2. BASES DE DATOS COLUMNARES

2.1. Column-Family Stores

Ejemplos de BD Column-Family stores

- Cassandra
- HBase
- Hypertable
- Amazon SimpleDB

Permiten almacenar datos con keys mapeados con valores, y valores agrupados en múltiples familias de columnas , cada familia representa a un mapa de datos.

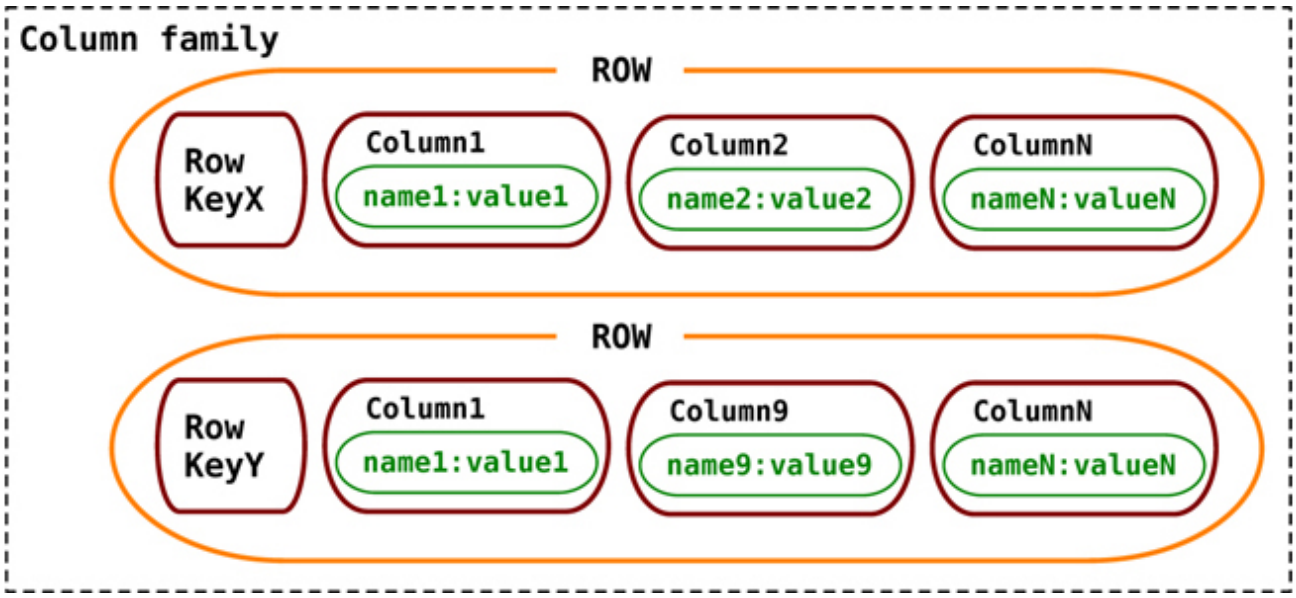
SImilitudes con RDBMS

- Instancia → cluster
- Base de datos → Keyspace
- Tabla → familia de columnas
- Row → Row
- Columna → columna

Los siguientes conceptos ilustran este patrón de almacenamiento empleando a Cassandra como base de datos de referencia.

2.1.1. Familia de columnas

Los datos se almacenan agrupándolos en ***familias de columnas***. Los registros están formados por un conjunto de columnas que tienen cierta correspondencia, por ejemplo, las columnas siempre se consultan juntas. Cada renglón se asocia a una ***row key.***



2.1.1.1. Columnas

- La unidad básica de almacenamiento es la ***columna***.
- Cada columna está formada por 3 elementos: name, value, y timestamp. El nombre de la columna actúa como una ***key***.
- El timestamp se emplea para detectar datos expirados, para resolver conflictos de escritura, etc.

