Azure Cosmos DB Powered Elastic Tables in Microsoft's PowerApps: Unraveling the Possibilities

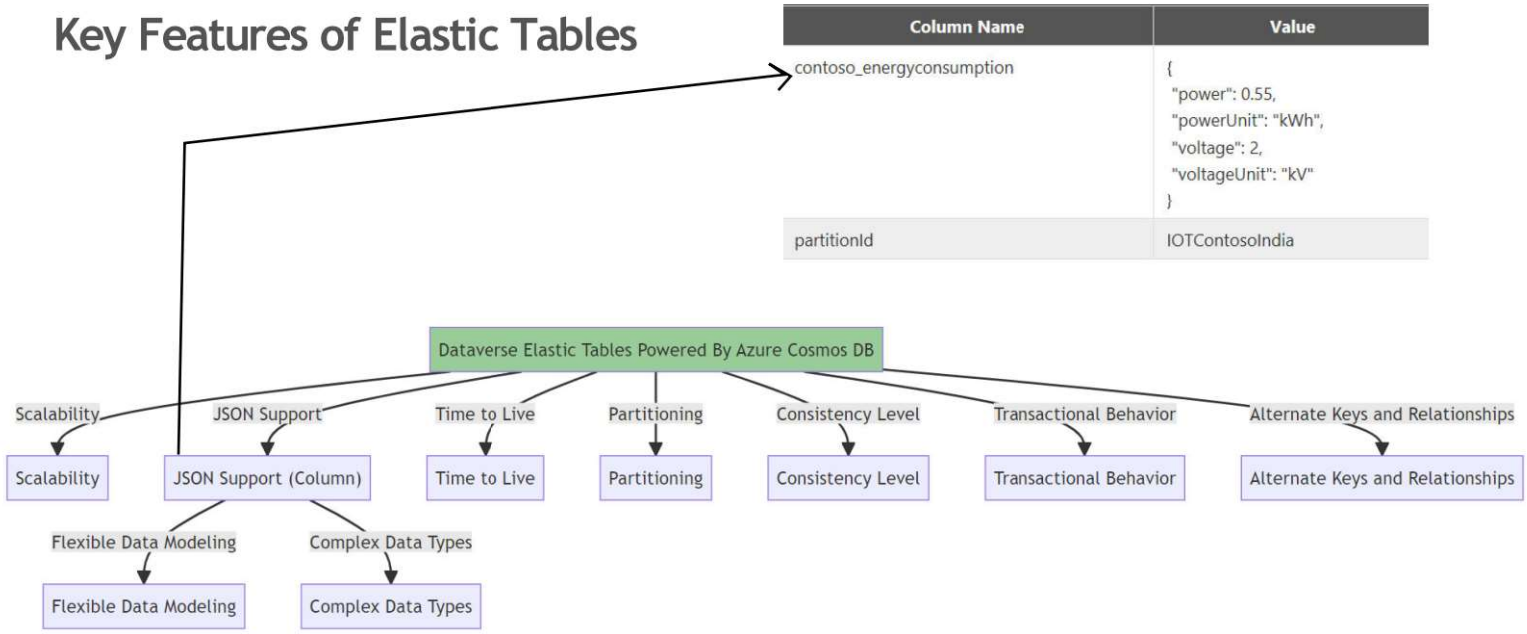
- Praveen

Microsoft's Power Apps is a powerful suite of tools that allows developers to create custom applications tailored to their business needs. One of the standout features of Power Apps is the Elastic Tables, a feature of Microsoft's Dataverse. In this blog post, we will delve into the features of Elastic Tables and explore their potential through various use cases and examples.

What are Elastic Tables?

Elastic Tables are a feature of Microsoft's Dataverse powered by Azure Cosmos DB. They are designed to handle large volumes of data and high levels of throughput with low latency. This makes them ideal for applications with unpredictable, spiky, or rapidly growing workloads.

