AZ-203 Developer Training

Develop for the cloud and for Azure storage

Agenda

- NoSQL & NoSQL Database
- Cosmos DB
- Local Emulator
- Multiple APIs and Data Models
- Request Units
- Exceeding Reserved Throughput Limits
- Whiteboarding the Cost
- Request Unit Calculator
- Achieving Elastic Scale
- Partition Key
- Cross Partition Queries
- Replication-Why
- Turnkey Global Distribution
- NoSQL & NoSQL Database
- Replication and Consistency
- Consistency Levels
- Document Database
- Data Modeling: Relational vs. Document
- Cosmos DB Resource Model
- Resource Properties, Self-Links, and URIs
- Data Migration Tool
- Rich Query with SQL
- SQL Operators and Functions
- Client Development
- Introducing the .NET SDK for the SQL API
- Indexing Policies

Agenda

- Users, Permissions, and Resource Tokens
- Server-side Programming Model
- Writing Server-side Code
- Working with Triggers
- Creating User-Defined Functions (UDFs)
- Table API
- Cosmos DB Graph Database
- Graph Database Scenarios
- Vertices and Edges
- Populating the Graph
- Bi-Directional Relationships
- Writing Gremlin Queries
- Demo:Busy World Traveler
- Multi-Model Comic Book Catalog
- Azure Table Storage

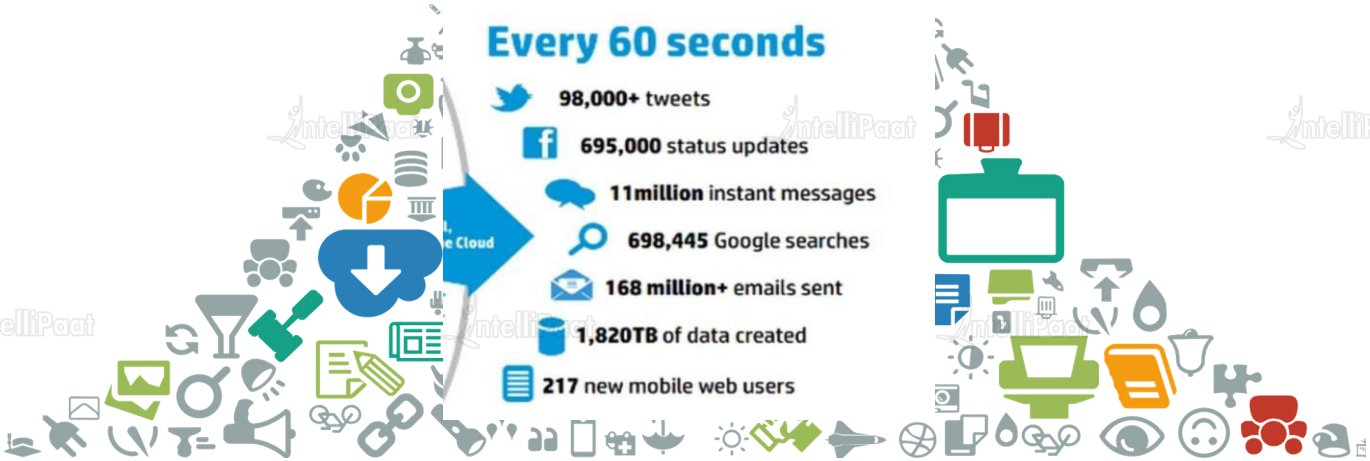
NoSQL & NoSQL Database

What Is NoSQL?

What Is NoSQL?

3Vs

- Volume
- Variety
- Velocity



What Is a NoSQL Database?

Distributed

Replicas ensure high throughput/availability, and low latency

Scale-out

Horizontal partitioning enables virtually limitless storage and throughput

Schema-free

Document, table, graph, and columnar data models

Cosmos DB

What Is Cosmos DB?

Evolution of DocumentDB

Scalable NoSQL document database Low latency (single-digit millisecond)

Virtually unlimited scale

Scale storage with server-side partitioning Scale throughput with variable request units

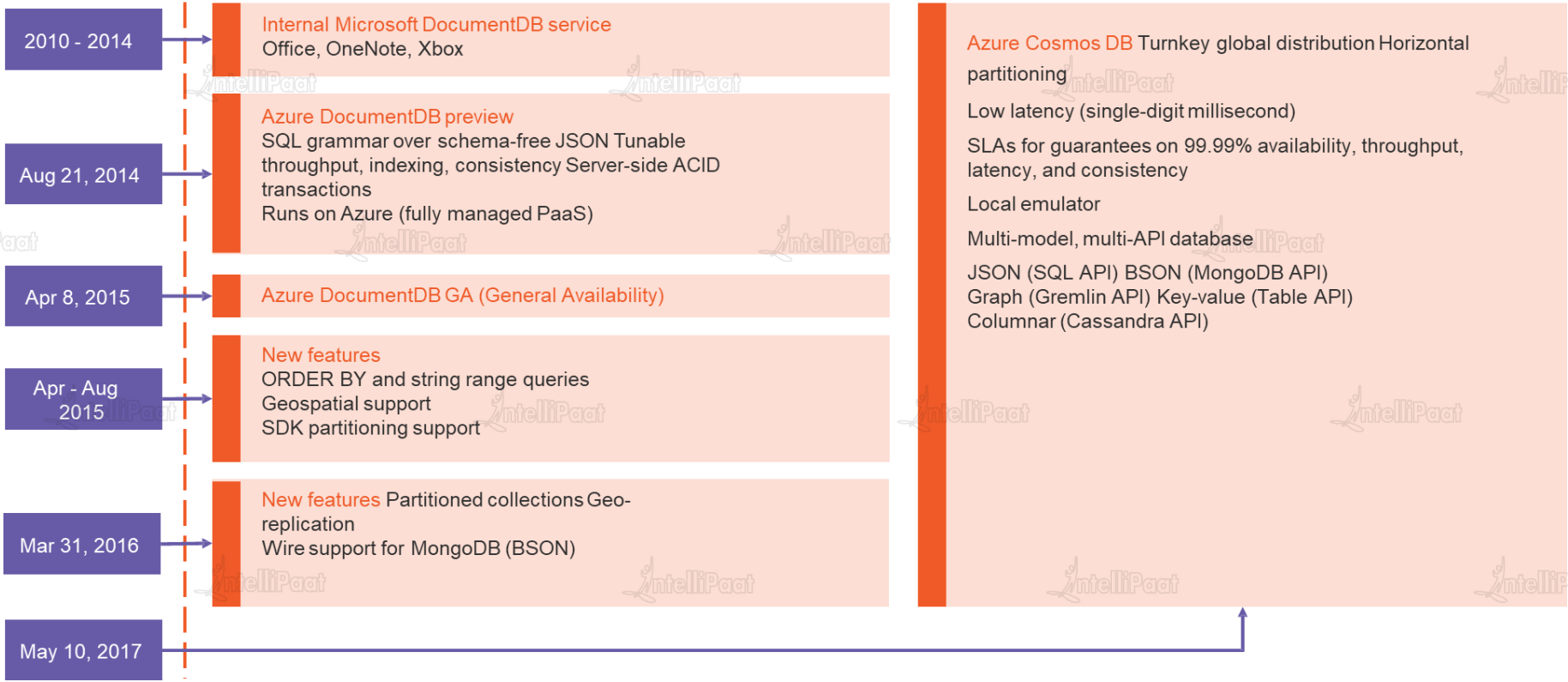
Turnkey global distribution

Point-and-click control over where your data gets geo-replicated

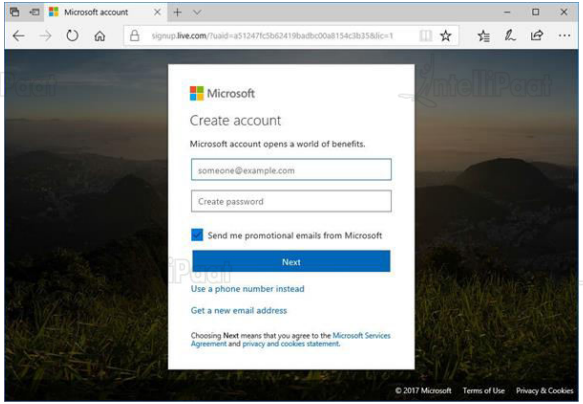
Multi-model / Multi-API

No longer exclusively a document database Also supports tables, graph, and columnar

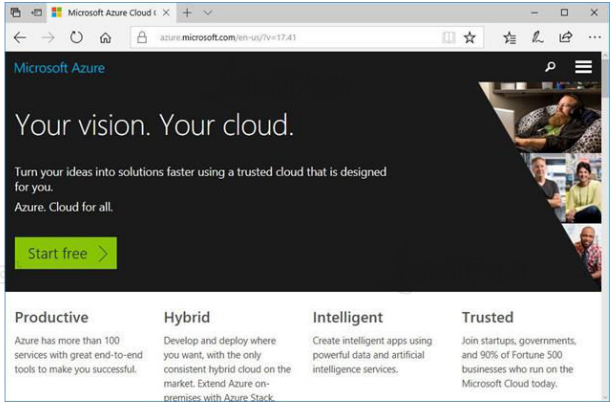
Cosmos DB Evolution



Getting Started



+



Get a Microsoft account
http://signup.live.com

Get a Microsoft Azure subscription
http://azure.microsoft.com

Or,

Download the local emulator
https://aka.ms/cosmosdb-emulator

Local Emulator

Introducing the Local Emulator

- Emulate Cosmos DB in a local development environment
  - Supports identical functionality as Azure Cosmos DB in the cloud

- No need for:
  - Azure subscription
  - Cosmos DB account
  - Internet connection

- Develop and test locally
  - Incur no costs
  - Deploy to the cloud when ready

Hands-On

- Creating a Cosmos DB account
- Installing the Cosmos DB local emulator
- Creating a collection
- Creating documents

Multiple APIs and Data Models

Multiple APIs and Data Models

Atom Record Sequence — ARS

It's all ARS under the covers, projected as any desired data model

Document

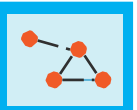
SQL API (JSON) MongoDB API (BSON)

Key-Value

Table API

(replaces Azure Table Storage)

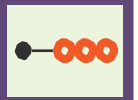
Graph



Gremlin API

(graph traversal language)

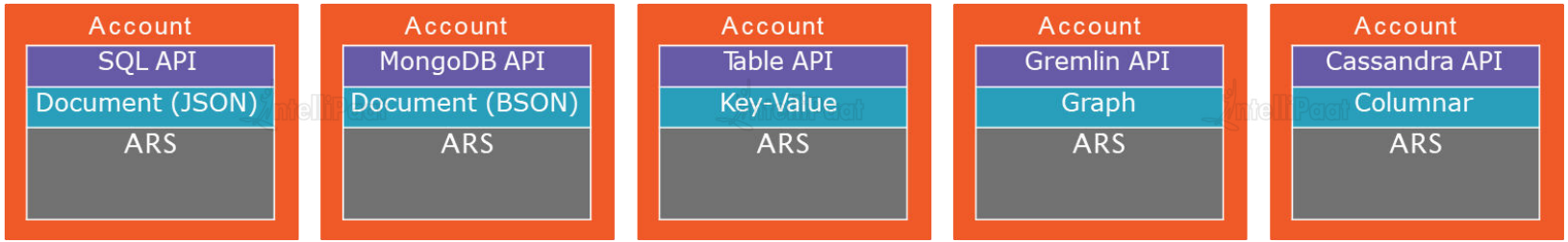
Columnar



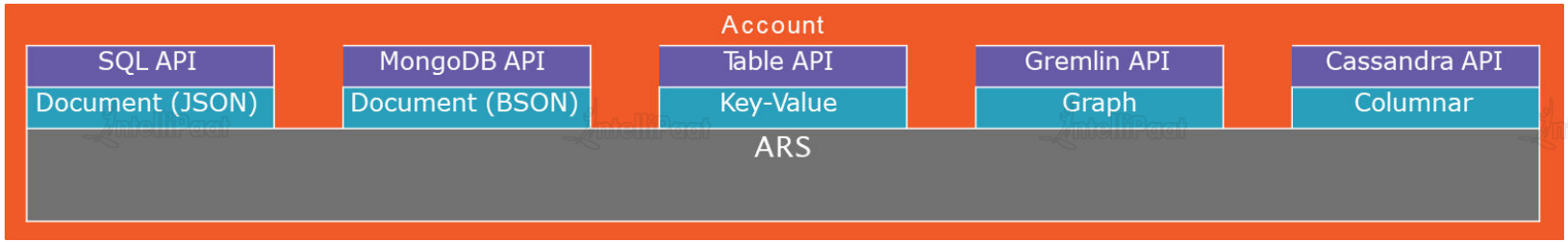
Cassandra API

Multiple APIs and Data Models

- Today



- In the future



Course Structure

Concepts

- Throughput
- Partitioning
- Global distribution

How-to

- SQL API
- Table API
- Gremlin API

Measuring Performance

Latency

How fast is the response for a given request?

Throughput

How many requests can be served within a specific period of time?

Request Units

Introducing Request Units

Throughput Currency

Blended measure of computational cost (CPU, memory, disk I/O, network I/O)

All Requests are Not Equal

Every Cosmos DB response header shows the RU charge for the request

Request Units are Deterministic

The same request will always require the same number of request units

Reserving Request Units

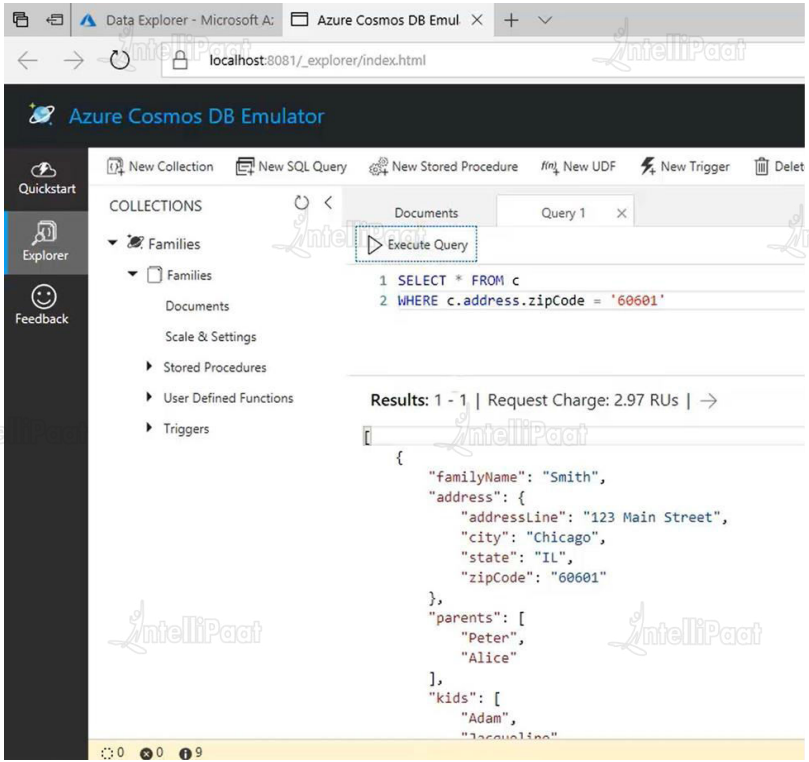
Provision request units per second (RU/s)

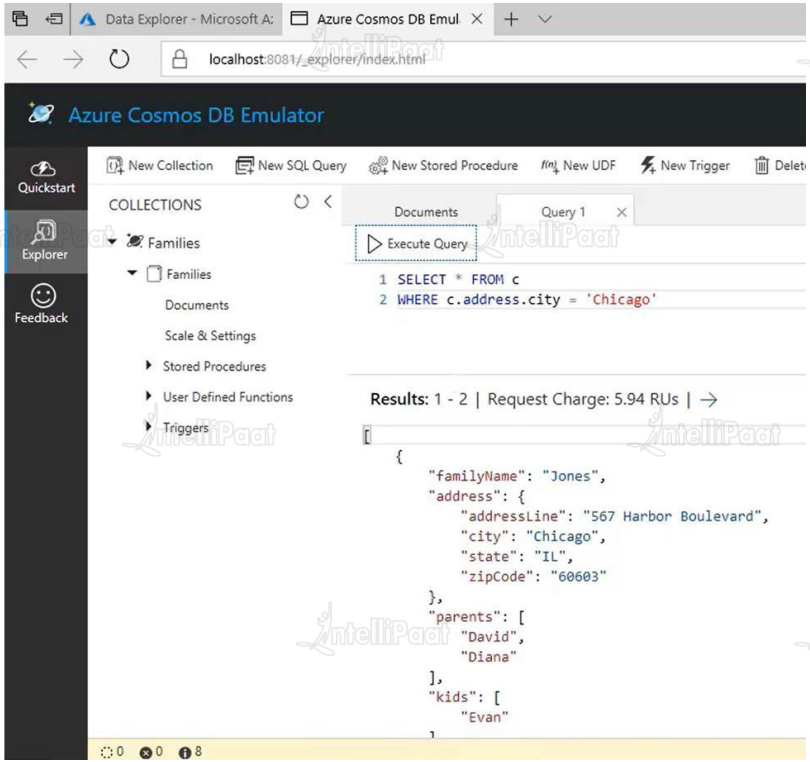
How many request units (not requests) per second are available to your application

Exceeding reserved throughput limits

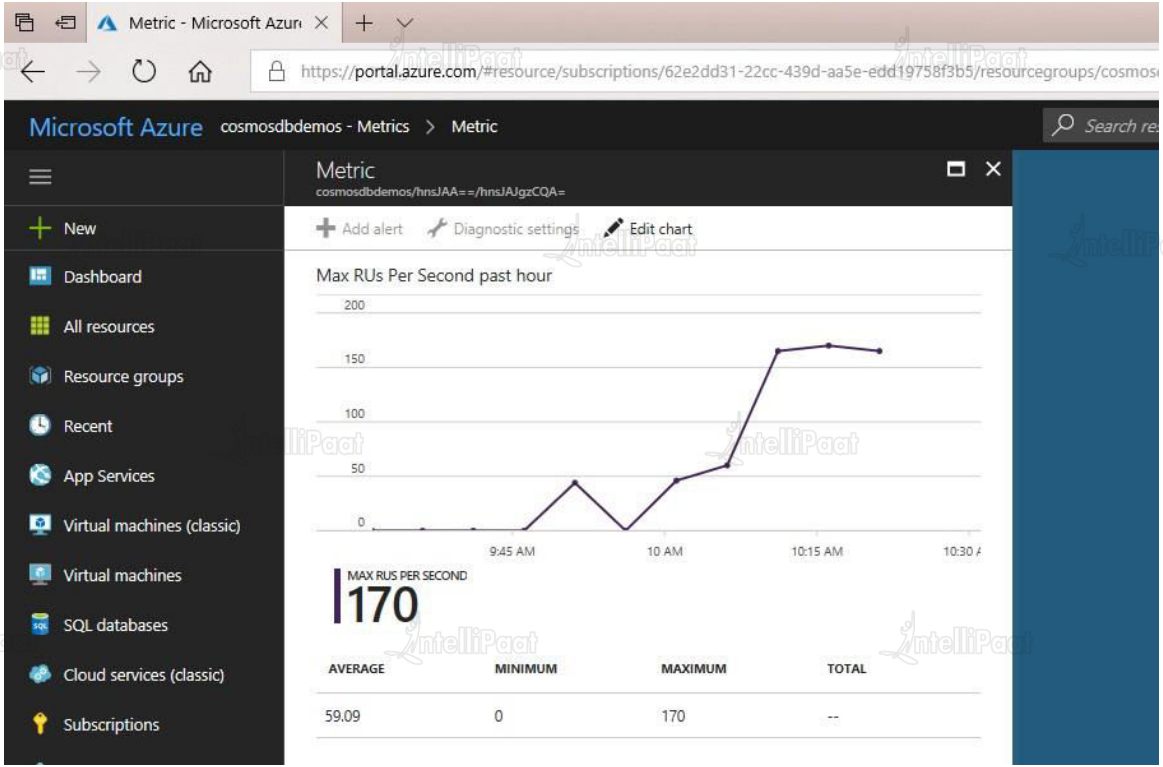
Requests are “throttled”

