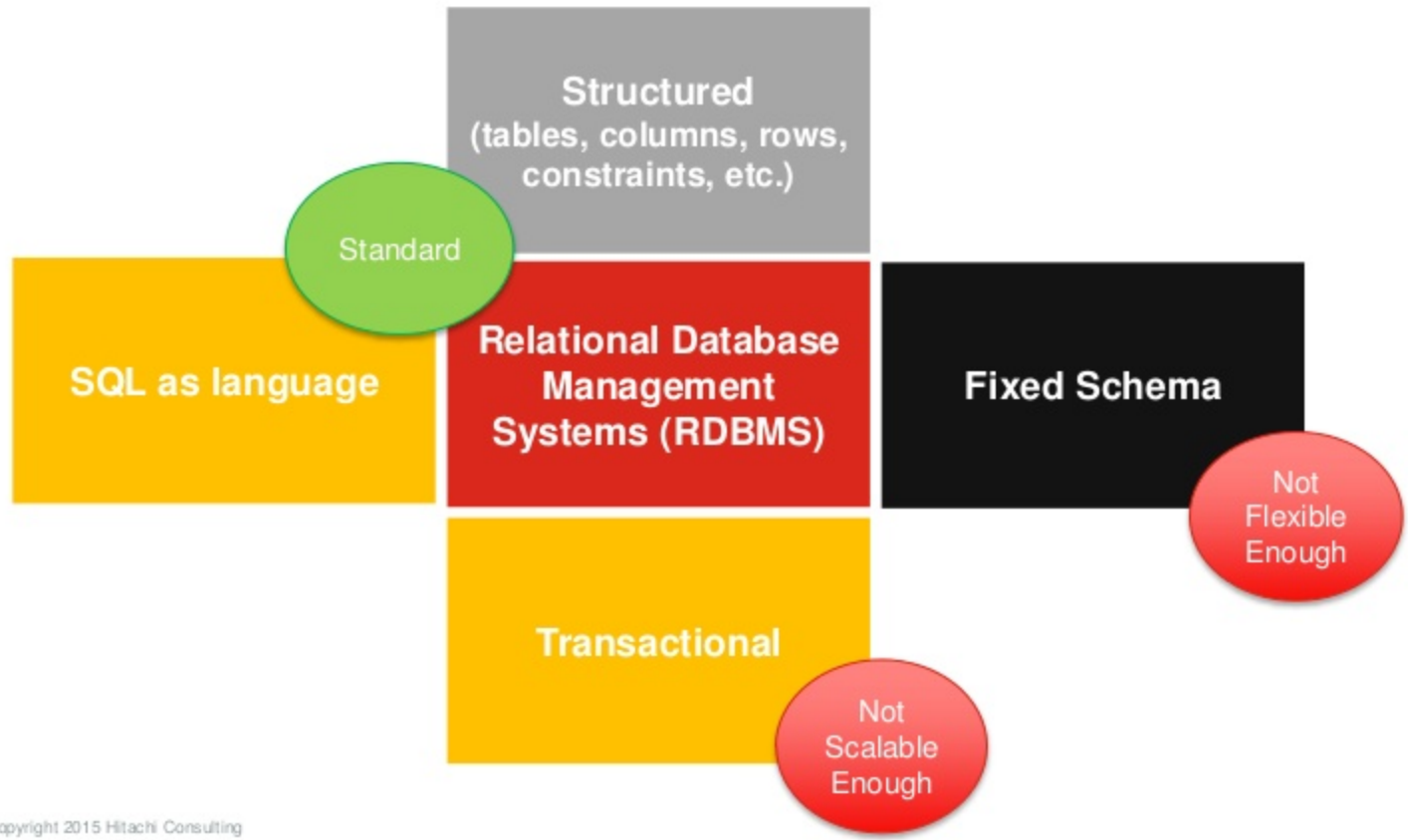
Hitachi Consulting
We make it happen. Better.



 |

What is SQL?

Hitachi Consulting
We make it happen. Better.



 |

Hitachi Consulting
We make it happen. Better.

What is NoSQL?

key attributes..

Non-relational

 |

Hitachi Consulting
We make it happen. Better.

What is NoSQL?

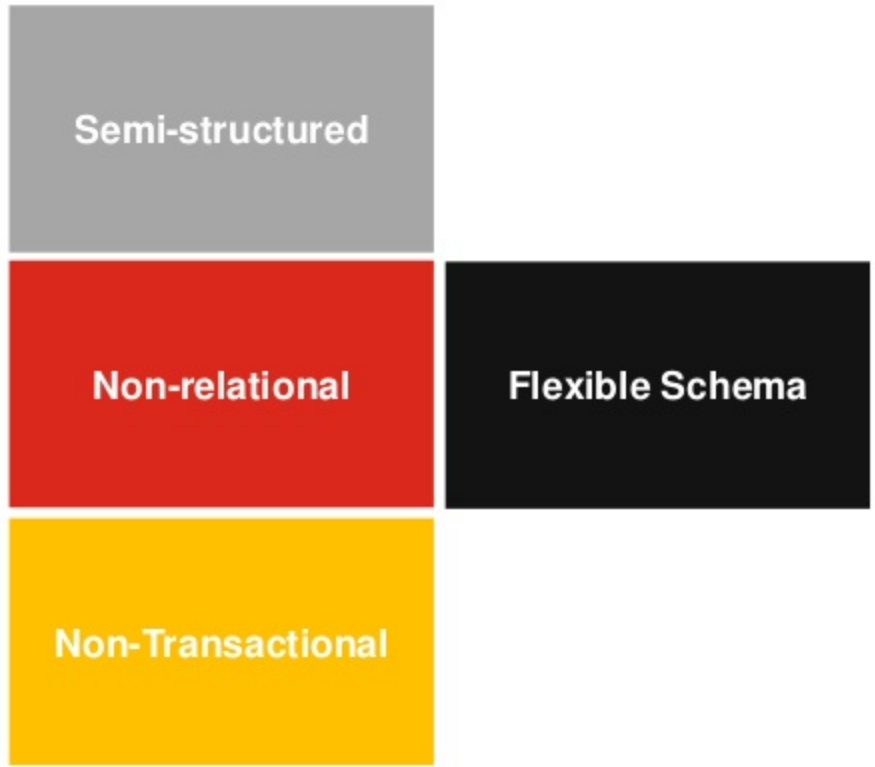
key attributes..

Non-relational

Non-Transactional

What is NoSQL?

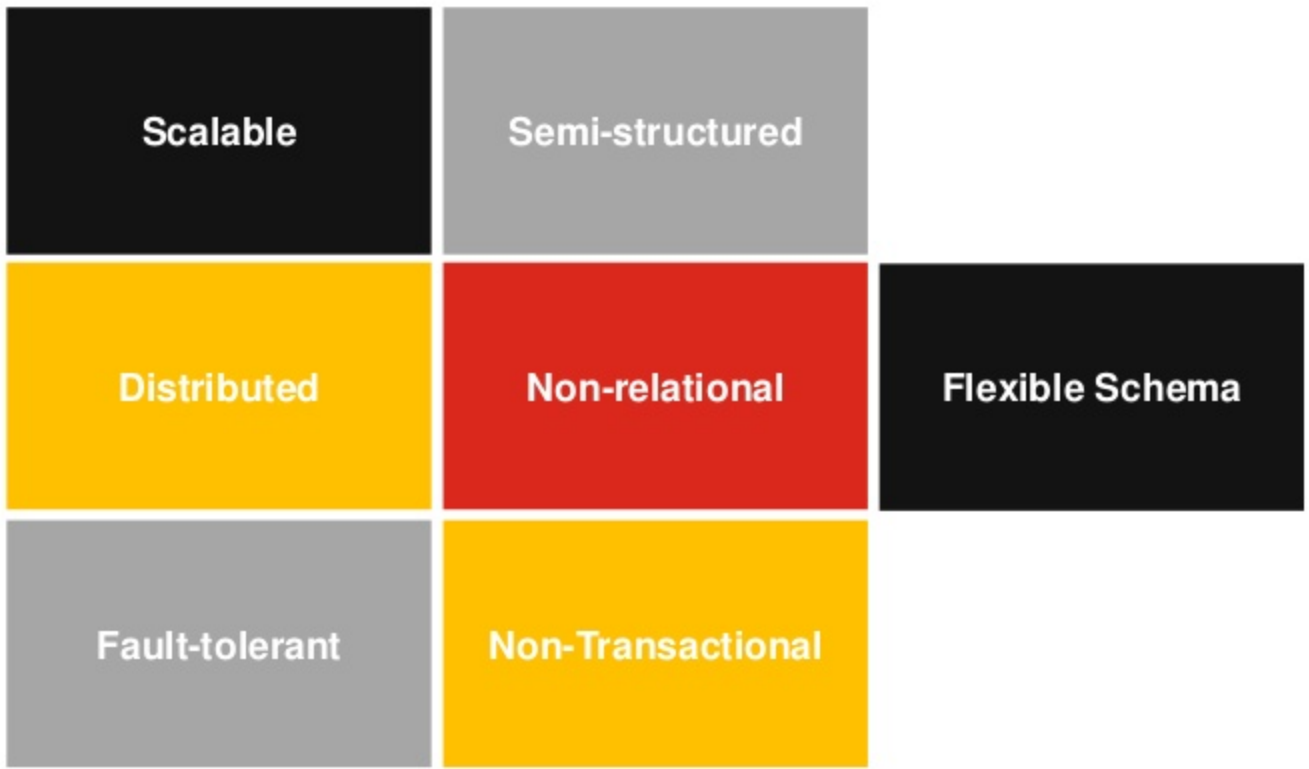
key attributes..



 |

What is NoSQL?

key attributes..



What is NoSQL?

key attributes..

Scalable	Semi-structured	Complement RDBMS
Distributed	Non-relational	Flexible Schema
Fault-tolerant	Non-Transactional	Random, real-time read/write access

Hitachi Consulting
We make it happen. Better.

Why NoSQL?

It is all about Big Data…

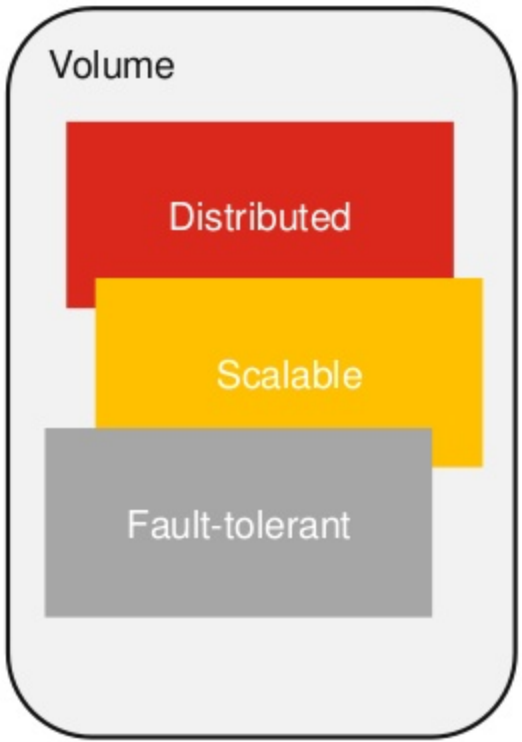
NoSQL data stores help overcoming Big Data challenges in real-time operational systems

Hitachi Consulting
We make it happen. Better.

Why NoSQL?

It is all about Big Data…

NoSQL data stores help overcoming Big Data challenges in real-time operational systems



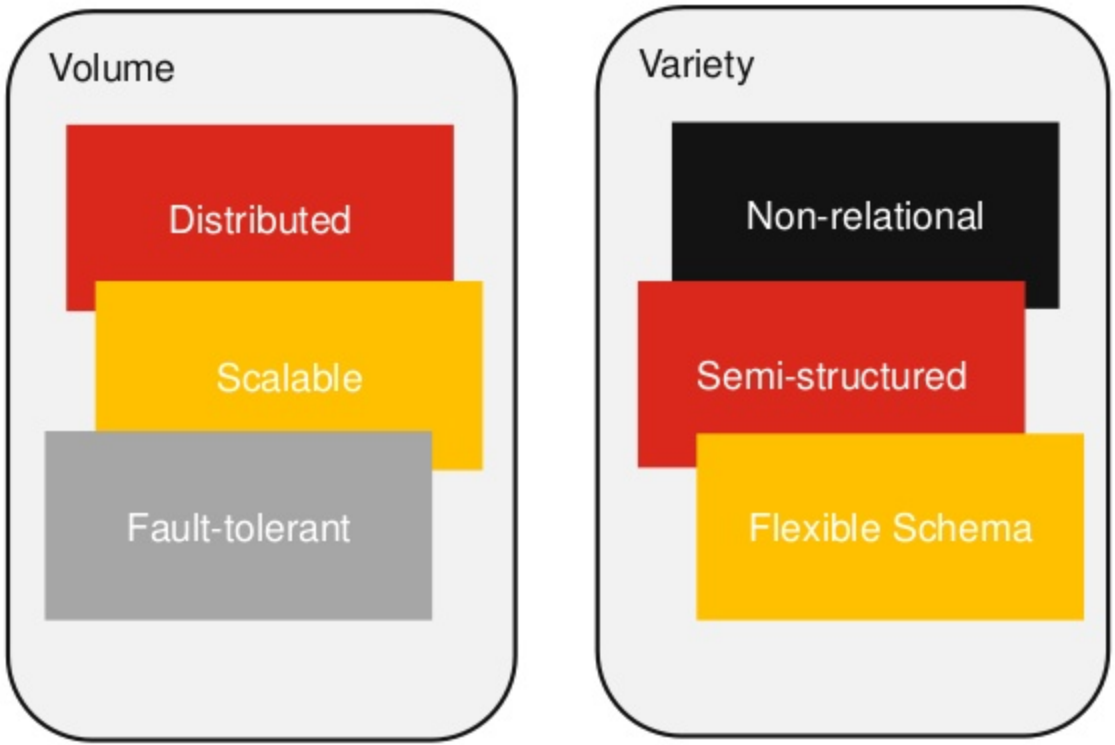
 |

Hitachi Consulting
We make it happen. Better.

Why NoSQL?

It is all about Big Data…

NoSQL data stores help overcoming Big Data challenges in real-time operational systems



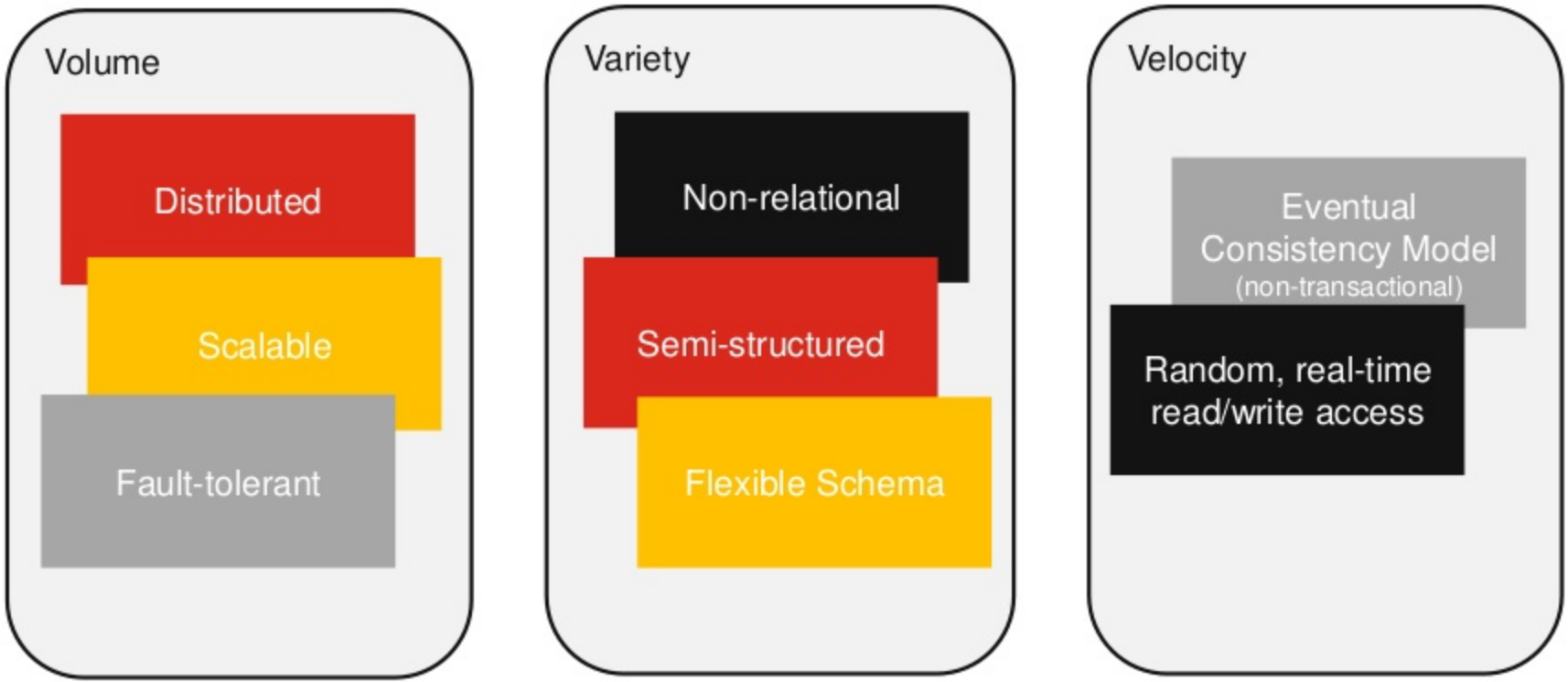
 |

Hitachi Consulting
We make it happen. Better.

Why NoSQL?

It is all about Big Data…

NoSQL data stores help overcoming Big Data challenges in real-time operational systems



Hitachi Consulting
We make it happen. Better.

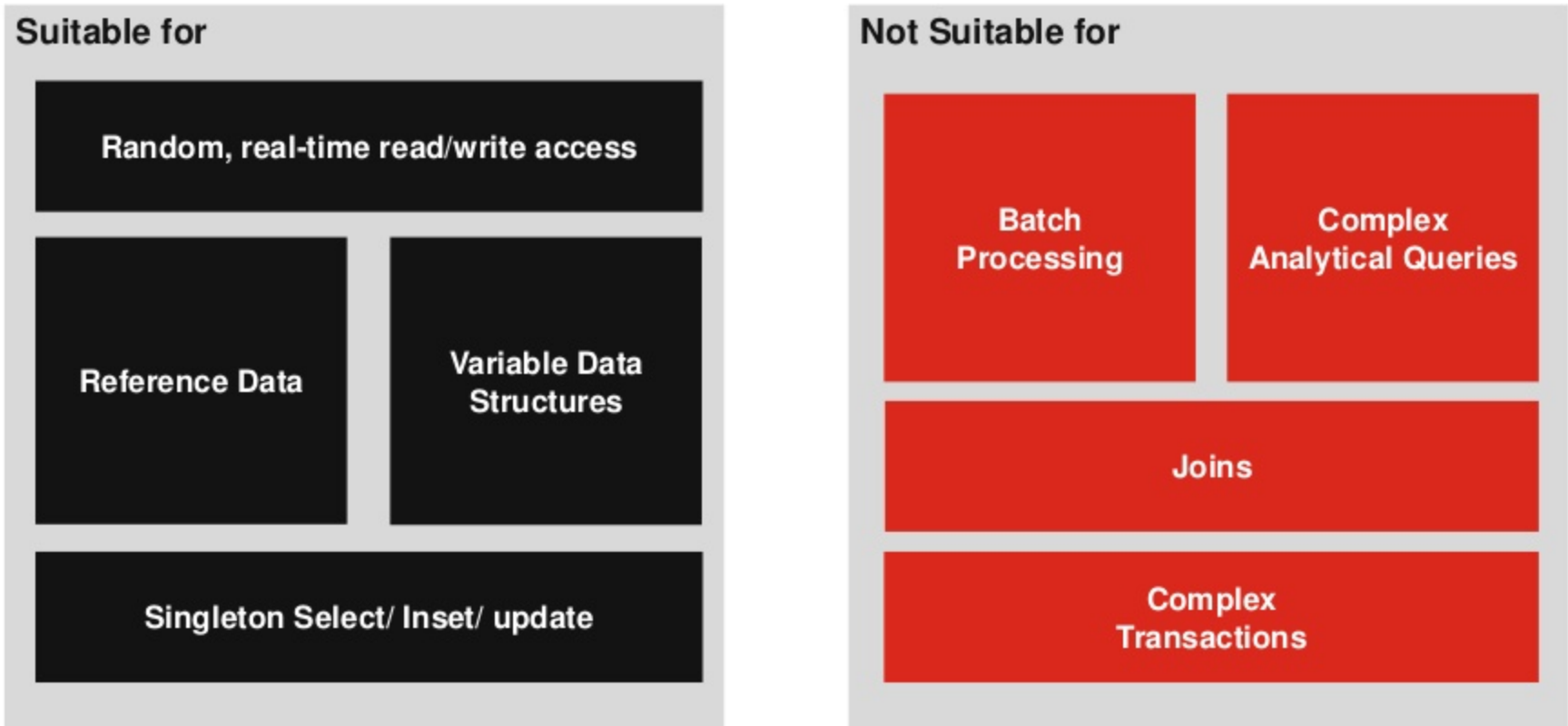
NoSQL Usage Patterns

suitability…

Hitachi Consulting
We make it happen. Better.