Key Features of Elastic Tables

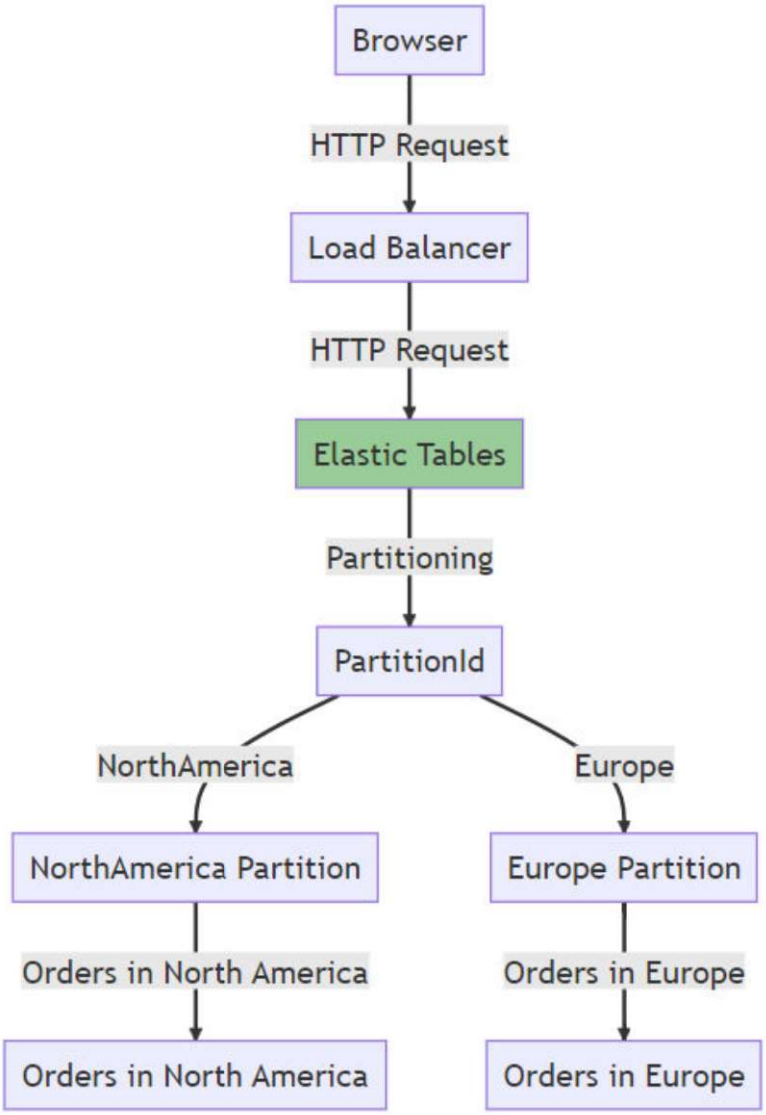



```
Partitioning and Horizontal Scaling
```

Partitioning is a method used in database management to divide a large table into smaller, more manageable parts, called partitions. Each partition can be managed and accessed independently of the others.

In the context of Elastic Tables, Azure Cosmos DB uses partitioning to scale individual tables according to the performance needs of your application. This is done by creating a system-defined Partition Id string column (with the schema name PartitionId and logical name partitionid) in all Elastic Tables.

Azure Cosmos DB ensures that the rows in a table are divided into distinct subsets, called logical partitions, based on the value of the partitionid column of each row.




```
Example
```

Imagine you're running an e-commerce platform that serves customers worldwide. You have a large table in your database that stores all the orders placed on your platform. As your business grows, this table becomes larger and larger, making it increasingly difficult to manage and query.

To handle this, you decide to use Elastic Tables, which support partitioning. Partitioning is the process of dividing a large table into smaller, more manageable parts, called partitions. Each partition can be managed and accessed independently, which can significantly improve the performance and scalability of your database.

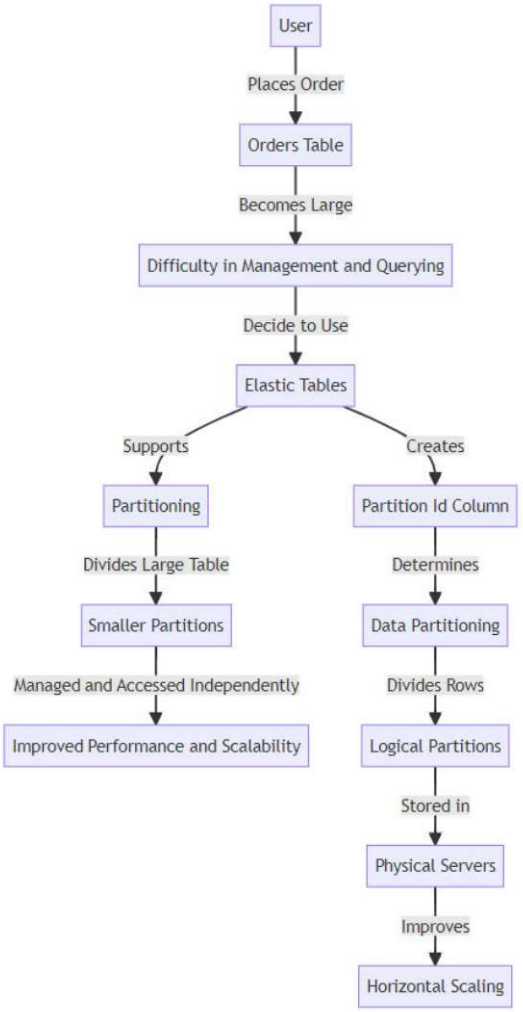
In Elastic Tables, Azure Cosmos DB handles the partitioning. It does this by creating a system-defined Partition Id string column (with the schema name PartitionId and logical name partitionid) in all Elastic Tables. The value of this column determines how the data is partitioned.