Monitoring Request Unit Consumption





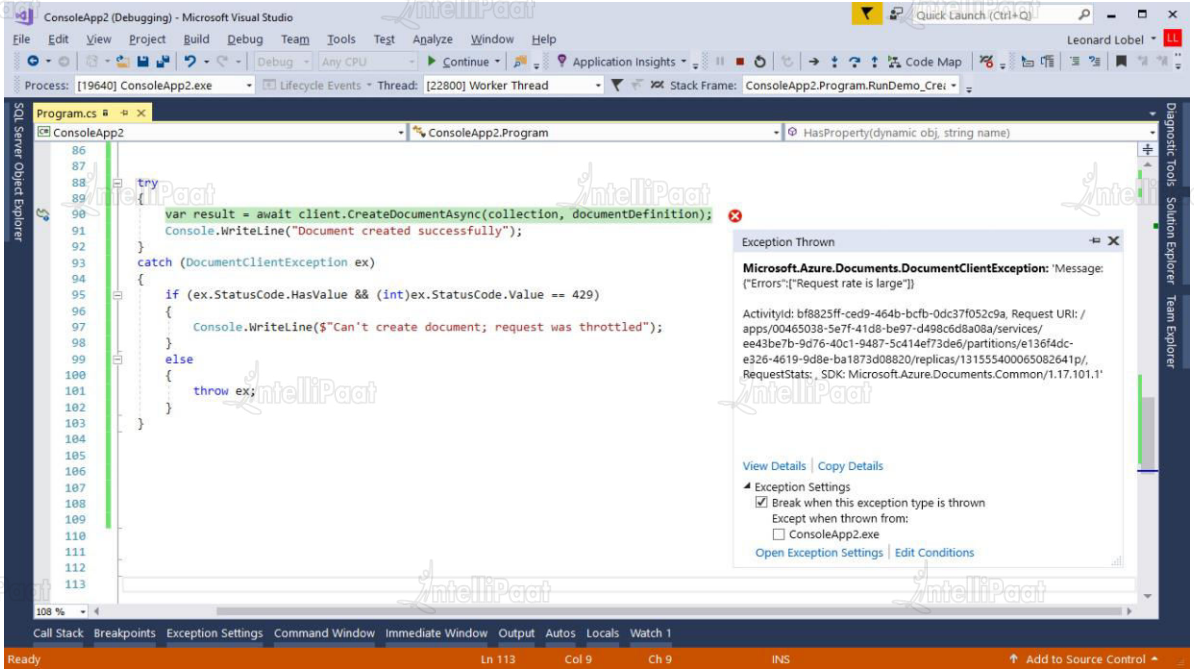
Monitoring Request Unit Consumption



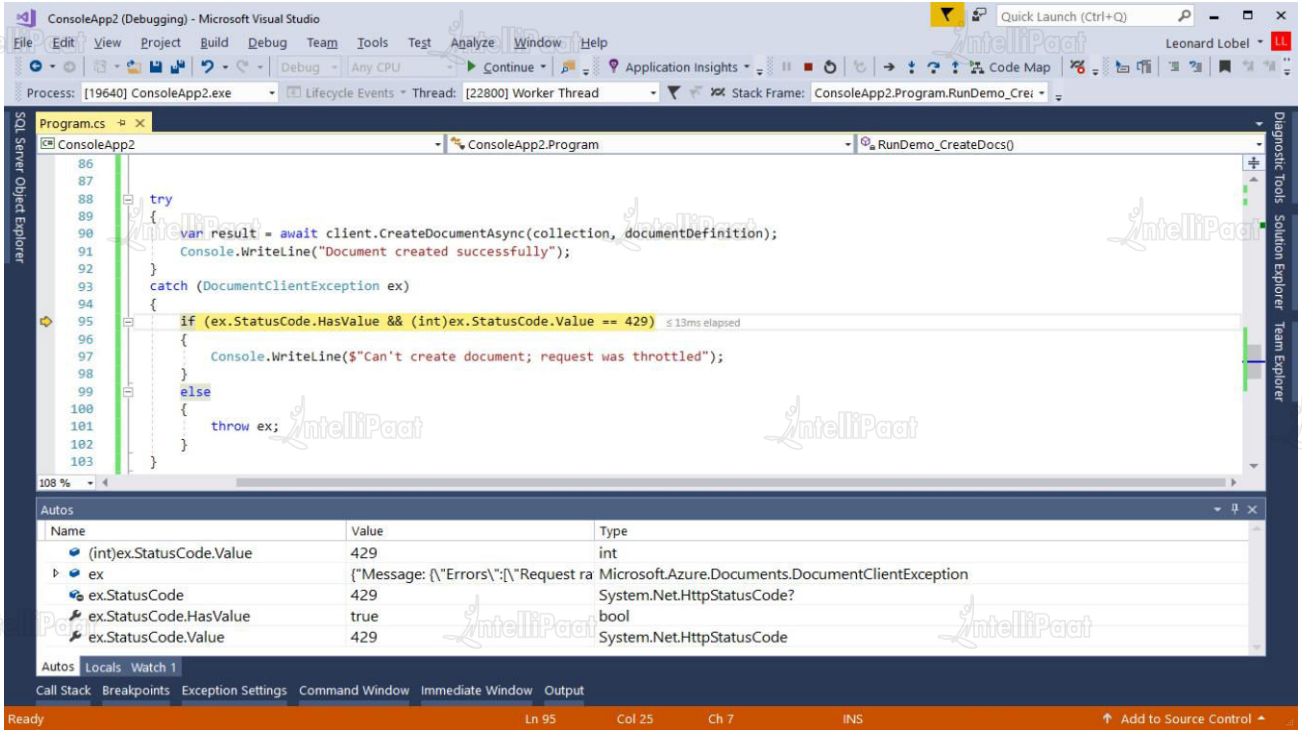
Monitoring Request Unit Consumption

Exceeding Reserved Throughput Limits

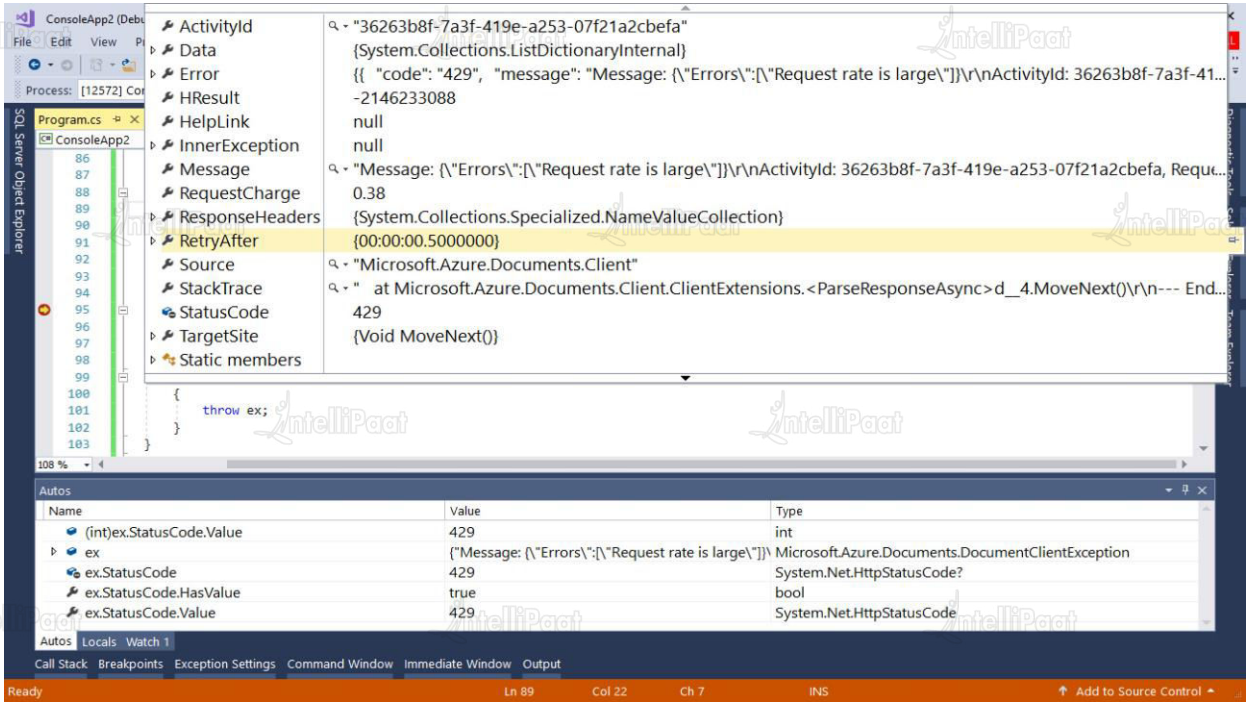
Exceeding Reserved Throughput Limits



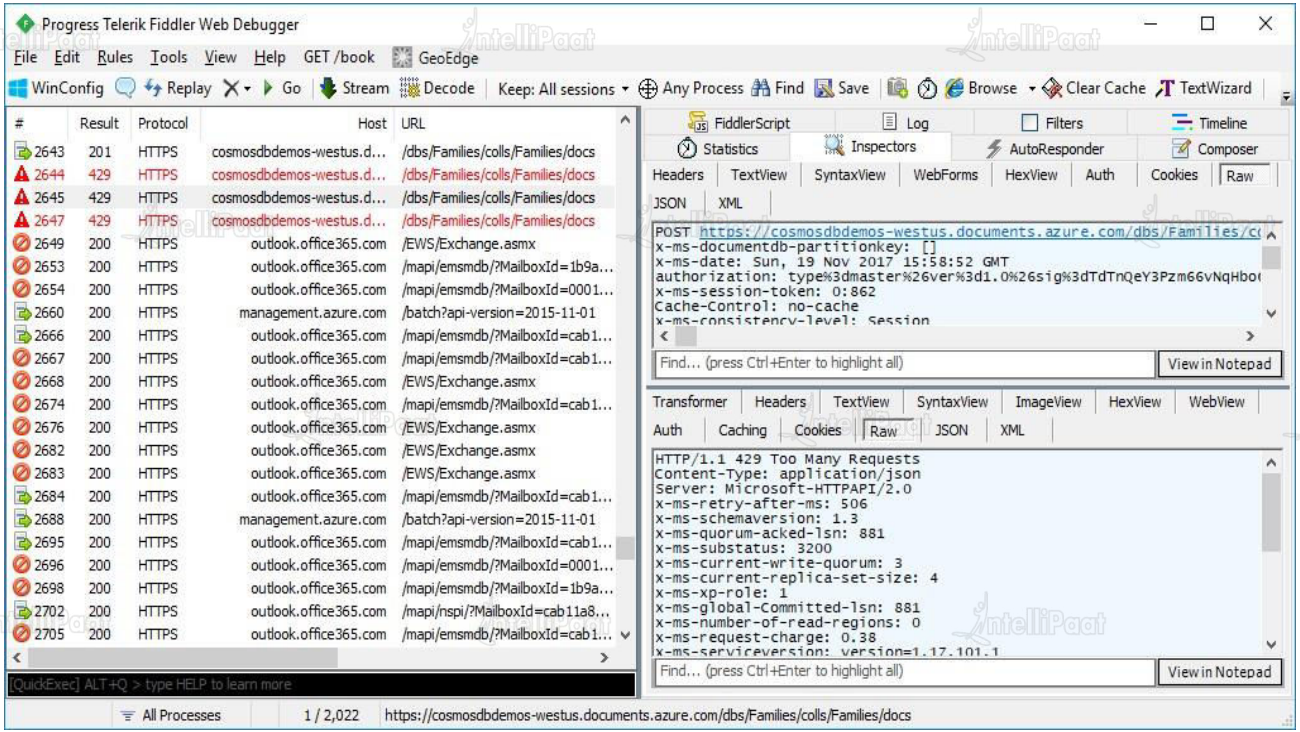
Exceeding Reserved Throughput Limits



Exceeding Reserved Throughput Limits



Exceeding Reserved Throughput Limits



Whiteboarding the Cost

Whiteboarding the Cost

Application checklist

1. What does a typical item look like?
2. What are the typical queries that users will run?
3. How many writes per second are required?
4. How many queries per second are required?
5. What is the acceptable consistency level?
6. What is the indexing policy?

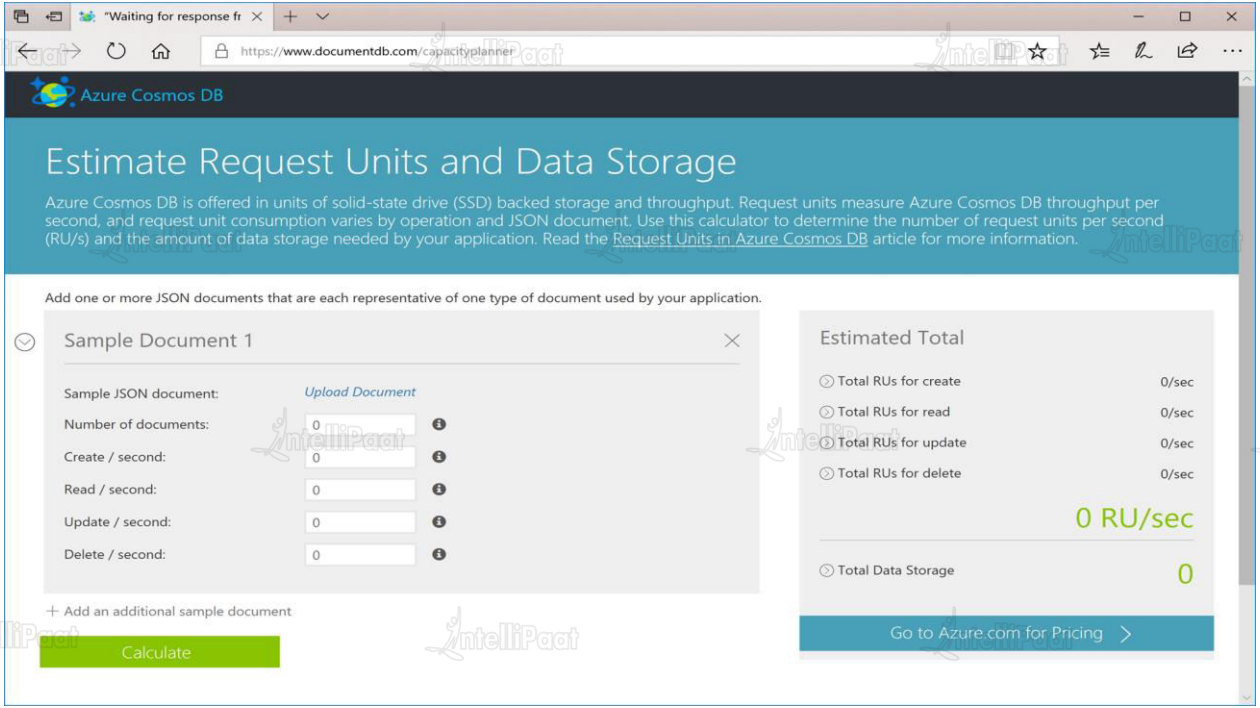
Estimating throughput needs

Item size	Reads/second	Writes/second	Request units
1 KB	500	100	(500 * 1) + (100 * 5) = 1,000 RU/s
1 KB	500	500	(500 * 1) + (500 * 5) = 3,000 RU/s
4 KB	500	100	(500 * 1.3) + (100 * 7) = 1,350 RU/s
4 KB	500	500	(500 * 1.3) + (500 * 7) = 4,150 RU/s
64 KB	500	100	(500 * 10) + (100 * 48) = 9,800 RU/s
64 KB	500	500	(500 * 10) + (500 * 48) = 29,000 RU/s

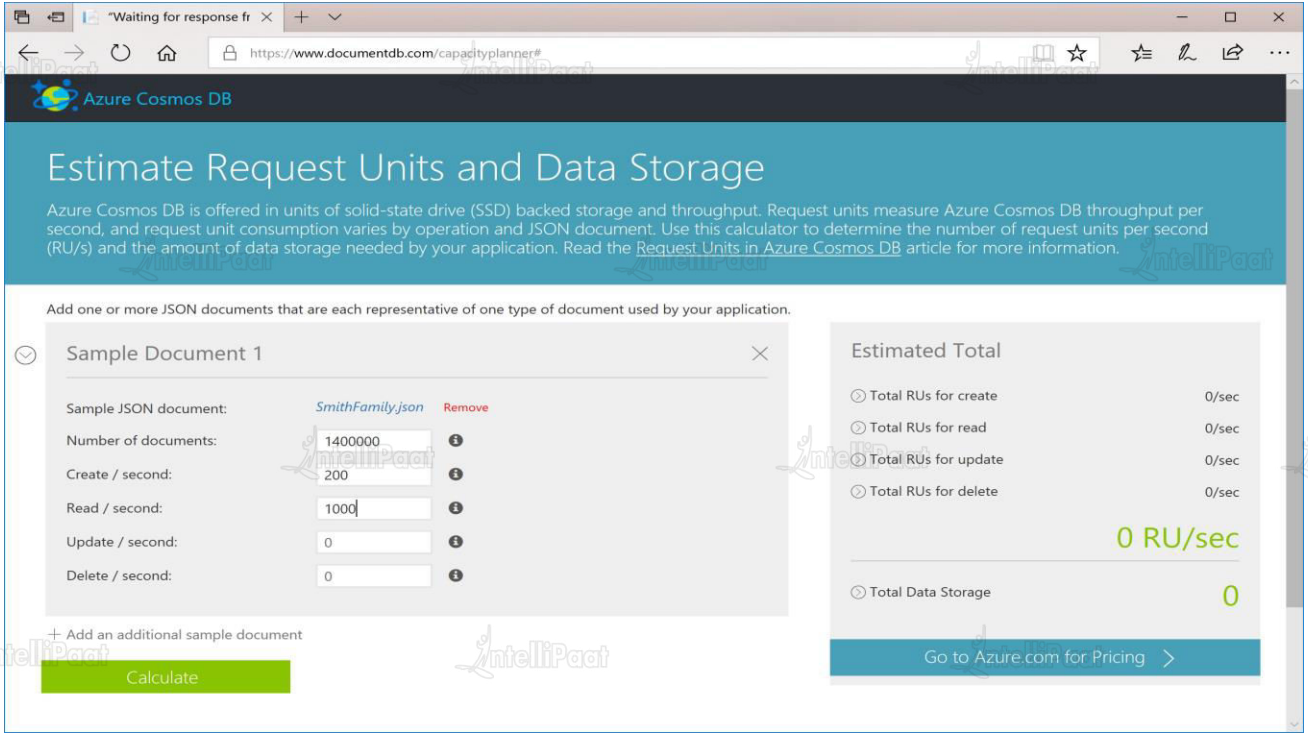
* Based on Session consistency indexing policy set to None.

Request Unit Calculator

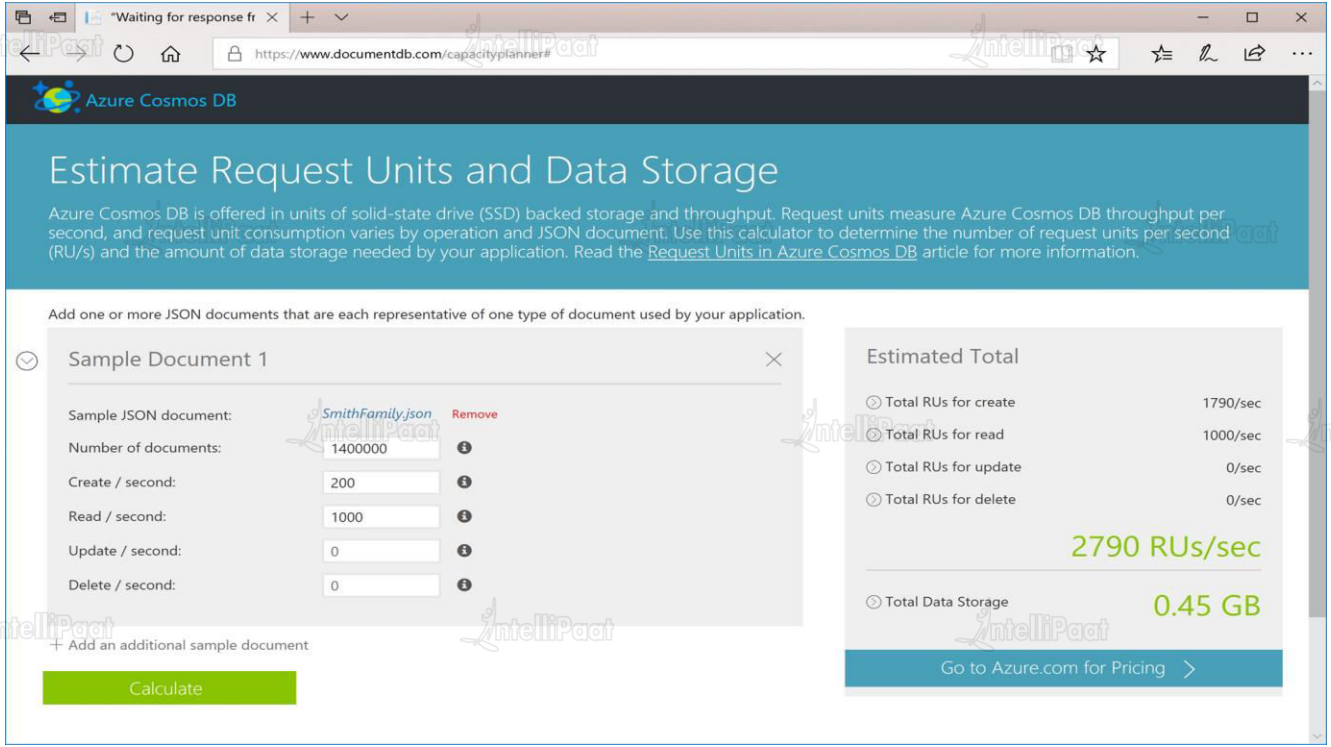
Request Unit Calculator



Request Unit Calculator



Request Unit Calculator



Pricing

SSD Storage

	A	B	C
1	1 GB	10 GB	
2	$ 0.25	$ 2.50	/month

Throughput

	A	B	C
1	100 RU/s	400 RU/s	
2	$ 0.008	$ 0.032	/hour
3	$ 0.192	$ 0.768	/day
4	$ 5.856	$ 23.424	/month

Achieving Elastic Scale

Achieving Elastic Scale

What Is Partitioning?