Ejemplo

```
{
  name: "firstName",
  value: "Martin",
  timestamp: 12345667890
}
```

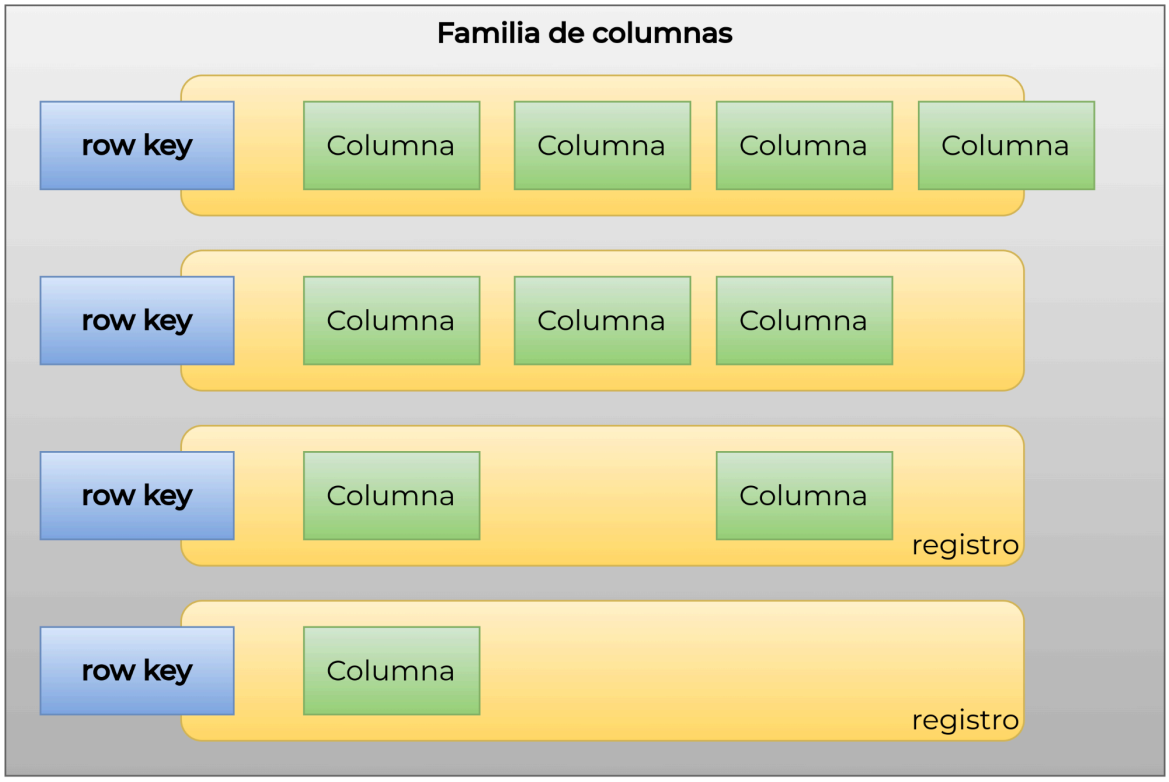
2.1.1.2. Registros

- Un registro está formado por una colección de columnas asociada o ligada a una key (***row key***).
- Un conjunto de registros asociados a estas columnas forman una Familia de columnas

```
//column family
{
//row
  "pramod-sadalage" : {
     firstName: "Pramod",
     lastName: "Sadalage",
     lastVisit: "2012/12/12"
  }
//row
  "martin-fowler" : {
     firstName: "Martin",
     lastName: "Fowler",
     location: "Boston"
  }
}
```

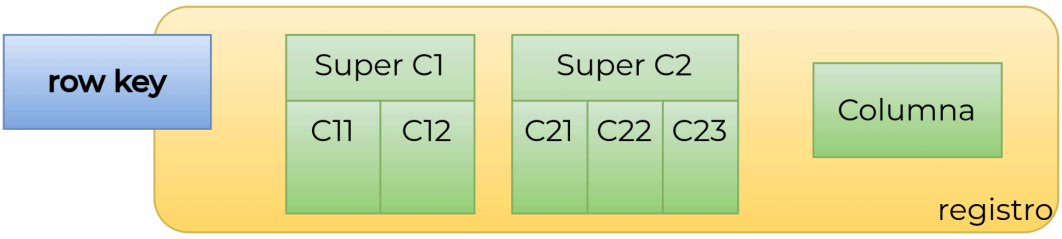
2.1.1.3. Familia de columnas

- Una familia de columnas puede compararse con una tabla en donde la PK define de forma única a un registro y cada registro está formado por varias columnas.
- La diferencia con una familia de columnas es que cada registro puede tener diferentes columnas (comparar los 2 registros del ejemplo anterior, `lastVisit` y location son columnas `diferentes` , columnas pueden ser agregadas a un registro en cualquier momento sin tener que agregarlas a otros registros.
- Cuando las columnas de la familia tienen valores simples, se le conoce como ***familia de columnas estándar*** Si el valor de una columna es a su vez un conjunto de columnas, se le conoce como súper columna.