NoSQL Usage Patterns

suitability…



CAP Theorem

 |

Hitachi Consulting
We make it happen. Better.

NoSQL & CAP Theorem

The trade-off…

- In order to handle large volume of data processing efficiently, we need to **scale out**, i.e. **partition** the data and **distribute** the computation

- Now we face a trade-off between **C**onsistency, **A**vailability, and **P**artition Tolerance
  - **Consistency**: Data is in a consistent state across all the nodes.
    That is, **all** the reads would get you the same, **most recent** write.
  - **Availability**: **Every** request to the system **gets a response** (i.e., executed) on success/failure.
    That is, system **responsiveness** (latency).
  - **Partition Tolerance**: The system **continuous to work** despite of message loss or **partition (node) failure**. That is, the system can sustain partial network failures.

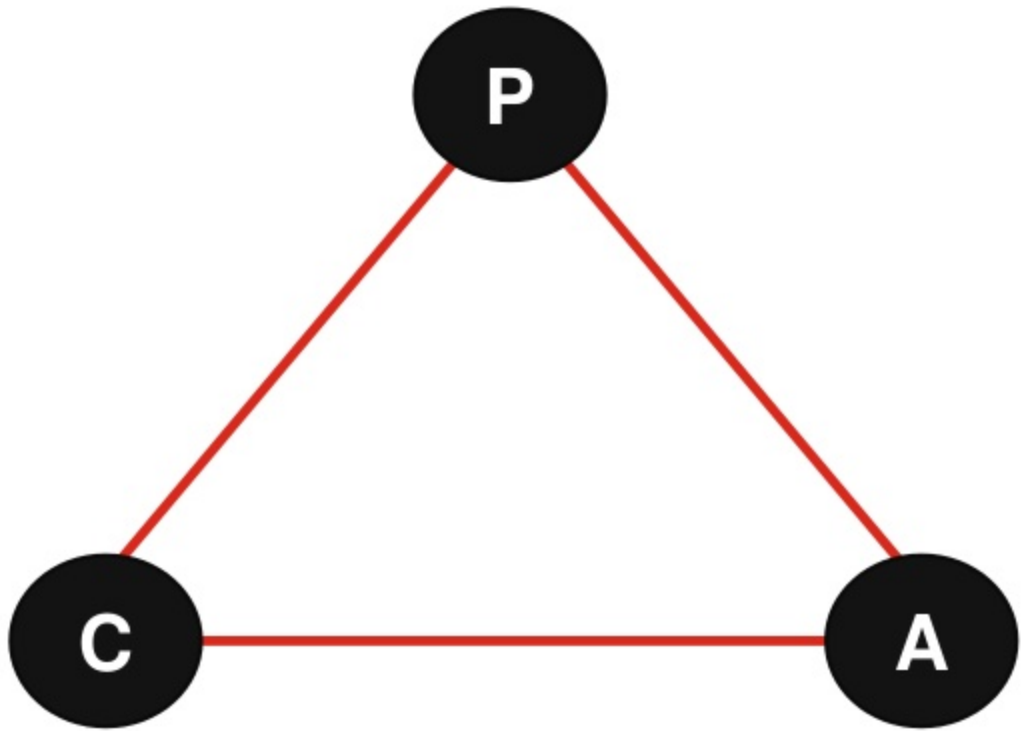
- **CAP Theorem**: only two out of three properties can be satisfied in a distributed data system. In facet, it is **consistency vs availability**, wrt partition tolerance!

Hitachi Consulting
We make it happen. Better.

NoSQL & CAP Theorem

The trade-off…

Continue working if a partition is not reachable by the system?



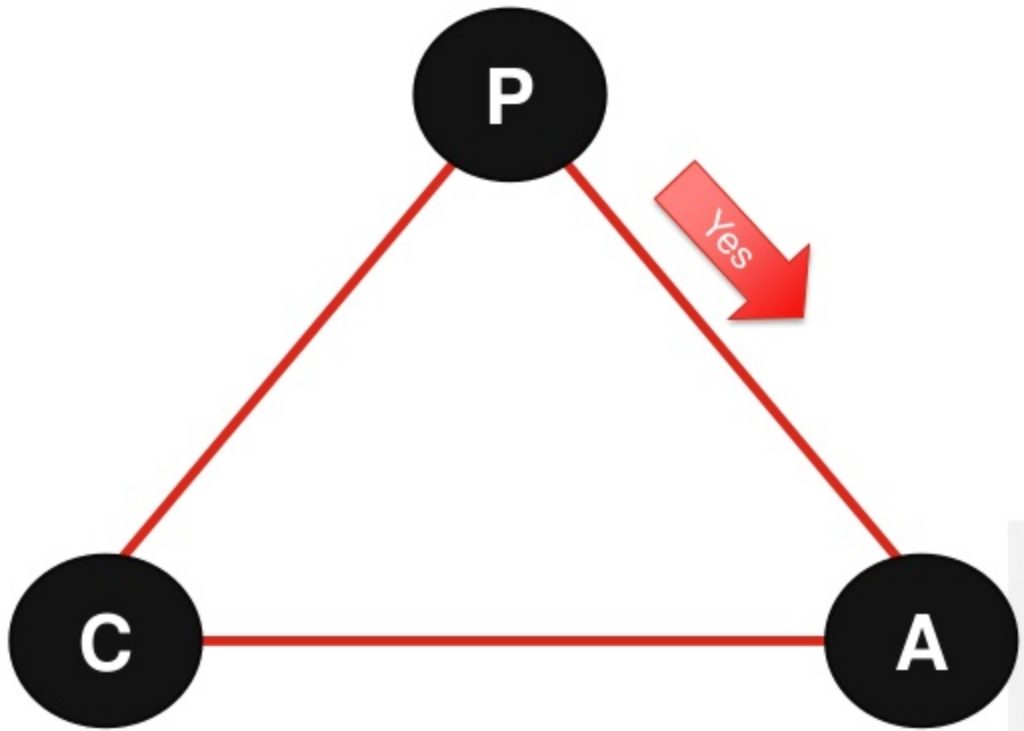
 |

Hitachi Consulting
We make it happen. Better.

NoSQL & CAP Theorem

The trade-off…

Continue working if a partition is not reachable by the system?



Big Data Systems

- BASE Mode – Eventually Consistency
- Remains available (operational & responsive)
- partition tolerant, i.e., **sacrifices consistency**

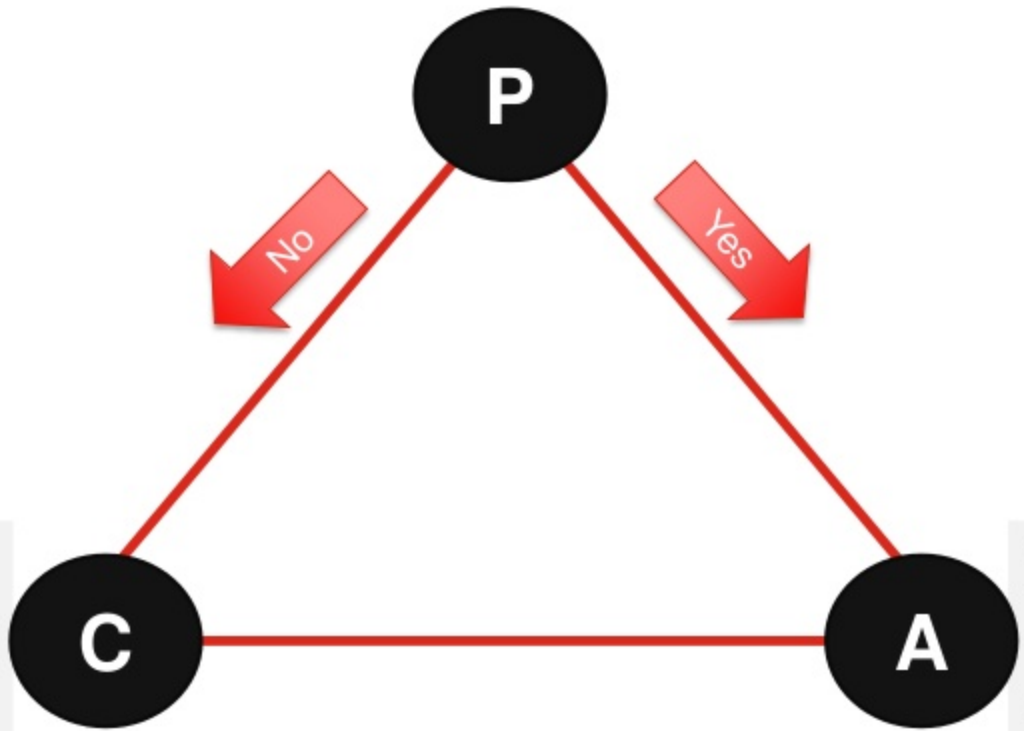
 |

Hitachi Consulting
We make it happen. Better.

NoSQL & CAP Theorem

The trade-off…

Continue working if a partition is not reachable by the system?



Transactional RDBMS

- ACID Mode – Strong Consistency
- Commits are atomic across the entre system
- Not partition tolerant, i.e., **sacrifices availability**

Big Data Systems

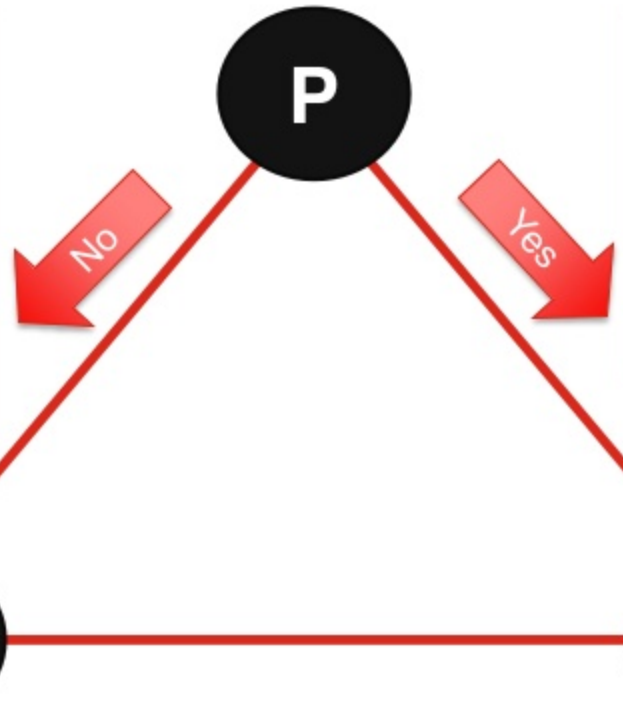
- BASE Mode – Eventually Consistency
- Remains available (operational & responsive)
- partition tolerant, i.e., **sacrifices consistency**

Hitachi Consulting
We make it happen. Better.

NoSQL & CAP Theorem

The trade-off…

Continue working if a partition is not reachable by the system?



ACID

- **Atomic**: Everything in a transaction succeeds or the entire transaction is rolled back.
- **Consistent**: A transaction cannot leave the database in an inconsistent state.
- **Isolated**: Transactions cannot interfere with each other.
- **Durable**: Completed transactions persist, even when servers restart etc.

BASE

- **Basic Availability**: data is sharded and replicated, and consistency is compromised for availability
- **Soft-state**: Allow data to be inconsistent and try to maintain consistency later.
- **Eventual consistency**: Consistency is maintained later.

Transactional RDBMS

- ACID Mode – Strong Consistency
- Commits are atomic across the entre system
- Not partition tolerant, i.e., **sacrifices availability**

Big Data Systems

- BASE Mode – Eventually Consistency
- Remains available (operational & responsive)
- partition tolerant, i.e., **sacrifices consistency**

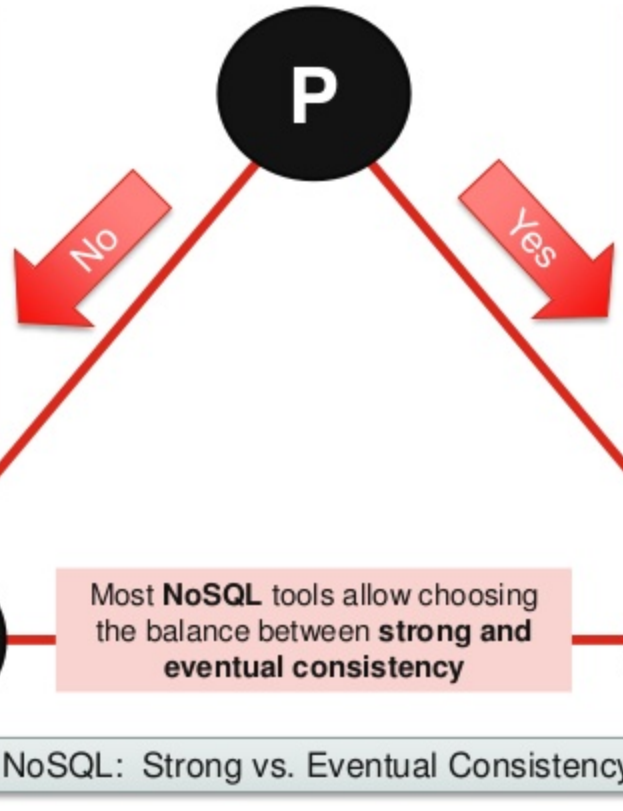
 |

Hitachi Consulting
We make it happen. Better.

NoSQL & CAP Theorem

The trade-off…

Continue working if a partition is not reachable by the system?



ACID

- **Atomic**: Everything in a transaction succeeds or the entire transaction is rolled back.
- **Consistent**: A transaction cannot leave the database in an inconsistent state.
- **Isolated**: Transactions cannot interfere with each other.
- **Durable**: Completed transactions persist, even when servers restart etc.

BASE

- **Basic Availability**: data is sharded and replicated, and consistency is compromised for availability
- **Soft-state**: Allow data to be inconsistent and try to maintain consistency later.
- **Eventual consistency**: Consistency is maintained later.

Transactional RDBMS

- ACID Mode – Strong Consistency
- Commits are atomic across the entre system
- Not partition tolerant, i.e., **sacrifices availability**

Big Data Systems

- BASE Mode – Eventually Consistency
- Remains available (operational & responsive)
- partition tolerant, i.e., **sacrifices consistency**

 |

NoSQL Data Stores

Hitachi Consulting
We make it happen. Better.

NoSQL Data Stores

Categories and Breads

 |

Hitachi Consulting
We make it happen. Better.

NoSQL – Key/Value Stores

The agile…

- Based on a **Hash Table** (Dictionary);
- **Unique key** and a pointer to a data (Value)
- Data Item can be **anything** (Blob).
- A key can only have one value.
- Simple APIs (GET, PUT, DELETE).
- Yet complex implementation (**no querying language**).
- Optimized for singleton operations.
- Example tools:
  - Amazon Dynamo (pioneer)
  - Apache Accumulo
  - Riak KV
  - Redis (memory cache)
  - Memcached (memory cache)

	Key	Value
Image name →	image-12345.jpg	Binary image file
Web page URL →	http://www.example.com/my-web-page.html	HTML of a web page
File path name →	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash →	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call →	view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SQL query →	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

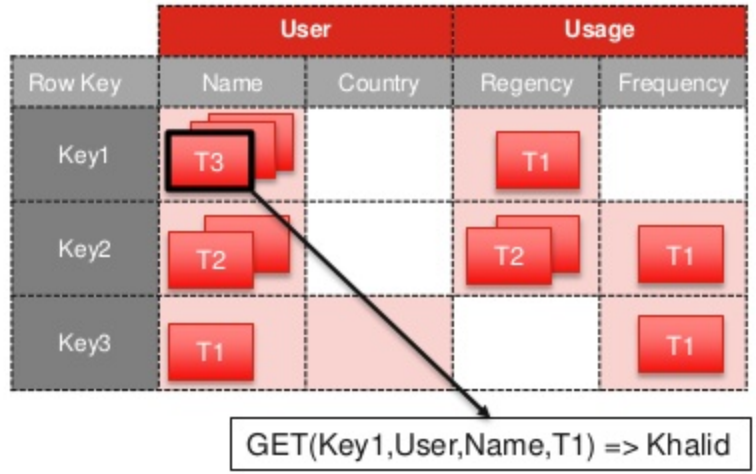
 |

Hitachi Consulting
We make it happen. Better.

NoSQL – Column Family Stores

The big…

- **Extensible Record** stores - **Wide Column** Store
- **Tabular** Data Structure
- Columns can be **extended** (organized in column family)
- Resulting in **Sparse Matrix**
- Millions of rows and Hundred of thousands of Columns
- APIs (GET, PUT, SCAN, DELETE, COUNT).
- A data item is accessed via (row key, column, timestamp)
- Example tools:
  - Google BigTable (pioneer)
  - Apache HBase
  - Apache Cassandra



Real-world Scenarios

- Website usage information in Google Analytics
- Geographic information in Google Maps
- Social Media Apps (Twitter, Facebook, etc..)
- Search Engine Web Crawling Results

 |

NoSQL – Document Stores

The popular…

Hitachi Consulting
We make it happen. Better.

Key	Document
John	{ "name": "John", "country": "Canada", "age": 43, "lastUse": "March 4, 2014" }
Eva	{ "name": "Eva", "country": "Germany", "age": 25 }
Lou	{ "name": "Lou", "country": "Australia", "age": 51, "firstUse": "May 8, 2013" }
Total	{ "docCount": 3, "last": "May 1, 2014" }

- Same as Key/Value store, where the value is a **document** with a specific format. E.g. **JSON, BSON, XML**, etc.
- A document is retrieved by a **key,** or
- Since the format of the documents is known (e.g. JSON), a **query language** can be used to retrieve documents.
- E.g. retrieve all the document that has attribute “country” and its value equals “Algeria”.
- Attributes in side the document are **indexed**. Documents are often **versioned**.
- Supports wide range of **OLTP/Real-time** applications
- Example Tools
  - IBM lotus Nots (pioneer)
  - MongoDB
  - CouchDB - Apache Couchbase
  - Microsoft Azure DocumentDB

Real-world Scenarios

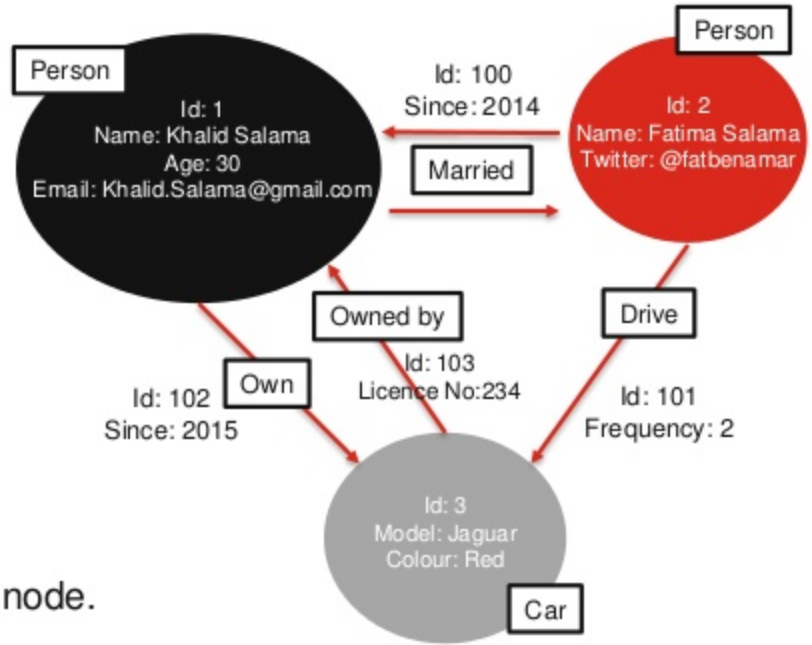
- Content Management and Personalization
- User Data Management
- Reference Data Management
- Lookups for Real-time Operational Intelligence

Hitachi Consulting
We make it happen. Better.