- Massive scale-out within a container

Unlimited Containers

- Logical resource composed of multiple partitions

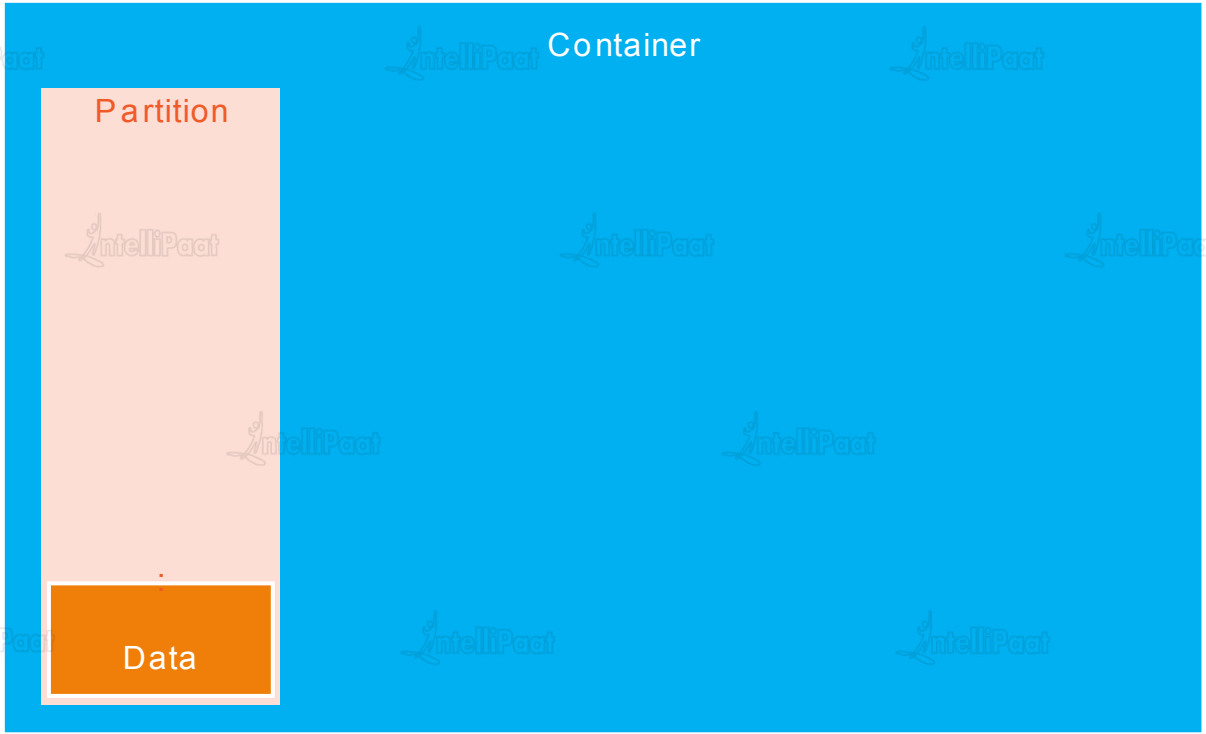
Partitions

- Physical fixed-capacity data buckets

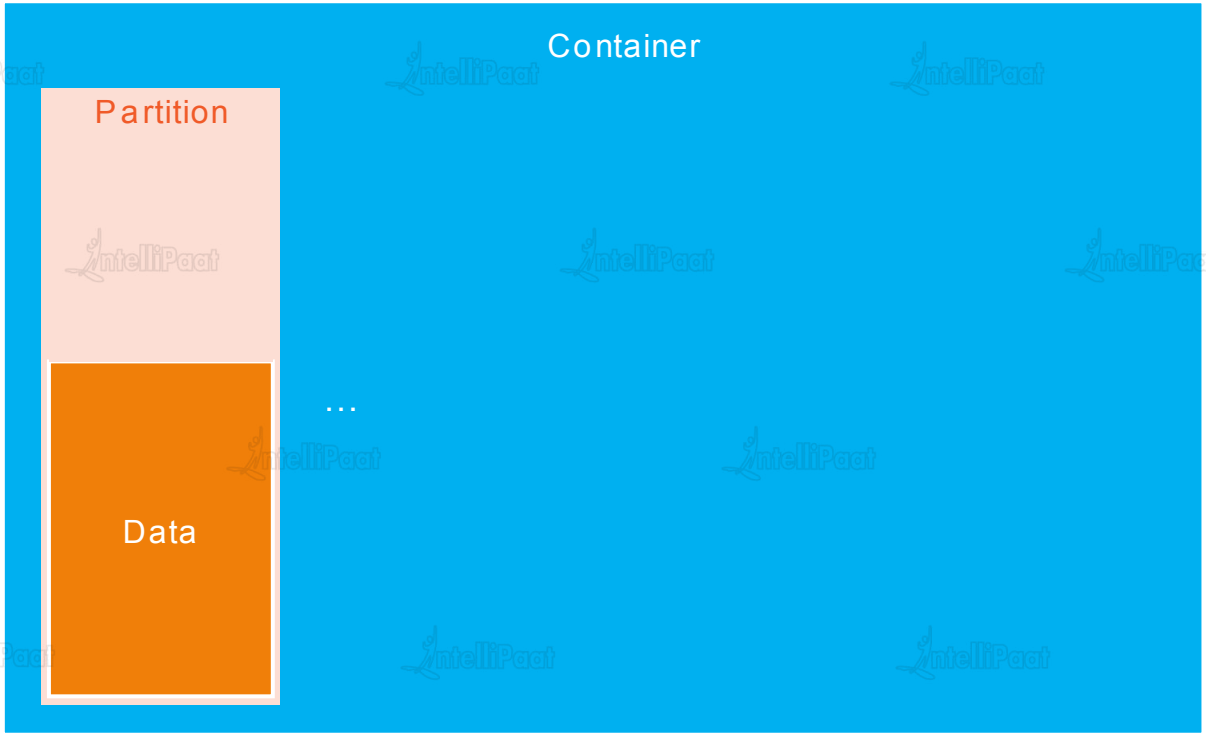
Automated Scale-out

- Cosmos DB transparently splits partitions to manage growth

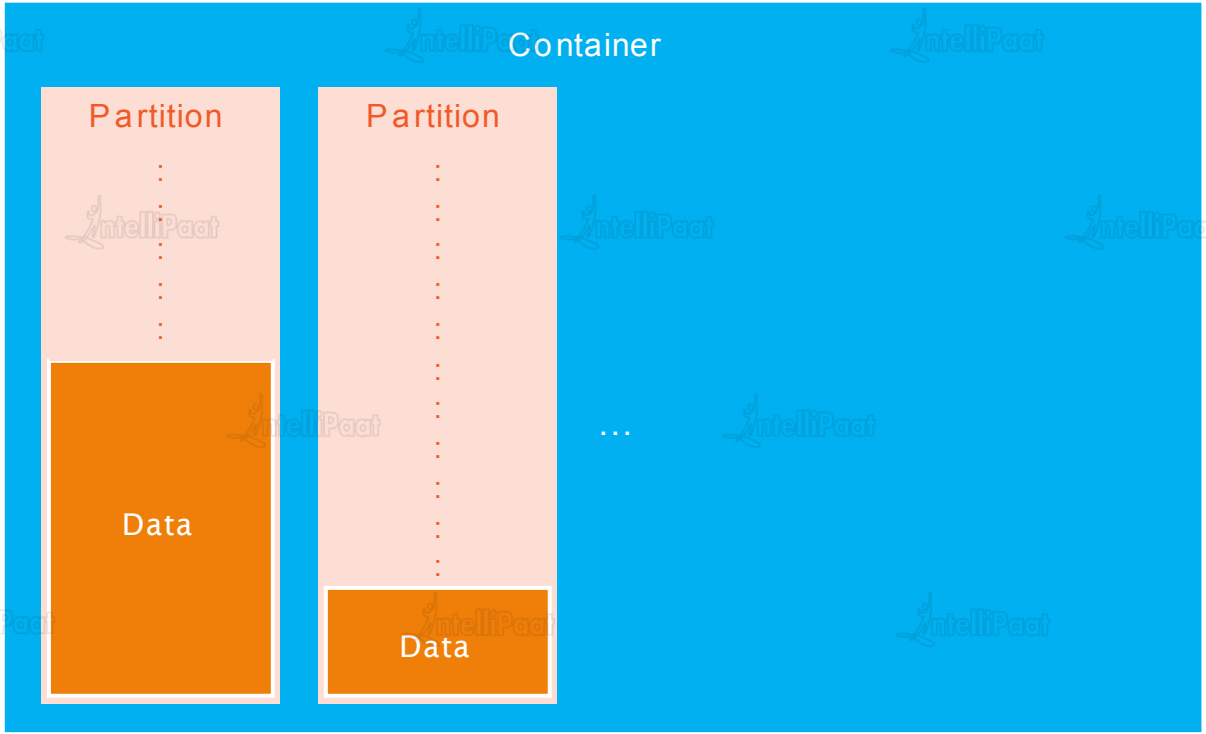
Achieving Elastic Scale



Achieving Elastic Scale



Achieving Elastic Scale



Partition Key

Selecting a Partition Key

Choosing the best partition key

- The right choice will deliver massive scale

Partition key values are hashed

- Hashed value determines the physical partition for storing each item

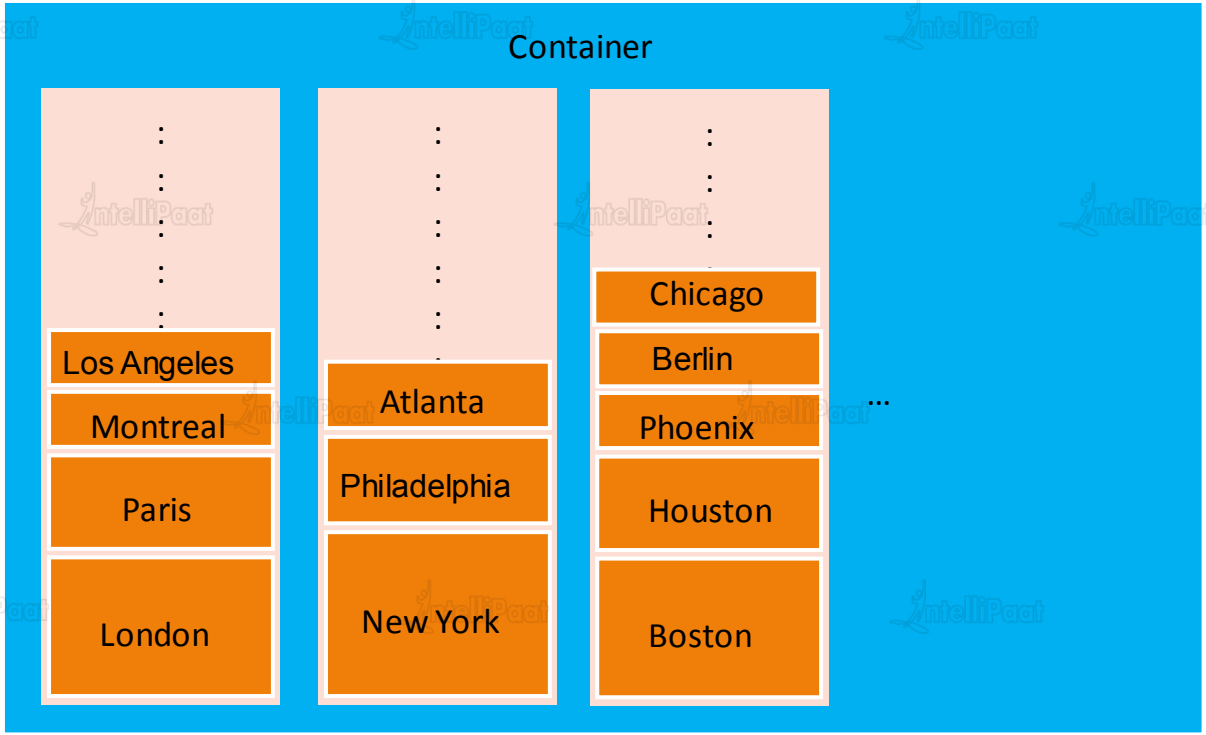
Partitions host multiple partition keys

- Items with the same partition key value are physically stored together on the same partition

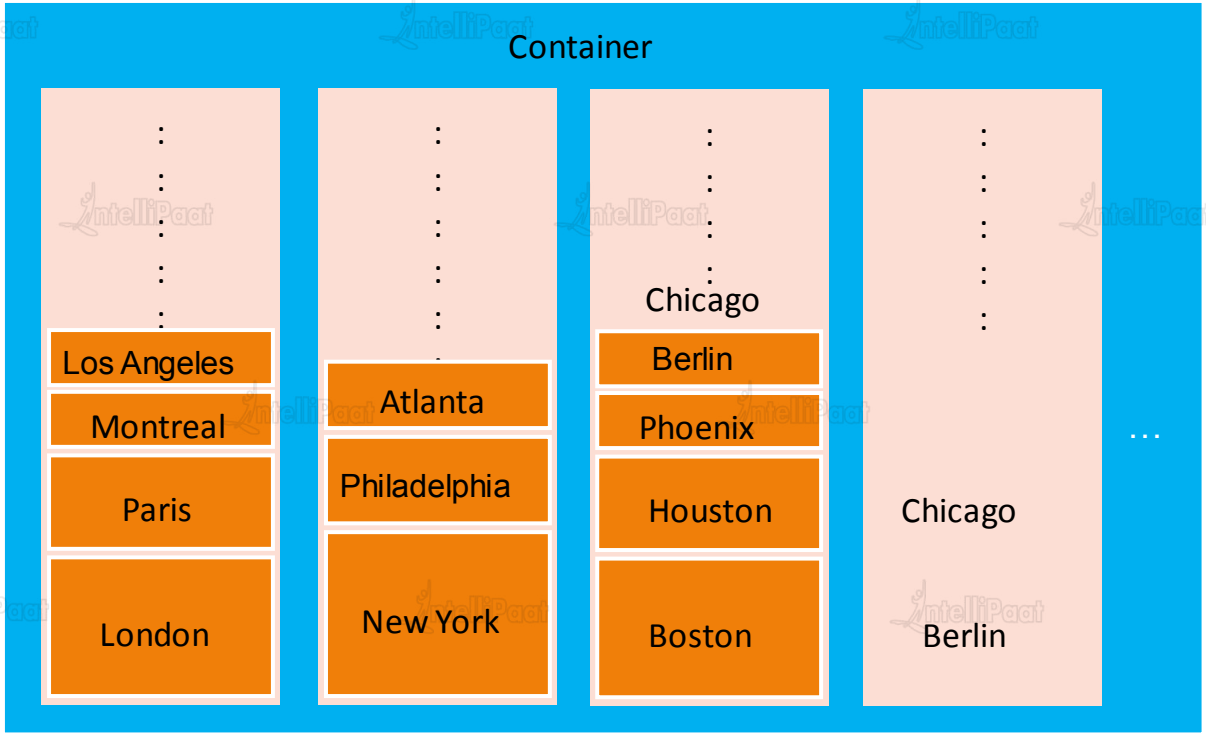
Two primary considerations

- Boundary for query and transactions
- No storage or performance bottlenecks

Selecting a Partition Key



Selecting a Partition Key



Choosing the Right Partition Key

Driven by data access patterns

Choose a property that groups commonly queried/updated items together

- User Profile Data
- User ID

- IoT

(e.g., device state)

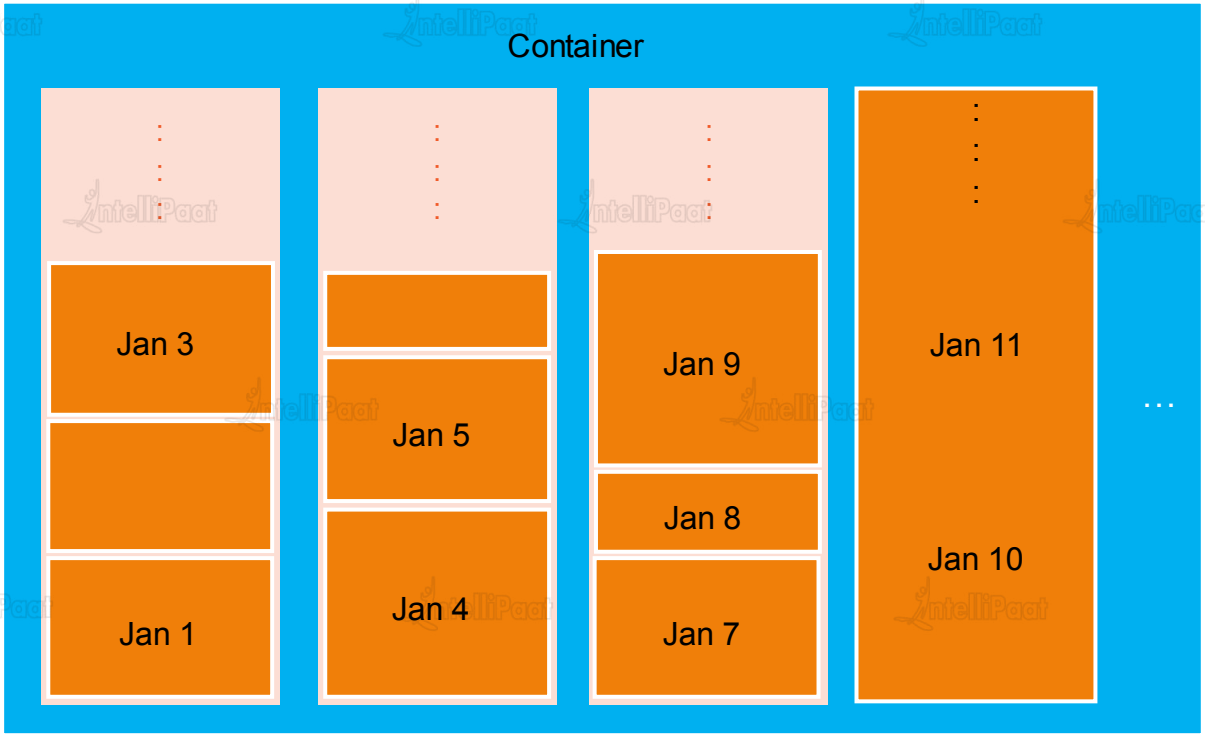
Device ID

- Multi-tenant Architecture
- Tenant ID

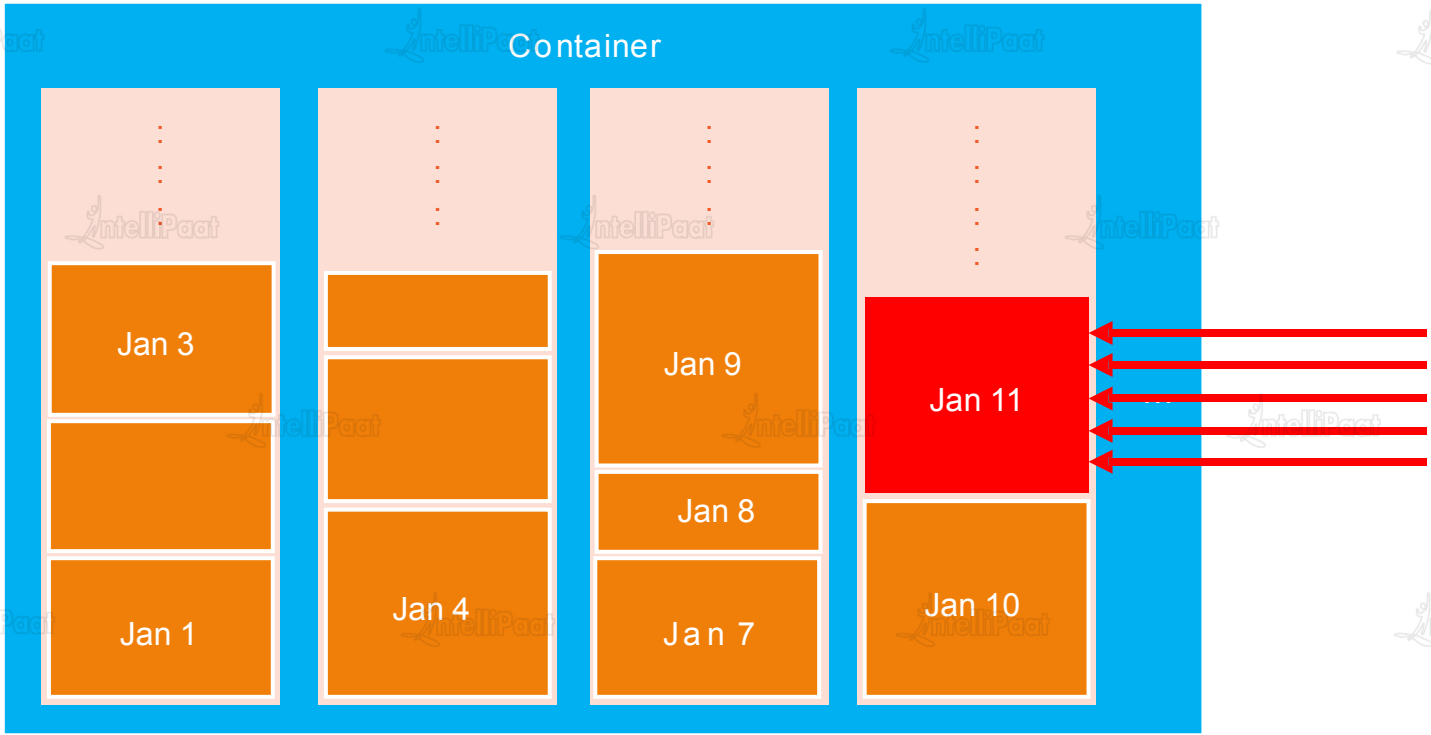
Choosing the Right Partition Key

- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
  - Partitioning by creation date
    - Bad idea! All writes of the day are directed to the same partition