2.1.2. Super columnas

Una súper columna está formada por un mapa de columnas, es decir, el ***value*** de la columna es un mapa de columnas. Este concepto puede ser comparado con un contenedor de columnas.



Ejemplos

```
{
  name: "book:978-0767905923",
  value: {
    author: "Mitch Albon",
    title: "Tuesdays with Morrie",
    isbn: "978-0767905923"
  }
}
```

- Super Column family: Se forma cuando la familia hace uso de super columnas.

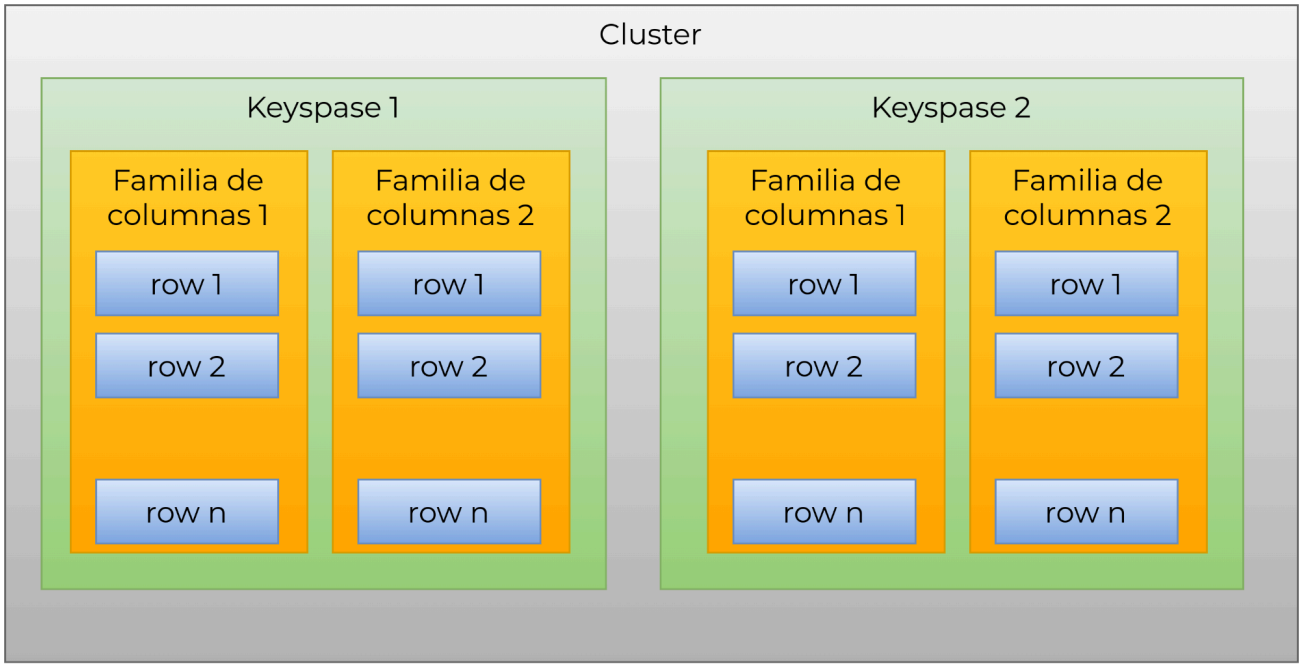
```
//super column family
{
//row
name: "billing:martin-fowler",
value: {
  address: {
    name: "address:default",
    value: {
      fullName: "Martin Fowler",
      street:"100 N. Main Street",
      zip: "20145"
    }
  },
  billing: {
    name: "billing:default",
    value: {
      creditcard: "8888-8888-8888-8888",
      expDate: "12/2016"
      }
    }
  }
//row
name: "billing:pramod-sadalage",
value: {
  address: {
    name: "address:default",
    value: {
      fullName: "Pramod Sadalage",
      street:"100 E. State Parkway",
      zip: "54130"
```

```
    }
   },
   billing: {
      name: "billing:default",
      value: {
        creditcard: "9999-8888-7777-4444",
        expDate: "01/2016"
        }
      }
   }
}
```

- Las familias de super columnas son útiles para mantener datos que se relacionan entre sí y que siempre se consultan juntos.
- Si se realiza una consulta y no se ocupan todas las columnas, se tiene un uso ***no óptimo*** de recursos.