NoSQL – Graph Stores

The clever…

- Represent data in **graphical structures**: Nodes and Edges.
- **Nodes** represent **entities**, **Edges** represent **relationships** between entities.
- Relationships are **directed**, semantics of the direction is up to the application. E.g. "Married" is reflexive, "Owns" is not.
- Each Node/Edge has a set of **Key/Value properties**
- Each Node/Edge has a **label** (type of entity/relationship)
- **Optimized** to process **graph-related queries**, E.g. the number of steps needed to get from one node to another node.
- Example Tools
  - Neo4j
  - OrientDB
  - Titan
  - Apache Giraph
  - Microsoft Graph Engine (Trinity)



Real-world Scenarios

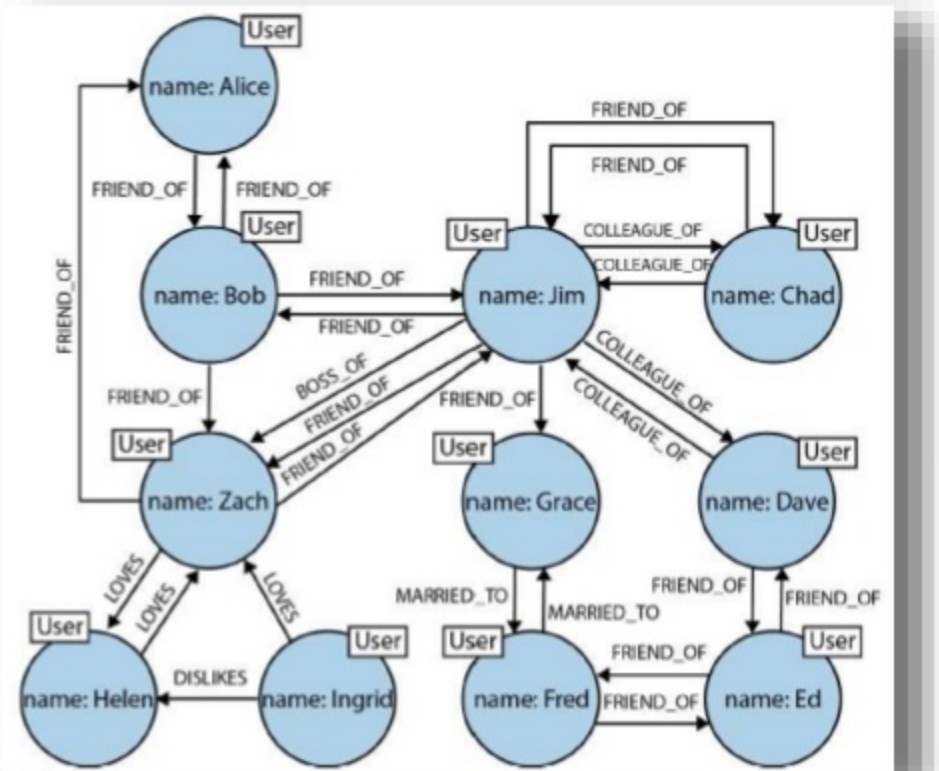
- Social Networks
- Network and IT Operations
- Fraud Detection
- Digital Assets Management

 |

NoSQL – Graph Stores

The clever…

Hitachi Consulting
We make it happen. Better.



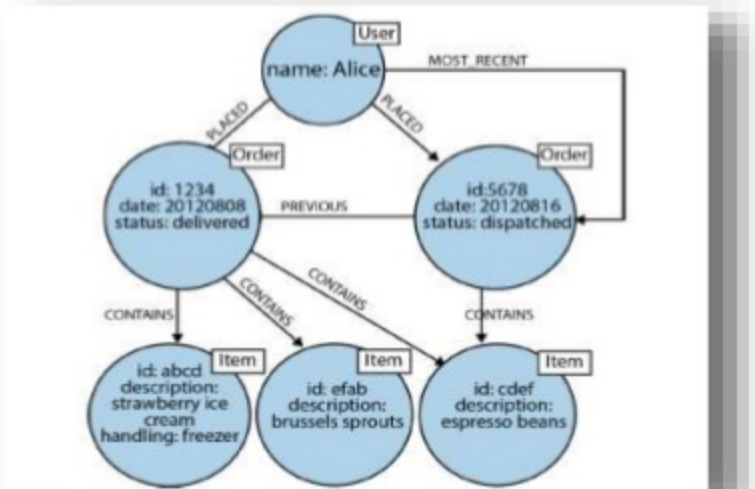


Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

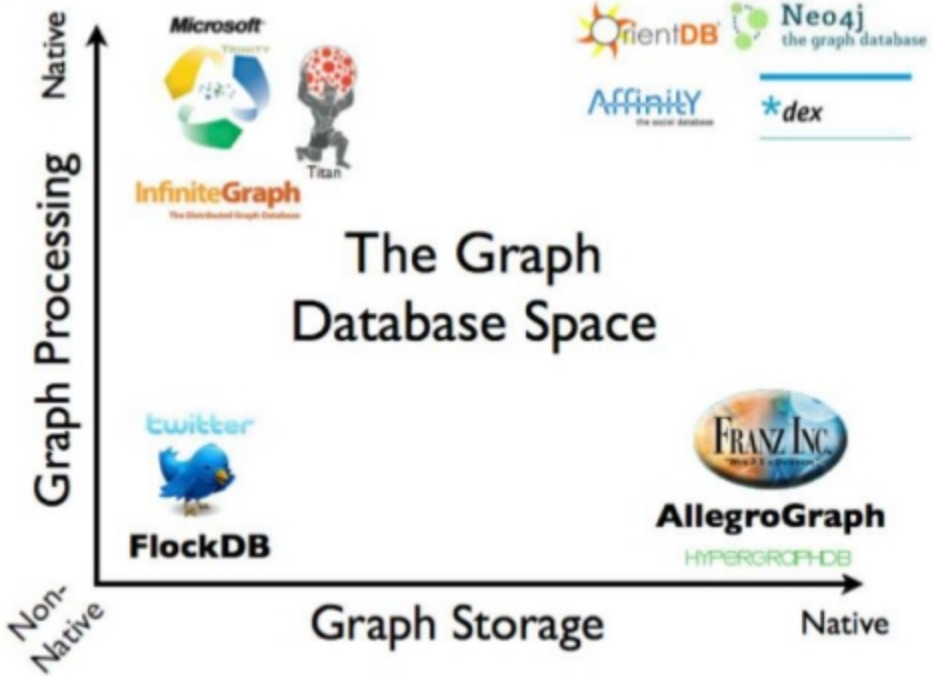
O'REILLY - GRAPH DATABASES

Hitachi Consulting
We make it happen. Better.

NoSQL – Graph Stores

The clever…

index-free adjacency; connected nodes physically "point" to each other in the database



Any database *behaves* like a graphDB; exposes a graph data model through CRUD operations

Graphs are serialized in any database; Relational, Document, or objectDBs

Storage is designed and optimize to store, process, and query graph data structures

Hitachi Consulting
We make it happen. Better.

NoSQL – Memory Cache Stores

The fast….

- Usually a **simple data structure** (Key/Value)
- The value can be a simple data type (String) or complex data objects.
- A **memory-based** store, usually, with **persistence** option.
- Used to optimize “hot” data access.
- Manages distributed web application session states.
- Can help to survive service downtime.
- Evolves **read/write strategies** (write-through, write-behind, etc.), **data expiration**, and **conflict resolution** techniques.
- Example Tools:
  - Memcached (pioneer)
  - Redis

 |

Hitachi Consulting
We make it happen. Better.

NoSQL – Other Data Stores

Storage for other data structures

Object-oriented databases

- Developed in the 1980s motivated by the common use of **object-oriented programming.**
- Simply store the objects in a database in a way that corresponds to their representation in the application, without the need of conversion or decomposition.
- The relationships between the objects, e.g. inheritance should also be maintained in the database.
- **Examples**: Caché, Db4o, Versant Object Database

Resource Description Framework (RDF) Data stores – Triple Data Stores

- Originally developed for describing metadata of IT resources.
- Used in connection with the sematic web, and other applications.
- The RDF model represents information as **triples** in the form of **subject-predicate-object**.
- **Examples**: MarkLogic, Virtuoso, Jena, Sesame, Algebraix

Multi-dimensional databases, Multi-value databases, Time-series databases, Event Sourcing databases, Multi-modal data stores, etc.

NoSQL Tools & Technologies

Hitachi Consulting
We make it happen. Better.

	Typical Usage	Lineage	Tools
Key/Value	▪ Flexible data structure ▪ Dictionary/ lookup ▪ Value can be anything	Amazon's Dynamo	▪ Apache Accumulo ▪ Riak KV ▪ *Redis*
Column Family (Tabular/wide column)	▪ Column-oriented access ▪ BigData with real-time read/write random access ▪ Extensible	Google's BigTable	▪ Apache Cassandra ▪ Apache HBase ▪ HBase on Microsoft HDInsight
Document	▪ Query-able data ▪ Objects (complex structure) in JSON, BSON, XML, etc. ▪ CRUD apps	IBM's Lotus Notes	▪ MongoDB ▪ CouchDB ▪ Apache CouchBase ▪ Microsoft Azure DocumentDB
Graph	▪ Social networks ▪ Fraud detection ▪ Relationship-heavy data	Graph Theory	▪ Neo4j ▪ OrientDB ▪ Titan ▪ Apache Giraph ▪ Microsoft Graph Engine
Memory Cache	▪ Non-durable data ▪ Fast access	LiveJournal's Memcached	▪ Redis ▪ Microsoft Azure Redis ▪ Microsoft Azure Memcached (Preview)

http://db-engines.com/en/ranking

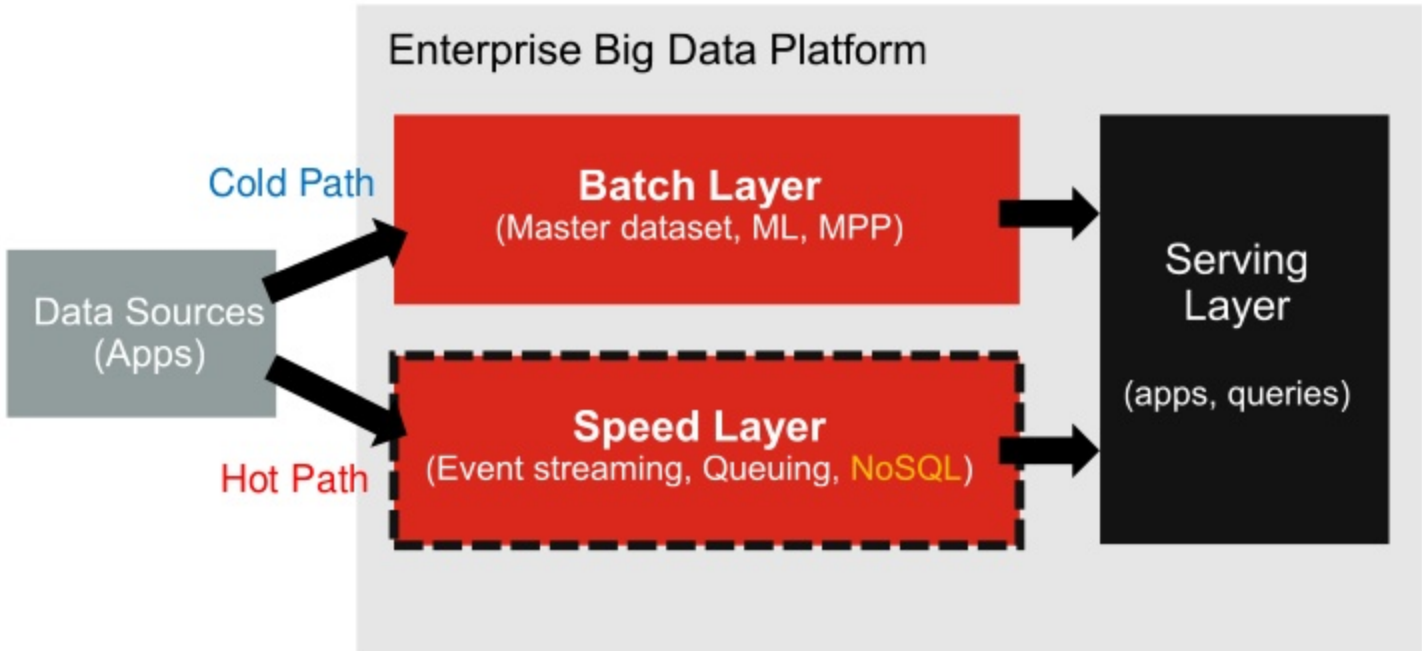
Solution Architecture Patterns

 |

Hitachi Consulting
We make it happen. Better.

Lambda Architecture

NoSQL and Speed Layer

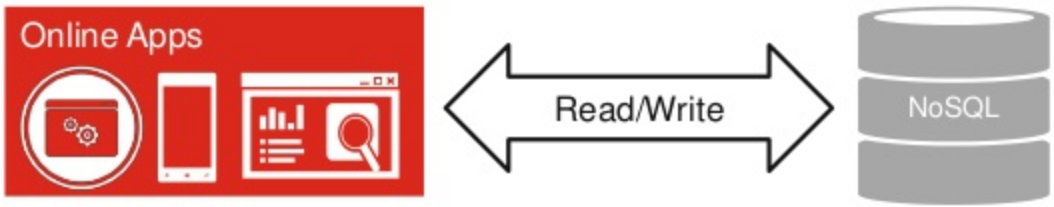


 |

Hitachi Consulting
We make it happen. Better.

NoSQL Usage Patterns

Common Scenarios



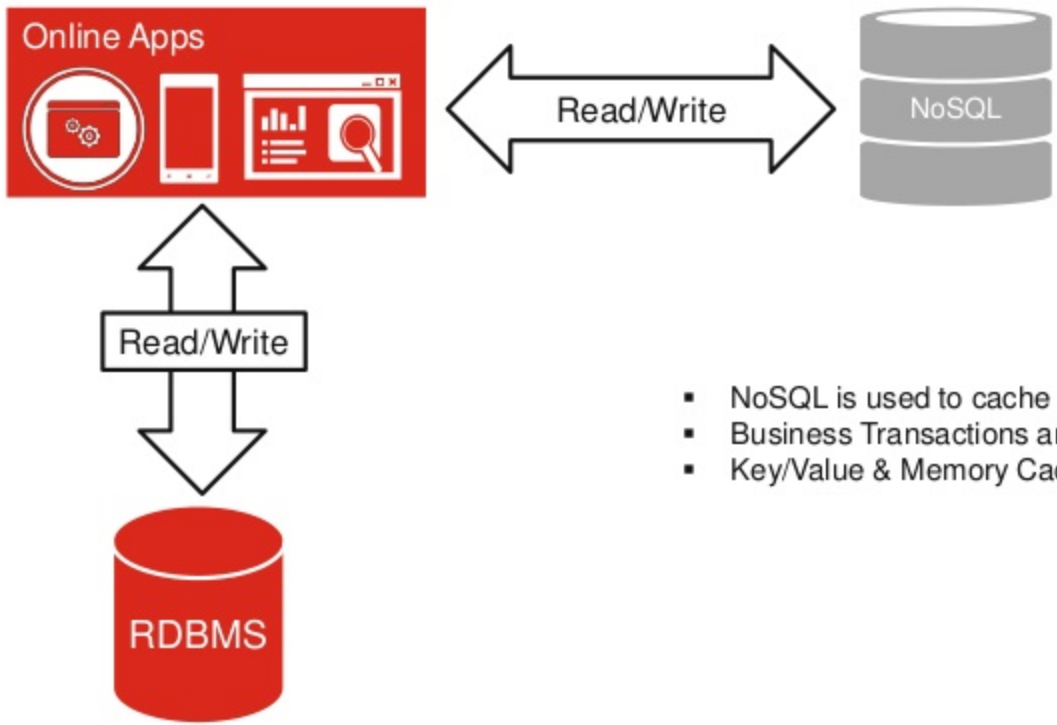
- Apps with NoSQL Backend data store
- High Throughputs
- Scalability and Availability
- Column Family, Graph & Document stores

 |

Hitachi Consulting
We make it happen. Better.

NoSQL Usage Patterns

Common Scenarios



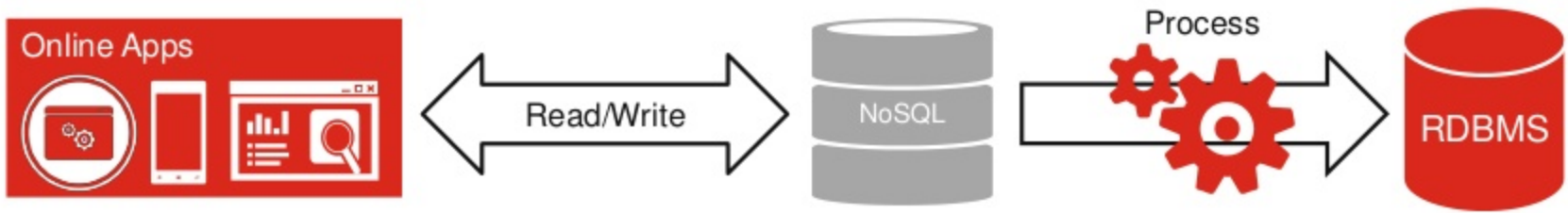
- NoSQL is used to cache web content, personalization, reference data
- Business Transactions are invoked and stored into RDBMS
- Key/Value & Memory Cache stores

 |

Hitachi Consulting
We make it happen. Better.

NoSQL Usage Patterns

Common Scenarios

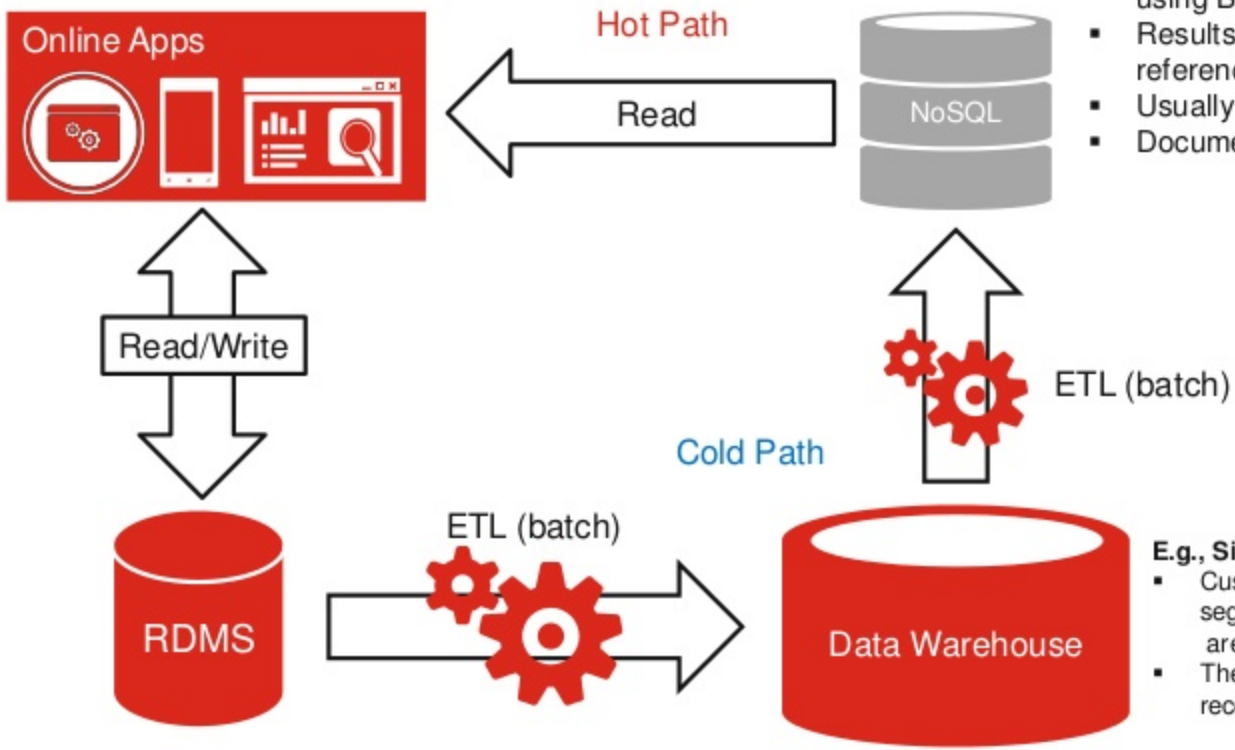


- NoSQL stores in-progress transactions/activities (i.e., purchase basket, forms, user session, etc.)
- When transactions are submitted, they are processed into a RDBMS
- Document Stores

NoSQL Usage Patterns

Common Scenarios

Hitachi Consulting
We make it happen. Better.