Choosing the Right Partition Key



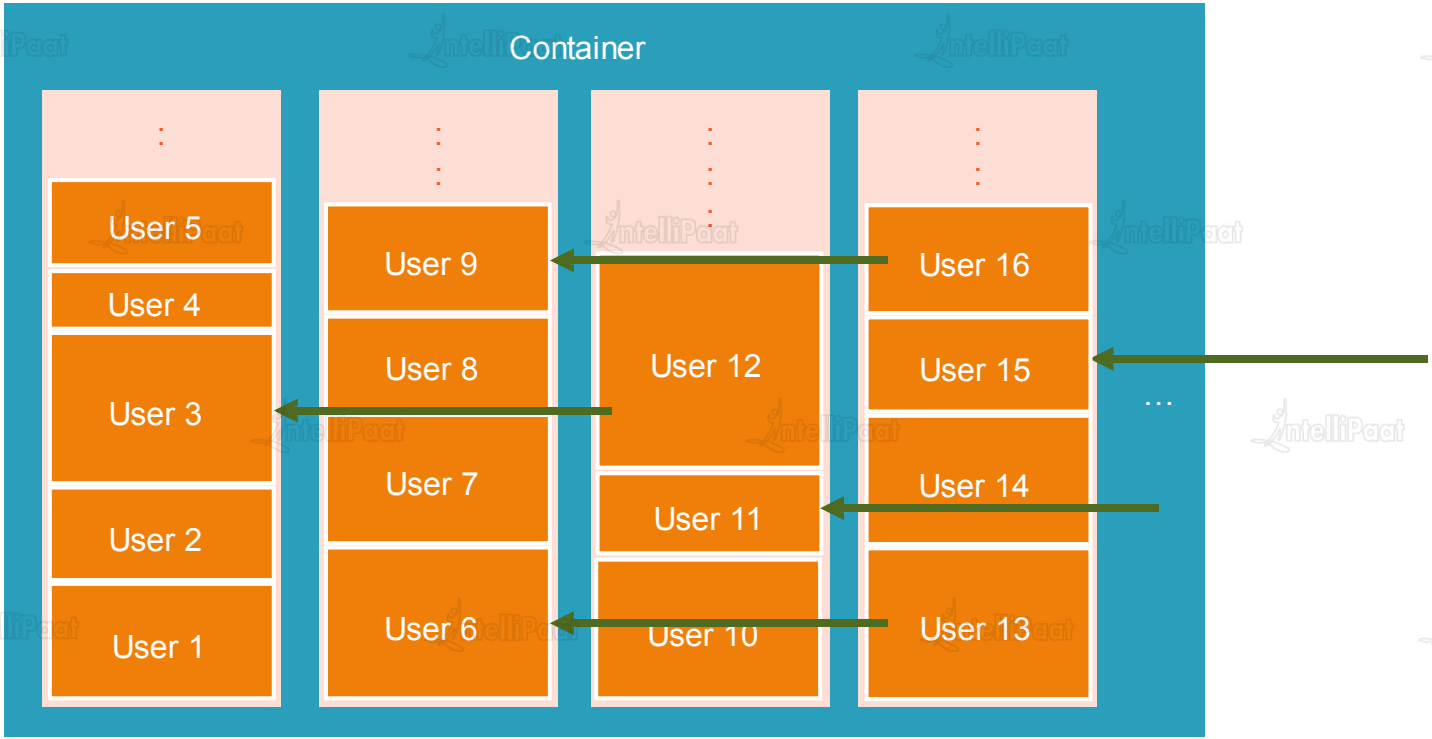
Choosing the Right Partition Key



Choosing the Right Partition Key

- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
  - Partitioning by creation date
    - Bad idea! All writes of the day are directed to the same partition
  - Partitioning by user ID
    - Much better! Writes are directed to different partitions per user

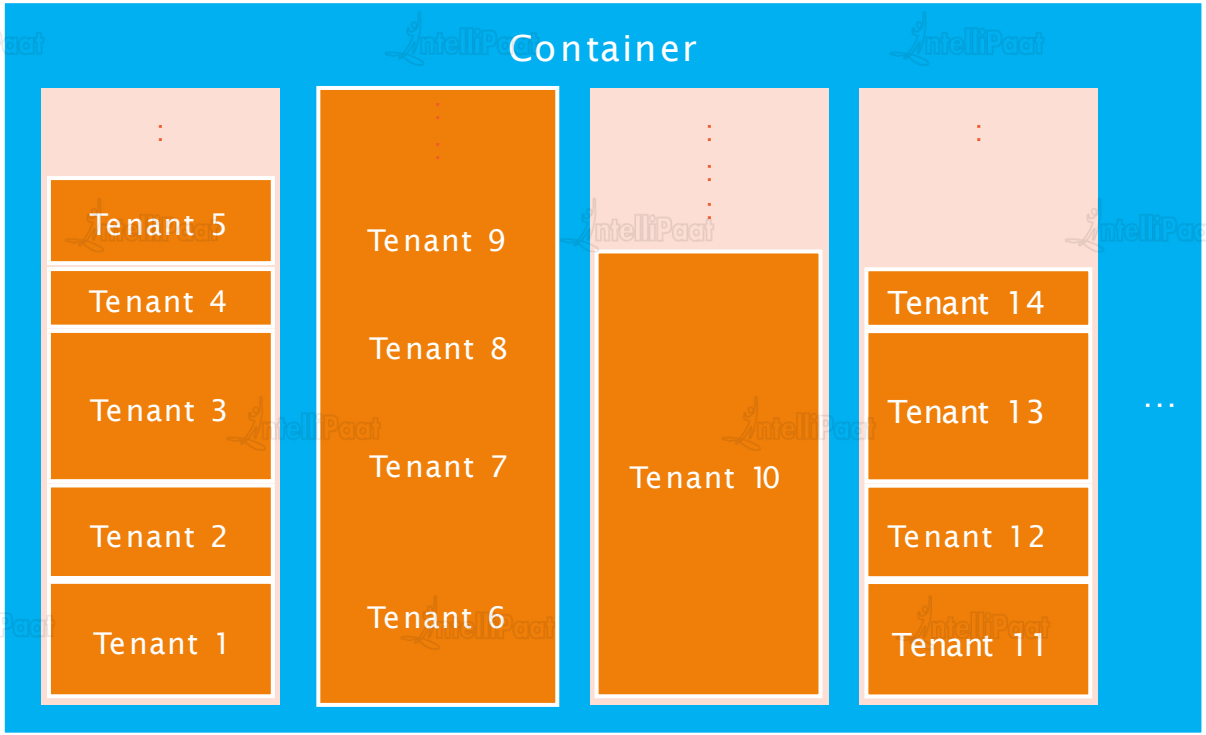
Choosing the Right Partition Key



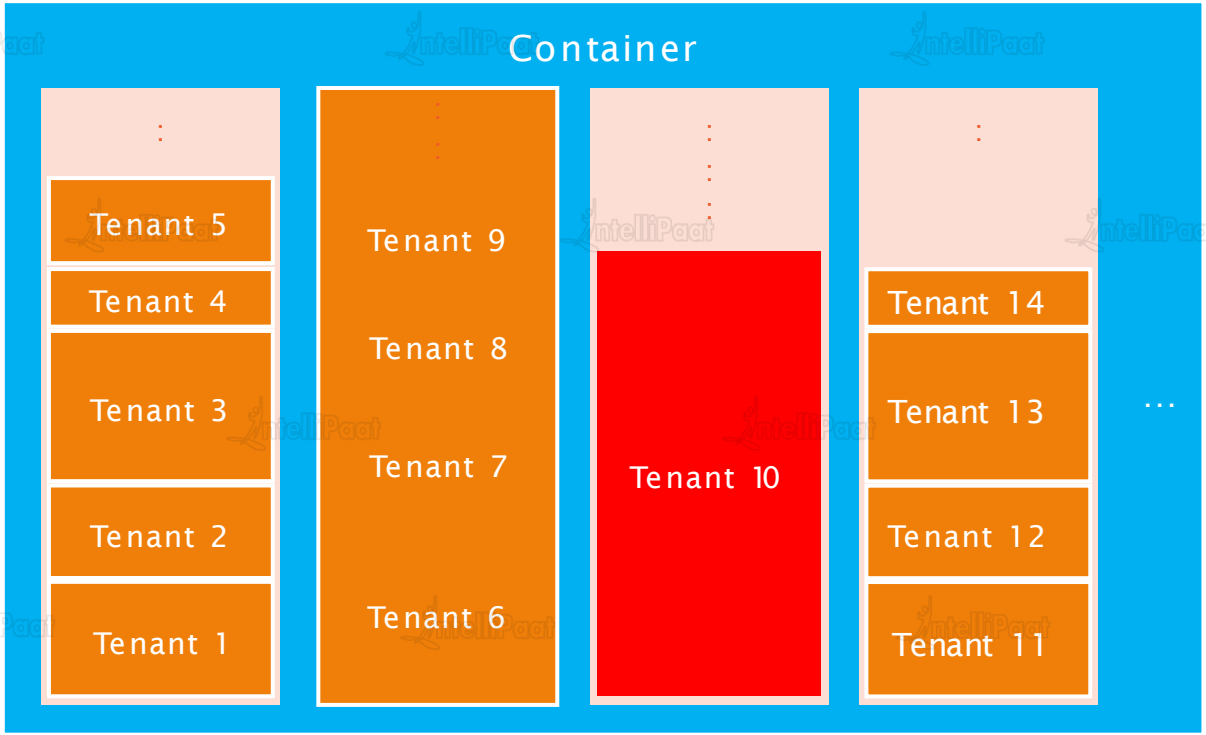
Choosing the Right Partition Key

- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
  - Partitioning by creation date
    - Bad idea! All writes of the day are directed to the same partition
  - Partitioning by user ID
    - Much better! Writes are directed to different partitions per user
- Create multiple containers for varying throughput needs
  - Throughput is purchased at the container level

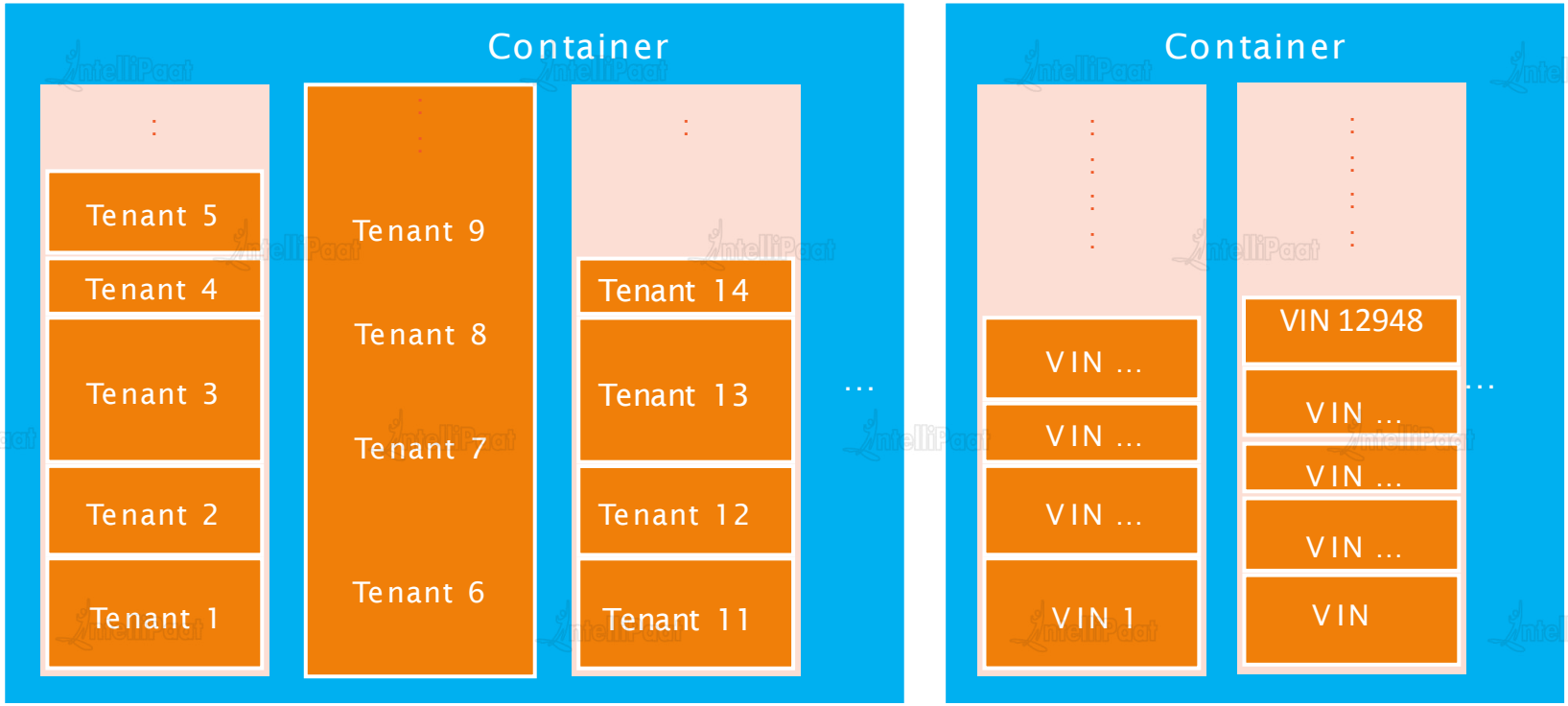
Choosing the Right Partition Key



Choosing the Right Partition Key



Choosing the Right Partition Key



Cross Partition Queries

Cross Partition Queries

Stored Procedures

- Always scoped to a single partition key

Queries

- Typically scoped to a single partition key

Cross-Partition Queries

- Span multiple partition keys
- Fan-out execution

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList(); ≤ 7ms elapsed

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";  ≤ 261ms elapsed
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message); ≤ 78ms elapsed
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);  ≤ 78ms elapsed
    }
}
```

ex Count = 1

Data	{System.Collections.ListDictionaryInternal}
HResult	-2146233088
HelpLink	null
InnerException	{"Cross partition query is required but disabled. Please set x-ms-documentdb-query-enablecross
InnerExceptions	Count = 1
Message	"One or more errors occurred."

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message); ≤ 78ms elapsed
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();  ≤ 3ms elapsed

    await client.DeleteDatabaseAsync(dbUri);
}
```

Cross Partition Queries

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri); ≤ 936ms elapsed
}
```

Replication-Why

Replication-Why?

Performance

- Within a region, ensures SLA on RUs purchased
- Across regions, brings data closer to the consumer

Business continuity

- In the event of major failure or natural disaster

Turnkey Global Distribution

Turnkey Global Distribution

Turnkey Global Distribution

- Associate any number of regions with your Cosmos DB account
  - Limited to geo-fencing policies
- Dynamically add/remove regions
  - Associate (and disassociate) regions with the click of a mouse
- Failover priorities
  - Choose preferred region, followed by regions for failover in order of preference

Hands-On

Hands-On

- Global Distribution

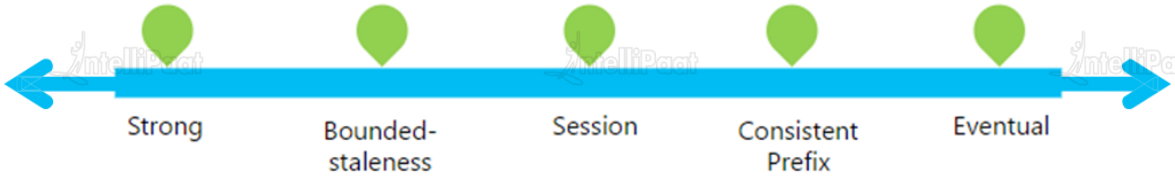
Replication and Consistency

Replication and Consistency

How do you ensure consistent reads across replicas?

Define a consistency level

Stronger consistency
Higher latency
Lower availability



Weaker consistency
Lower latency Higher availability

Replication within a region

Data moves extremely fast (typically, within 1ms) between neighboring racks

Global replication

It takes hundreds of milliseconds to move data across continents

Consistency Levels

Five Consistency Levels

Strong

- No dirty reads

Session

- No dirty reads for writers (read your own writes)
- Dirty reads possible for other users

Bounded Staleness

- Dirty reads possible
- Bounded by time and updates

Consistent Prefix

- Dirty reads possible
- Reads never see out-of-order writes

Eventual

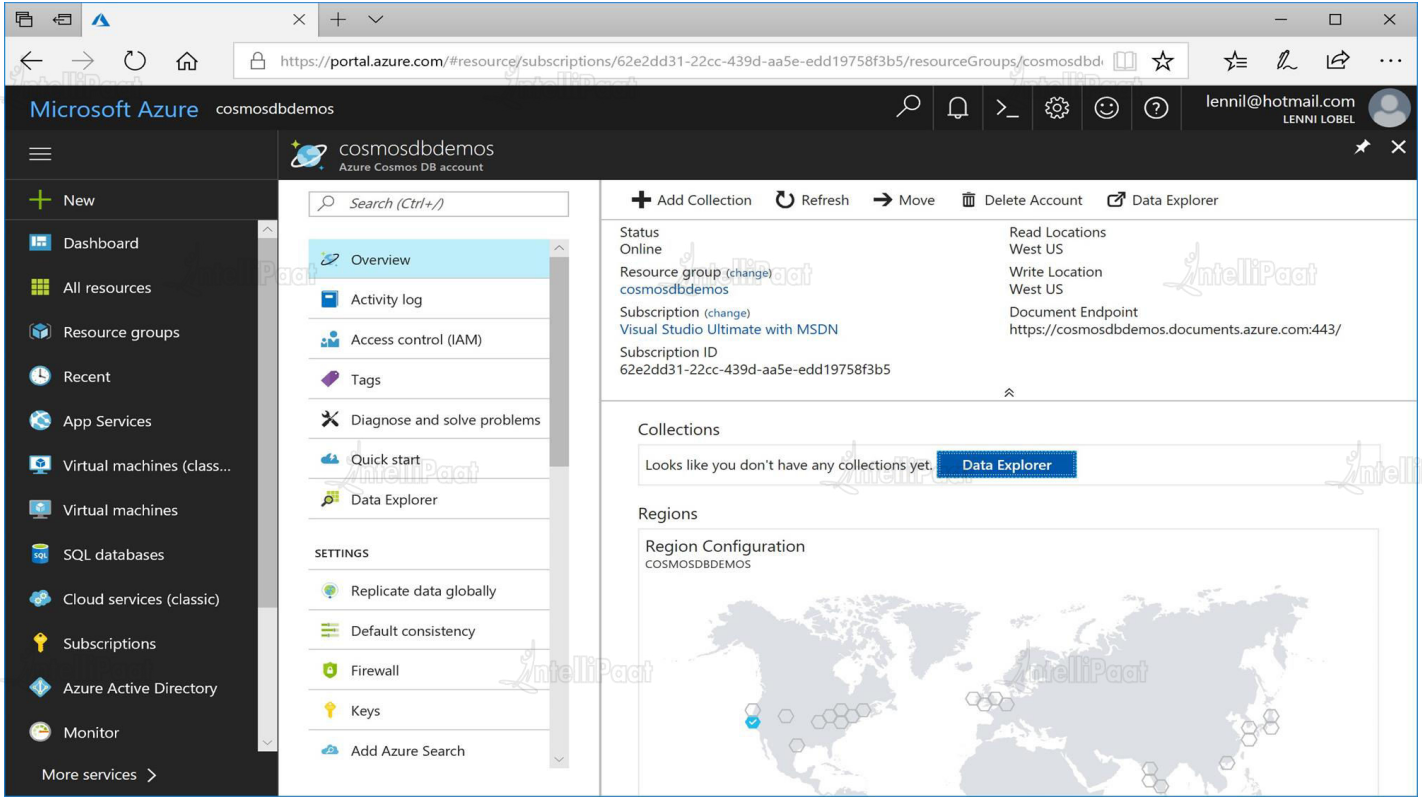
- Stale reads possible
- No guaranteed order

Setting the Consistency Level

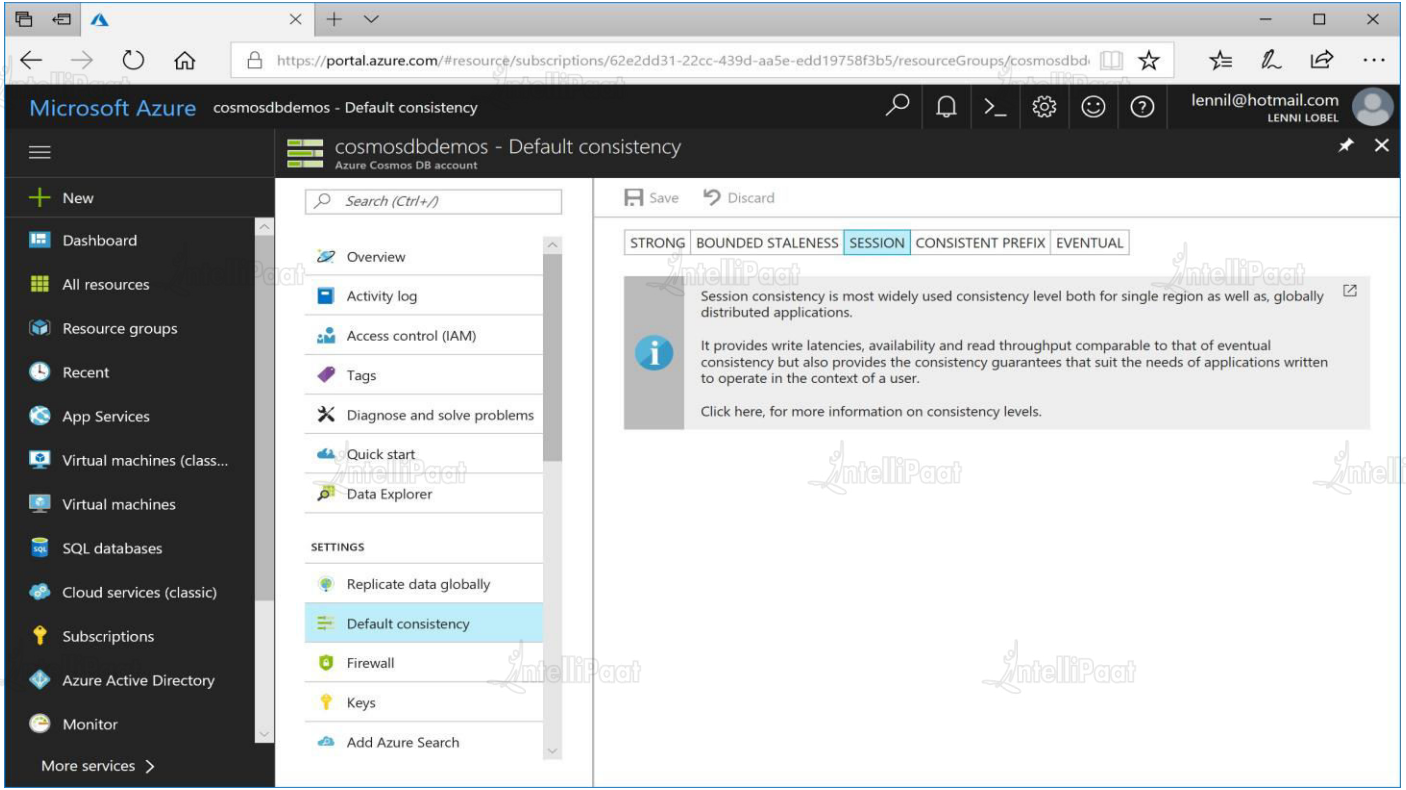
Set default for entire account

Can be changed at any time

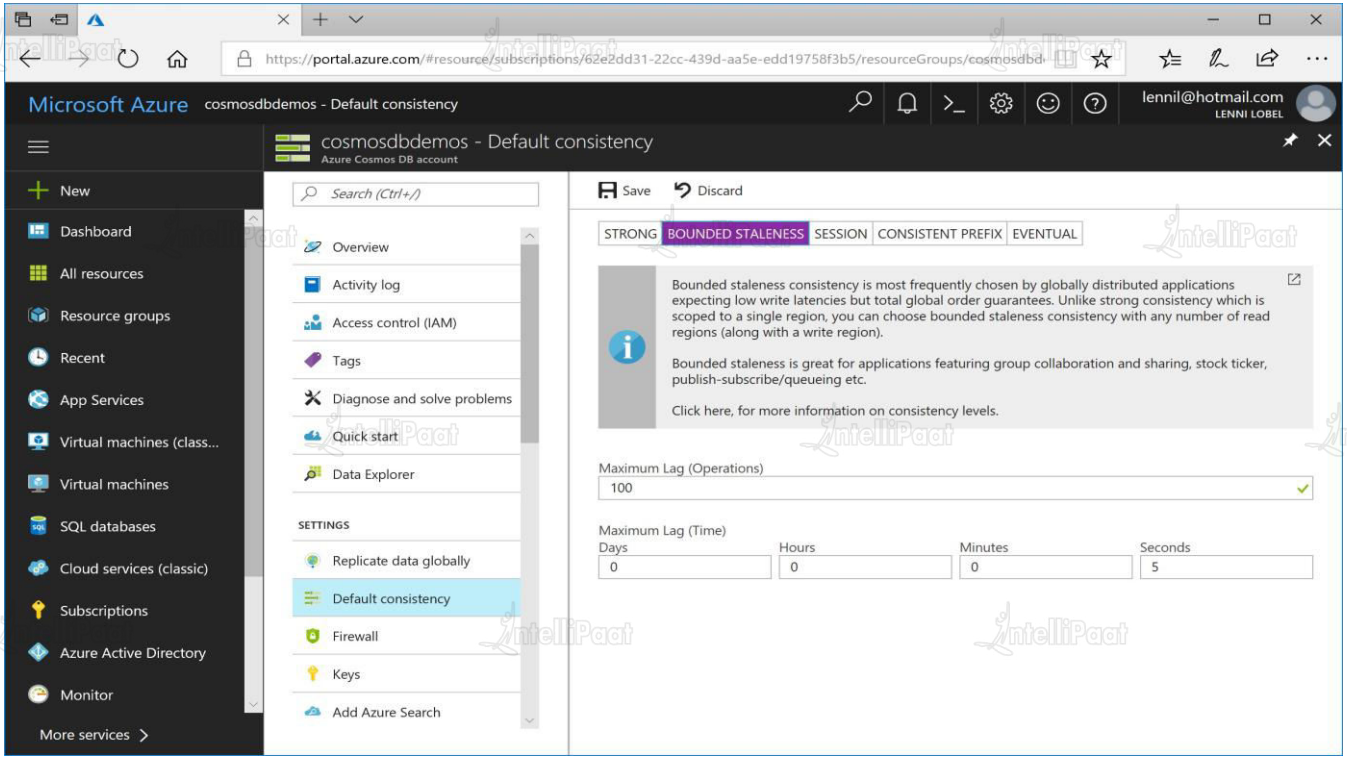
Setting the Consistency Level



Setting the Consistency Level



Setting the Consistency Level



Setting the Consistency Level

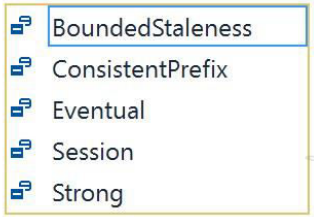
Set default for entire account

- Can be changed at any time

Override at the request level

- Any request can weaken the default consistency level