Let's say you decide to partition your orders table based on the region where the order was placed. You could use the region as the value for the PartitionId column. Azure Cosmos DB would then ensure that the rows in the table are divided into distinct subsets, called logical partitions, based on the value of the partitionid column of each row.

For example, all orders placed in North America would have a PartitionId of "NorthAmerica" and would be stored in the same logical partition. Similarly, all orders placed in Europe would have a PartitionId of "Europe" and would be stored in a separate logical Partition

This partitioning strategy allows your database to scale horizontally. As your business expands into new regions, new partitions can be added to accommodate the additional data. This makes it easier to manage the data and can significantly improve the performance of queries that are limited to a specific region.

Furthermore, because each partition can be managed independently, Azure Cosmos DB can distribute these partitions across multiple physical servers, spreading out the workload and further improving performance. This is known as horizontal scaling, and it's one of the key advantages of using Elastic Tables.



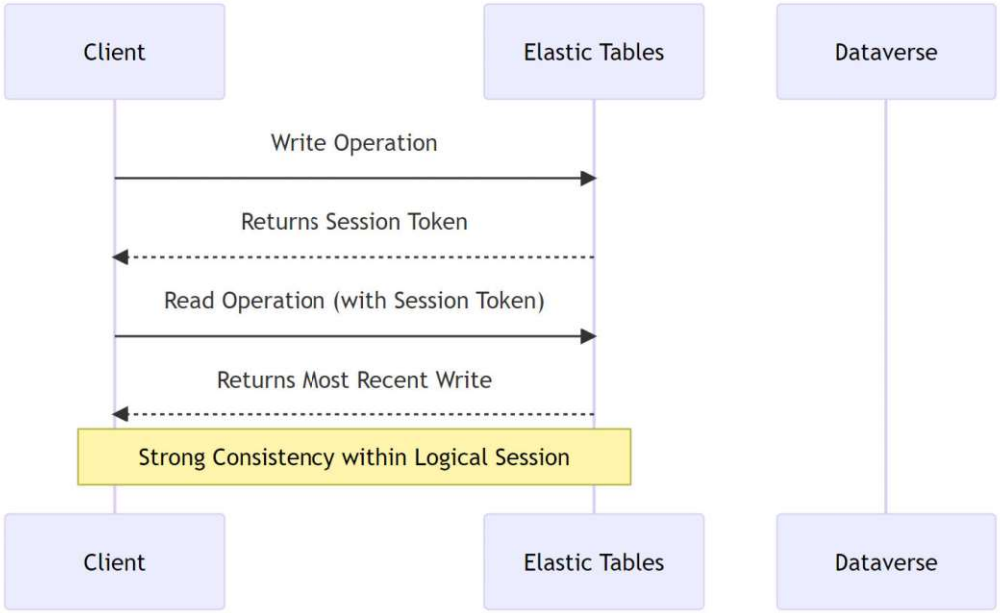

```
Consistency Level
```

Consistency in a database refers to ensuring that all changes to a database bring it from one valid state to another. Strong consistency means that a read operation will always return the most recent write operation.

Elastic Tables support strong consistency within a logical session. A logical session is a connection between a client and Dataverse. When a client performs a write operation on an Elastic Table, it receives a session token that uniquely identifies this logical session.

To maintain strong consistency, you need to include this session token with all subsequent requests. This ensures that all read operations performed within the same logical session context return the most recent write made within that logical session.