- Data is Extracted, Transformed, loaded from OLTPs to a DW
- Aggregations, KPIs, and scores are computed in using Batch Processing
- Results are populated to a NoSQL data store for reference use in apps
- Usually App hot read and ETL batch Write
- Document & Graph stores

E.g., Single Customer View:

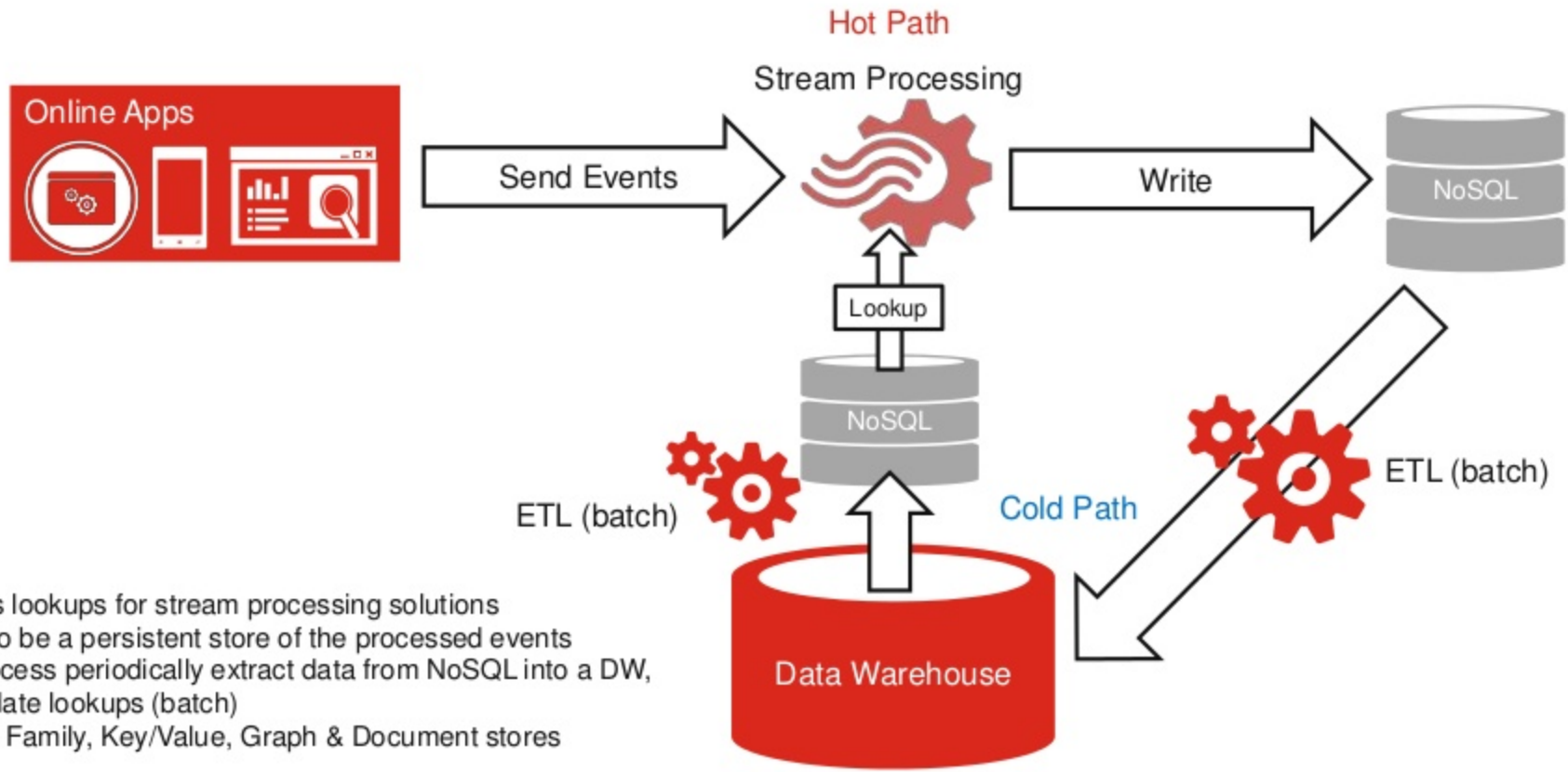
- Customer Matching, customer KPIs, segment assignment, and propensity scoring are performed as batch processing in DW
- The output goes to NoSQL to be used for real-time recommendation, campaigning, targeted advertising, etc.

 |

Hitachi Consulting
We make it happen. Better.

NoSQL Usage Patterns

Common Scenarios



- Used as lookups for stream processing solutions
- Can also be a persistent store of the processed events
- ETL process periodically extract data from NoSQL into a DW, and update lookups (batch)
- Column Family, Key/Value, Graph & Document stores

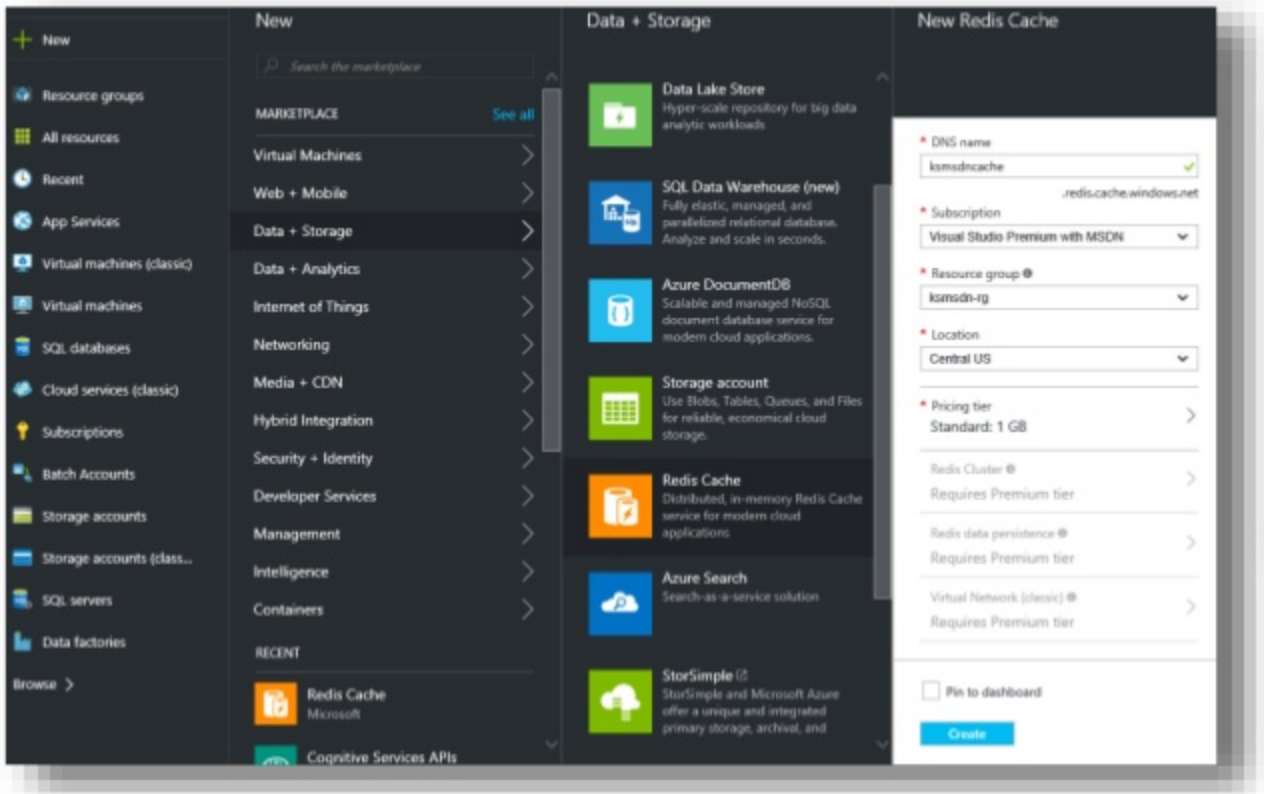
Azure Redis Cache

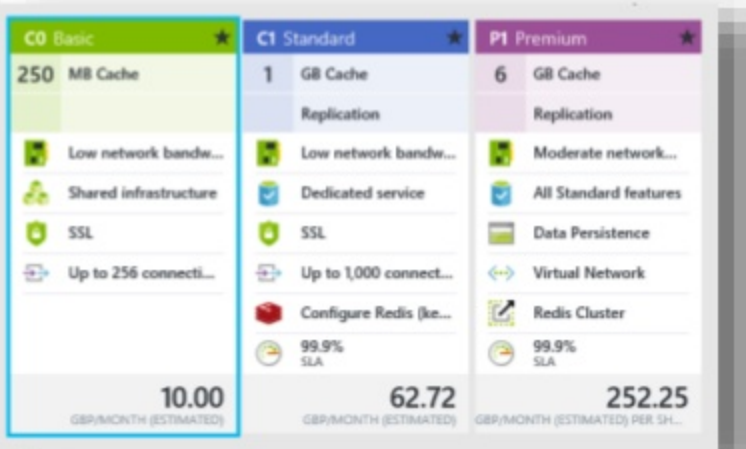
 |

Hitachi Consulting
We make it happen. Better.

Azure Redis Cache

A Key/Value in memory store – a.k.a Data Structures Store

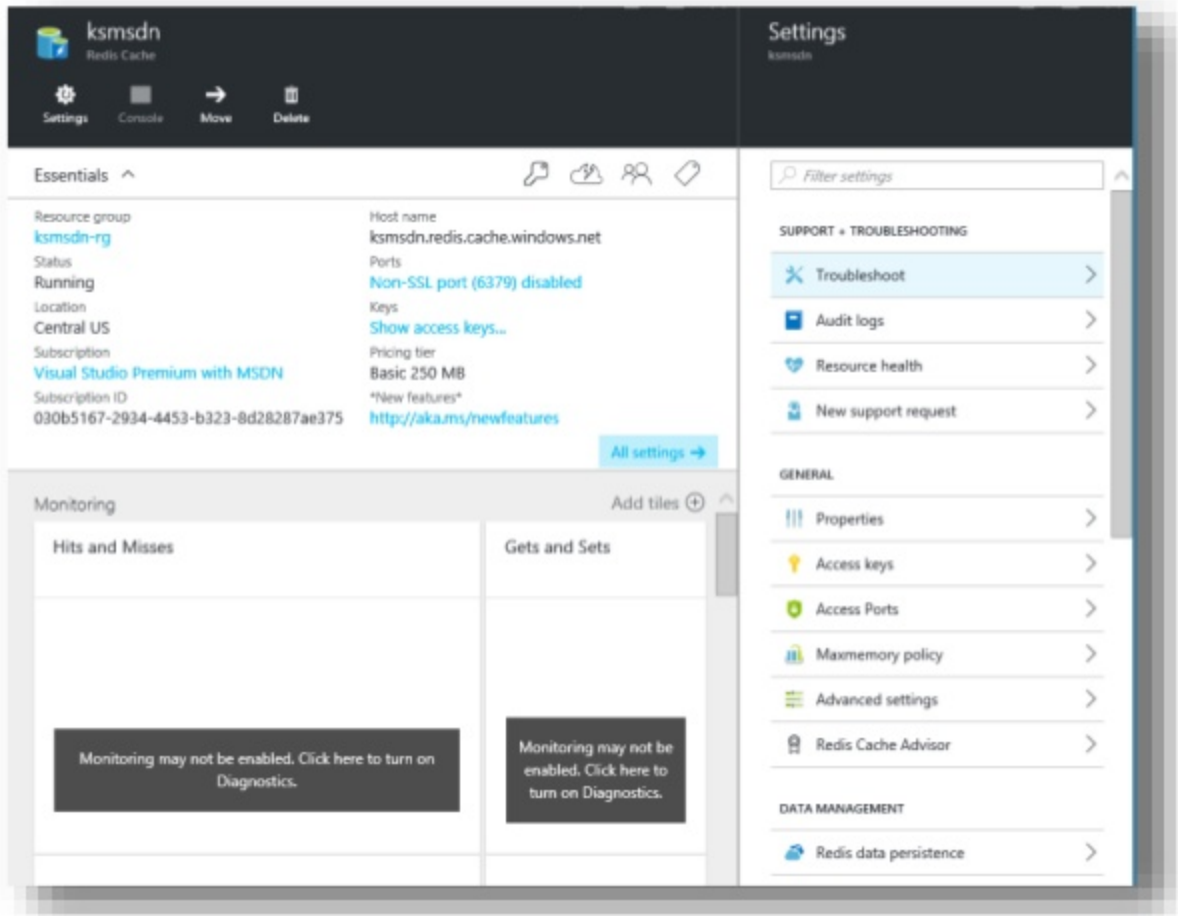




Hitachi Consulting
We make it happen. Better.

Azure Redis Cache

A Key/Value in memory store – a.k.a Data Structures Store



Hitachi Consulting
We make it happen. Better.

Azure Redis Cache

A Key/Value in memory store – a.k.a Data Structures Store

In the example, Radis caches sql query (key), and query xml result (value)

```
using StackExchange.Redis;
```

```
private static Lazy<ConnectionMultiplexer> lazyConnection = new Lazy<ConnectionMultiplexer>(() =>
{
    return  ConnectionMultiplexer.Connect(RedisConnectionString);

});

public static ConnectionMultiplexer Connection
{
    get
    {
        return lazyConnection.Value;
    }
}
```

You can also store hash tables, lists, and sorted lists

```
private DataTable GetDataFromCache(string sqlQuery)
{
    DataSet ds = new DataSet();

    DataTable tbResults = null;


    IDatabase cache = Connection.GetDatabase();

    string xmlContent = null;

    if (!cache.KeyExists(sqlQuery))
    {
        tbResults = this.GetDataFromDatabase(sqlQuery);


        using (StringWriter sw = new StringWriter())
        {
            tbResults.WriteXml(sw);
            xmlContent = sw.ToString();
        }

        cache.StringSet(sqlQuery, xmlContent);

    }

    xmlContent = cache.StringGet(sqlQuery);

    using(StringReader sr= new StringReader(xmlContent))
    {
        ds.ReadXml(sr);
        tbResults = ds.Tables[0];
    }


    return tbResults;
}
```

Azure Table Storage

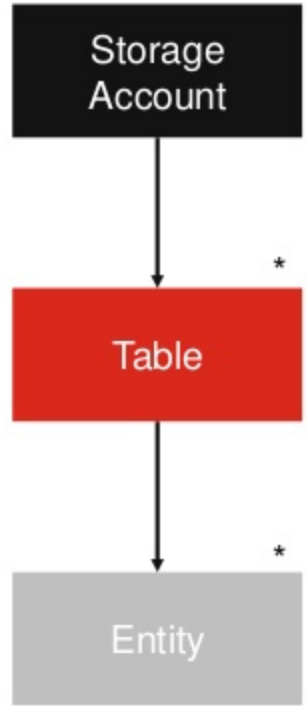
 |

Hitachi Consulting
We make it happen. Better.

Azure Table Storage

A Key/Value NoSQL Store

- Key/attribute store with a schema-less design.
- Adapts your data as the needs of your application evolve.
- Access to data is fast and cost-effective for all kinds of applications.
- Significantly lower in cost than traditional SQL for similar volumes of data.
- Web applications, address books, device information, metadata, etc.
- Row Key and Partition Key must be defined for the entity
- No complex joins, foreign keys, or stored procedures.
- Supports OData protocol and LINQ with WCF Data Service .NET Libraries.

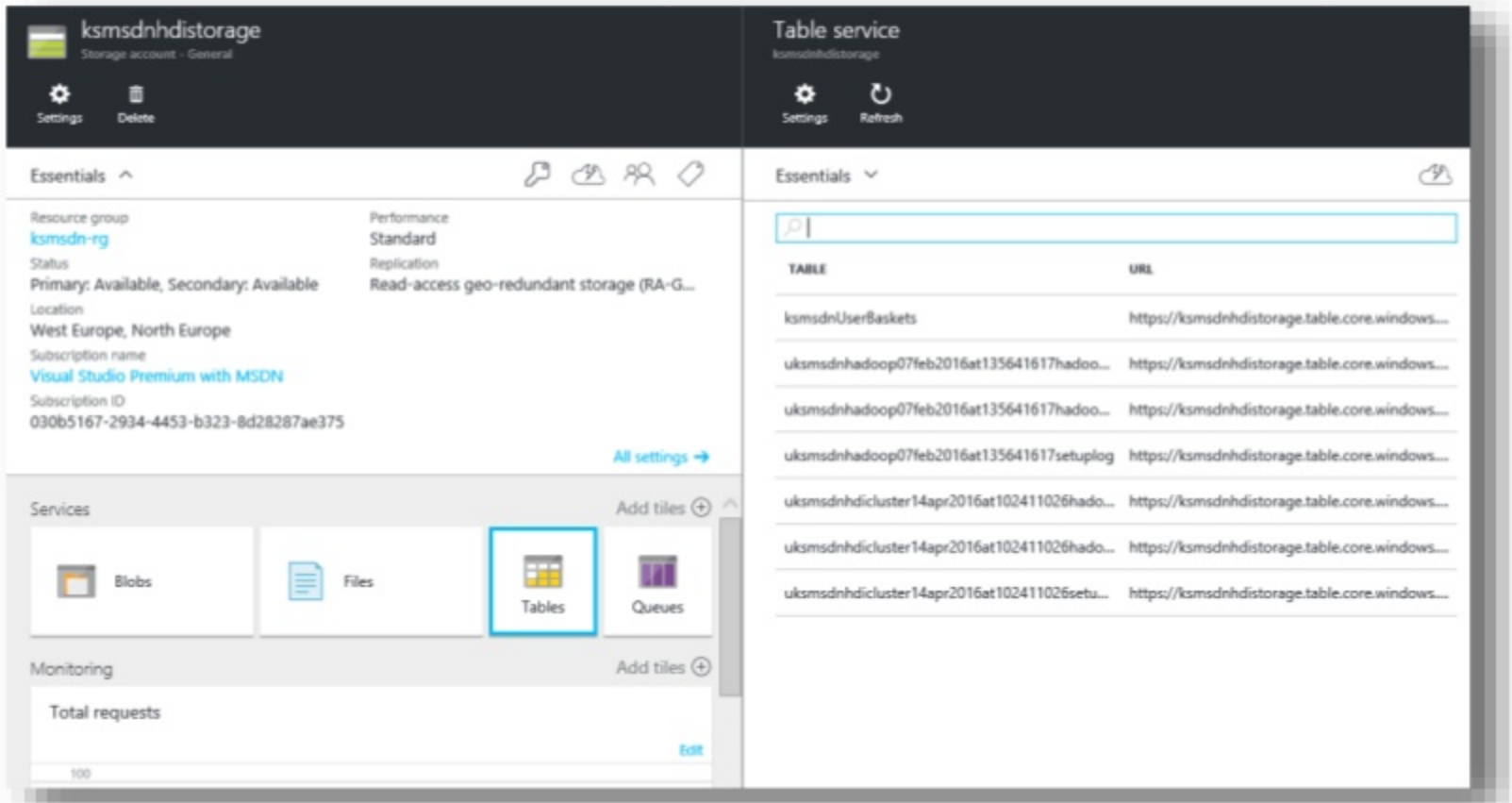


 |

Azure Table Storage

A Key/Value NoSQL Store

Hitachi Consulting
We make it happen. Better.



 |

Hitachi Consulting
We make it happen. Better.

Azure Table Storage

A Key/Value NoSQL Store

```
using Microsoft.WindowsAzure.Storage.Table;

namespace HCUK.AzureNoSQL.Model
{
    public class UserBasket : TableEntity
    {
        private string _accountId;
        private string _region;

        public string AcountId
        {
            get
            {
                return this._accountId;
            }
            set
            {
                this._accountId = value;
                this.RowKey = value;
            }
        }

        public string Region
        {
            get
            {
                return this._region;
            }
            set
            {
                this._region = value;
                this.PartitionKey = value;
            }
        }
```

Namespace to use

Class to inherit

Row Key and Partition Key properties to set

 |

Azure Table Storage

A Key/Value NoSQL Store

Hitachi Consulting
We make it happen. Better.