```
new DocumentClient(new Uri(endpoint), masterKey, connectionPolicy, ConsistencyLevel.)
```



Document Database

What Is a Document Database?

Object graph

Semi-structured JSON (JavaScript Object Notation) documents

What Is a Document Database?

Object graph

Semi-structured JSON (JavaScript Object Notation) documents

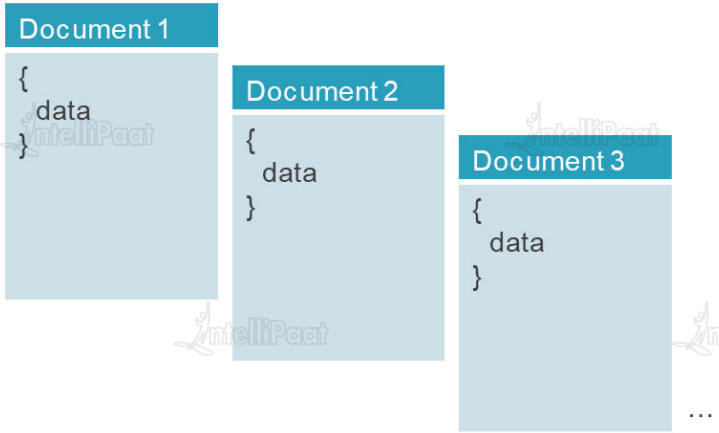
Two API choices

1. SQL API(fka DocumentDB API)
2. MongoDB API

Data Modeling: Relational vs. Document

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents



Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free

ID	Name	IsActive	Dob
1	John Smith	True	8/30/1964
2	Sarah Jones	False	2/18/2002
3	Adam Stark	True	7/13/1987

Document 1

```
{
  "id": 1,
  "name": "John Smith",
  "isActive": true, "dob":
  "1964-30-08"
}
```

Document 2

```
{
  "id": 2,
  "fullName": "Sarah Jones",
  "dob": "2002-02-18"
}
```

Document 3

```
{
  "id": 3,
  “fullName”:
  {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized

User Table

UserID	Name	Dob
1	John Smith	8/30/1964

Holdings Table

StockID	UserID	Qty	Symbol
1	1	100	MSFT
2	1	75	WMT

Document

```
{
  "id": 1,
  "name": "John Smith",
  "dob": "1964-30-08",
  "holdings": [
     { "qty": 100, "symbol": "MSFT" },
     { "qty": 75, "symbol": "WMT" }
   ]
}
```

Data Modeling: Relational vs. Document

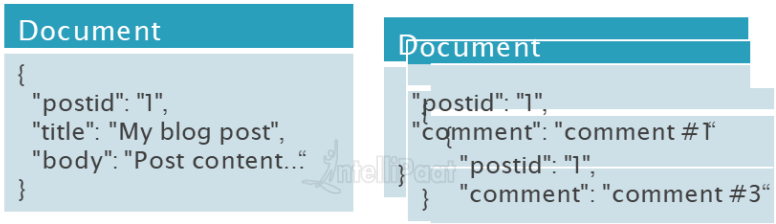
Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized

Document

```
{
  "postid": "1",
  "title": "My blog post",
  "body": "Post content..",
  "comments": [
     "comment #1",
     "comment #2",
     "comment #3",
     "comment #4“,
          :
     "comment #1598873",
          :
```

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized



Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized

Document

```
{
  "postid": "1",
  "title": "My blog post",
  "body": "Post content...",
  "comments": [
     "comment #1",
     "comment #2",
          :
     "comment #100"
  ]
}
```

Document

```
"postid": "1",
"comments": [
     "comment #301",
     "comment #302",
          :
     "comment #400"
]
```

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized

Document

```
{
  “itemid”: "1",
  “part”: “1”,
  “prop1”: “Typically read",
  “prop2”: “Typically read",
  “prop3”: “Typically read",
  “prop4”: “Typically read",
      :
  “prop200”: “Typically read",
}
```

Document

```
{
  “itemid”: "1",
  “part”: “2”,
  “prop1”: “Typically updated",
  “prop2”: “Typically updated",
:
  “prop10”: “Typically updated",
}
```

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized

Document

```
{
  "name": "John Smith",
  "orderDate": "2015-30-03",
  "details": [
     { "qty": 5, "code": "BC", "desc": "Black chair" },
     { "qty": 1, "code": "RT", "desc": "Red table" },
     { "qty": 2, "code": "YC", "desc": "Yellow clock" },
   ]
}
```

Document

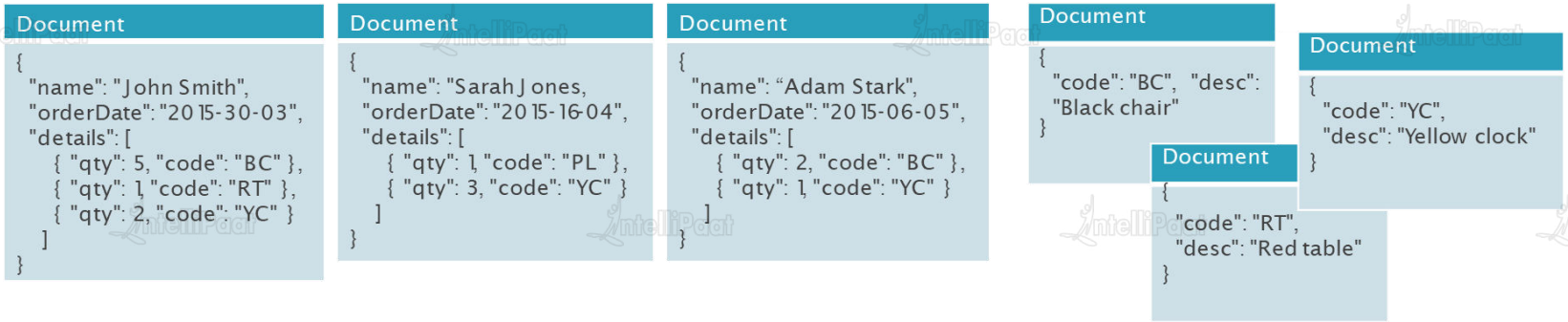
```
{
  "name": "Sarah Jones,
  "orderDate": "2015-16-04",
  "details": [
     { "qty": 1, "code": "PL", "desc": "Purple lamp" },
     { "qty": 3, "code": "YC", "desc": "Yellow clock" },
   ]
}
```

Document

```
{
  "name": “Adam Stark",
  "orderDate": "2015-06-05",
  "details": [
     { "qty": 2, "code": "BC", "desc": "Black chair" },
     { "qty": 1, "code": "YC", "desc": "Yellow clock" },
   ]
}
```

Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized



Data Modeling: Relational vs. Document

Relational Store	Document Store
Rows	Documents
Columns	Properties
Strongly-typed schemas	Schema-free
Highly normalized	Typically denormalized

Document

```
{
  "id": "1",
  "name": "John Smith",
  "dob": "1964-30-08",
  "holdings": [
     { "qty": 100, "symbol": "MSFT" },
     { "qty": 75, "symbol": "WMT" }
   ]
}
```

Cosmos DB Resource Model

Cosmos DB Resource Model

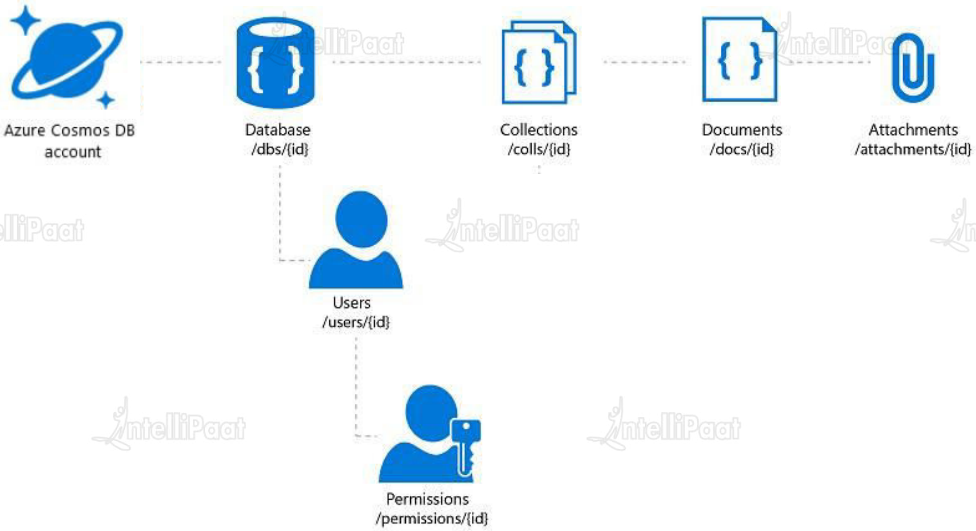
Cosmos DB Resource Model

Cosmos DB Resource Model

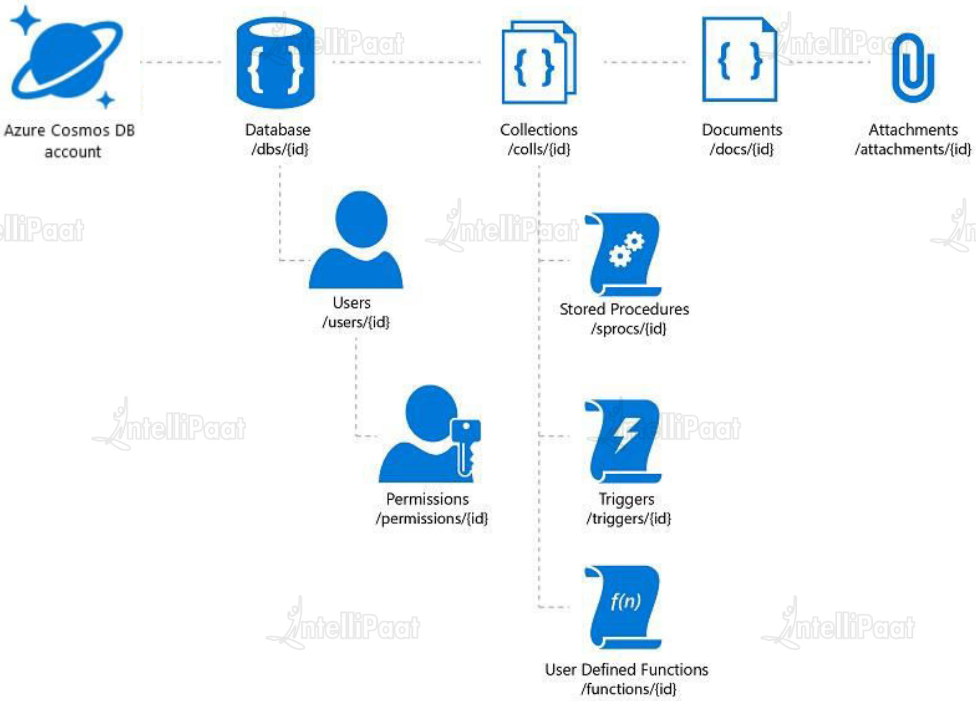
Cosmos DB Resource Model

Cosmos DB Resource Model

Cosmos DB Resource Model



Cosmos DB Resource Model



Resource Properties, Self-Links, and URIs

Resource Properties, Self-Links, and URIs

Every resource contains these properties

Property	Value
id	User-defined unique ID (string)

Resource Properties, Self-Links, and URIs

Every resource contains these properties

Property	Value
id	User-defined unique ID (string)
_rid	Resource ID

Resource Properties, Self-Links, and URIs

Every resource contains these properties

Property	Value
id	User-defined unique ID (string)
_rid	Resource ID
_ts	Timestamp (last updated) epoch value

Resource Properties, Self-Links, and URIs

Every resource contains these properties

Property	Value
id	User-defined unique ID (string)
_rid	Resource ID
_ts	Timestamp (last updated) epoch value
_etag	GUID used for optimistic concurrency

Resource Properties, Self-Links, and URIs

Every resource contains these properties

Property	Value
id	User-defined unique ID (string)
_rid	Resource ID
_ts	Timestamp (last updated) epoch value
_etag	GUID used for optimistic concurrency
_self	URI path to the resource

Resource Properties, Self-Links, and URIs

Every resource contains these properties

Property	Value
id	User-defined unique ID (string)
_rid	Resource ID
_ts	Timestamp (last updated) epoch value
_etag	GUID used for optimistic concurrency
_self	URI path to the resource
_attachments	URI path suffix to the resource attachments

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ]
}
```

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d"
}
```

Resource Properties, Self-Links, and URIs


```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d",
    "_rid": "v84pAO7nPQABAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQABAAAAAAAABg==/",
    "_etag": "\"00004700-0000-0000-0000-5a3e991a0000\"",
    "_attachments": "attachments/",
    "_ts": 1514051866
}
```


```
var updatedAt = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc)
    .AddSeconds(1514051866)
    .ToLocalTime()
    .ToString("ddd M/d/yyyy h:m:s tt");
```

Resource Properties, Self-Links, and URIs

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d",
    "_rid": "v84pAO7nPQABAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQABAAAAAAAABg==/",
    "_etag": "\"00004700-0000-0000-0000-5a3e991a0000\"",
    "_attachments": "attachments/",
    "_ts": 1514051866
}
```

```
// Identify document by...

// ...self link
string id1 = "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQABAAAAAAAABg==/";

// ...ID-based routing
string id2 = "dbs/Families/colls/Families/docs/ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d";

// ...URI
Uri id3 = UriFactory.CreateDocumentUri("Families", "Families", "ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d");
```

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d",
    "_rid": "v84pAO7nPQABAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQABAAAAAAAABg==/",
    "_etag": "\"00004700-0000-0000-0000-5a3e991a0000\"",
    "_attachments": "attachments/",
    "_ts": 1514051866
}
```

```
// Identify document by...

// ...self link
string id1 = "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQABAAAAAAAABg==/";

// ...ID-based routing
string id2 = "dbs/Families/colls/Families/docs/ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d";

// ...URI
Uri id3 = UriFactory.CreateDocumentUri("Families", "Families", "ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d");
```

```
id3 {dbs/Families/colls/Families/docs/ab7aafc4-6a6e-f19e-bf5f-f7985b2f156d}
```

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "SmithFamily"
}
```

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "SmithFamily",
    "_rid": "v84pAO7nPQACAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQACAAAAAAAABg==/",
    "_etag": "\"00004800-0000-0000-0000-5a3e9a210000\"",
    "_attachments": "attachments/",
    "_ts": 1514052129
}
```

```
// Identify document by...

// ...self link
string id1 = "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQACAAAAAAAABg==/";

// ...ID-based routing
string id2 = "dbs/Families/colls/Families/docs/SmithFamily";

// ...URI
Uri id3 = UriFactory.CreateDocumentUri("Families", "Families", "SmithFamily");
```

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "SmithFamily",
    "_rid": "v84pAO7nPQACAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQACAAAAAAAABg==/",
    "_etag": "\"00004800-0000-0000-0000-5a3e9a210000\"",
    "_attachments": "attachments/",
    "_ts": 1514052129
}
```

```
// Identify document by...