2.1.3. Keyspaces, clusters y particiones

Un conjunto de familias de columnas tanto estándar como super columnas forman un keyspace. Un keyspace ***debe*** ser creado antes de poder crear a sus familias de columnas



Ejemplo

```
-- Create a keyspace
create keyspace if not exists store with replication = {
  'class' : 'simplestrategy',
   'replication_factor' : '1'
 };
```

Un clúster está conformado por uno o más keyspaces. En un sistema de bases de datos relacionales, un keyspace corresponde con una base de datos. Cada keyspace tiene asociado un conjunto de atributos que definen el comportamiento global del keyspace. Los atributos más importantes que se pueden configurar son:

Factor de replicación

El factor de replicación hace referencia al número de copias que debe guardarse de cada dato de la base de datos. Este parámetro impacta directamente en el nivel de consistencia y de rendimiento del sistema. A mayor factor de replicación, más rendimiento y menos consistencia

.

Estrategia de localización de réplicas

Este parámetro establece qué política se seguirá para alojar las réplicas en el anillo, y para decidir qué nodos serán los que alojen qué copias. Las políticas de localización de réplicas tienen en cuenta aspectos como la tomar en consideración el rack donde están alojados los servidores, los centros de datos, la topología de la red, etc.

Column families

Los keyspaces en Cassandra actúan como ***contenedores*** de familias de columnas, de igual manera que en el modelo relacional una database sería un contenedor de tablas. Cada column family contendrá varias columnas, y las filas se conforman de estas estructuras para almacenar la información final.

2.1.3.1. Particiones

Existe una parte del row key de cada registro que permite identificar al nodo dentro del cluster en el que se almacenará dicho registro. Una consulta puede especificar un partition key para mejorar desempeño al consultar solo los nodos asociados.

2.1.4. Consistencia

Cuando se realiza una escritura, los datos se almacenan en un ***commit log***, posteriormente, los datos se mantienen en caché llamado ***memtable.*** Para que la escritura se considere exitosa, esta se debe escribir en las 2 estructuras anteriores.

Todas las lecturas tienen un valor `consistency=1` por default. Esto significa que al solicitar una lectura, el dato se obtiene de la primera réplica sin importar que el dato pudiera ya estar desactualizado. Lecturas posteriores podrán obtener los valores actualizados: ***read repair.***

De forma similar, las escrituras tienen un valor `consistency=1`. Esto significa que basta con escribir en el commit log de un nodo para considerar la escritura válida.

Existe otro nivel de consistencia: `consistency=quorum`. En este caso, se asegura que la mayoría de los nodos coinciden empleando el timestamp más reciente.

El nivel `consistency=all` requiere que todos los nodos respondan.

Para sincronizar las réplicas se emplea el comando `repair`.

```
repair ecommerce customerInfo
```

2.1.5. Transacciones

Cassandra no implementa el concepto tradicional de transacciones. Una escritura es atómica a nivel renglón o registro. Para sincronizar réplicas se pueden emplear soluciones externas como Zookeeper o Cages.

2.1.6. Consultas básicas

Formadas por los comandos `GET`, `SET`, `DEL`

```
CREATE COLUMN FAMILY Customer
WITH comparator = UTF8Type
AND key_validation_class=UTF8Type
AND column_metadata = [
  {column_name: city, validation_class: UTF8Type}
  {column_name: name, validation_class: UTF8Type}
  {column_name: web, validation_class: UTF8Type}
];
```

SET:

```
SET Customer['mfowler']['city']='Boston';
SET Customer['mfowler']['name']='Martin Fowler';
SET Customer['mfowler']['web']='www.martinfowler.com';
```