If a Table Entity has an attribute that is a collection or object, Table Storage will ignore it.
Thus, needs to serialized/ de-serialized

```
public List<Item> Items
{ get; set; }

[Newtonsoft.Json.JsonIgnore]
public string ItemString
{
    get
    {
        return String.Join("|", this.Items.Select(t => t.ToString()).ToArray());
    }
    set
    {
        this.Items = new List<Item>();
        foreach (var itemSting in value.Split('|'))
        {
            var elemets = itemSting.Trim('(', ')').Split(',');
            this.Items.Add(new Item() { ItemId = elemets[0], Quantity = int.Parse(elemets[1]), Value = double.Parse(elemets[2]) });
        }

    }
}
```

 |

Azure Table Storage

A Key/Value NoSQL Store

Hitachi Consulting
We make it happen. Better.

```
CloudTable _azureTable;

public AzureTableStorageServices(string accountName, string keyValue, string tableName)
{
    var credentials = new Microsoft.WindowsAzure.Storage.Auth.StorageCredentials(accountName, keyValue);
    CloudStorageAccount storageAccount = new CloudStorageAccount(credentials, false);
    CloudTableClient azureTableclient = storageAccount.CreateCloudTableClient();

    this.TableName = tableName;
    this._azureTable = azureTableclient.GetTableReference(this.TableName);
}
```

```
public T RetrieveEntity(string partitionKey, string rowKey)
{
    TableOperation retrieveOperation = TableOperation.Retrieve<T>(partitionKey, rowKey);
    var result = _azureTable.Execute(retrieveOperation);
    return result.Result as T;
}
```

```
public void RemoveEntity(T entity)
{
    var deleteOperation = TableOperation.Delete(entity);
    this._azureTable.ExecuteAsync(deleteOperation);
}

public void RemoveEntity(string paritionKey, string rowKey)
{
    var entity = RetrieveEntity(paritionKey, rowKey);

    if (entity != null)
    {
        var deleteOperation = TableOperation.Delete(entity);
        this._azureTable.ExecuteAsync(deleteOperation);
    }

}
```

```
public void AddEntities(IEnumerable<T> entities)
{
    var partitions = entities.Select(e => e.PartitionKey).Distinct();

    foreach (var partition in partitions)
    {
        TableBatchOperation batch = new TableBatchOperation();
        foreach (var entity in entities.Where(e=>e.PartitionKey==partition))
            batch.Add(TableOperation.InsertOrReplace(entity));
        _azureTable.ExecuteBatchAsync(batch).Wait();
    }

}
```

```
public List<T> RetrieveEntitiesByPartition(string partitionKey)
{
    List<T> results = new List<T>();

    TableQuery<T> query = new TableQuery<T>()
        .Where(TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal, partitionKey));

    return results;

}
```

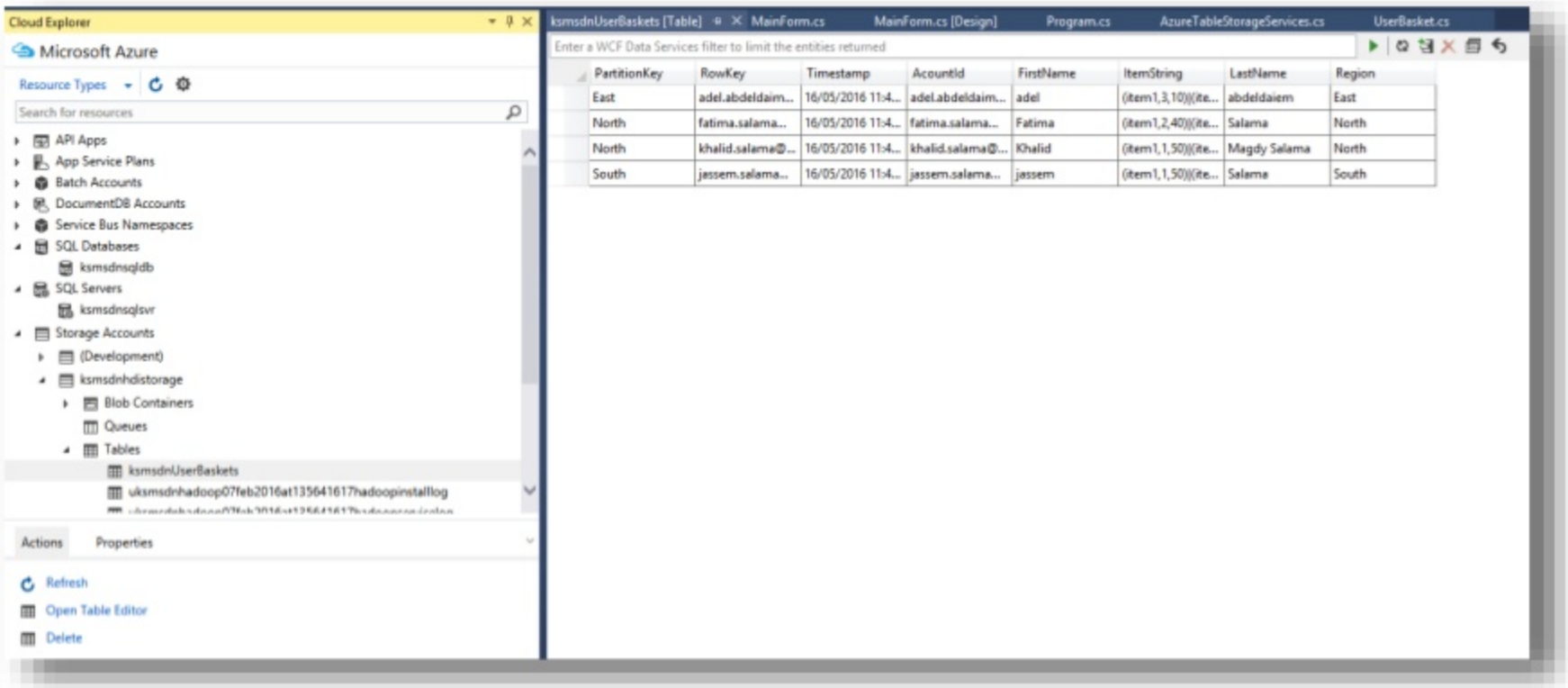
```
public void AddEntity(T entity, InsetBehaviour insertBehavior = InsetBehaviour.Normal)
{
    TableOperation insertOperation = null;

    switch (insertBehavior)
    {
        case InsetBehaviour.Normal: insertOperation = TableOperation.Insert(entity); break;
        case InsetBehaviour.Replace: insertOperation = TableOperation.InsertOrReplace(entity); break;
        case InsetBehaviour.Merge: insertOperation = TableOperation.InsertOrMerge(entity); break;
    }

    _azureTable.Execute(insertOperation);
}
```

```
public static List<UserBasket> GetUserBasketByCondition
    (CloudTable azureTable, string filterCondition)
{
    TableQuery<UserBasket> query = new TableQuery<UserBasket>()
        .Where(filterCondition);
    return azureTable.ExecuteQuery<UserBasket>(query).ToList();
}
```

Azure Table Storage

A Key/Value NoSQL Store

Hitachi Consulting
We make it happen. Better.



 |

Azure DocumentDB

 |

Hitachi Consulting
We make it happen. Better.

Azure DocumentDB

A cloud-based Document store

- JSON Database
- Elastically scalable throughput and storage
- Ad hoc queries with familiar SQL syntax
- JavaScript execution within the database
- Tunable consistency levels
- Fully managed
- Open by design

Hitachi Consulting
We make it happen. Better.

Azure DocumentDB

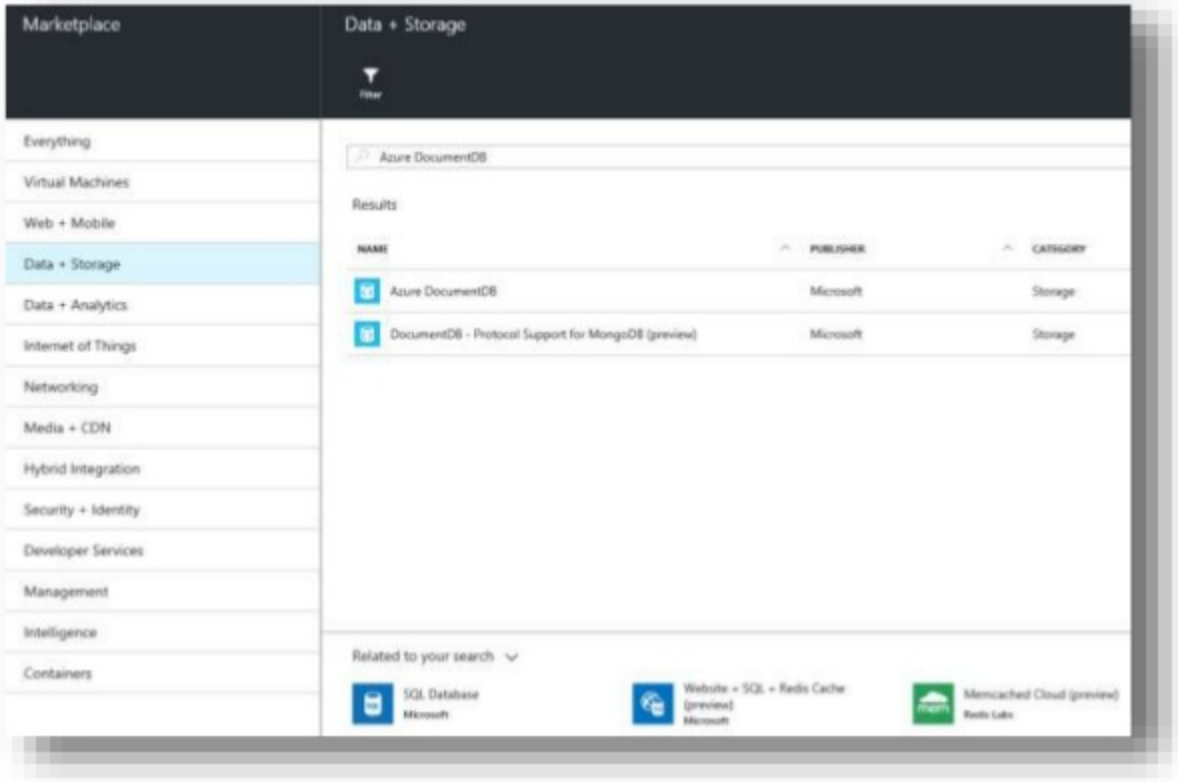
Azure DocumentDB Structure

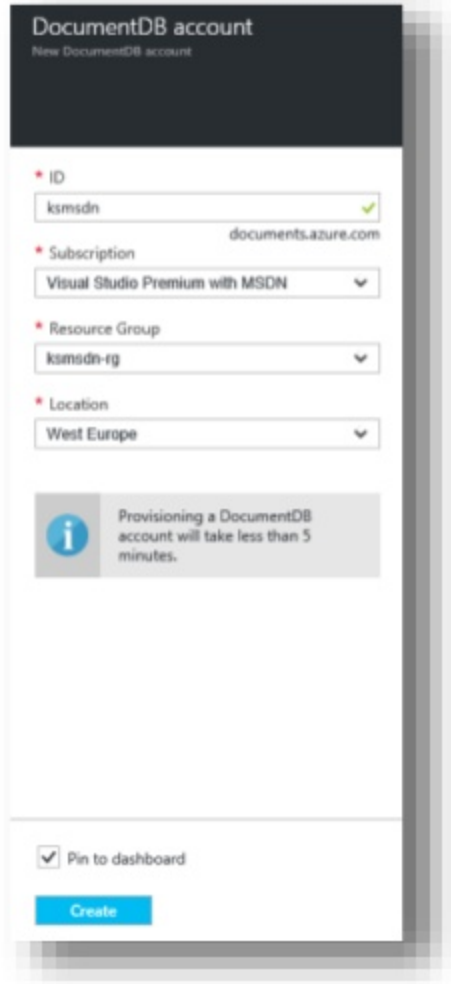
 |

Azure DocumentDB

Getting Started

Hitachi Consulting
We make it happen. Better.



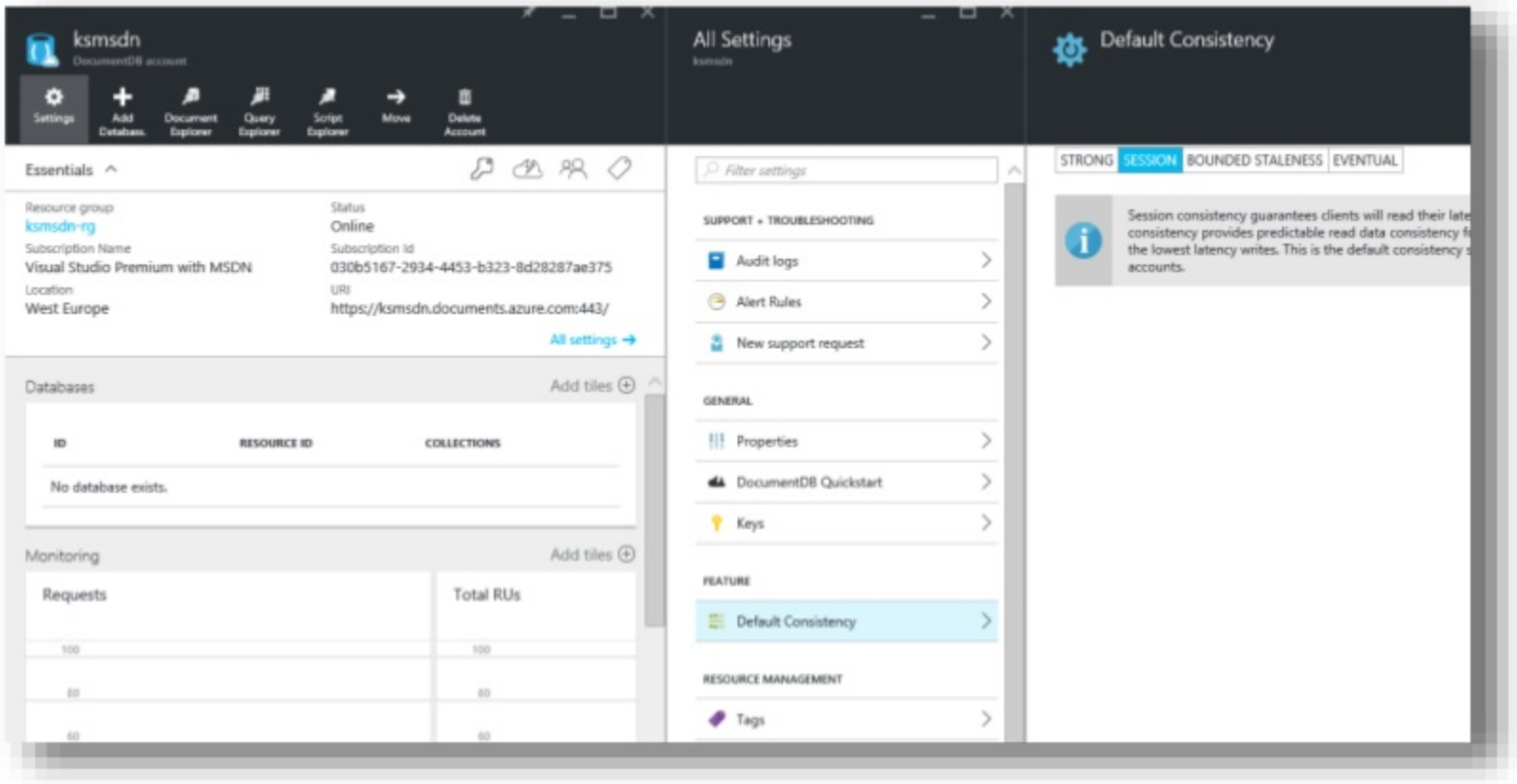


 |

Azure DocumentDB

Getting Started

Hitachi Consulting
We make it happen. Better.



 |

Hitachi Consulting
We make it happen. Better.

Azure DocumentDB

DocumentDB Consistency Model

	Description
Strong	The slowest of the four, but is guaranteed to always return correct data.
Session	Ensures that an application always sees its own writes correctly, but allows access to potentially out-of-date or out-of-order data written by other applications.
Bound Staleness	Ensures that an application will see changes in the order in which they were made. This option does allow an application to see out-of-date data, but only within a specified window, e.g., 500 milliseconds.
Eventual	Provides the fastest access, but also has the highest chance of returning out-of-date data.

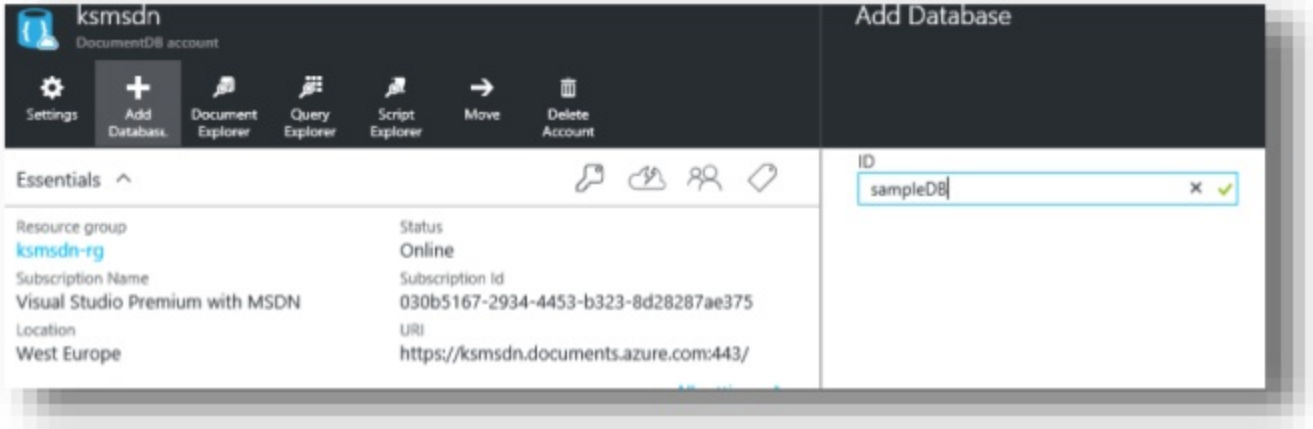
 |

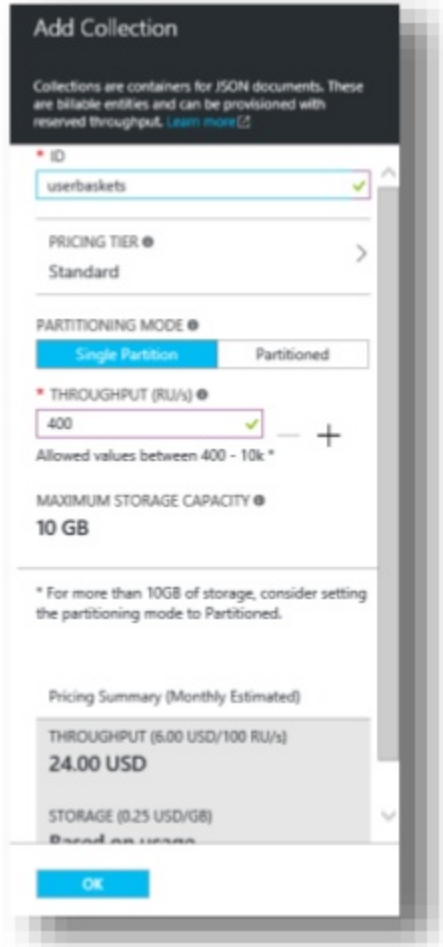
https://en.wikipedia.org/wiki/Consistency_model

Azure DocumentDB

Getting Started

Hitachi Consulting
We make it happen. Better.