// ...self link
string id1 = "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQACAAAAAAAABg==/";

// ...ID-based routing
string id2 = "dbs/Families/colls/Families/docs/SmithFamily";

// ...URI
Uri id3 = UriFactory.CreateDocumentUri("Families", "Families", "SmithFamily");
```

id3 {dbs/Families/colls/Families/docs/SmithFamily}

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "SmithFamily",
    "_rid": "v84pAO7nPQACAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQACAAAAAAAABg==/",
    "_etag": "\"00004800-0000-0000-0000-5a3e9a210000\"",
    "_attachments": "attachments/",
    "_ts": 1514052129
}
```

```
// Identify collection by...

// ...self link
string id1 = "dbs/v84pAA==/colls/v84pAO7nPQA=/";

// ...ID-based routing
string id2 = "dbs/Families/colls/Families";

// ...URI
Uri id3 = UriFactory.CreateDocumentCollectionUri("Families", "Families");
```

Resource Properties, Self-Links, and URIs

```
{
    "familyName": "Smith",
    "address": {
        "addressLine": "123 Main Street",
        "city": "Chicago",
        "state": "IL",
        "zipCode": "60601"
    },
    "parents": [
        "Peter",
        "Alice"
    ],
    "kids": [
        "Adam",
        "Jacqueline",
        "Joshua"
    ],
    "id": "SmithFamily",
    "_rid": "v84pAO7nPQACAAAAAAAABg==",
    "_self": "dbs/v84pAA==/colls/v84pAO7nPQA=/docs/v84pAO7nPQACAAAAAAAABg==/",
    "_etag": "\"00004800-0000-0000-0000-5a3e9a210000\"",
    "_attachments": "attachments/",
    "_ts": 1514052129
}
```

```
// Identify collection by...

// ...self link
string id1 = "dbs/v84pAA==/colls/v84pAO7nPQA=/";

// ...ID-based routing
string id2 = "dbs/Families/colls/Families";

// ...URI
Uri id3 = UriFactory.CreateDocumentCollectionUri("Families", "Families");
```

id3 {dbs/Families/colls/Families}

Data Migration Tool

Data Migration Tool

Open source project

Microsoft Download Center

http://www.microsoft.com/en-us/download/details.aspx?id=46436

Documentation and How-To

https://azure.microsoft.com/en-us/documentation/articles/documentdb-import- data

Source code on GitHub

https://github.com/azure/azure-documentdb-datamigrationtool

Import from

- SQL Server
- JSON
- CSV
- MongoDB
- Azure Table Storage
- and more…

Hands-On

- Importing data from SQL Server

Rich Query with SQL

Rich Query with SQL

Cosmos DB SQL

Special version of SQL designed for JSON

.NET SDK also includes a LINQ provider

Familiar keywords

SELECT, FROM, WHERE, JOIN, IN, BETWEEN, ORDER BY

Work with JSON

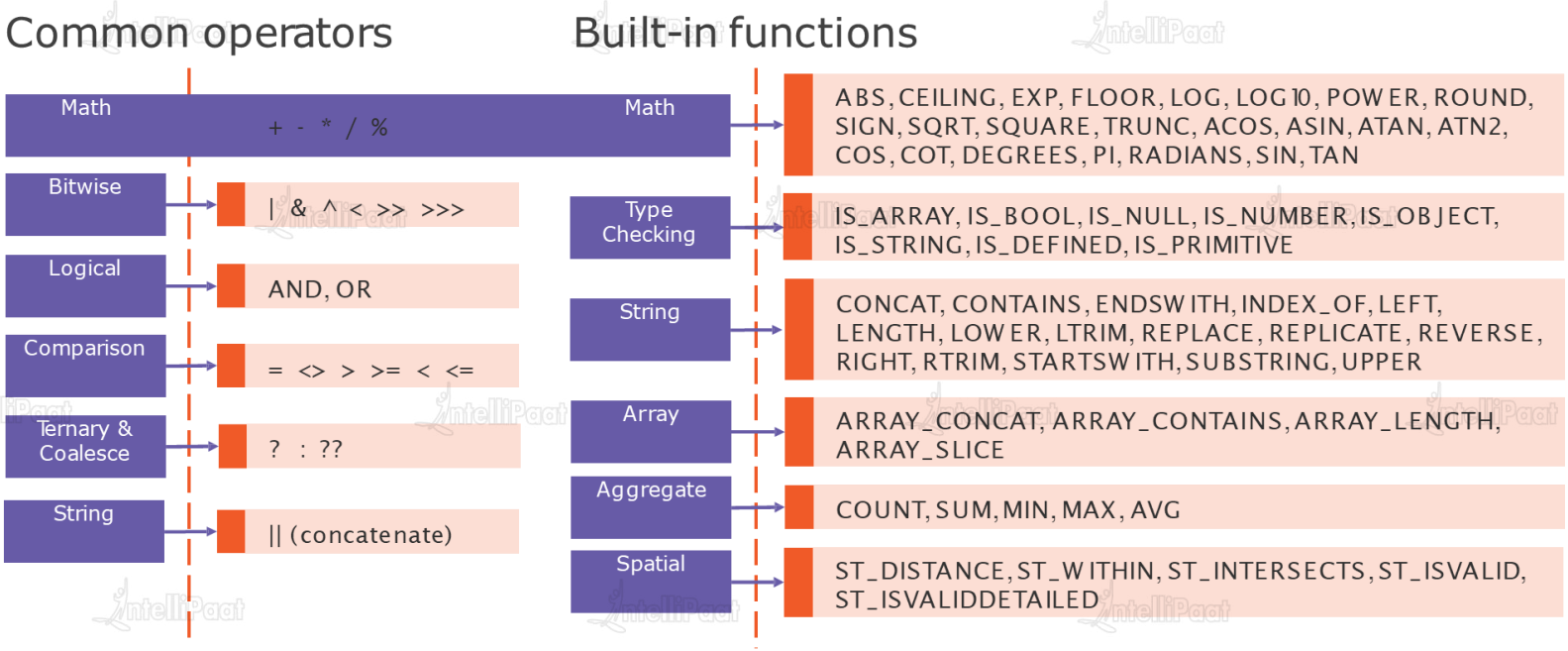
Descend into any subsection of a document Iterate nested arrays for intra-document join Return documents as-is, or as custom shape

Example

```
SELECT ch.name, ch.birthdate
FROM Families AS f
JOIN ch IN f.children
WHERE f.address.state = 'CA'
```

SQL Operators and Functions

SQL Operators and Functions



Hands-On

- Simple SQL Query
- Scalar Expression Queries
- Query Operators and Built-in Functions
- Querying Documents in a Collection
- More SQL Queries
- Aggregation Queries
- Spatial Queries

Client Development

Client Development

Build web-scale applications

Uses REST/HTTP

Platform SDKs

- .NET / .NET
- Core Java
- Node.js
- Python

Introducing the .NET SDK for the SQL API

Introducing the .NET SDK for the SQL API

Create a DocumentClient instance

- Supply connection information (endpoint and key)

Invoke methods to access resources

- Create, modify, and delete resources
- Use POCOs or dynamics for document objects

Task Parallel Library (TPL)

- Simplified asynchronous programming
- Use async/await keywords with Task objects

Introducing the .NET SDK for the SQL API

Synchronous code

```
private void Main()
{
    DoSomething();
}

private void DoSomething()
{
    // do some work
}
```

Asynchronous code

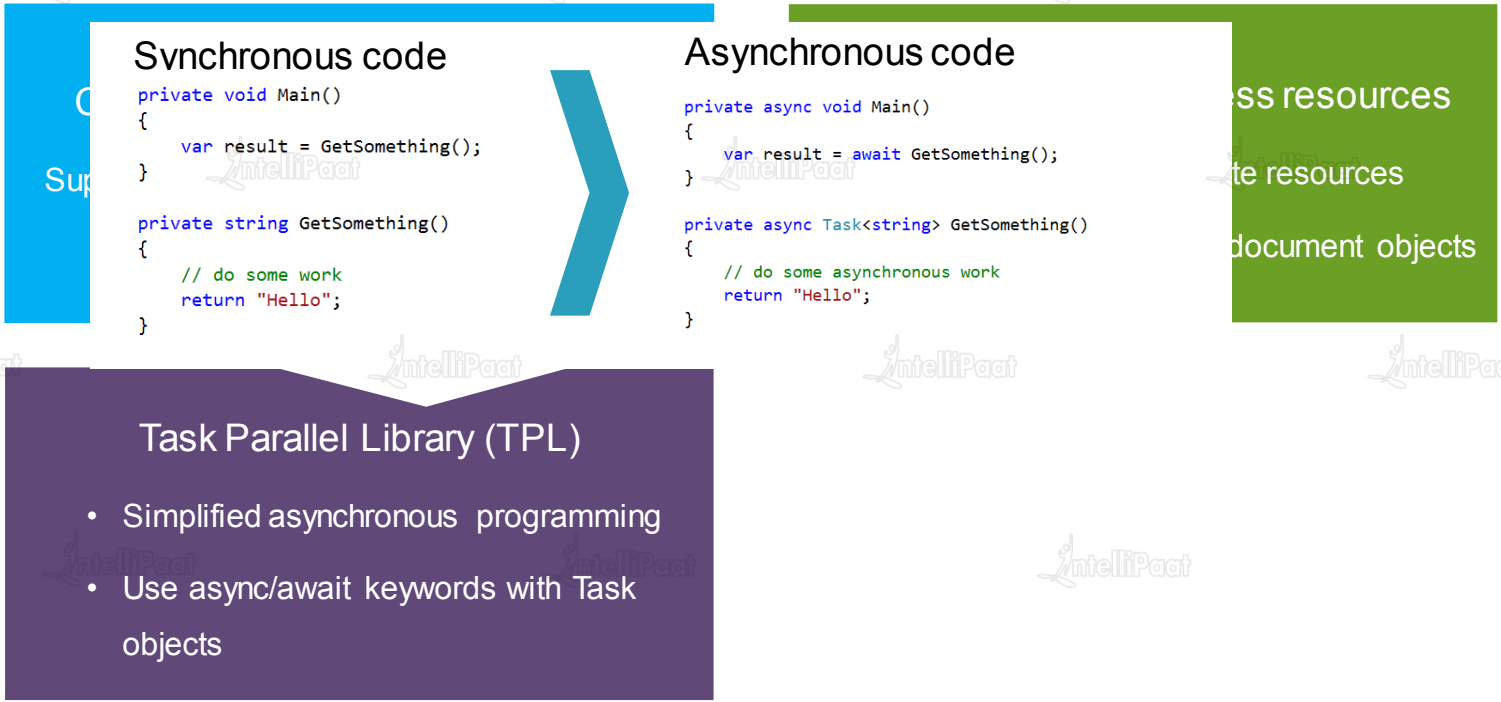
```
private async void Main()
{
    await DoSomething();
}

private async Task DoSomething()
{
    // do some asynchronous work
}
```

Task Parallel Library (TPL)

- Simplified asynchronous programming
- Use async/await keywords with Task objects

Introducing the .NET SDK for the SQL API



Introducing the .NET SDK for the SQL API

Create a DocumentClient instance

- Supply connection information (endpoint and key)

Invoke methods to access resources

- Create, modify, and delete resources
- Use POCOs or dynamics for document objects

Task Parallel Library (TPL)

- Simplified asynchronous programming
- Use async/await keywords with Task objects

LINQ provider

- Automatically translates LINQ queries to SQL

Hands-On

- Getting started with the .NET SDK
- Working with databases
- Working with collections
- Creating documents
- Querying for documents
- Replacing and deleting documents

Indexing Policies

Indexing Policies

Hash index

Equality queries Strings and numbers

Range index

Equality, range, ORDER BY Strings and numbers

Spatial index

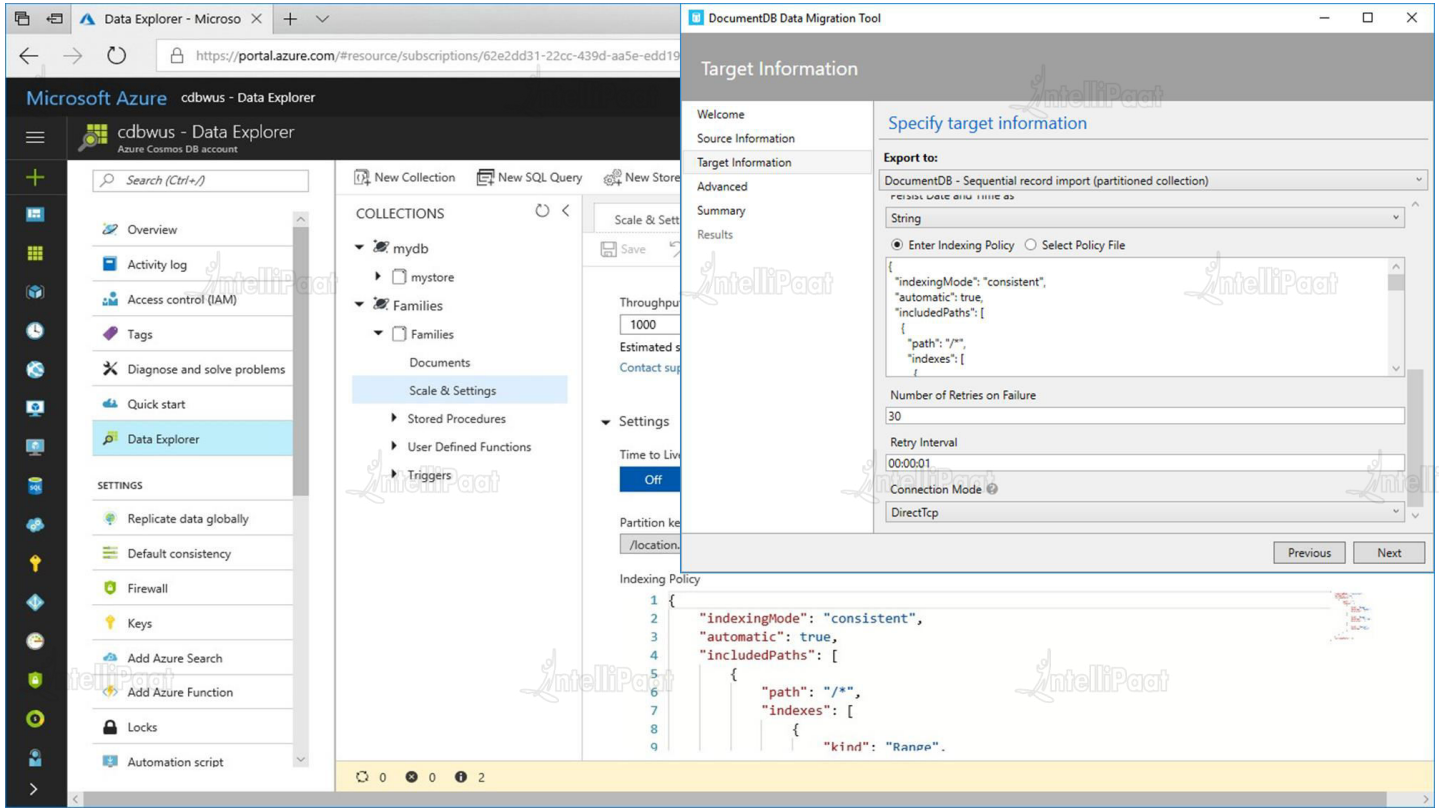
Distance and intersection Points, polygons, line strings

Indexing Policies

Collection-wide policy

Established when creating a collection Can be changed after collection is created

Indexing Policies



Indexing Policies

Collection-wide policy

Established when creating a collection Can be changed after collection is created

Automatic indexing

Can switch to manual

Can override on a per-document basis

Selective indexing

Include/exclude selected property paths

Individual properties (?)

/name/?

/address/state/?

Recursive properties (*)

/address/*

/item/colors[]/*

Indexing Policies

Collection-wide policy

Established when creating a collection can be changed after collection is created

Automatic indexing

Can switch to manual

Can override on a per-document basis

Selective indexing

Include/exclude selected property paths

Indexing modes

Consistent (synchronous)

Lazy (asynchronous)

Users, Permissions, and Resource Tokens

Users, Permissions, and Resource Tokens

Resource tokens vs. master key

Provides granular control over security

Create database users

Then create permissions for each user

Get resource token from permission

Read or All access to a single resource

(collection, document, stored procedures, triggers, user-defined functions)

Connect using resource tokens

Access will be granted based on all supplied resource tokens

Hands-On

- Custom indexing
- Working with users and permissions

Server-side Programming Model

Server-side Programming Model

Run code inside Cosmos DB

- Stored procedures
- Triggers
- User-defined functions

Scoped by partition key

- Read/write documents with that partition key
- Fully transactional updates (ACID guarantee)

Bounded execution

- Implement continuation model for long running processes

Handle throttling (code 429)

- Implement retry logic when exceeding reserved throughput

Writing Server-side Code

Writing Server-side Code

Write a JavaScript function

- Obtain context object

Context exposes

- Collection
- Request
- Response

Working with Triggers

Working with Triggers

Attach custom behavior

- Access document being created or replaced

Two trigger types

- Pre-trigger
- Post-trigger

Not fired automatically

- Must be explicitly requested with an operation

Creating User-Defined Functions (UDFs)

Creating User-Defined Functions (UDFs)

Extend the query language

- Write custom business logic that can be called from inside queries

Use with care

- Querying on a UDF requires a scan (cannot use index)

No access to context

- Compute-only JavaScript

Hands-On

- Creating triggers
- Pre-triggers
- Post-triggers
- User-Defined Functions (UDFs)

Table API

What Is the Table API?

IntelliPaat

Replaces Azure Table storage

- Implement a key-value data model

Key-value data model

- Key = PartitionKey + RowKey
- Value = key-value pairs!

Leverage Cosmos DB back end

- Predictable throughput
- Global distribution
- Automatic indexing

SQL API vs. Table API

Cosmos DB	SQL API	Table API

SQL API vs. Table API

Cosmos DB	SQL API	Table API
Container	Collection	Table

SQL API vs. Table API

Cosmos DB	SQL API	Table API
Container	Collection	Table
Item	Document	Row

Document (SQL API)

```
{
  "genre": "sci-fi",
  "id": "Star Trek II",
  "year": 1982,
  "length": "1h, 53m",
  "description": "Khan is back!"
}
```

Row (Table API)

PartitionKey	RowKey	Year	Length	Description
sci-fi	Star Trek II	1982	1h, 53m	Khan is back!

SQL API vs. Table API

Cosmos DB	SQL API	Table API
Container	Collection	Table
Item	Document	Row
Partition Key	Any property	PartitionKey

SQL API vs. Table API

Cosmos DB	SQL API	Table API
Container	Collection	Table
Item	Document	Row
Partition Key	Any property	PartitionKey
ID	id	RowKey

Why Use the Table API?

Migrate existing applications

- Just change the connection string

Upgrade SDK

- Cosmos DB Table SDK uses native protocol

No advantage for new applications

- Table API is a layer over SQL API

Hands-On

- Simple Azure Table Storage application

Cosmos DB Graph Database

Cosmos DB Graph Database

Graph container

Horizontal partitioning, provisioned throughput, global distribution, indexing

Vertex and Edge objects

Entities and relationships

Both can hold arbitrary key-value pairs

Create/Retrieve/Update/Delete

GraphSON and Gremlin

Apache TinkerPop

http://tinkerpop.apache.org

Focus on relationships

Chain relationship queries with Gremlin

Graph Database Scenarios

Graph Database Scenarios

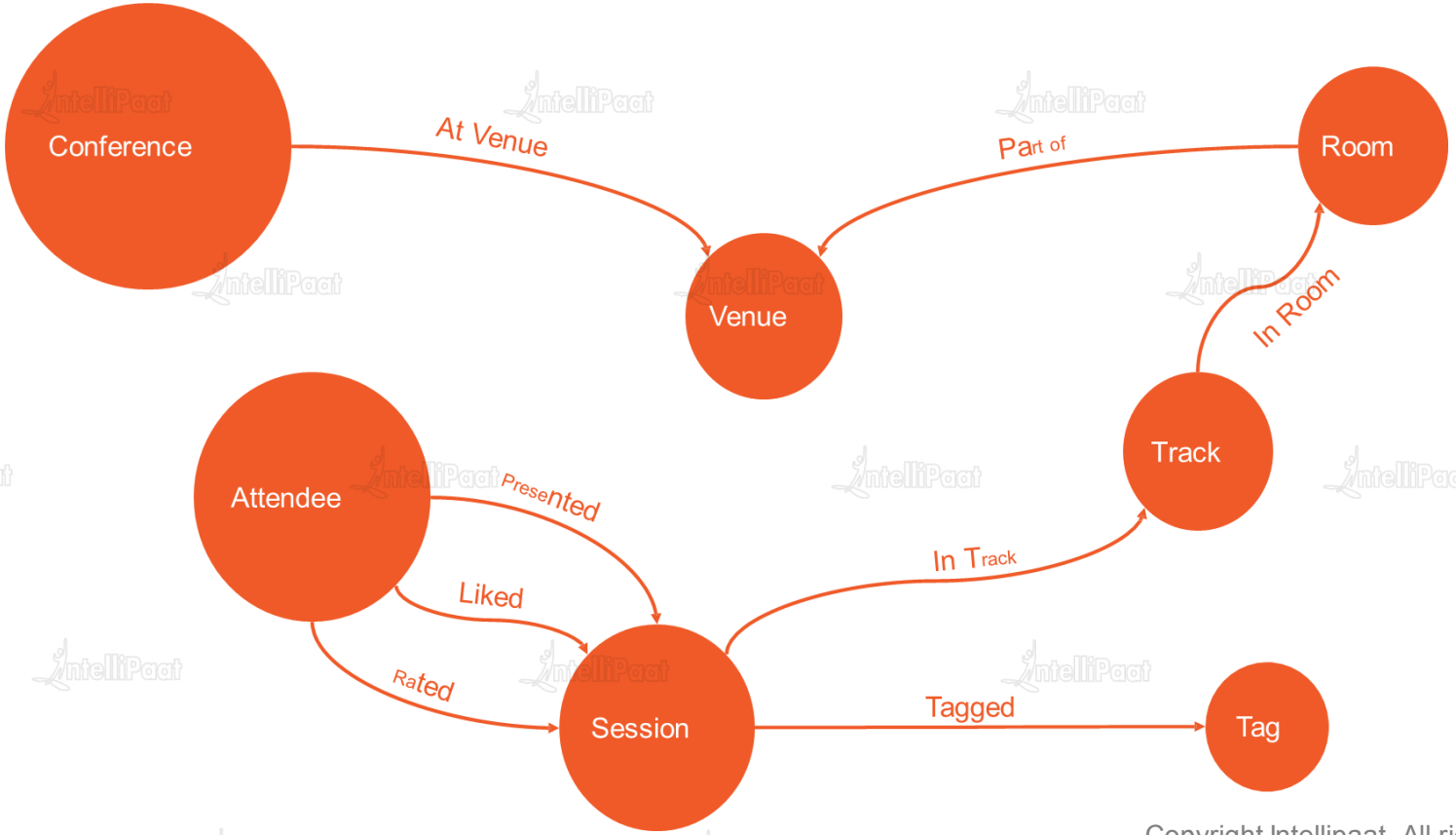
Complex relationships

- Many “many-to-many” relationships
- Excessive JOINs
- Analyze interconnected data and relationships