For instance, if you update a customer's address in a session, any subsequent reads within the same session will reflect this change. However, if a different session performs a write operation, other sessions may not see those changes immediately.




```
Transactional Behavior
```

In database systems, a transaction is a sequence of operations performed as a single logical unit of work. A multi-record transaction involves multiple operations on multiple records.

Elastic Tables do not support multi-record transactions. This means that for a single request execution, multiple write operations happening in the same or different synchronous plugin stages aren't transactional with each other.

Note:

It's important to note that Elastic Tables are designed for specific types of workloads - particularly those with large volumes of data and high throughput requirements. The trade-off for not having multi-record transactions is improved performance and scalability, as each operation can be performed independently.

In many use cases, especially those dealing with large-scale, distributed systems, this trade-off can be worth it. It's also worth noting that there are patterns and practices that can be used to manage consistency at the application level, such as compensating transactions or eventual consistency models.

Let's explore the transactional behavior of Elastic Tables with a couple of more examples:

```
Example 1: Inventory Management
```

Let's say you're building an inventory management system using Elastic Tables. In a traditional relational database, you might have a transaction that involves two steps:

Decreasing the quantity of a product in stock when a sale is made. Increasing the total sales for that product. In a traditional database, these two operations would be part of a single transaction. If the second operation fails (for example, due to a system error), the first operation would be rolled back to ensure data consistency.

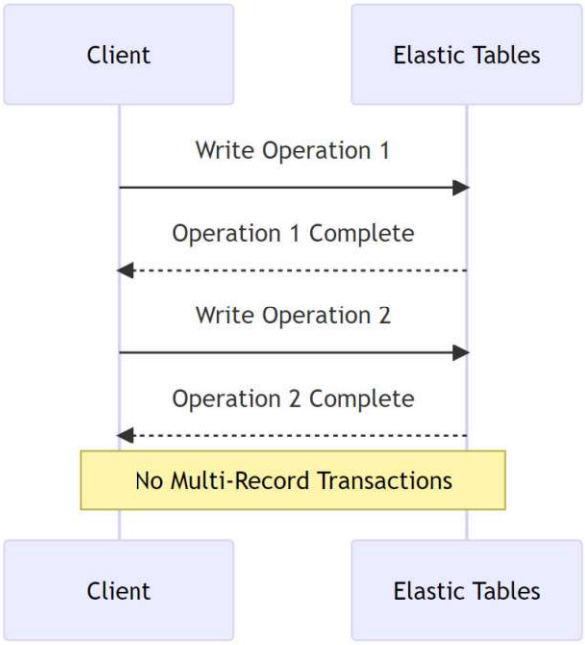
However, in Elastic Tables, these operations are not transactional. If the second operation fails, the first operation (decreasing the quantity of the product in stock) would not be rolled back. This means you would end up with an inconsistent state where your stock levels have decreased, but your total sales have not increased.

```
Example 2: Banking System
```

Consider a banking system where you want to transfer money from one account to another. This operation typically involves two steps:

Deducting the amount from the sender's account. Adding the same amount to the receiver's account. In a traditional database, these two operations would be part of a single transaction. If the second operation fails (for example, if the receiver's account number is incorrect), the first operation would be rolled back, and the sender's account balance would remain unchanged.

However, in Elastic Tables, these operations are not transactional. If the second operation fails, the first operation (deducting the amount from the sender's account) would not be rolled back. This means you would end up with an inconsistent state where money has been deducted from the sender's account but has not been added to the receiver's account.




```
Time to Live
```

The Time to Live (TTL) feature is indeed specific to Elastic Tables in Microsoft's Dataverse. This feature allows you to set an expiration time for data in your table, after which the data will be automatically deleted.

The TTL feature is particularly useful in scenarios where you need to store temporary data. For example, consider a scenario where you're storing session data for users of a web application. Each session might include data about the user's activity, such as pages visited, items added to a shopping cart, etc.