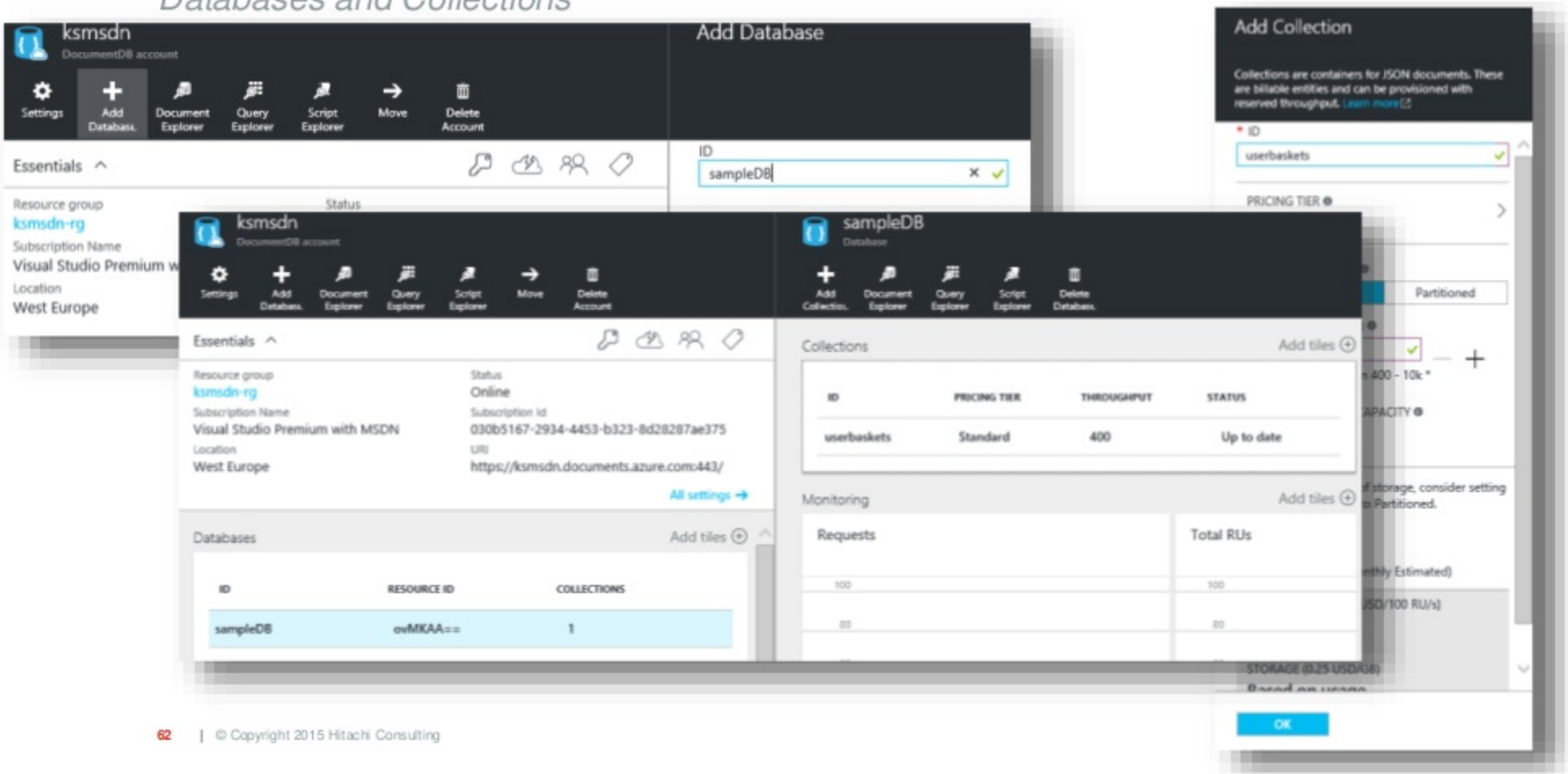


 |

Azure DocumentDB

Databases and Collections

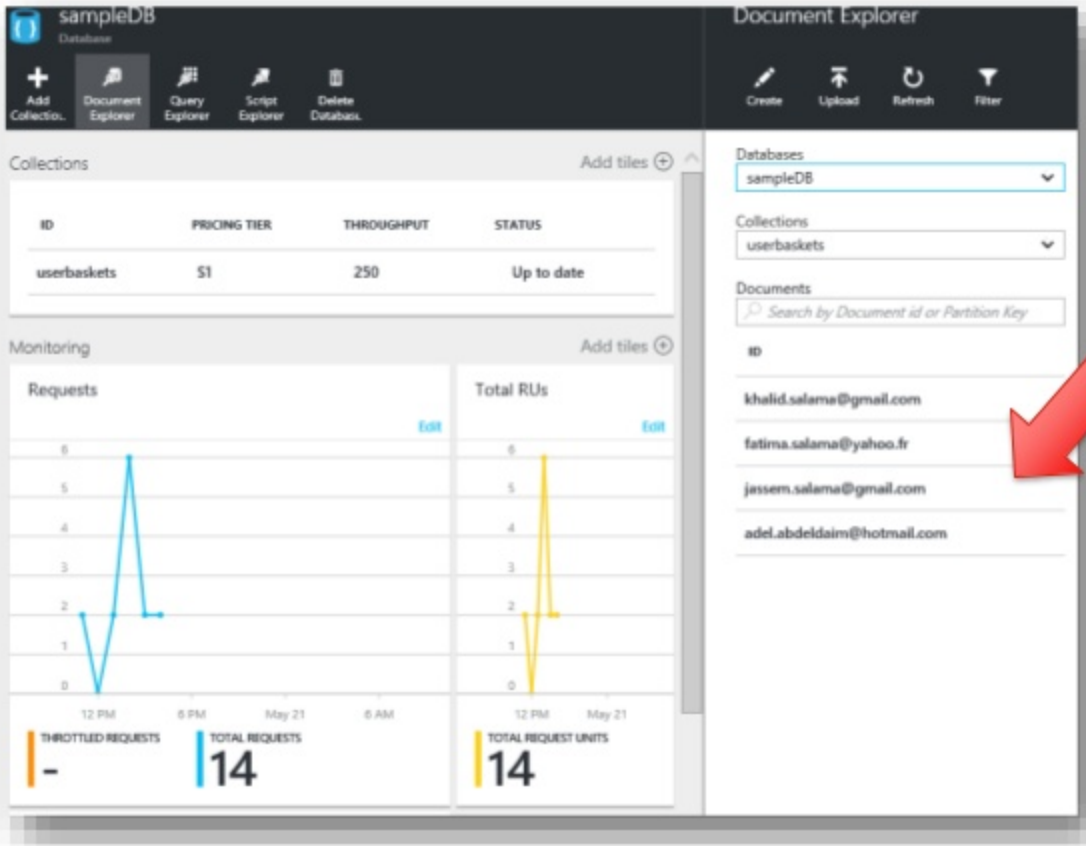
Hitachi Consulting
We make it happen. Better.



 |

Azure DocumentDB

Document Explorer

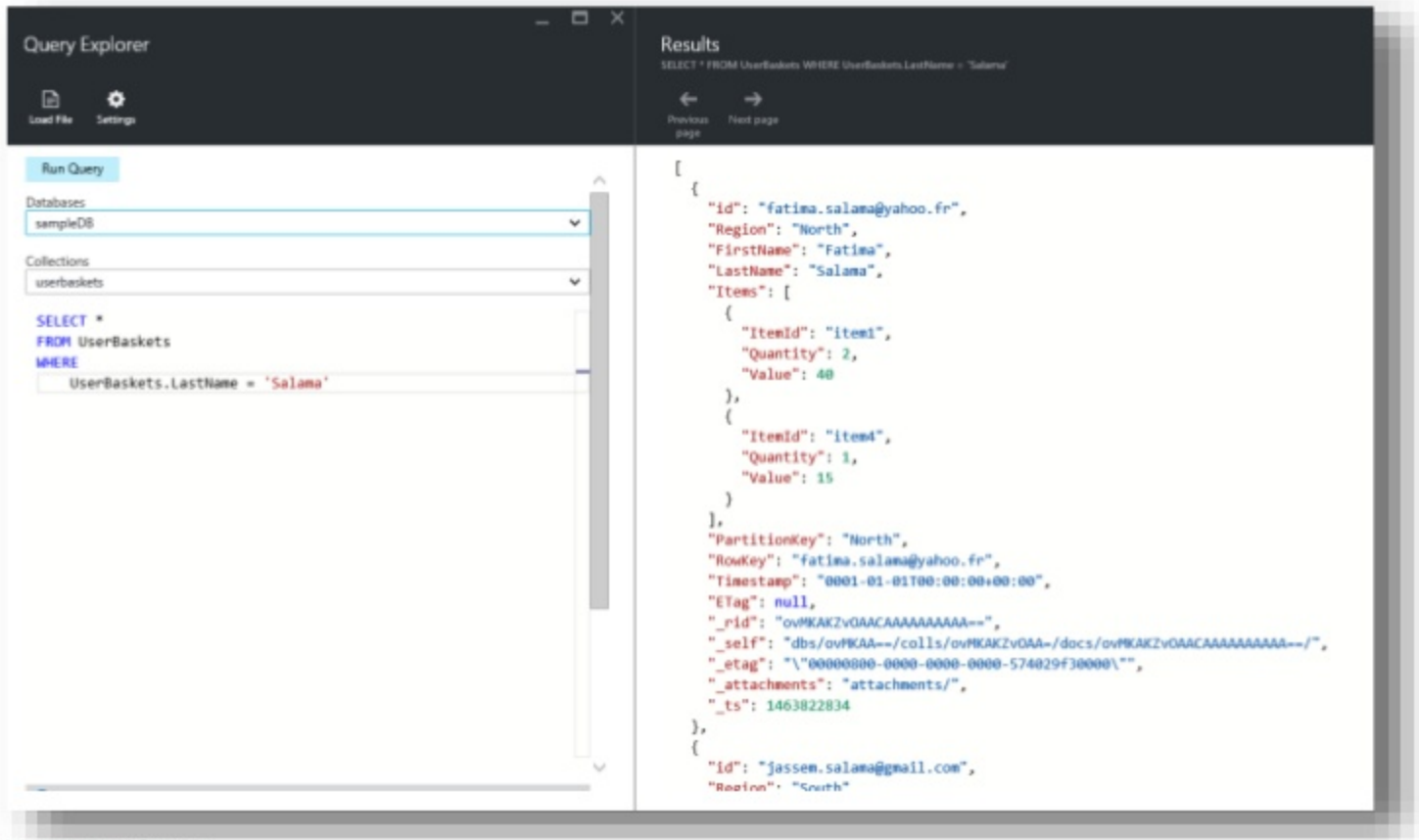



```
public class UserBasket : TableEntity
{
    private string _accountId;
    private string _region;

    [JsonProperty(PropertyName = "id")]
    public string AcountId
    {
        get
        {
            return this._accountId;
        }
        set
        {
            this._accountId = value;
            this.RowKey = value;
        }
    }

    public string Region
    {
        get
        {
            return this._region;
        }
        set
        {
            this._region = value;
            this.PartitionKey = value;
        }
```

 |

Azure DocumentDB

Query Explorer



 |

Azure DocumentDB

.NET Code

Microsoft.Azure.Documents.Client.DocumentClient

Includes the following operations,

- Create/Delete
- Read
- Replace/Upsert,

for the following objects,

- Database
- Collection
- Document
- Attachment
- User
- Permission
- USerDefinedFunction
- StoredProcedure
- Trigger

Azure DocumentDB

.NET Code

Hitachi Consulting
We make it happen. Better.

```
using Microsoft.Azure.Documents;
using Microsoft.Azure.Documents.Client;
using Microsoft.Azure.Documents.Linq;
```

```
public class DocumentDBServices<T>
{
    public Database Database...

    public DocumentCollection Collection...

    private DocumentClient Client...

    public DocumentDBServices(string endPoint, string accountKey, string databaseId, string collectionId, bool dropCollection=false)
    {
        this.Client = new DocumentClient(new Uri(endPoint), accountKey);
        this.Collection =  this.ReadOrCreateCollection(databaseId, collectionId, dropCollection);
    }
```

```
public string AddItem(T item)
{
    return CreateItemAsync(item).Result.Id;
}
```

```
public Document GetDocument(string id)
{
    return Client.CreateDocumentQuery(Collection.DocumentsLink)
        .Where(d => d.Id == id)
        .AsEnumerable()
        .FirstOrDefault();


}
```

```
public T GetItem(Expression<Func<T, bool>> predicate)
{
    return Client.CreateDocumentQuery<T>(Collection.DocumentsLink)
                .Where(predicate)
                .AsEnumerable()
                .FirstOrDefault();
}

public IEnumerable<T> GetItems(Expression<Func<T, bool>> predicate)
{
    return Client.CreateDocumentQuery<T>(Collection.DocumentsLink)
                .Where(predicate)
                .AsEnumerable();

}
```

```
public IEnumerable<T> GetItems(string sqlQuery)
{
    FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1 };

    IQueryable<T> query = this.Client.CreateDocumentQuery<T>(
        UriFactory.CreateDocumentCollectionUri(Database.Id, Collection.Id),
        sqlQuery,
        queryOptions);

    var result = query.ToList();

    return result;
}
```

```
public async Task<Document> UpdateItemAsync(string id, T item)
{
    Document doc = GetDocument(id);
    return await Client.ReplaceDocumentAsync(doc.SelfLink, item);
}
```


Hitachi Consulting
We make it happen. Better.

HBase on Azure

 |

Introducing Apache HBase

HBase & Hadoop Big Data Ecosystem

Hitachi Consulting
We make it happen. Better.

Applications

Acquisition	Batch	In-Memory	Stream	SQL	NoSQL	Machine Learning	Search	Orchest. Mgmnt
▪ Sqoop ▪ Flume ▪ HDFS (APIS)	▪ Map Reduce ▪ Pig	▪ Spark	▪ Storm ▪ Flume ▪ Kafka ▪ Spark Streaming	▪ Hive ▪ Phoenix ▪ Spark-SQL	▪ Accumulo ▪ HBase ▪ CouchBase	▪ Mahout ▪ Spark-MIlib	▪ Solr ▪ Elastic-search	▪ Oozie ▪ Ambari ▪ Zoo-Keeper

Named Node

Yet Another Resource Negotiator (YARN)

Hadoop Distributed File System (HDFS)

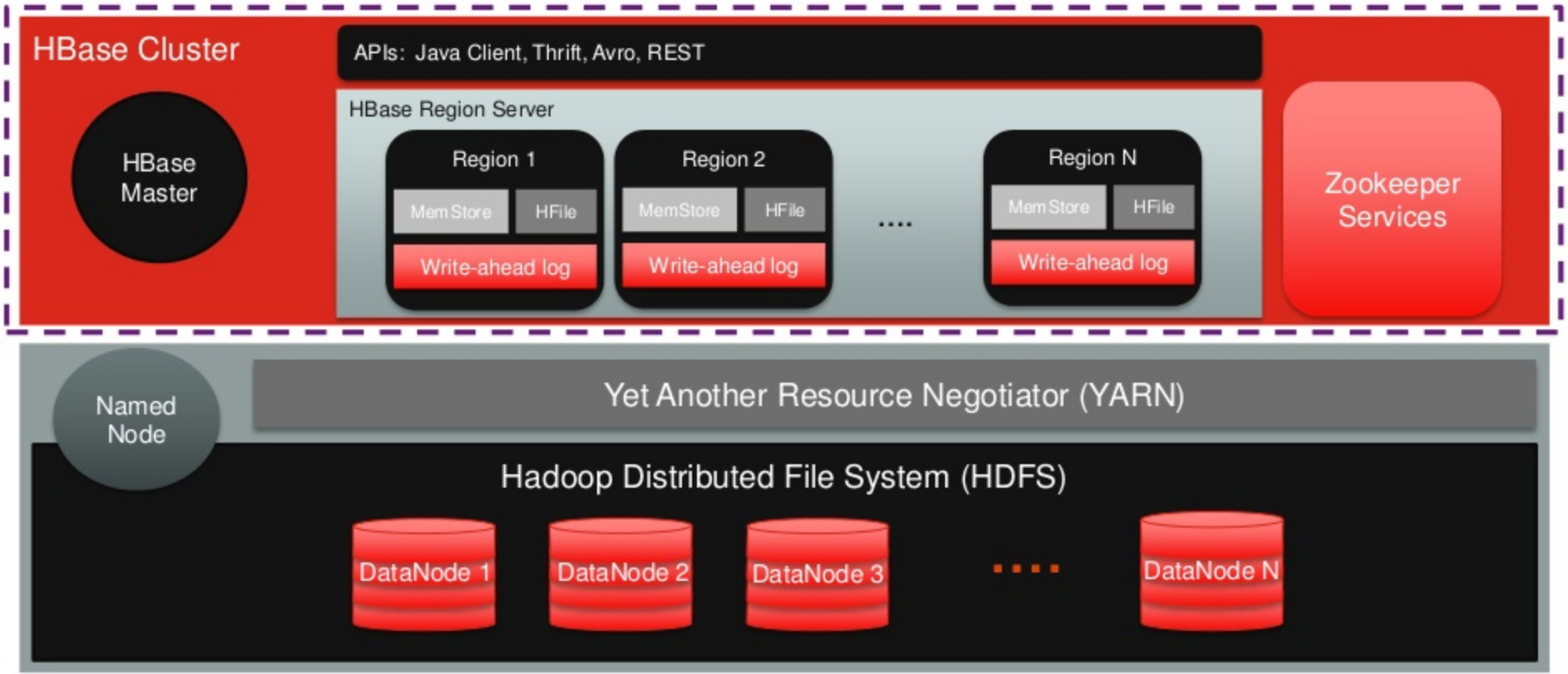
DataNode 1 | DataNode 2 | DataNode 3 | .... | DataNode N

 |

Hitachi Consulting
We make it happen. Better.

Introducing Apache HBase

HBase & Hadoop Big Data Ecosystem



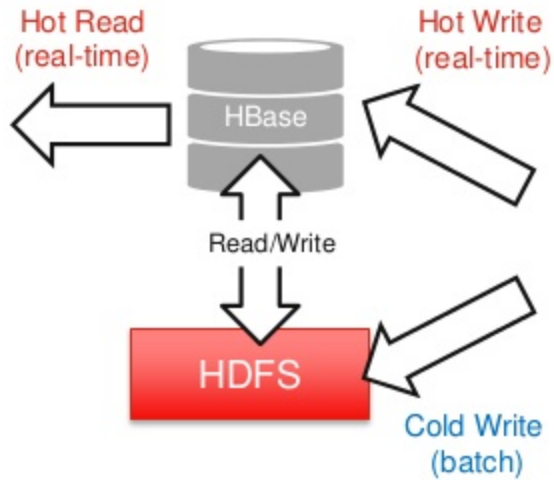
 |

Hitachi Consulting
We make it happen. Better.

HBase on Azure HDInsight

a brief introduction to HBase

- HDFS is suitable for batch processing (i.e., scan over big data files)
- HBase is optimized for **fast record lookups**, and singleton operations
- HDFS is usually the file system for HBase
- Rows maintained in **sorted lexicographical order** for efficient rows scan
- Row ranges are **partitioned** into tablets
- Column are grouped **into column families for locality indication**
- Simple commandlet: create, alter, drop, list, describe, get, put, incr, scan, count, delete, truncate, etc. https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/
- APIs support batch operations
- A common choice with stream processing solutions

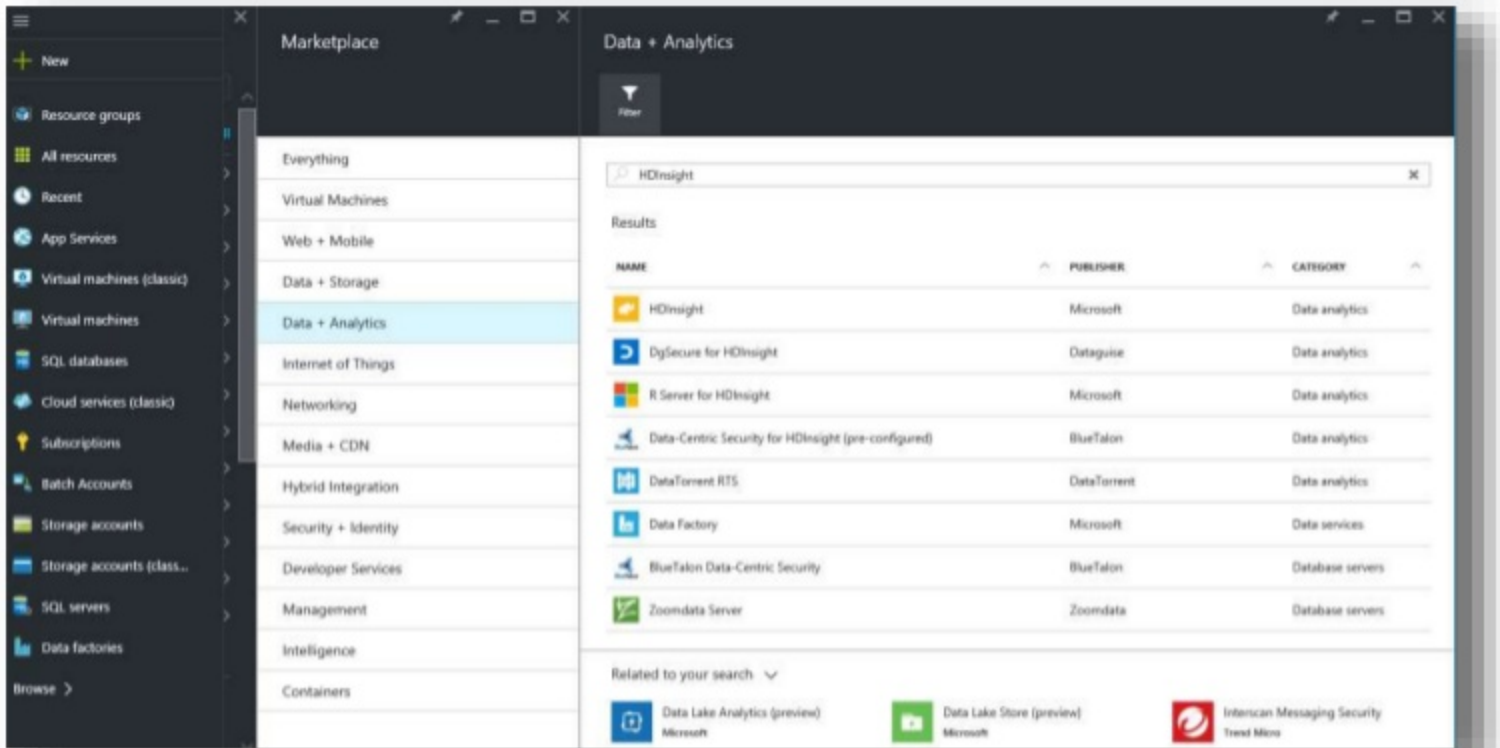


 |

HBase on Azure HDInsight

Getting Started

Hitachi Consulting
We make it happen. Better.