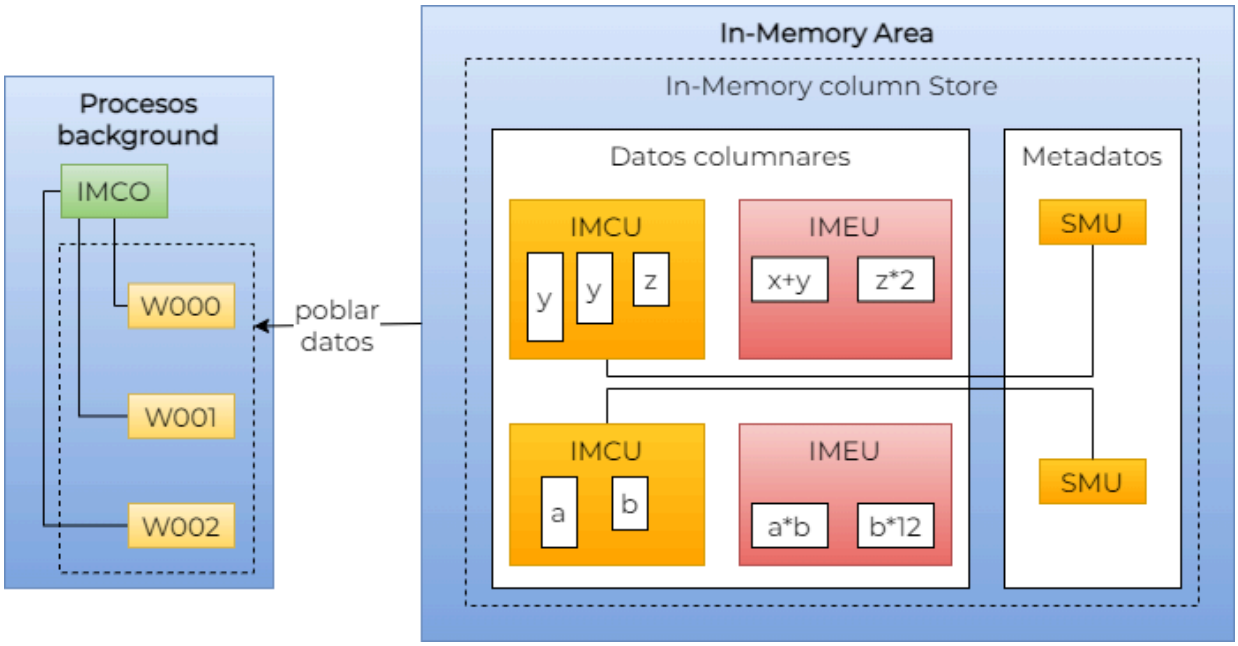
```
En Java

ColumnFamilyTemplate<String, String> template =
  cassandra.getColumnFamilyTemplate();
```

```
ColumnFamilyUpdater<String, String> updater =
  template.createUpdater(key);
for (String name : values.keySet()) {
  updater.setString(name, values.get(name));
}
try {
  template.update(updater);
} catch (HectorException e) {
  handleException(e);
}
```

Ejercicio práctico 04
Realizar las actividades del documento correspondiente, entregar con base a las fechas de entrega configuradas.

2.2. Bases de datos columnares en RDBMS - In Memory Area: Oracle



Es un componente opcional de la SGA que permite a los objetos almacenarse en la memoria en un formato de columnas. Este formato facilita la realización de operaciones, como lecturas y joins de tablas, llevándolas a cabo mucho más rápido que el formato tradicional en disco, lo que mejora considerablemente el rendimiento, en especial para operaciones de consulta.

- El conjunto de características que conforman la ***In-Memory Area*** han estado presentes en el software de base de datos de Oracle desde su versión 12.1.0.2.
- Surgió como una forma de mejorar el rendimiento para realizar tareas de analítica de datos en tiempo real.

2.2.1. Formato por filas (Row format)

Por default, una base de datos almacena filas o registros de forma continua en bloques de datos. Típicamente en un mismo bloque de datos se pueden almacenar los datos de uno o más registros.

- Cada fila contiene los valores de todas las columnas para dicha fila. Los datos con este formato están optimizados para el procesamiento de transacciones (OLTP): Non Uniform access patterns.
- Por ejemplo, actualizar todas las columnas de unas cuantas filas modifica solo una pequeña cantidad de bloques

2.2.2. Formato por columnas (Columnar format)

Una base de datos de este tipo almacena columnas, no filas, de forma contigua.

