Creación y uso de índices en MongoDB

En esta sección se analizan distintos tipos de índices con el objetivo de evaluar su impacto en el rendimiento de las consultas.

Se comparará el plan de ejecución de MongoDB antes y después de crear cada índice, observando:

- Tipo de acceso a datos (COLLSCAN vs IXSCAN)
- Número de documentos examinados
- Tiempo de ejecución

Esto permitirá comprobar cómo el uso de índices optimiza el acceso a la información en el modelo documental construido.

Conexión

In [1]:
```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017")
db = client["actividad_comercial_madrid"]
col = db["locales"]

print("Documentos:", col.count_documents({}))
```

```
Documentos: 151162
```

Función de análisis de rendimiento

Se define una función que ejecuta una consulta y devuelve estadísticas reales de ejecución obtenidas mediante `explain("executionStats")`. Esto permite comparar cuantitativamente el impacto de los índices.

In [3]:
```
def stats(query):
    exp = col.find(query).explain()
    return {
```

```
        "docs_examined": exp["executionStats"]["totalDocsExamined"],
        "execution_time_ms": exp["executionStats"]["executionTimeMillis"],
        "stage": exp["queryPlanner"]["winningPlan"]["stage"]
    }
```

Índice simple sobre el barrio

Se crea un índice simple sobre el campo `desc_barrio_local`.

Este tipo de índice optimiza búsquedas que filtran por un único campo, evitando recorrer toda la colección.

In [4]:
```
#CONSULTA SIN ÍNDICE

# Borramos los índices para comparar el rendimiento de la consulta sin índices (no se borra bien reiniciando el kernel)
col.drop_indexes()

query = {"desc_barrio_local": "JUSTICIA"}
stats(query)
```

Out[4]:
```
{'docs_examined': 151162, 'execution_time_ms': 382, 'stage': 'COLLSCAN'}
```

In [5]:
```
#CONSULTA CON ÍNDICE

# Creamos un índice en el campo "desc_barrio_local"
col.create_index({"desc_barrio_local": 1})

#consulta con índice
stats(query)
```

Out[5]:
```
{'docs_examined': 1950, 'execution_time_ms': 11, 'stage': 'FETCH'}
```

Tras la creación del índice, el plan de ejecución cambia de COLLSCAN (escaneo completo de colección) a IXSCAN (búsqueda indexada).

Se reduce significativamente el número de documentos examinados, demostrando la mejora de rendimiento en consultas puntuales.

Índice compuesto (distrito + barrio)

Muchas consultas utilizan simultáneamente distrito y barrio, por lo que se crea un índice compuesto.

El orden del índice es importante: primero distrito y después barrio, ya que las consultas suelen filtrar primero por distrito.

In [ ]:
```
#CONSULTA SIN ÍNDICE

# Borramos los índices para comparar el rendimiento de la consulta sin índices (no se borra bien reiniciando el kernel)
col.drop_indexes()

query = {
    "desc_distrito_local": "CENTRO",
    "desc_barrio_local": "JUSTICIA"
}
stats(query)
```

Out[ ]:
```
{'docs_examined': 151162, 'execution_time_ms': 96, 'stage': 'COLLSCAN'}
```

In [7]:
```
#CONSULTA CON ÍNDICE COMPUESTO

# Creamos un índice compuesto en los campos "desc_distrito_local" y "desc_barrio_local"
col.create_index({
    "desc_distrito_local": 1,
    "desc_barrio_local": 1
})

stats(query)
```

Out[7]:
```
{'docs_examined': 1950, 'execution_time_ms': 10, 'stage': 'FETCH'}
```

El índice compuesto permite resolver consultas multicampo sin combinar índices individuales.

MongoDB puede localizar directamente los documentos que cumplen ambas condiciones, reduciendo aún más los documentos examinados respecto al índice simple.

Índice sobre arrays (multikey)

El campo `actividad_economica.desc_epigrafe` es un array embebido. Al crear un índice sobre él, MongoDB genera automáticamente un índice multikey.

Este índice permite buscar documentos que contengan un valor concreto dentro del array sin recorrer toda la colección.

In [8]:
```
#CONSULTA SIN ÍNDICE

# Borramos los índices para comparar el rendimiento de la consulta sin índices (no se borra bien reiniciando el kernel)
col.drop_indexes()

query = {"actividad_economica.desc_epigrafe": "BAR CON COCINA"}
stats(query)
```

Out[8]:
```
{'docs_examined': 151162, 'execution_time_ms': 156, 'stage': 'COLLSCAN'}
```

In [9]:
```
#CONSULTA CON ÍNDICE COMPUESTO

# Creamos un índice compuesto en el campo anidado "actividad_economica.desc_epigrafe"
col.create_index({"actividad_economica.desc_epigrafe": 1})

stats(query)
```

Out[9]:
```
{'docs_examined': 4540, 'execution_time_ms': 13, 'stage': 'FETCH'}
```

MongoDB crea automáticamente un índice multikey al indexar campos que contienen arrays.

Esto permite localizar rápidamente documentos que contienen un valor dentro del array, optimizando especialmente consultas analíticas sobre modelos embebidos.

Si creamos el indice sin .desc_epigrafe al final lo que obtenemos es otro COLLSCAN y tarda lo mismo que la query anterior sin indice.

Conclusión

La utilización de índices mejora significativamente el rendimiento de las consultas:

- El índice simple optimiza búsquedas puntuales

- El índice compuesto mejora consultas multicampo
- El índice multikey permite explotar eficientemente arrays embebidos

Se demuestra que el modelo documental diseñado es compatible con consultas analíticas eficientes mediante el uso adecuado de índices.

Esto podemos confirmarlo viendo los resultados:

- Para la primera consulta sin indice se escanean 151162 documentos y tarda 88ms. Mientras que con indice escanea 1950 y tarda 11ms.
- Para la segunda consulta sin indice compuesto se escanean 151162 documentos y tarda 93ms. Mientras que con indice escanea 1950 y tarda 11ms.
- Para la segunda terca sin indice se escanean 151162 documentos y tarda 132ms. Mientras que con indice escanea 1950 y tarda 3ms.

Con estos resultados confirmamos que es extremadamente más eficiente crear indices para las consultas.