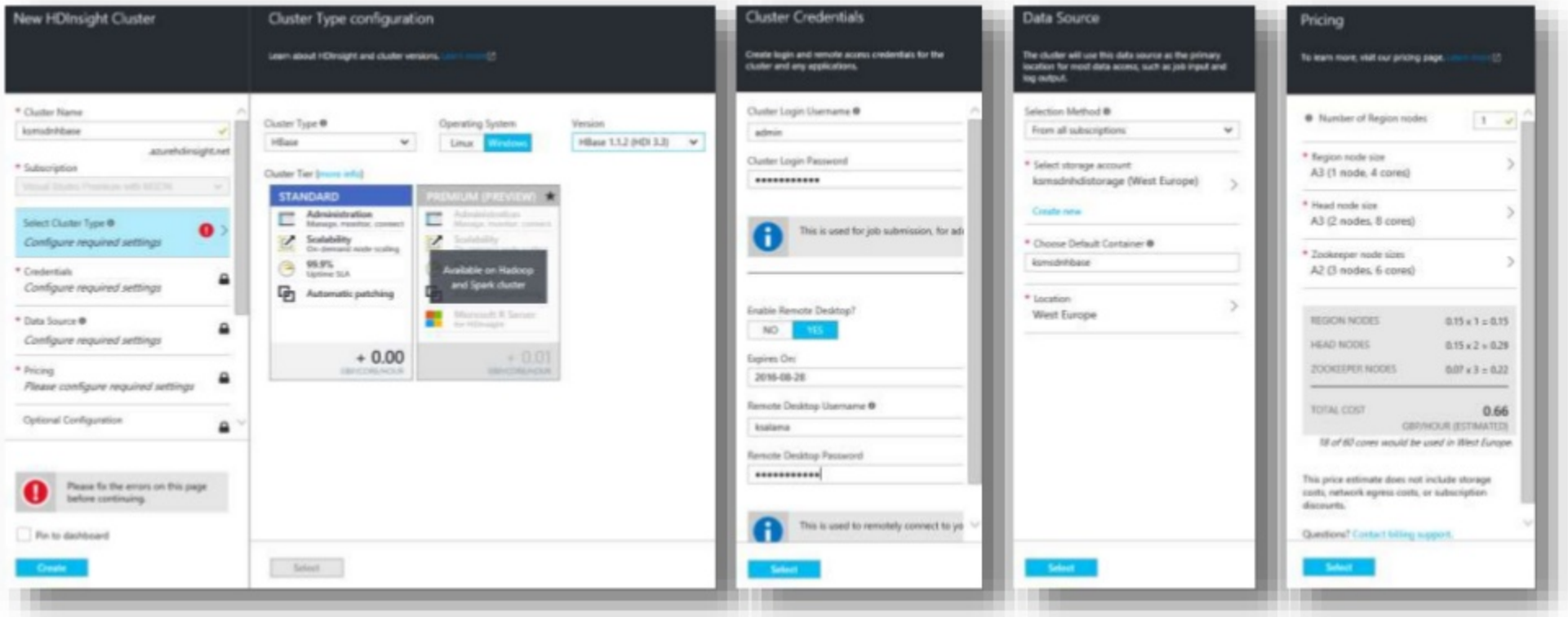


 |

HBase on Azure HDInsight

Getting Started

Hitachi Consulting
We make it happen. Better.

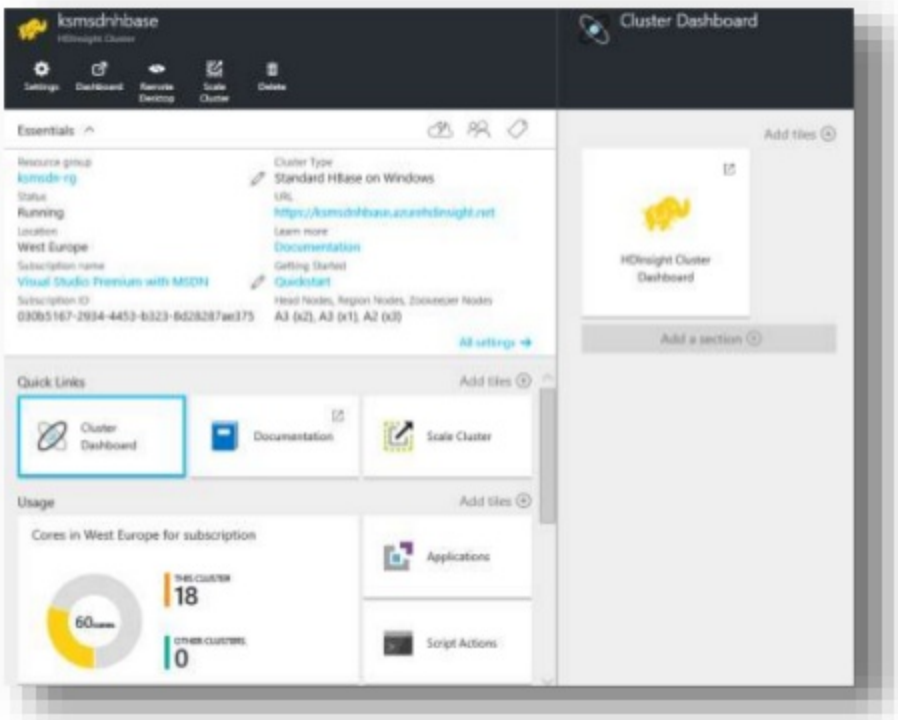


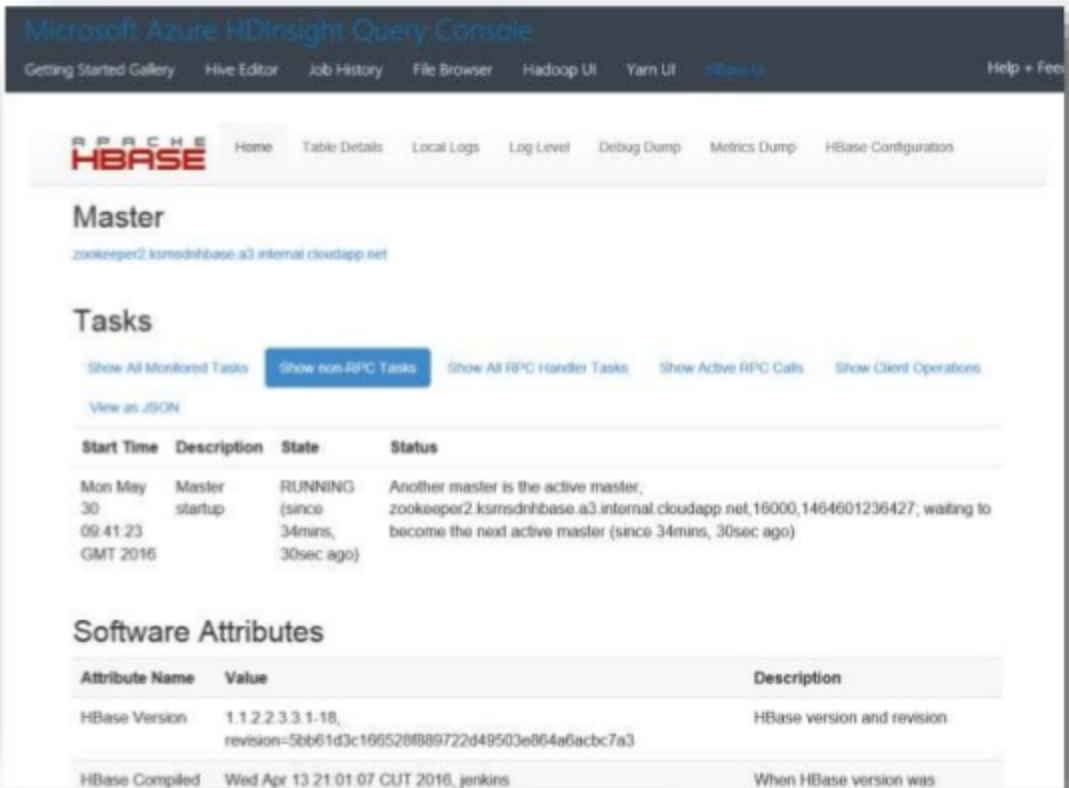
 |

HBase on Azure HDInsight

Exploring HBase

Hitachi Consulting
We make it happen. Better.



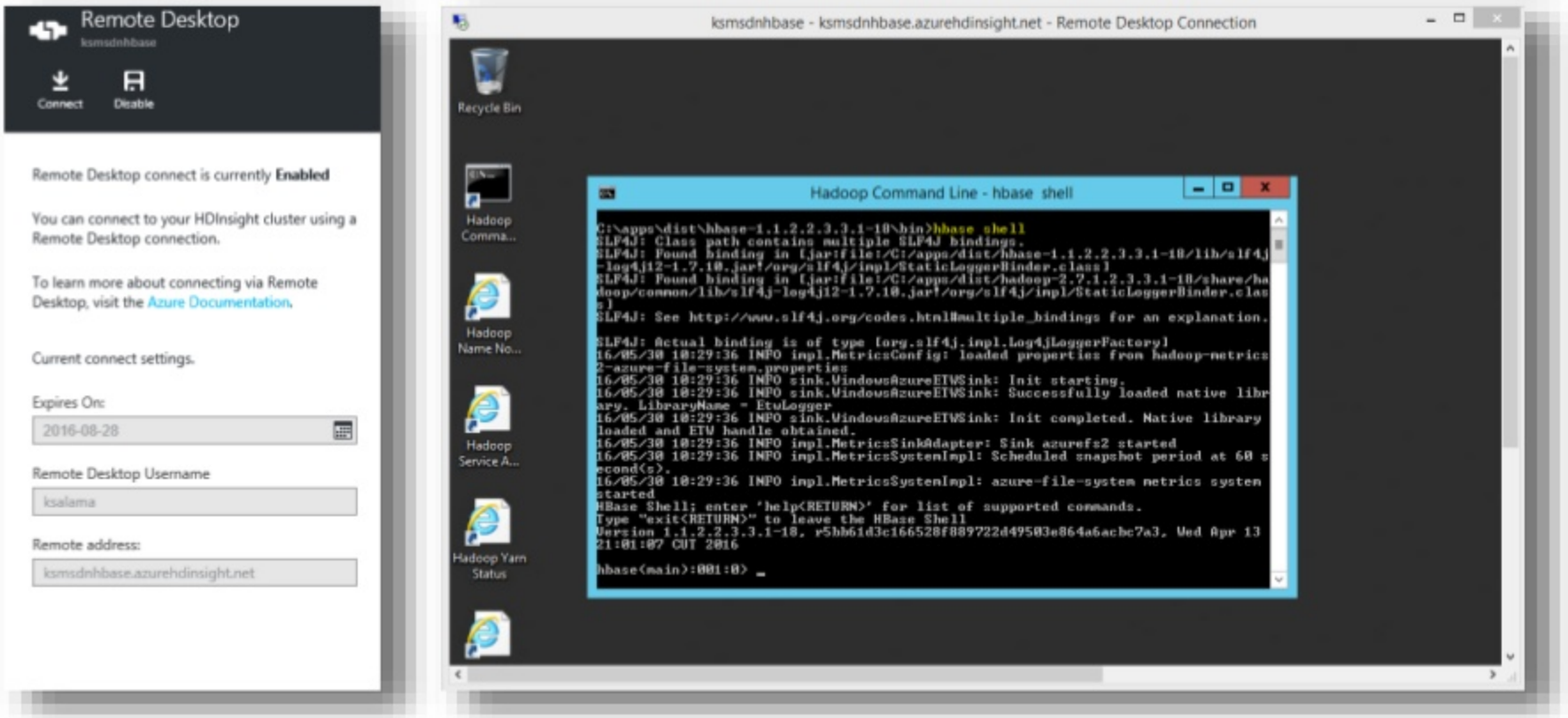


 |

HBase on Azure HDInsight

Exploring HBase

Hitachi Consulting
We make it happen. Better.



 |

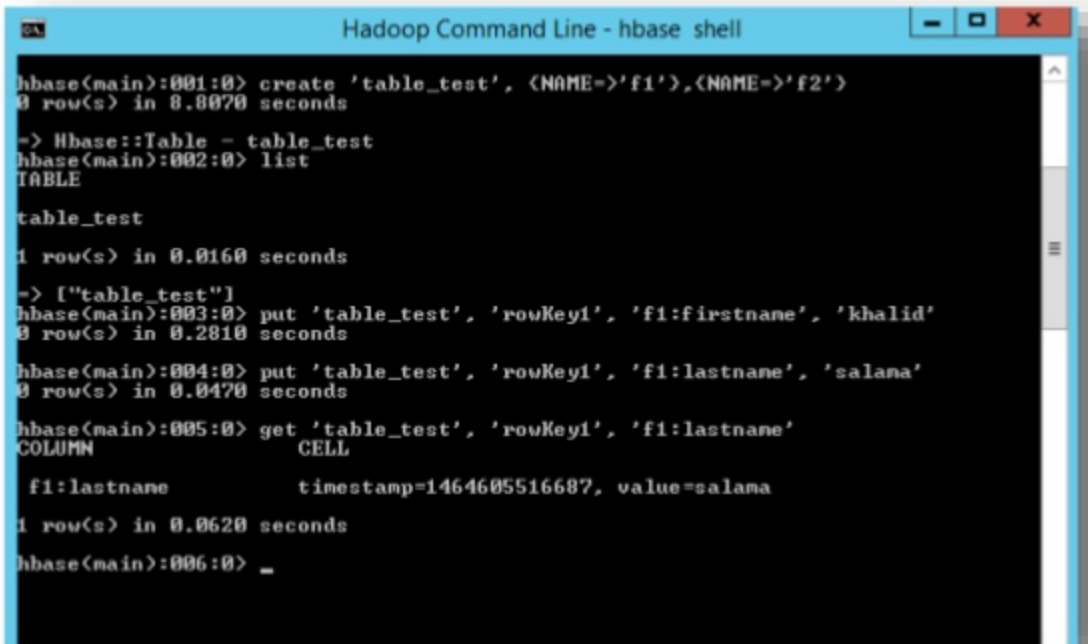
HBase on Azure HDInsight

Exploring HBase

Hitachi Consulting
We make it happen. Better.

Basic Commands

```
create 'table_test', {NAME=>'f1'},{NAME=>'f2'}

List

put 'table_test', 'rowKey1', 'f1:firstname', 'khalid'
put 'table_test', 'rowKey1', 'f1:lastname', 'salama'
put 'table_test', 'rowKey1', 'f2:level', '3'

put 'table_test', 'rowKey2', 'f1:firstname', 'paul'
put 'table_test', 'rowKey2', 'f1:lastname', 'linehame'
put 'table_test', 'rowKey2', 'f1:email', 'plinhame@hitachi.com'


get 'table_test', 'rowKey1', 'f1:lastname'

disable 'table_test'

drop 'table_test'
```



 |

HBase on Azure HDInsight

Exploring HBase

Hitachi Consulting
We make it happen. Better.

Other Commands

```
get 'table test' , 'rowkey1'

get 'table test',  'rowkey1' , {COLUMN => [f1:lastname]}

get 'table test', 'rowkey2', {TIMERANE => [0:1000]}

scan 'table test' {LIMIT =>100}

scan 'table test' {STARTROW=>'rowkey5' , STOPROW='rowkey10'}
```


HBase on Azure HDInsight

HBase Reader/Writer using .NET

Hitachi Consulting
We make it happen. Better.

```
using Microsoft.HBase.Client;
using org.apache.hadoop.hbase.rest.protobuf.generated;
```

```
public HBaseServices(Uri clusterUri, string userName, string password, string tableName, bool maintainRowCount=true)
{

    var credentials = new ClusterCredentials(clusterUri, userName, password);
    this.TableName = tableName;
    this._maintainRowCount = maintainRowCount;

    client = new HBaseClient(credentials);

    if (!client.ListTablesAsync().Result.name.Contains(this.TableName))
    {
        // Create the table
        var tableSchema = new TableSchema();
        tableSchema.name = this.TableName;
        if(maintainRowCount)
            tableSchema.columns.Add(new ColumnSchema { name = "d" });
        client.CreateTableAsync(tableSchema).Wait();
        Console.WriteLine("Table \"{0}\" created.", TableName);
    }
}
```

```
public class HBaseItem
{
    public string ColumnFamily
    { get; set; }

    public string ColumnName
    { get; set; }

    public string Value
    { get; set; }

    public long Timestamp
    { get; set; }
}
```

 |

HBase on Azure HDInsight

HBase Reader/Writer using .NET

Hitachi Consulting
We make it happen. Better.

```
private async Task AddRow(string rowKey, IEnumerable<HBaseItem> items)
{
    var row = new CellSet.Row { key = Encoding.UTF8.GetBytes(rowKey) };

    foreach (var item in items)
    {
        var cell = new Cell
        {
            column = Encoding.UTF8.GetBytes(item.ColumnFamily+":"+item.ColumnName),
            data = Encoding.UTF8.GetBytes(item.Value)
        };
        row.values.Add(cell);

    }

   await this.client.CheckAndPutAsync(this.TableName, row,null);

   if (_maintainRowCount)
   {
       long count = this.RowCount;
       count++;
       await this.SetRowCount(count);
   }
}
```

```
public async Task AddRows(Dictionary<string, List<HBaseItem>> hbaseRows)
{
    var set = new CellSet();

    foreach (string rowKey in hbaseRows.Keys)
    {
        var row = new CellSet.Row { key = Encoding.UTF8.GetBytes(rowKey) };
        var items = hbaseRows[rowKey];

        foreach (var item in items)
        {
            var cell = new Cell
            {
                column = Encoding.UTF8.GetBytes(item.ColumnFamily + ":" + item.ColumnName),
                data = Encoding.UTF8.GetBytes(item.Value)
            };
            row.values.Add(cell);

        }

        set.rows.Add(row);
    }

    await this.client.StoreCellsAsync(this.TableName, set);

    if (_maintainRowCount)
    {
        long count = this.RowCount;
        count+=set.rows.Count;
        await this.SetRowCount(count);
    }
}
```

 |

HBase on Azure HDInsight

HBase Reader/Writer using .NET

Hitachi Consulting
We make it happen. Better.

```
public List<HBaseItem> GetRow(string rowKey)
{
    List<HBaseItem> items = new List<HBaseItem>();

    var cells = this.client.GetCellsAsync(this.TableName, rowKey).Result.rows[0].values;
    foreach (var cell in cells)
    {
        var parts = Encoding.UTF8.GetString(cell.column).Split(':');

        var item = new HBaseItem()
        {
              ColumnFamily = parts[0]
            , ColumnName = parts[1]
            , Value = Encoding.UTF8.GetString(cell.data)
            , Timestamp = cell.timestamp
        };

        items.Add(item);
    }

    return items;
}
```

```
public string GetValue(string rowKey, string columnFamily, string columnName)
{
    return this.GetRow(rowKey)
               .Where(i=> i.ColumnFamily==columnFamily &&  i.ColumnName==columnName)
               .First().Value;
}
```

```
public async Task<List<string>> GetRowsByColumnValues(IEnumerable<HBaseItem> items
    , string startRowKey = null, string endRowKey = null)
{
    var list = new List<string>();

    RequestOptions scanOptions = RequestOptions.GetDefaultOptions();
    scanOptions.AlternativeEndpoint = "hbaserest0/";
    var scanSettings = new Scanner
    {
        batch = 100000
        ,startRow = Encoding.UTF8.GetBytes(startRowKey)
        ,endRow = Encoding.UTF8.GetBytes(endRowKey)
    };
    ScannerInformation scannerInfo = null;
    try
    {
        scannerInfo = await client.CreateScannerAsync(this.TableName, scanSettings, scanOptions);

        CellSet next;
        while ((next = await client.ScannerGetNextAsync(scannerInfo, scanOptions)) != null)
            foreach (CellSet.Row row in next.rows)
                if (Match(row,items))
                    list.Add(Encoding.UTF8.GetString(row.key));


        return list;
    }
    finally
    {
        if (scannerInfo != null)
        {
            client.DeleteScannerAsync(this.TableName, scannerInfo, scanOptions).Wait();
        }
    }
}
```

 |

How to Get Started with NoSQL?

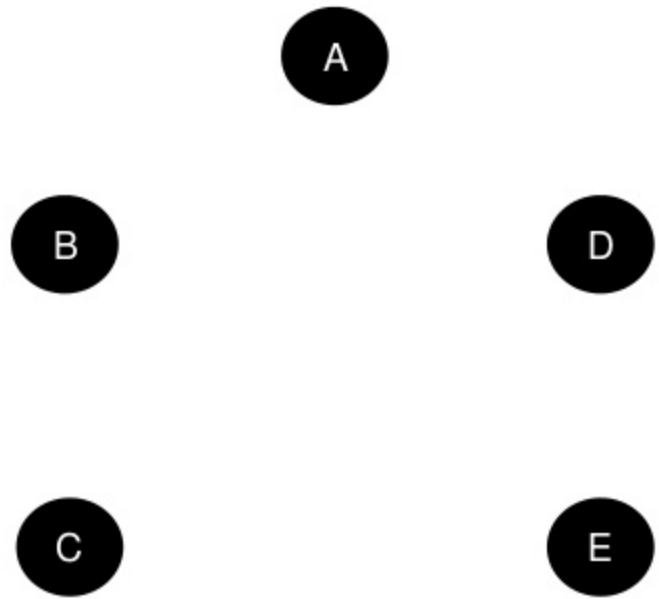
Hitachi Consulting
We make it happen. Better.