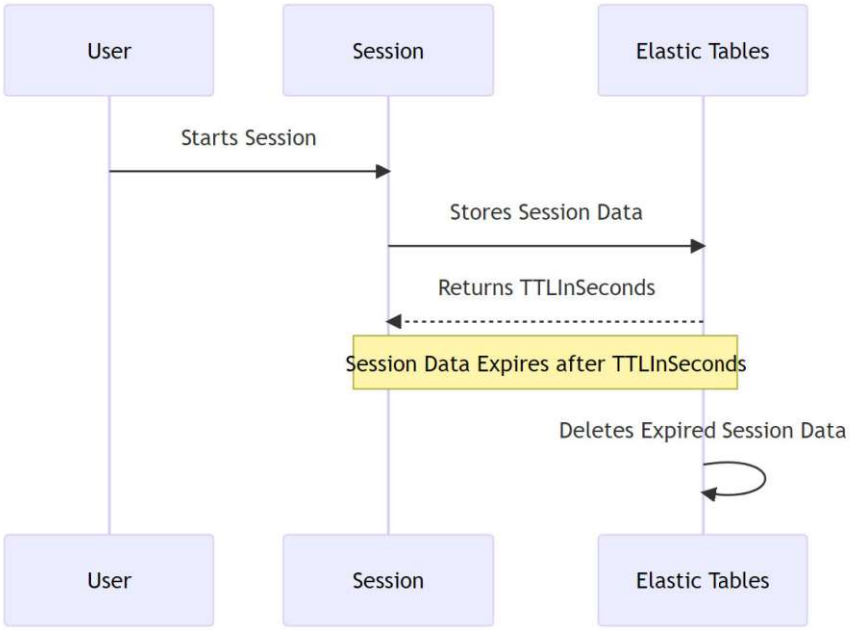
In this scenario, you might only need to keep this session data for a limited period of time, such as 24 hours. After this time, the session data is no longer relevant and can be safely deleted. By setting the TTL value for each row in your Elastic Table, you can ensure that old session data is automatically cleaned up, saving storage space and reducing clutter in your table.

```
Here's how it works:
```

Dataverse automatically creates an integer column with the name Time to Live, schema name TTLInSeconds, and logical name ttlinseconds in each Elastic Table. When you insert a new row into the table, you can set a value for this column. The value represents the time in seconds after which the row will be automatically deleted.

For example, if you set the TTLInSeconds value to 86400 (the number of seconds in 24 hours) when inserting a new row, that row will automatically be deleted 24 hours after it was inserted.

This feature can be a powerful tool for managing temporary data in your applications, helping to keep your Elastic Tables clean and efficient.




```
Handling Unstructured or Semi-Structured Data
```

Elastic Tables are ideal when your data may be unstructured or semi-structured, or if your data model may constantly be changing. For instance, if you are dealing with data from various sources that do not conform to a specific structure, Elastic Tables can be a great fit. The ability to store JSON data within a column allows you to keep complex data structures within a single field.

```
Handling High Volume of Read and Write Requests
```

If your application needs to handle a high volume of read and write requests, Elastic Tables can be a good choice. For example, if you are building a social media application that needs to handle a large number of posts, comments, and likes, Elastic Tables can provide the scalability and performance you need.

Advanced Features of Elastic Table

```
Alternate Keys:
```

In many databases, an alternate key is a column or set of columns that can be used to uniquely identify a row, just like a primary key.

However, in the context of Elastic Tables in Microsoft's Dataverse, you cannot create custom alternate keys. This means that you cannot specify your own set of columns to serve as an alternate key.

Despite this limitation, each Elastic Table is automatically created with one predefined alternate key.

This alternate key is composed of two values: **the primary key of the table and the partitionid.**

Here are the details of this predefined alternate key:

Display Name: "Entity key for NoSql Entity that contains PrimaryKey and PartitionId attributes"

Name: "KeyForNoSqlEntityWithPKPartitionId"

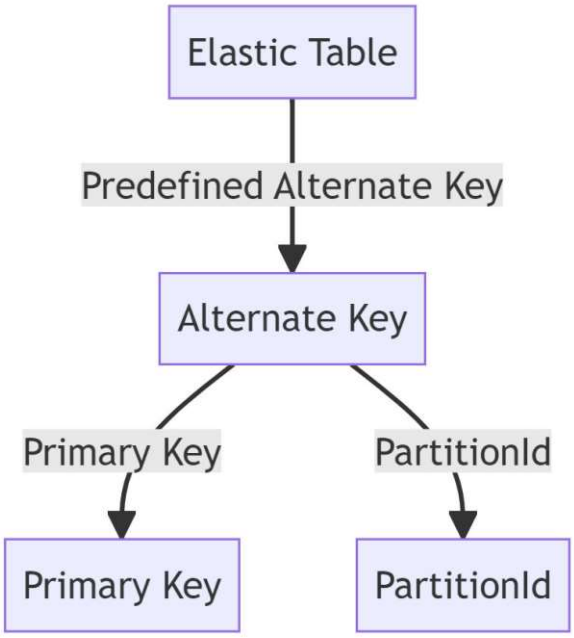
LogicalName: "keyfornosqlentitywithpkpartitionid"

The key values for this alternate key are the primary key of the table and the **partitionid**.

This predefined alternate key can be useful when you need to reference a record that has a **partitionid** value set. Instead of referencing the record by its primary key alone, you can use the alternate key, which includes both the primary key and the **partitionid**.

For example, if you have a table with a primary key **id** and a **partitionid** 'NorthAmerica', you can reference a specific record in this table using the alternate key composed of these two values.

This feature allows you to work with Elastic Tables in a way that is similar to working with partitioned tables in other database systems, where a record is often identified by both its primary key and its partition.




```
Relationships:
```

Many-to-Many Relationships:

Currently, Dataverse does not support creating Many-to-Many relationships with Elastic Tables. A Many-to-Many relationship is when multiple records in one table are associated with multiple records in another table.

For example, in a bookstore database, a book can have multiple authors, and an author can write multiple books. This limitation means that you cannot directly link multiple records in an Elastic Table to multiple records in another Elastic Table or standard table.

One-to-Many Relationships:

One-to-Many relationships are supported for Elastic Tables, but with some limitations. A One-to-Many relationship is when a record in one table can be associated with multiple records in another table. For example, in a customer-orders scenario, one customer can have many orders, but each order is associated with only one customer.

Here are the limitations for One-to-Many relationships in Elastic Tables:

Cascading Isn't Supported:

Cascading refers to the automatic propagation of changes from one record to related records. For example, if you delete a record, any related records would also be deleted. In Elastic Tables, you must set the cascading behavior to Cascade.None when creating a relationship, meaning that changes to a record will not automatically propagate to related records.

Formatted Values for Lookup Columns Aren't Returned:

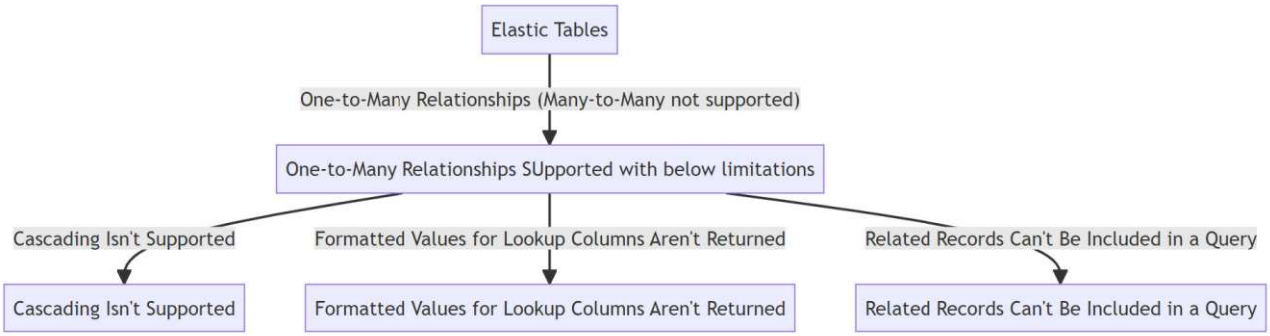
In certain conditions, the formatted values for lookup columns (columns that reference another table) are not returned. This happens when the table being retrieved is a standard table and the lookup refers to an Elastic Table, and when the partitionid value of the Elastic Table is set to a custom value (not the default primary key value of the Elastic Table row).

Related Records Can't Be Included in a Query:

While you can retrieve related rows when retrieving a record from an Elastic Table, you cannot include related records in a query. This means that you cannot write a query that retrieves records from an Elastic Table along with their related records in one operation.

These limitations are important to keep in mind when designing your data model and writing queries for Elastic Tables.

Despite these limitations, One-to-Many relationships can still be a powerful tool for modeling complex data structures in Elastic Tables.