- Las cargas de trabajo analíticas suelen acceder a pocas columnas mientras escanean, pero leen todo el conjunto de datos de las mismas.
- Debido a que las columnas se almacenan por separado, una consulta analítica puede acceder solo a las columnas requeridas y evitar la lectura de bloques de datos innecesarios.
- Cuando los datos de una columna son almacenados en el IM Column Store se realiza una compresión para realizar escaneos optimizados así como la creación de In-Memory storage indexes.

En resumen, IM Column store puede representar una solución atractiva para:

- Escaneos de tablas grandes que hacen uso de filtros con operadores =, <, >, in.
- Consultas que obtienen un número bajo de columnas de una tabla con una gran cantidad de registros.
- Consultas que hacen join entre una tabla grande con una pequeña.
- Consultas que hacen uso de funciones de agregación.

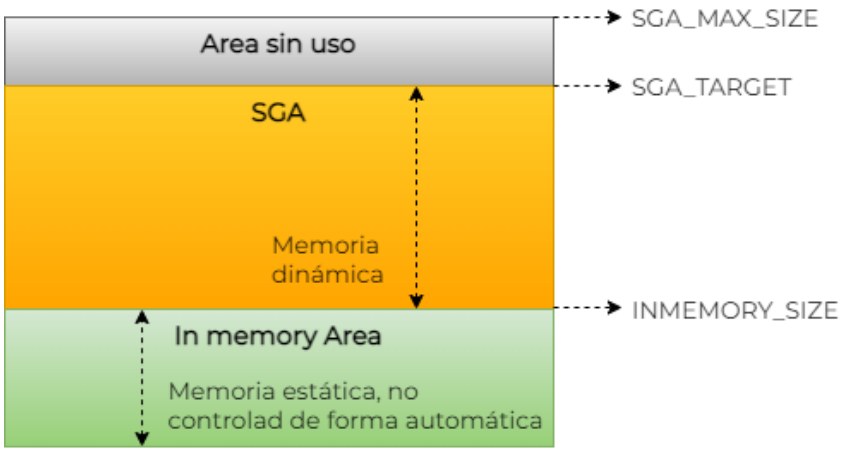
Escenarios donde IM Column store no es tan recomendada:

- Consultas con predicados complejos
- Consultas que regresan un número grande de columnas
- Consultas que regresan un número grande de registros
- Consultas que hacen join entre tablas grandes.

2.2.3. Administración de la In-Memory Area

La In-Memory Área no es administrada de forma automática por la base de datos, por lo que no se modificará su tamaño en automático bajo ninguna circunstancia.

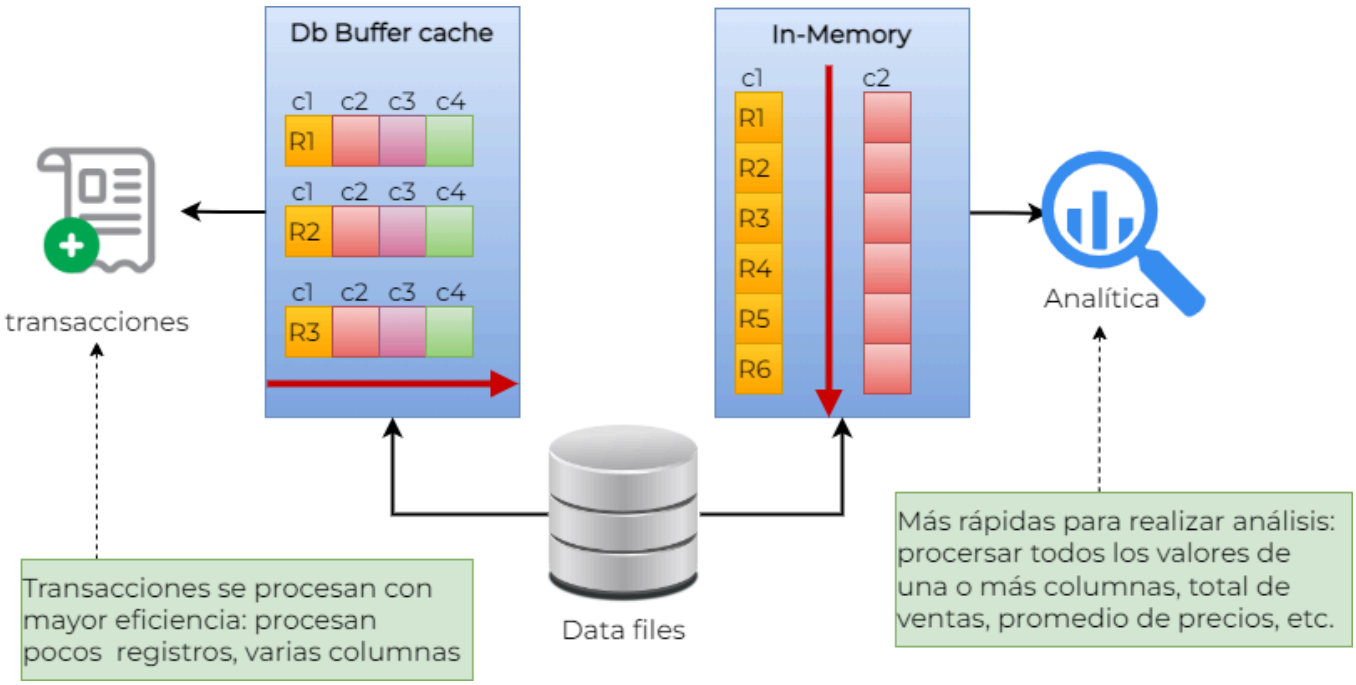
- El tamaño de la In-Memory Área está determinado por el parámetro `inmemory_size`.
- Este parámetro tiene un valor default igual a cero, lo que implica que el área se encuentra deshabilitada, y puede ser consultado en la vista `v$sga`.
- Para habilitar el área, se debe asignar un valor mínimo de 100 MB a dicho parámetro.
- Es posible modificar el parámetro `inmemory_size` para incrementar el tamaño del área mediante la instrucción alter system.
- El tamaño está limitado por el valor de `sga_target`. Su valor es estático por lo que la memoria disponible para la SGA no puede hacer uso de la memoria asignada para la In memory area.