- Read the slides!
- Azure Documentation – Azure Table Storage
  https://azure.microsoft.com/en-gb/documentation/articles/storage-dotnet-how-to-use-tables/
- Azure Documentation – Azure DocumentDB
  https://azure.microsoft.com/en-gb/documentation/services/documentdb/
- Azure Documentation – Azure Redis Cache
  https://azure.microsoft.com/en-gb/documentation/services/redis-cache/
- Azure Documentation – HBase on HDInsight
  https://azure.microsoft.com/en-gb/documentation/articles/hdinsight-hbase-overview/
- GitHub – tweet-sentiment with HBase
  https://github.com/maxluk/tweet-sentiment
- Azure Documentation – Understanding NoSQL on Azure
  https://azure.microsoft.com/en-gb/documentation/articles/fundamentals-data-management-nosql-chappell/
- db-engines - Knowledge Base of Relational and NoSQL Database Management Systems
  http://db-engines.com/en/
- NoSQL Databases
  http://nosql-database.org/

How to Play with neo4j

A widely-used GraphDB

Hitachi Consulting
We make it happen. Better.

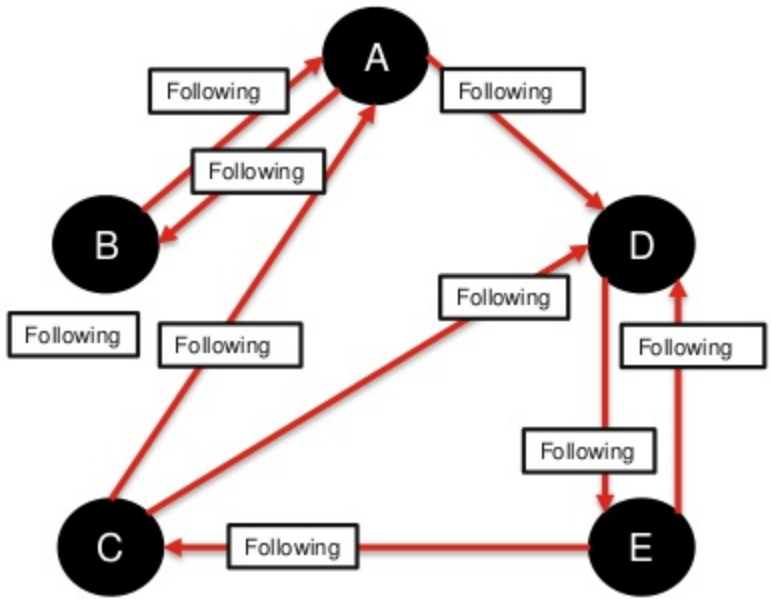


 |

How to Play with neo4j

A widely-used GraphDB

Hitachi Consulting
We make it happen. Better.

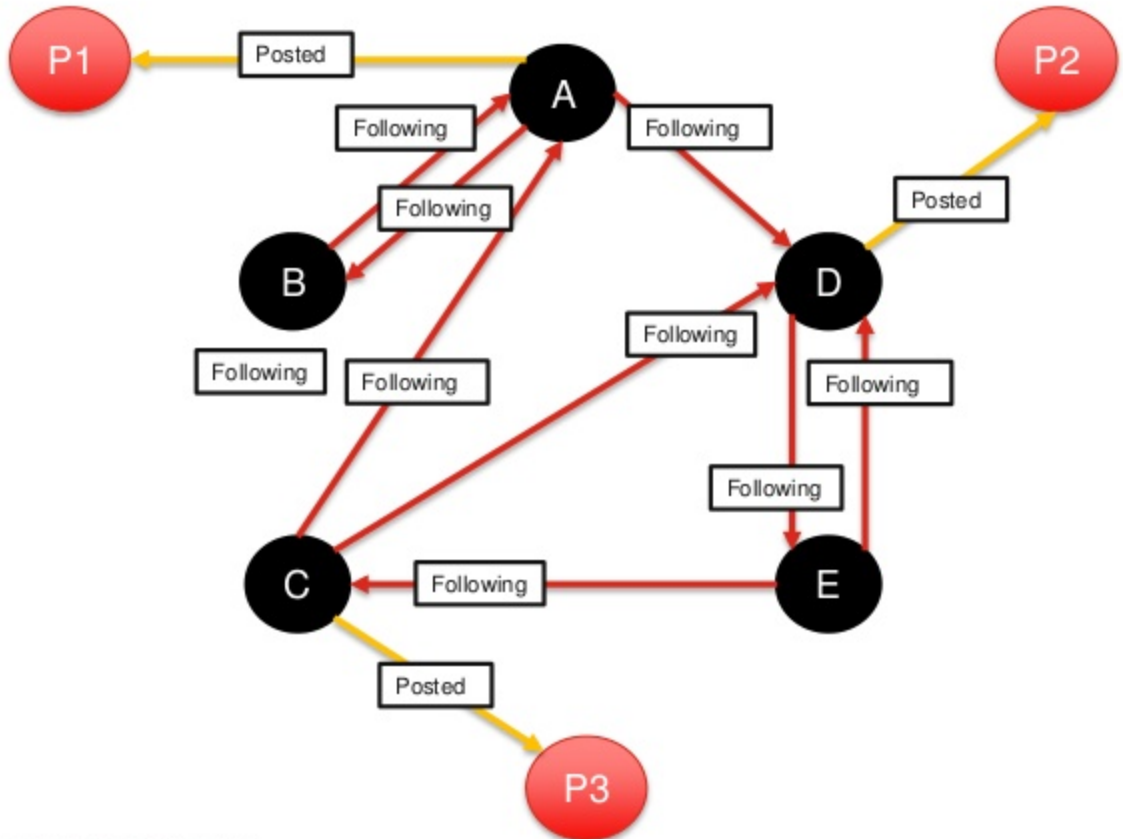


 |

Hitachi Consulting
We make it happen. Better.

How to Play with neo4j

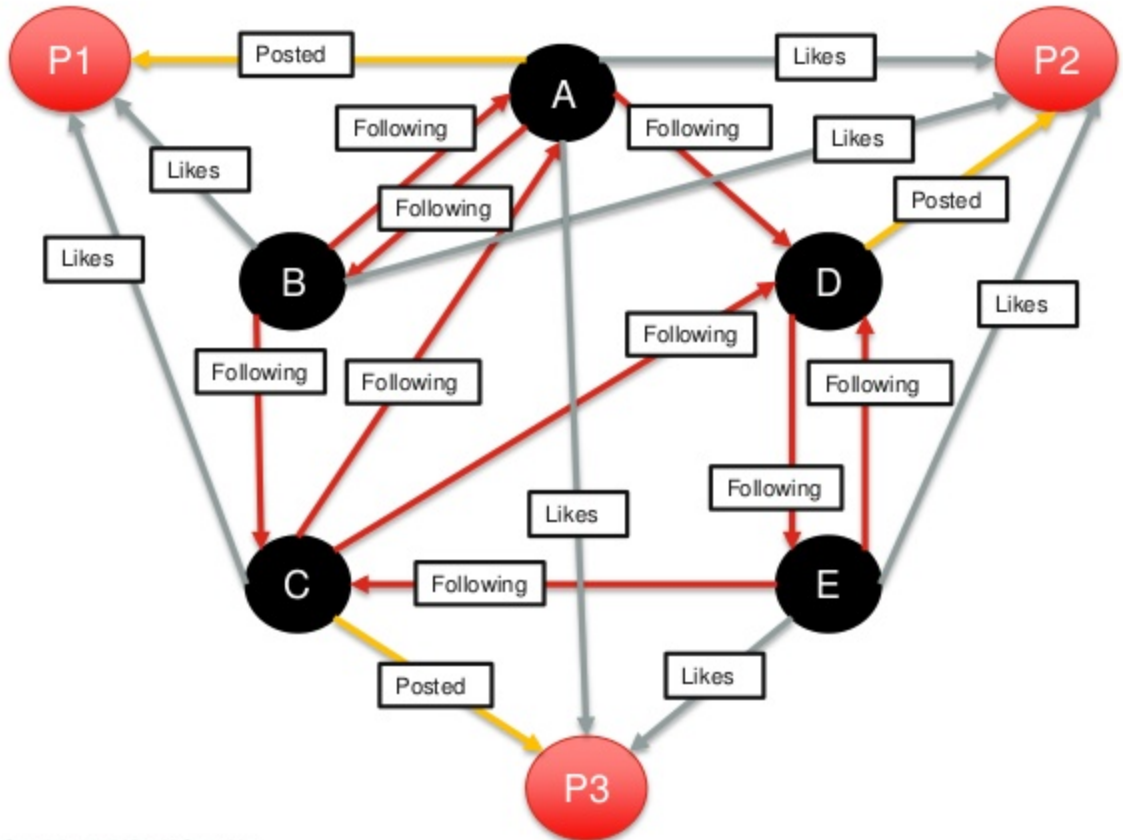
A widely-used GraphDB



 |

How to Play with neo4j

A widely-used GraphDB



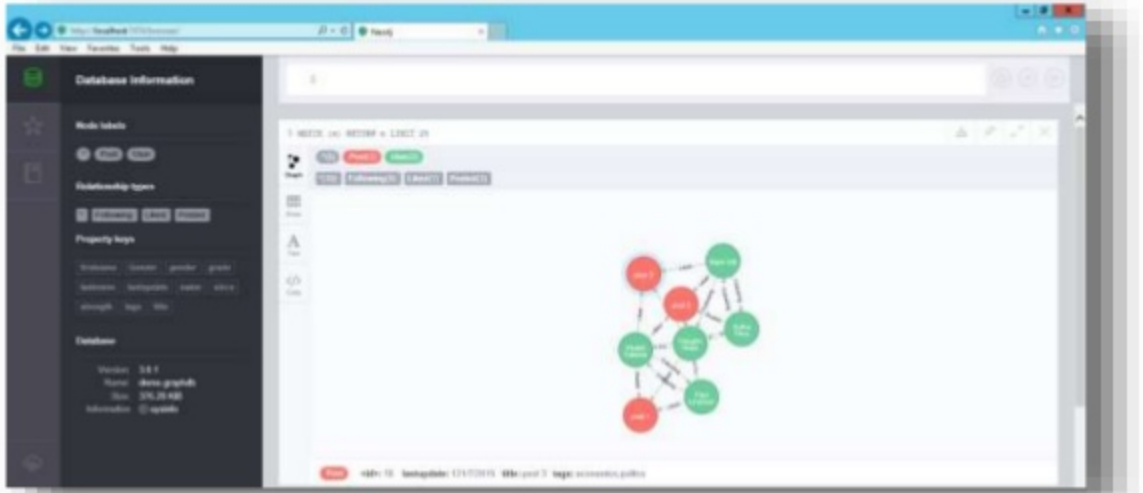
 |

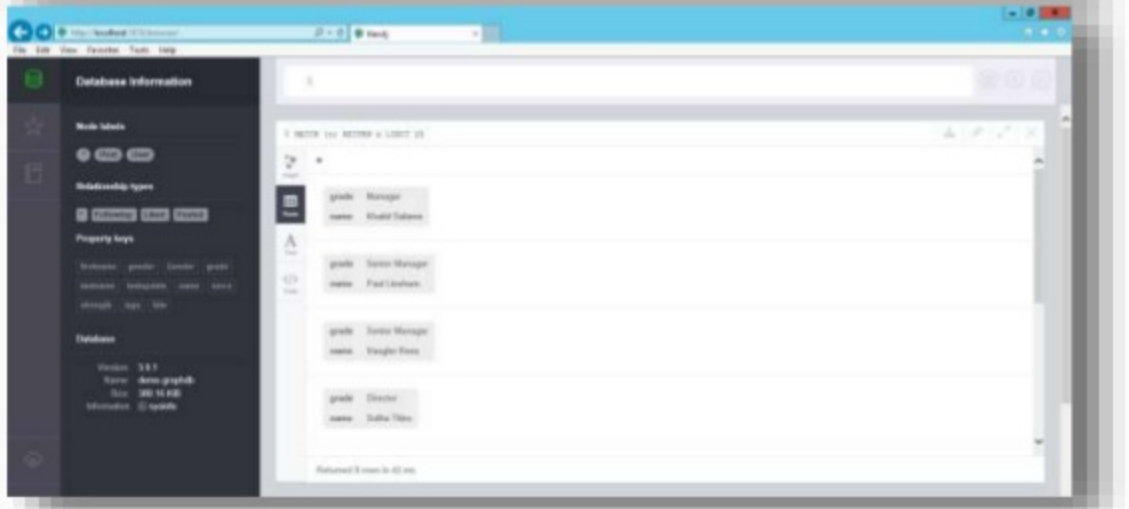
Hitachi Consulting
We make it happen. Better.

How to Play with neo4j

A widely-used GraphDB

```
CREATE

(a:User{name:"Khalid Salama", grade:"Manager"}),
(b:User{name:"Paul Lineham", grade:"Senior Manager"}),
(c:User{name:"Vaughn Rees", grade:"Senior Manager"}),
(d:User{name:"Sutha Thiru", grade:"Director"}),
(e:User{name:"Mark Hill", grade:"VP"}),

(a)-[:Following{since:'2014'}]->(d),
(a)-[:Following{since:'2014'}]->(b),
(b)-[:Following{since:'2010'}]->(a),
(d)-[:Following{since:'2011', strength:"high"}]->(e),
(e)-[:Following{since:'2014'}]->(d),
(e)-[:Following{since:'2015'}]->(c),
(c)-[:Following]->(d),
(c)-[:Following{since:'2013', strength:"low"}]->(a),
(b)-[:Following]->(c),

(p1:Post{title:"post 1", lastupdate:"01/01/2016", tags:['sports','life style']}),
(p2:Post{title:"post 2", lastupdate:"03/05/2015"}),
(p3:Post{title:"post 3", lastupdate:"121/7/2015", tags:['economics','politcs']}),

(a)-[:Posted]->(p1),
(d)-[:Posted]->(p2),
(c)-[:Posted]->(p3),

(b)-[:Liked]->(p1),
(c)-[:Liked]->(p1),
(a)-[:Liked]->(p2),
(b)-[:Liked]->(p2),
(e)-[:Liked]->(p2),
(a)-[:Liked]->(p3),
(e)-[:Liked]->(p3)
```





 |

How to Play with neo4j

A widely-used GraphDB

```
-- fetch one node
MATCH (u:User{name:"Khalid Salama"}) RETURN u

-- fetch an attribute of a node
MATCH (u:User{name:"Khalid Salama"}) RETURN u.grade

-- fetch nodes by conditions
MATCH (u:User{grade:"Senior Manager"}) RETURN u
--
MATCH (u:User)
WHERE u.grade = 'Senior Manager'
RETURN u
--
MATCH (u:User)
WHERE u.name =~ "Sutha.+"   -- START WITH, END WITH, CONTAIN, IN [,],
RETURN u
--
MATCH ()-[r:Posted]->(p:Post)
WHERE 'sports' IN p.tags
RETURN p

-- Whom khalid is following?
MATCH (x:User{name:"Khalid Salama"})-[r:Following]->(y:User)
RETURN x,r,y

-- Who is Following Khalid
MATCH (x:User{name:"Khalid Salama"})<-[r:Following]-(y:User)
RETURN x,r,y

-- Update
MERGE (u:User { name:"Khalid Salama" })
SET u.practice = "Data Insights & Analytics"
RETURN u
```


Hitachi Consulting
We make it happen. Better.

```
- Get Count of Posts
MATCH (p:Post) RETURN COUNT(p)
-- Get User Count By Grade
MATCH (u:User) RETURN u.grade, COUNT(u)
-- Get User and Followers
MATCH (u:User)<-[:Following]-(f:User)
RETURN u.name AS User,COLLECT(f.name) AS followrs,COUNT(f) AS Total

-- Constraint
CREATE CONSTRAINT ON (u:User) ASSERT u.name IS UNIQUE
-- Index
CREATE INDEX ON :User(grade)

-- Get users following each other
MATCH (u1:User)-[:Following]->(u2:User)-[:Following]->(u1)
RETURN u1.name,u2.name

-- Get Users likes a post posted by a follower
MATCH (u:User)-[:Liked]->(p:Post)<-[:Posted]-(u2:User)-[:Following]->(u)
RETURN u,p,u2

-- Get Following of Following
MATCH (u:User)-[:Following]->()-[:Following]->(u2:User)
Return u.name,COLLECT(DISTINCT u2.name)

-- Get User with max 3 steps from Paul
MATCH (u:User)-[:Following*..3]->(us:User{name:"Paul Lineham"})
Return u

-- Shortest path

MATCH
        (u1:User{name:"Mark Hill"}),
        (u2;User{name:"Paul Lineham"}),
        p=SHORTESTPATH((u1)-[:Following*..10]->(u2))
RETURN p

-- Get nodes having a property
MATCH(p)
WHERE EXSITS(p.tags)
```

http://neo4j.com/docs/developer-manual/current/#cypher-query-lang

Hitachi Consulting
We make it happen. Better.

My Background

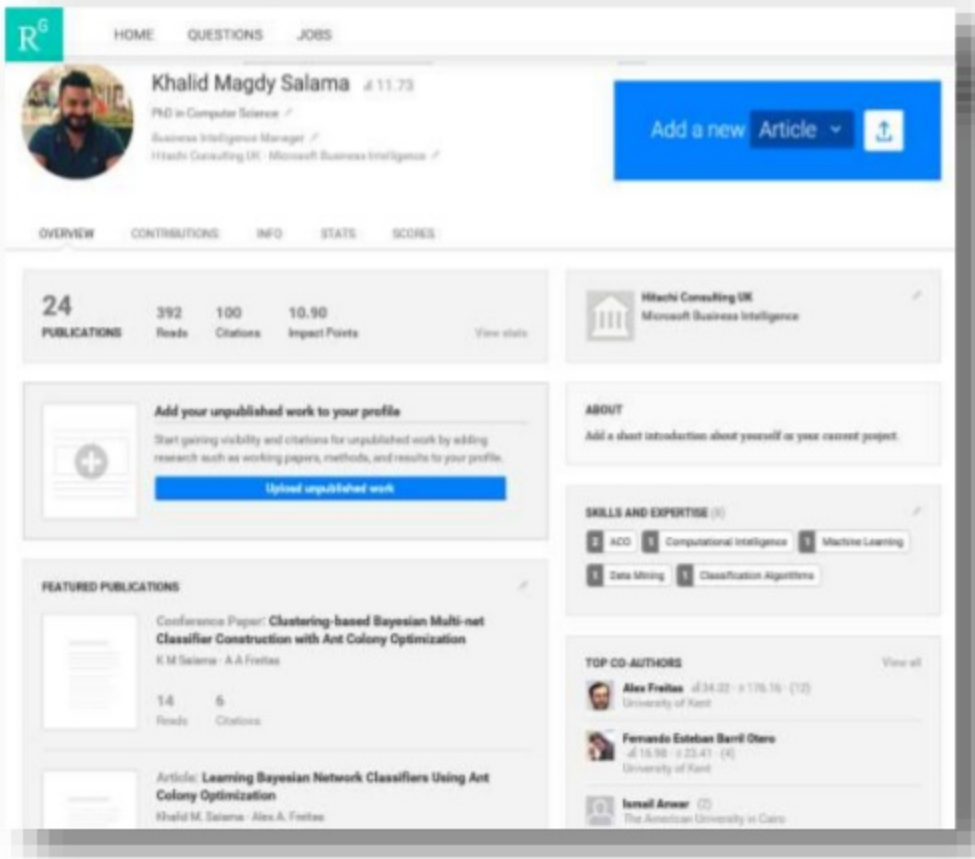
Applying Computational Intelligence in Data Mining

- Honorary Research Fellow, School of Computing , University of Kent.
- Ph.D. Computer Science, University of Kent, Canterbury, UK.
- M.Sc. Computer Science , The American University in Cairo, Egypt.

- 25+ published journal and conference papers, focusing on:
  - *classification rules induction,*
  - *decision trees construction,*
  - *Bayesian classification modelling,*
  - *data reduction,*
  - *instance-based learning,*
  - *evolving neural networks*, and
  - *data clustering*

- **Journals**: *Swarm Intelligence, Swarm & Evolutionary Computation, , Applied Soft Computing, and Memetic Computing.*

- **Conferences**: *ANTS, IEEE CEC, IEEE SIS, EvoBio, ECTA, IEEE WCCI and INNS-BigData.*



ResearchGate.org

 |

Hitachi Consulting
We make it happen. Better.

Thank you!

 |