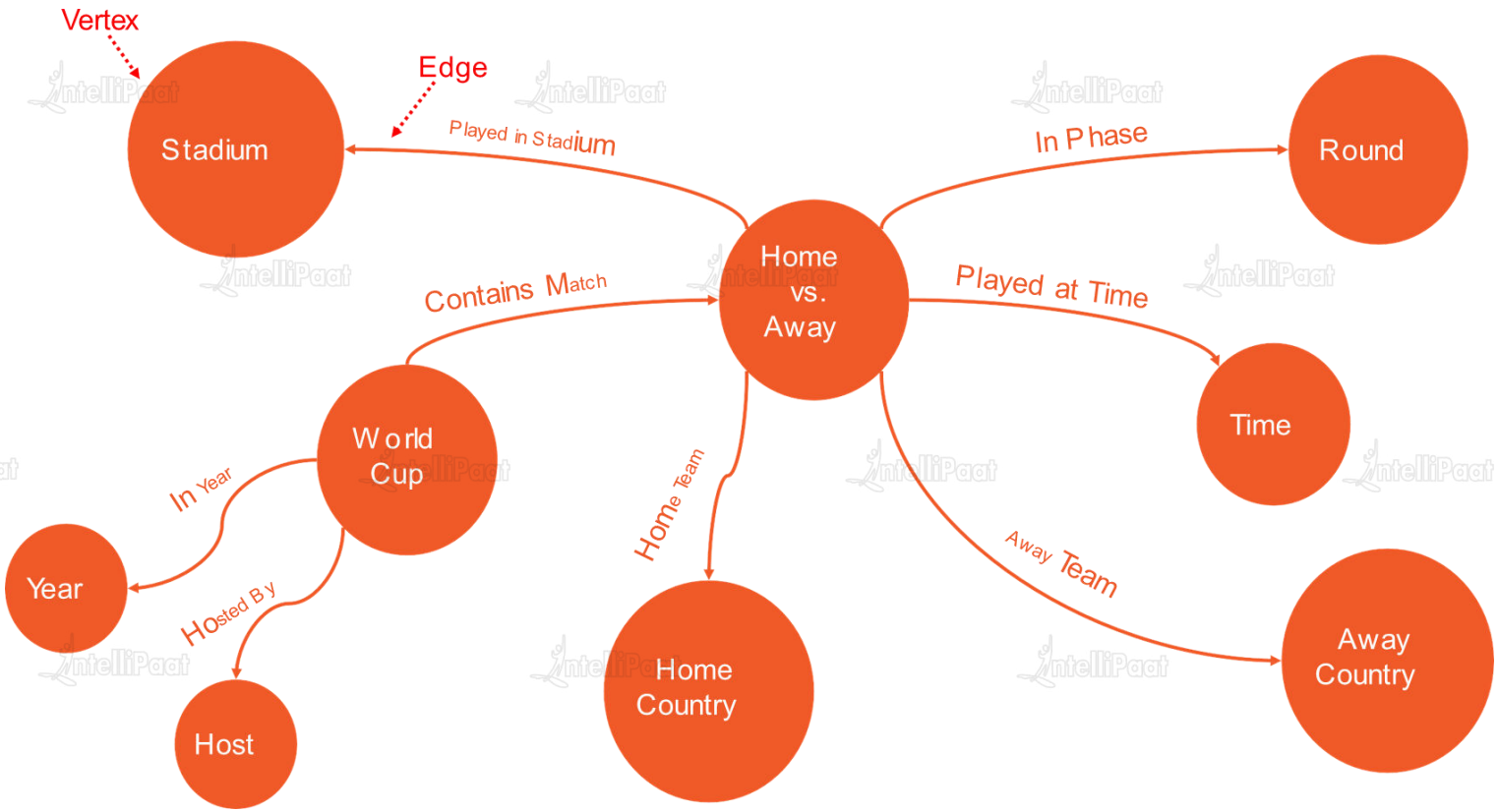
Typical graph applications

- Social networks
- Recommendation engines
- Knowledge graphs
- Many more…

Graph Database Scenarios



Graph Database Scenarios



Vertices and Edges

Vertices and Edges

Vertex and Edge properties

- id (within partition key) label (type)
- Additional arbitrary properties (including the partition key)

Additional Edge properties

- Cardinality(in-and-out vertices)
- Create two edges for bi-directionality

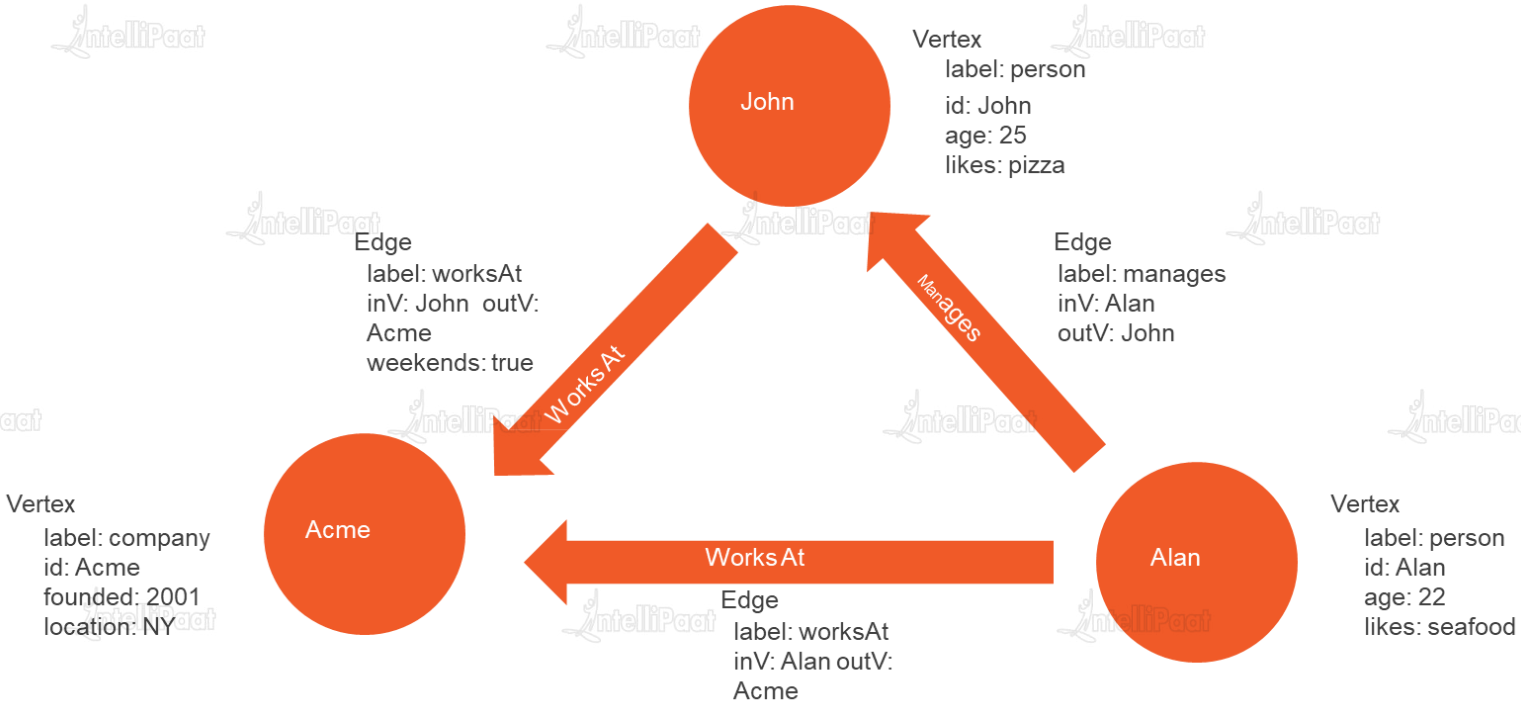
Vertices and Edges

Vertex
label: person
id: John
age: 25
likes: pizza

Vertex
label: company
id: Acme
founded: 2001
location: NY

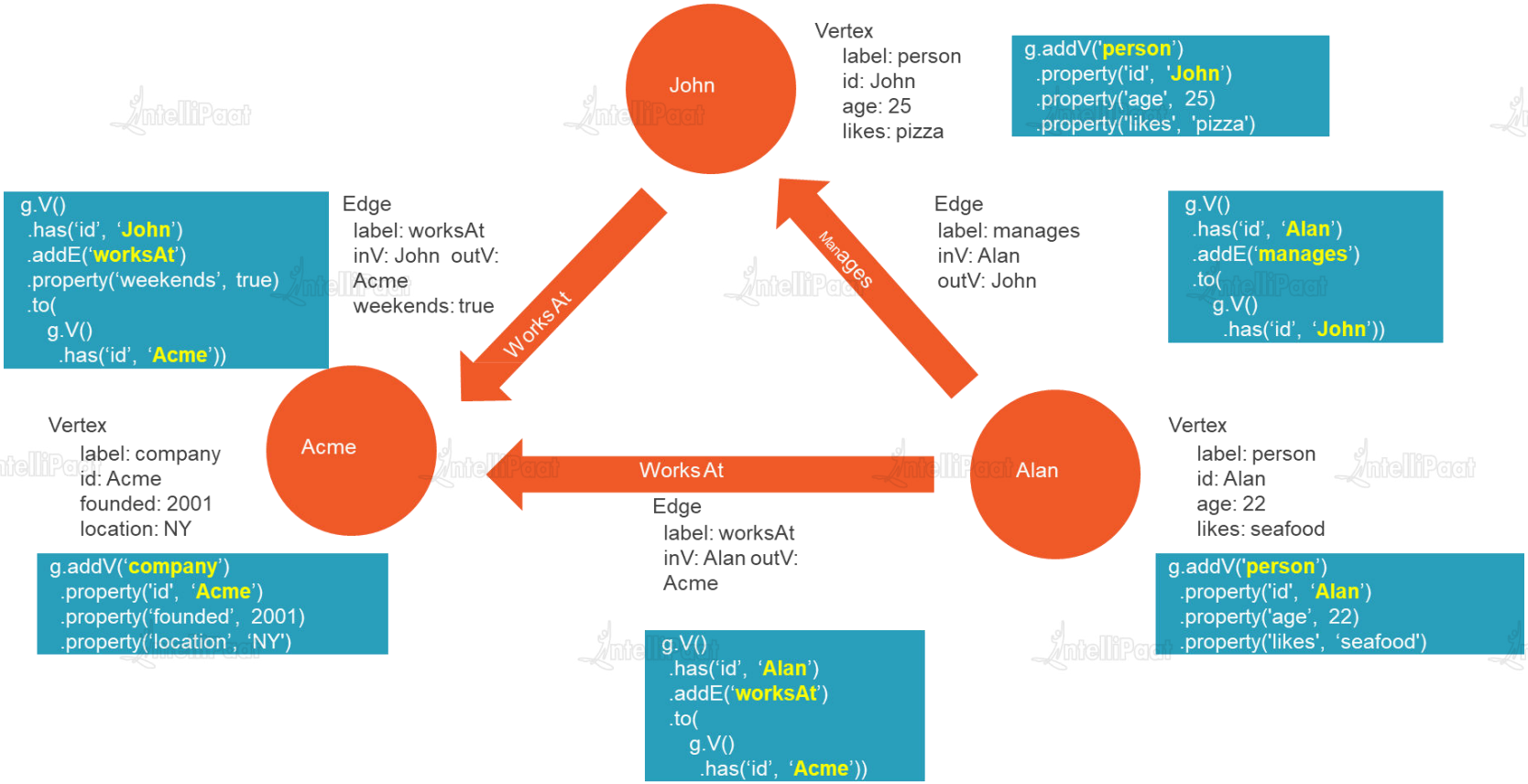
Vertex
label: person
id: Alan
age: 22
likes: seafood