When to Use Elastic Tables

Elastic Tables are a powerful tool in Microsoft's Dataverse, but they are not suitable for every scenario. Here are some situations where Elastic Tables can be particularly beneficial:

```
High Volume of Data:
```

Elastic Tables are designed to handle large volumes of data efficiently. If your application needs to store and manage millions or even billions of records, Elastic Tables can be a good choice.

```
High Throughput:
```

Elastic Tables can handle a high level of throughput, making them suitable for applications that need to perform a large number of read and write operations per second.

```
Unpredictable Workloads:
```

Elastic Tables can automatically scale to meet the demands of your workload. If your application has unpredictable, spiky, or rapidly growing workloads, Elastic Tables can adjust to meet these demands.

```
JSON Data:
```

Elastic Tables support storing JSON data in string columns. If your application needs to store complex data structures or semi-structured data, Elastic Tables can be a good choice.

```
Time-bound Data:
```

Elastic Tables support Time to Live (TTL) for data rows. If your application needs to store data that is only relevant for a certain period of time, you can use the TTL feature to automatically delete this data when it's no longer needed.

Use Cases and Examples

Use Case	Description
IoT Data Management	Elastic Tables can be used to handle large volumes of data generated by Internet of Things (IoT) devices. For example, a company could use an Elastic Table to store sensor data from thousands of devices, with each device's data stored in a separate partition.
Log Data Storage	Applications often generate large volumes of log data. Elastic Tables can be used to store this data, allowing for scalable, efficient storage and querying.
Real-Time Analytics	Elastic Tables can be used to store data for real-time analytics. Their ability to handle high levels of throughput makes them suitable for applications that need to process large volumes of data in real time.
Content Management Systems	Elastic Tables can be used in content management systems to store and manage large volumes of content. For example, a news website could use an Elastic Table to store articles, with each article stored as a JSON object in a single row.
Social Media Applications	Elastic Tables can be used to handle the large volumes of data generated by social media applications. For example, a social media app could use an Elastic Table to store user posts, with each post stored as a JSON object in a single row.
Personalization and Recommendation Systems	Elastic Tables can be used to store user behavior data for personalization and recommendation systems. Their ability to handle large volumes of data and high levels of throughput makes them suitable for these types of applications.

Quick comparison of the actions that can be performed on Elastic Tables using SDK and Web API, along with a brief description of each action.

As you can see, while most actions can be performed using both SDK and Web API, the CreateMultiple, UpdateMultiple, and DeleteMultiple actions are currently only available in C# SDK.

Action	C# SDK	Web API	Description
Create Elastic Tables	Yes	Yes	Create new Elastic Tables.
Create a Column with JSON Format	Yes	Yes	Add a new column with JSON format to an existing Elastic Table.
Getting the Session Token	Yes	Yes	Retrieve the session token for maintaining strong consistency.
Sending the Session Token	Yes	Yes	Send the session token with requests to maintain strong consistency.
Using Alternate Key	Yes	Yes	Use an alternate key to uniquely identify a record.
Using PartitionId Parameter	Yes	Yes	Use the PartitionId parameter to divide data into logical partitions.
Create a Record in an Elastic Table	Yes	Yes	Add a new record to an Elastic Table.
Update a Record in an Elastic Table	Yes	Yes	Modify an existing record in an Elastic Table.
Retrieve a Record in an Elastic Table	Yes	Yes	Fetch a specific record from an Elastic Table.
Query Rows of an Elastic Table	Yes	Yes	Perform a query to retrieve multiple rows from an Elastic Table.
Upsert a Record in an Elastic Table	Yes	Yes	Create a new record or update an existing record in an Elastic Table.
Delete a Record in an Elastic Table	Yes	Yes	Remove a specific record from an Elastic Table.
Set JSON Column Data	Yes	Yes	Add or modify data in a JSON column of an Elastic Table.
Query JSON Column Data	Yes	Yes	Perform a query to retrieve data from a JSON column of an Elastic Table.
Use CreateMultiple with Elastic Tables	Yes	No	Perform a bulk create operation. Only available in SDK.
Use UpdateMultiple with Elastic Tables	Yes	No	Perform a bulk update operation. Only available in SDK.

Action	C# SDK	Web API	Description
Use DeleteMultiple with Elastic Tables	Yes	No	Perform a bulk delete operation. Only available in SDK.

References:

https://learn.microsoft.com/en-us/power-apps/developer/data-platform/elastic-tables

https://learn.microsoft.com/en-us/power-apps/developer/data-platform/create-elastic-tables?tabs=sdk

https://learn.microsoft.com/en-us/power-apps/developer/data-platform/bulk-operations-elastic-tables

https://github.com/microsoft/PowerApps-Samples/blob/master/dataverse/webapi/C%23-NETx/ElasticTableOperations/README.md