2.2.4. In-Memory Column Store (IM Column Store)

- Uno de los elementos más importantes de la In-Memory Area es el In-Memory Column Store.
- Únicamente los datos más relevantes para realizar analítica de de la base de datos deberían guardarse en el IM Column Store. Si bien se trata de una área optimizada para el almacenamiento, no debe sobrepoblarse.
- La cláusula `inmemory` permite añadir un objeto, columna o tablespace al IM Column Store.
- `inmemory` se puede habilitar en el momento de creación de la tabla o vía alter table.

2.2.5. Relación Buffer Caché - IM Column Store



- El DB Buffer Caché y el IM Column Store pueden trabajar en conjunto, siendo uno el complemento del otro.
- Los datos en el formato de filas se almacenan en el Buffer Caché, y aquellos en el formato de columnas se ubican en el IM Column Store.
- La base de datos suele enviar consultas de tipo OLTP al Buffer Caché, mientras que las consultas analíticas pasan directamente al IM Column Store.
- No obstante, también es posible recuperar información de ambas áreas en la misma consulta.

2.2.6. Conceptos: IMCU, IMEU, SMU, ESS

- El In memory columnstore se encarga de administrar tanto datos del usuario (valores de las columnas) como metadatos en unidades de almacenamiento optimizadas en lugar de utilizar bloques de datos

2.2.6.1. In-Memory Compression Units (IMCUs)

- Es una unidad de almacenamiento comprimida de solo de lectura que contiene datos de una o más columnas.
- Un IMCU está formado por 2 elementos:
  - Conjunto de Column Compression Units (CUs).
  - Header
- Respecto a los objetos de un esquema, los datos de una tabla pueden ser almacenados en un conjunto de IMCUs.
- Cada tabla puede incluir todas sus columnas en sus IMCUs, o puede incluir solo a un subconjunto de sus columnas.