Vertices and Edges

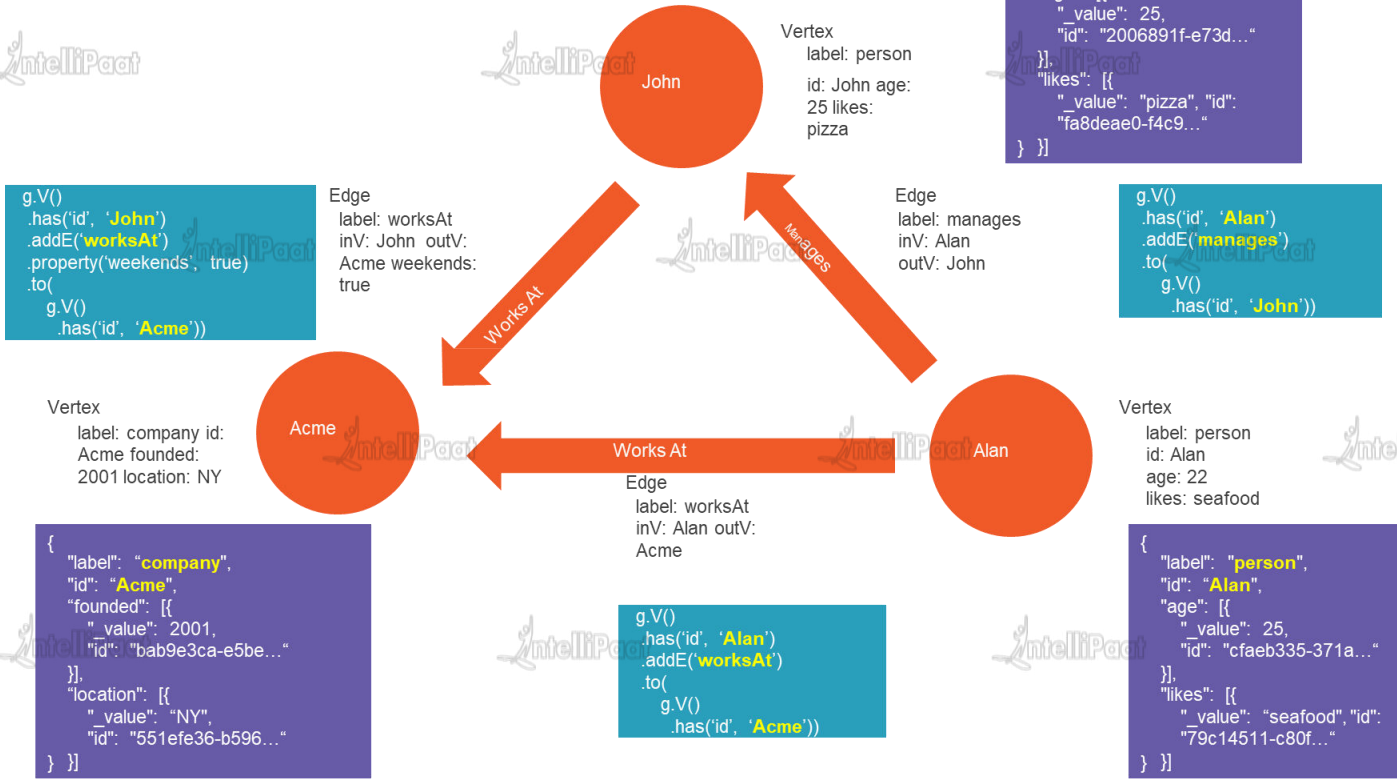


Populating the Graph

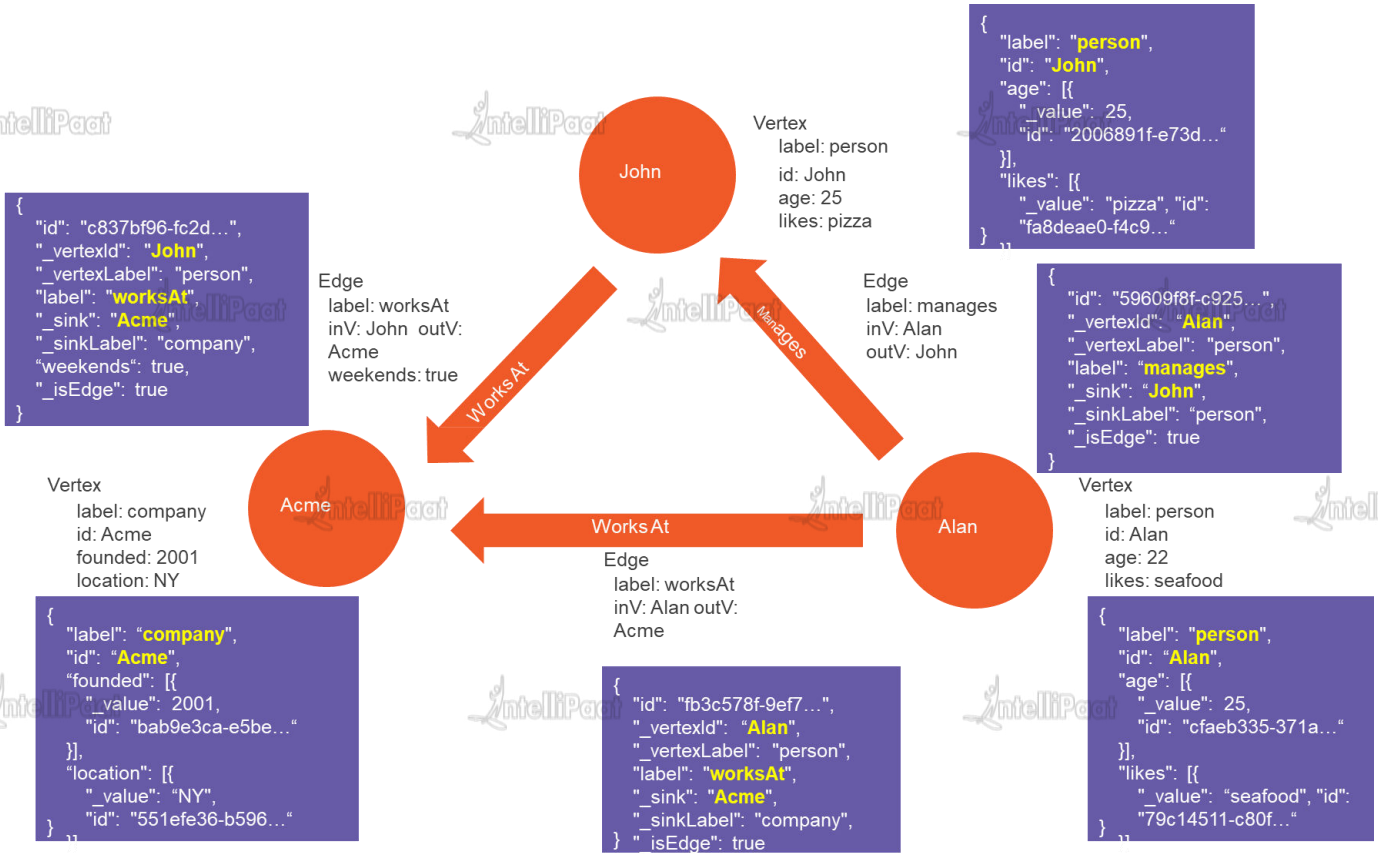
Populating the Graph



Populating the Graph

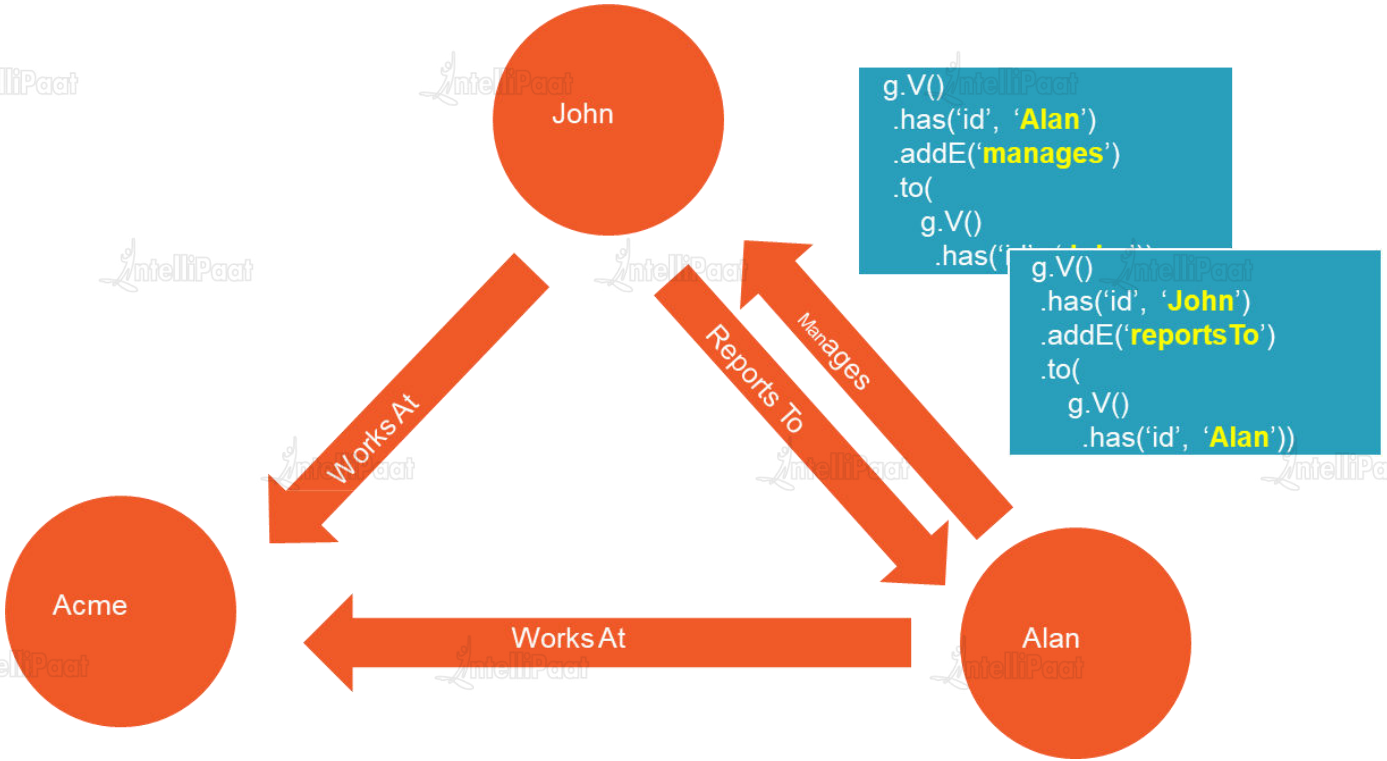


Populating the Graph

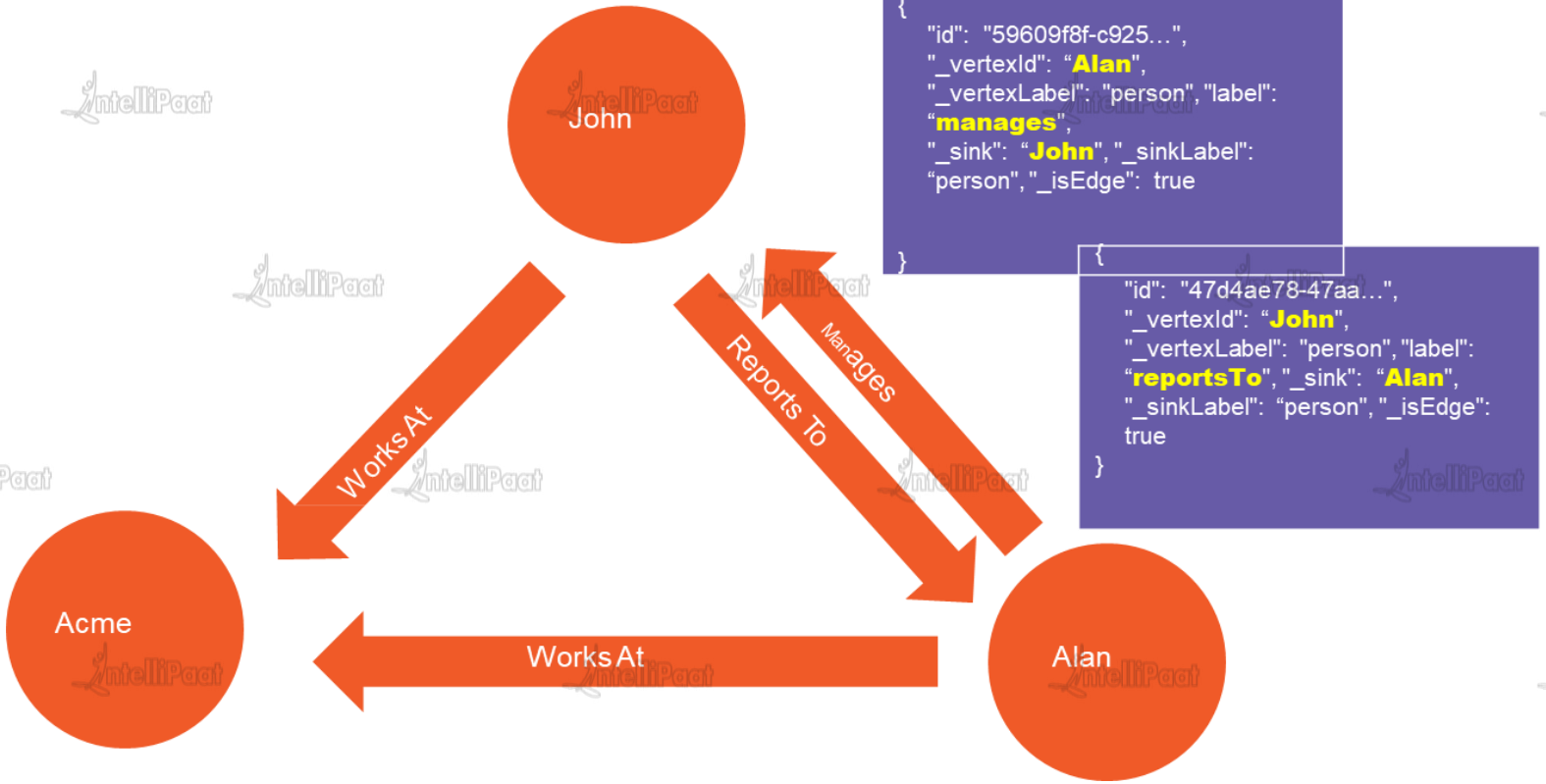


Bi-Directional Relationships

Bi-Directional Relationships



Bi-Directional Relationships



Hands-On

- Creating a simple graph

Writing Gremlin Queries

Writing Gremlin Queries

Cosmos DB Gremlin support

http://aka.ms/gremlin-support

Functional language

Chain multiple steps together

Define Vertices and Edges

.addV('label')

.addE('label')

.property('key', 'value')

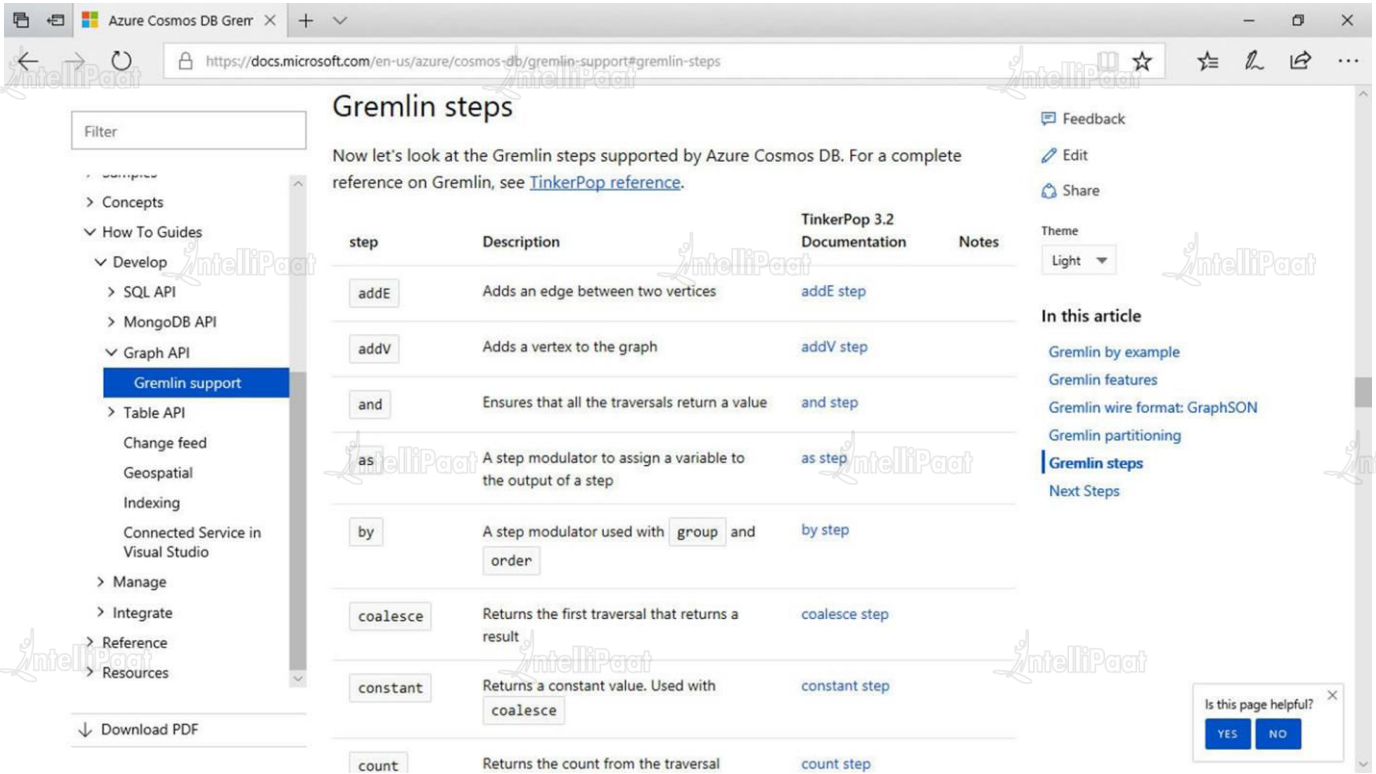
Query on filters and relationships

.V('label')

.out('label')

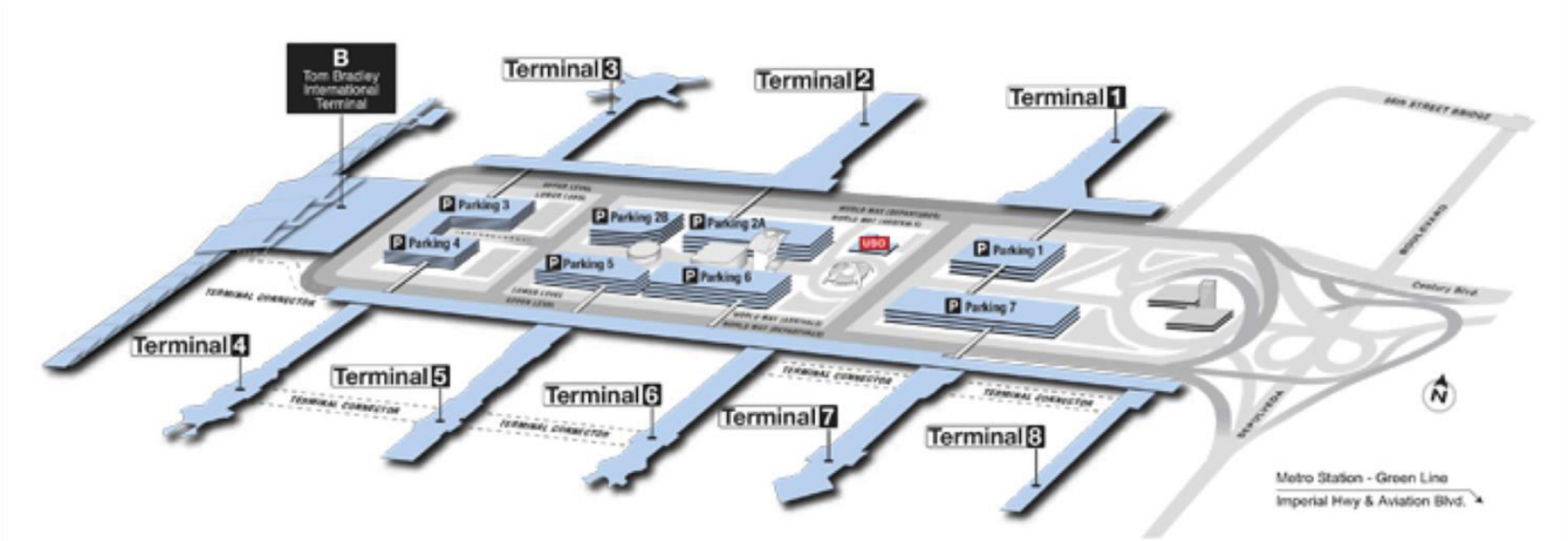
.has('property', condition)

Writing Gremlin Queries



Demo:Busy World Traveler

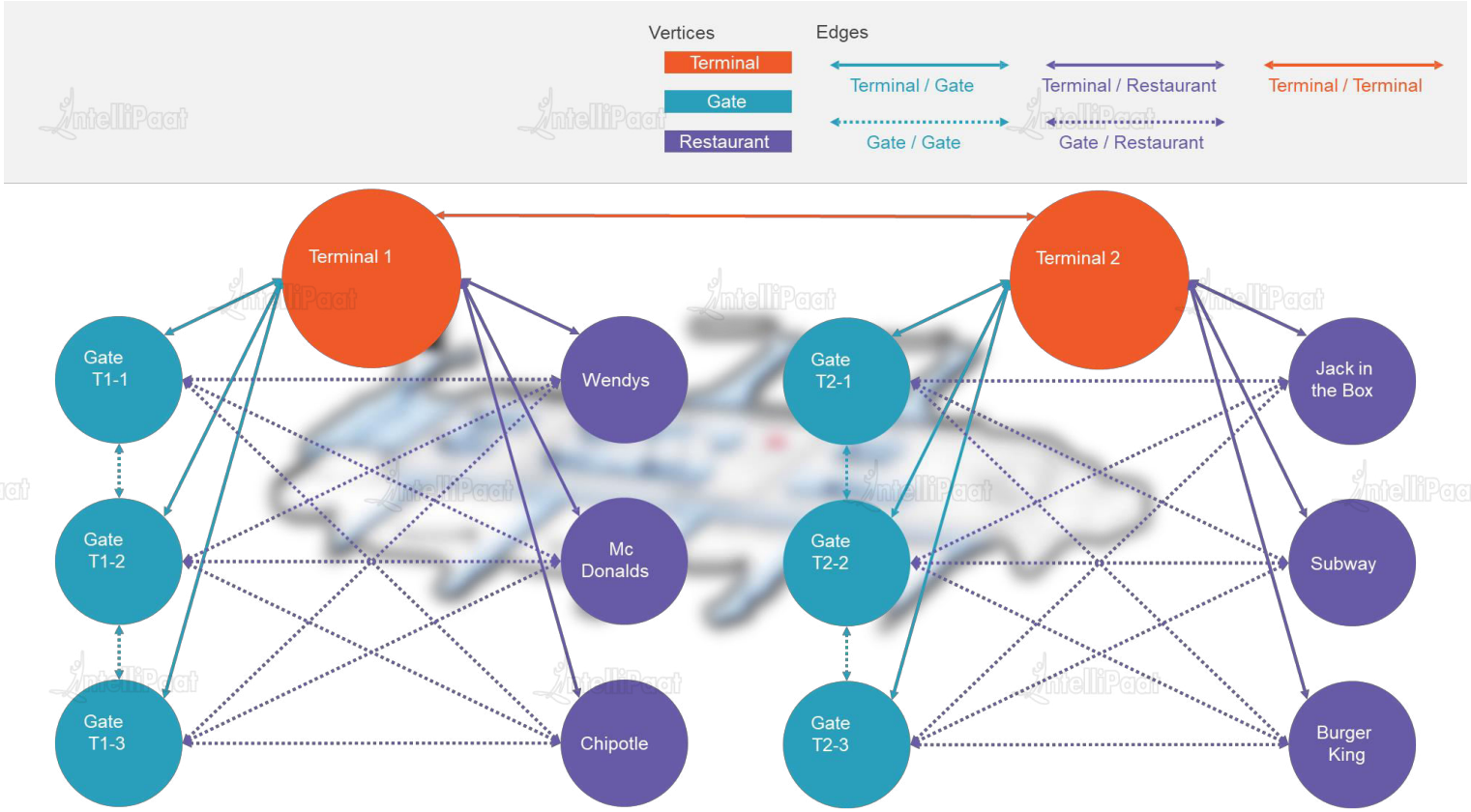
Demo:Busy World Traveler



Hands-On

- Creating a simple graph
- Busy world traveler

Demo:Busy World Traveler

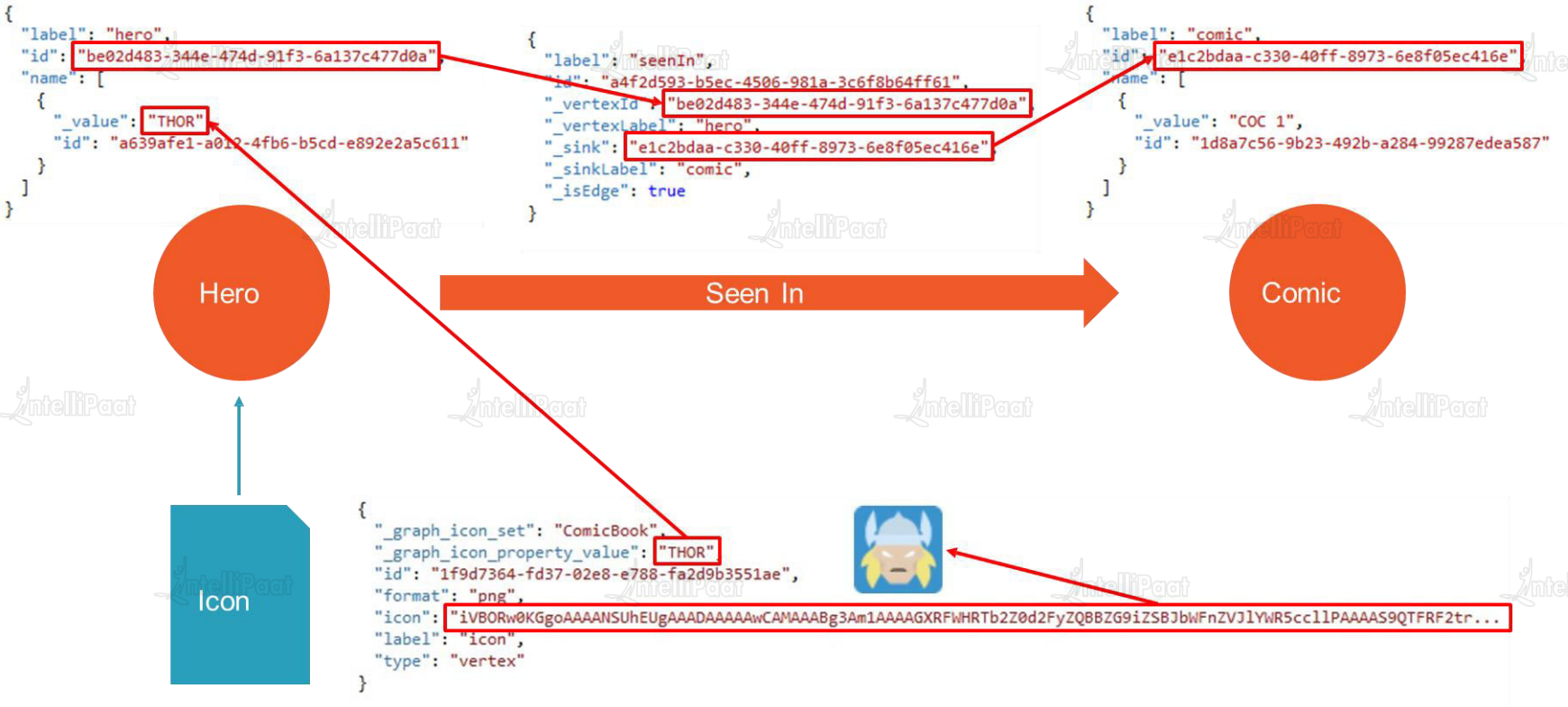


Hands-On

- Populating the airport graph
- Querying the airport graph
- Multi-model comic book catalog

Multi-Model Comic Book Catalog

Multi-Model Comic Book Catalog



Following This Course

Introduction & Overview

Storage Account, Table Storage Overview, Browsing and Creating Data

Integrating with Table Storage

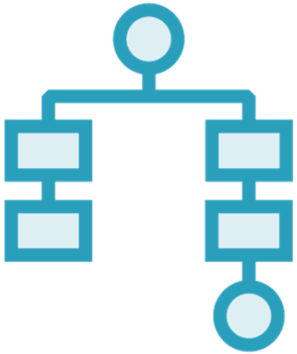
Working with the Azure Table Storage from .NET Applications to perform basic CRUD operations

Azure Table Storage

Why Choose the Azure Table Storage?

Low Latency

Scalable Apps & High Availability



Flexible data structure

Azure Table Storage

- NoSQL
- Key-value storage
- Uses JSON to serialize the data
- Flexible schema changes
- Scalable and highly available
- Single region with optional secondary read-only region

Different Azure Table Storage Options

Azure Table Storage

- No global distribution
- Additional read-only region
- Low latency
- Can only index using a primary key
- Eventual consistency using a second region
- Price optimized for Storage

Azure Cosmos DB Table API

- Globally distributed
- Slightly higher latency, single digit latency
- No need index management
- Can index on any property (automatic)
- Consistency can be tuned to the application specific needs
- Price optimized for Throughput

Hands-On

- Setting up Azure Table Storage from the Microsoft Azure Portal
- Using the Table Service REST API
- Setting up Azure Table Storage with the Azure Cosmos DB Table API
- Migrating to Azure Cosmos DB Table API

Hands-On

- Introducing Azure Table Storage in a .NET Application
- Modifying the Structure of Entities
- Migrating to Azure Cosmos DB Table API
- Generating Shared Access Signatures from Code
- Applying and Enforcing Table Policies
- Exporting Metrics

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor