> *Bases de Datos 2, Resumen 4*
> *Miguel Ku Liang - 2019061913*

## Introduction

**Azure DocumentDB** is Microsoft's multi-tenant distributed database service for managing JSON documents at Internet scale. DocumentDB's indexing subsystem needs to support automatic indexing of documents without requiring a schema or secondary indices, DocumentDB's query language, real-time consistent queries in the face of sustained high document ingestion rates and multi-tenancy under extremely frugal resource budgets while providing predictable performance guarantees and remaining cost effective.

### Overview of the Capabilities

- The query language supports rich relational and hierarchical queries.
- The database engine is optimized to serve consistent queries in the face of sustained high volume document writes.
- Transactional execution of application logic provided via stored procedures and triggers.
- DocumentDB offers consistency levels and corresponding performance guarantees.
- All machine and resource management is abstracted from users.

### Design Goals for Indexing

- Automatic indexing
- Configurable storage/performance tradeoffs
- Efficient, rich hierarchical and relational queries
- Consistent queries in face of sustained volume of document writes
- Multi-tenancy

## Schema Agnostic Indexing

### No Schema, No Problem!

The schema of a document describes the structure and the type system of the document independent of the document instance. DocumentDB exploits the simplicity of JSON and its lack of a schema specification.

### Documents as Trees

Representing JSON documents as trees normalizes the structure and the instance values across documents into a dynamically encoded path structure.

### Index as a Document

Every path in a document tree is indexed. Each update of a document to a collection leads to update of the structure of the index.

There are two possible mappings of document and the paths:

- **forward index mapping**: keeps a map of (document id, path) tuples
- **inverted index mapping**: keeps a map of (path, document id) tuples

### DocumentDB Queries

Developers can query DocumentDB collections using queries written in SQL and JavaScript. They operate directly against the tree representation.

## Logical Index Organization

### Directed Paths as Terms

A term represents a unique path in the index tree.

### Encoding Path Information

The number of segments in each term is important in terms of trade-off between query functionality, performance and indexing cost. We default three segments because most of the JSON documents have root, a key and a value in most of the paths and it helps distinguish various paths from each other.

### Partial Forward Path Encoding Scheme

It involves parsing of the document from the root and selcting three suffix nodes successively to yield a distinct path consisting of three segments. This scheme is used to do range and spatial indexing.

### Partial Reverse Path Encoding Scheme

It is similar to the previous scheme, but the term generated is in reverse order. This scheme is suitable for point query performance.

### Bitmaps as Postings Lists

A **postings list** captures the document ids of all the documents which contain the given term. To represent a postings list, we do:

- **Partitioning a Postings List**: each insertion of a new document to a DocumentDB collection is assigned an increasing document id.
- **Dynamic Encoding of Posting Entries**: each document needs 14 bits which can be captured with a short word.

### Customizing the Index

The default indexing policy indexes all properties of all documents and provides consistent queries. Developers can override the indexing policy by configuring the following:

- Including/Excluding documents and paths to/from index
- Configuring various index types
- Configuring index update modes

## Physical Index Organization

### The "Write" Data Structure

Consistent indexing in DocumentDB provides fresh query results for sustained document ingestion. This is a problem in a multi-tenant setting with frugal budget for memory, CPU and IOPS.

### The Bw-Tree for DocumentDB

Extending the Bw-Tree could meet the requirements that were described. It uses latch-free in-memory updates and log structured storage for persistence.

### High Concurrency

The Bw-Tree operates in a latch-free manner, allowing high concurrency. A modification of a page is done by appending a delta record on top of the page.

#### Write Optimized Storage Organization

In a B+-Tree, storage is organized into fixed size pages. The page is read into memory, updated in-place and written back to storage.

#### Index Updates

#### Document Analysis

It is performed by the document analyzer in the indexing subsystem. The document analysis function takes the document content, a logical timestamp when it was last updated and the indexing policy yields a set of paths.

#### Efficient and Consistent Index Updates

DocumentDB's database engine uses a merge callback function to consolidate Bw-Tree pages by combining fragments of key values across base page and delta records.

#### Lazy Index Updates with Invalidation Bitmap

DocumentDB collection configured with the lazy indexing mode is performed in the background. Maintain a counting invalidation bitmap which is a bitmap representing the document ids from the deleted and replaced document images and a count representing the number of times the document update has been recorded.

#### Index Replication and Recovery

DocumentDB follows a single master model for writes. The primary replica considers the write operation successful if it is durably committed to local disk by the write quorum of replicas. For read operations, the client contacts the read quorum to determine the correct version of the resource.

#### Index Resource Governance

DocumentDB offers richer access functionality than a key-value store. A Request Unit encapsulates a chunk of CPU, memory and IOPS. RU/second provides the unit for accounting, provisioning, allocating and consuming throughput guarantees.

#### Index Resource Governance

- CPU resources: DocumentDB database engine manages its thread scheduler.
- Memory resources: DocumentDB database engine and its components operates within a given memory budget.
- Storage IOPS resources: DocumentDB database engine needs to operate within a given IOPS budget.

### Insights from the Production Workloads

#### Document Frequency Distribution

Document frequency distribution for the unique terms universally follow Zipf's Law.

#### Query Performance

It is defined in terms of the number of false positives in postings for a given term lookup.

#### Blind Incremental Updates

Highly performant index updates within an extremely frugal memory and IOPS budget is the reason behind the blind incremental update access method.