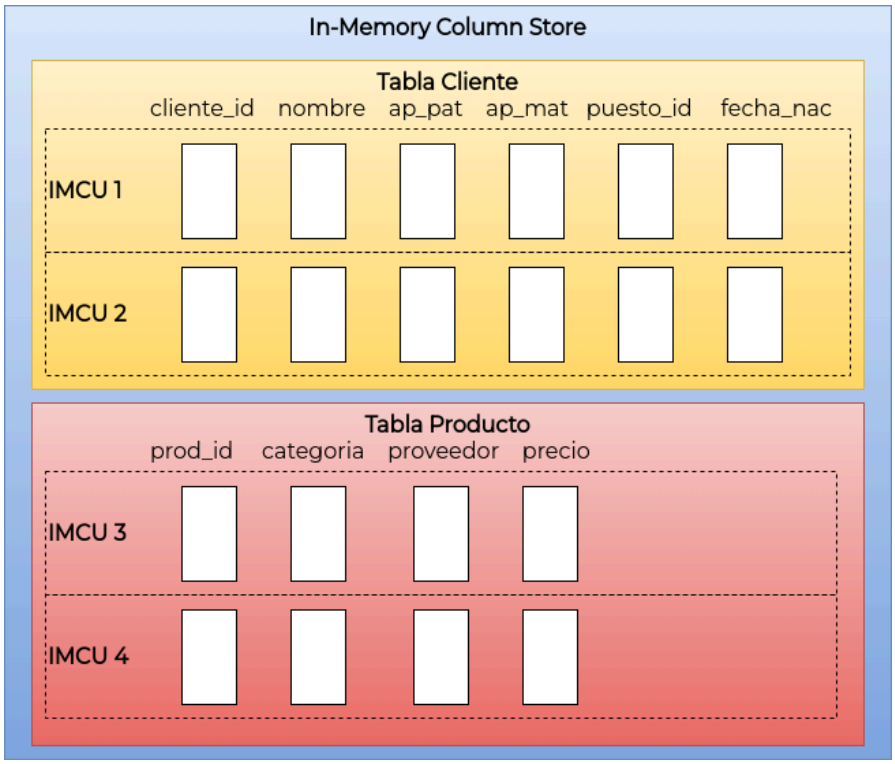
- En el siguiente ejemplo todas las columnas serán almacenadas en formato columnar

```
alter table mytable inmemory memcompress for query low;
```

En el siguiente ejemplo, las siguientes columnas se excluyen para ser almacenadas en formato columnar:

```
alter table mytable inmemory memcompress for query low
  no inmemory (genero, anio_nacimiento)
```

Importante: Si una consulta hace referencia a columnas no incluidas en el In Memory columnstore, los datos se tendrán que obtener del almacenamiento tradicional por renglón. Gráficamente, el almacenamiento columnar de una tabla se verá de la siguiente forma:



- Como se puede ver en la imagen, cada IMCU contiene todos los valores de una sola columna (incluyendo nulos) para un subconjunto de registros llamado ***granule***.
- Todos los IMCUs contienen aproximadamente el mismo número de registros. La instancia realiza esta distribución basado en el tipo de dato, formato de los datos, tipo de compresión.
- El tamaño por default de un IMCU de forma contigua es de 1MB. Por esta razón, al pool de memoria donde se almacenan los IMCUs se le llama 1M Pool.
- En términos de los 2 componentes que integran a un IMCU la imagen anterior se puede visualizar de la siguiente manera:

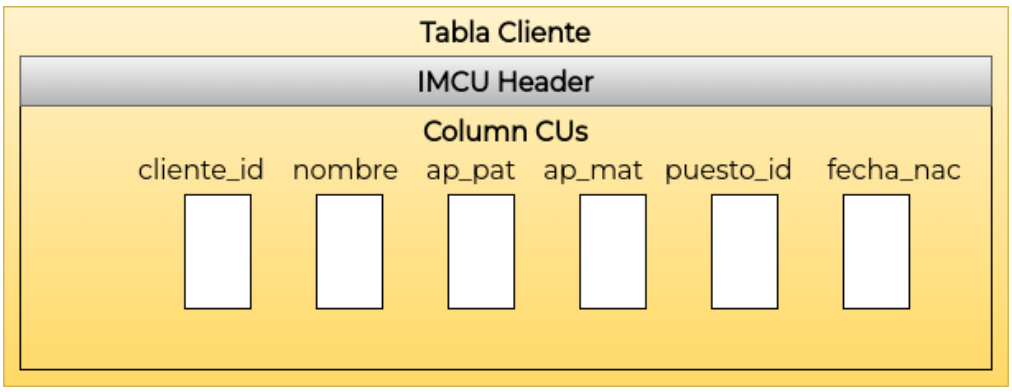


Tabla Cliente
IMCU Header
Column CUs
cliente_id
nombre
ap_pat
ap_mat
puesto_id
fecha_nac