Caso de uso de Base de Datos Orienta a Columnas: ClickHouse

Selección de la Base de Satos

Para la realización del caso de uso de una base de datos orientada a columnas, se escogió ClickHouse. A diferencia de otras alternativas comerciales como Google BigQuery y Amazon Redshift, ClickHouse tiene una versión de código abierto que se puede instalar fácilmente en local (ClickHouse, Inc., 2025f). Como se mostrará en la sección correspondiente a la instalación e implementación, cuando se trabaja con tamaños de datos moderados, ClickHouse no requiere de grandes cantidades de recursos computacionales. Esto permite su instalación en computadoras personales, alineándose con el objetivo didáctico de este documento.

Asimismo, ClickHouse es una base de datos popular que se utiliza en entornos de producción para el procesamiento analítico en línea (OLAP por las siglas de *online analytical processing*) de grandes volúmenes de datos (ClickHouse, Inc., 2025f). De manera que podremos ejemplificar las principales ventajas de las bases de datos orientadas a columnas mencionadas en el documento teórico, así como las principales diferencias con las bases de datos transaccionales.

Instalación y Ejecución de ClickHouse en un Entorno Local

Para este caso de uso, se utilizó la versión *25.8.4.13* de ClickHouse. La instalación se realizó mediante una imagen de Docker (ClickHouse, Inc., 2025a). El proceso de instalación, configuración y ejecución se puede encontrar en el documento del manual de usuario adjunto o en el archivo *README* del código correspondiente. Como se explica en estos documentos, al momento de iniciar el entorno de Docker Compose, también se instancian contenedores para la interfaz gráfica Tabix (Spoonest, 2025) y para Node.js (OpenJS Foundation, 2025). Este último se utilizó para la construcción de una API REST con conexión a la instancia de ClickHouse para consultas simples de tipo CRUD.

Interfaz gráfica de Usuario (GUI)

ClickHouse incluye un servidor Web ligero con varias funciones, entre las que se incluye un editor SQL y un dashboard con información de rendimiento, consultas y conexión de la base de datos. Se puede acceder a este servidor conectándose a la dirección IP y al puerto en que se inició la instancia de la base de datos, el proceso detallado se puede encontrar en el manual de usuario. Como veremos más adelante, el editor SQL es suficiente para realizar todos los tipos de consultas que se realizan en ClickHouse. Si se quiere una alternativa ligeramente más completa, se puede utilizar la interfaz Web Tabix (Spoonest, 2025), ver el proceso de configuración y conexión con ClickHouse en el manual de usuario. Esta también incluye un editor SQL y métricas de la base de datos, así como opciones para visualizar la estructura y datos de las tablas pertenecientes a esquemas propios del sistema o creados por el administrador de la base de datos. Existen otras alternativas como DBeaver (ClickHouse, Inc., 2025c) y DbVisualizer (ClickHouse, Inc., 2025d). Para las consultas presentadas en este documento, consideramos que es suficiente con utilizar el editor SQL incluido en ClickHouse.

Lenguaje de Consulta SQL para ClickHouse

ClickHouse utiliza un lenguaje declarativo basado en el estándar ANSI SQL (ClickHouse, Inc., 2025b). Como veremos a continuación, las sentencias utilizadas para crear tablas, insertar datos y seleccionar datos (incluyendo la presencia de las clausulas GROUP BY, ORDER BY, FROM, JOIN, IN y la posibilidad de realizar subconsultas) son muy similares a SQL (ClickHouse, Inc., 2025b). Sin embargo, la actualización y eliminación de datos presentan diferencias importantes. Para este caso de uso, vamos a construir una base de datos simple que almacena datos de comidas de un restaurante. Para mantener el ejemplo simple, únicamente vamos a construir la tabla que almacena los tipos de comida (a modo de catalogo) y la tabla que almacena las comidas y/o platillos. Con este ejemplo se espera resaltar alguna de las principales diferencias de ClickHouse con las bases de datos transaccionales clásicas y demostrar por qué sus características implementadas para OLAP

no la hacen adecuada para el manejo de transacciones.

Sentencias de Definición de Datos

Para organizar de manera adecuada los elementos de la base de datos, primero vamos a construir un esquema con el siguiente comando (ClickHouse, Inc., 2025b):

```
CREATE DATABASE comida_db;
```

En caso de que se quiera eliminar el esquema, se puede utilizar:

```
DROP DATABASE comida_db;
```

Para crear las tablas, podemos utilizar los siguientes comandos:

```
CREATE TABLE comida_db.tipo_comida (
    id UInt32, nombre String
) ENGINE = MergeTree() ORDER BY id;

CREATE TABLE comida_db.comida (
    id UInt32, nombre String, tipo_id UInt32,
    precio_unitario Float32, descripcion String DEFAULT ‘’
) ENGINE = MergeTree() ORDER BY id;
```

Si se quiero borrar una de las tablas, utilizar:

```
DROP TABLE comida_db.detalle_orden;
```

Notar que en la creación de las tablas se usa la misma sintaxis de SQL, de manera que se define el nombre de las filas junto con el tipo de los datos. De manera predeterminada, todas las filas son obligatorias, si no se quiere que siempre se deba ingresar un valor en el INSERT, se puede definir un valor por defecto con la cláusula *DEFAULT* (como en la creación de la tabla *comida*), o utilizando el modificador *Nullable(tipo de dato)* (ClickHouse, Inc., 2025b). Al momento de crear la tabla, también es necesario indicar el

motor de tablas (*table engine*) con la palabra *ENGINE*. El motor utilizado va a determinar la forma en que se almacenan los datos, las consultas soportadas y el tipo de optimización. En este caso se utilizó *MergeTree*, el cual es el motor más utilizado en ClickHouse para instancias de un único nodo por su robustez, alta tasa de ingesta de datos y alto rendimiento en la recuperación de datos. Soporta particiones arbitrarias e índices primarios dispersos. Para entornos de producción y distribuidos, se recomienda *ReplicatedMergeTree* que permite la replicación entre nodos y favorece la alta disponibilidad (ClickHouse, Inc., 2025e). Asimismo, se utiliza *ORDER BY* para indicar el *sorting key* o clave de ordenamiento que determina cómo se organizan físicamente los datos en disco y sobre qué columnas se construye el índice primario disperso. De cierta forma se puede considerar como un tipo de clave primaria, pero no garantiza la unicidad de los datos (dos o más filas pueden tener la misma clave de ordenamiento). El concepto de clave foránea no existe en ClickHouse (ClickHouse, Inc., 2025b).

Sentencias tipo CRUD o de Manipulación de Datos

Inserción de datos:

De manera general, ClickHouse está pensado para la ingesta de grandes volúmenes de datos. En las siguientes consultas se realizan inserciones en lote utilizando la función generadora *numbers(...)*, que permite crear rápidamente múltiples registros sin necesidad de escribir cada fila manualmente (ClickHouse, Inc., 2025b).

```
INSERT INTO comida_db.tipo_comida
SELECT
    number + 1 AS id,
    ['Entrada', 'Plato Principal', 'Postre', 'Bebida', 'Snack', 'Ensalada',
    'Sopa', 'Pizza', 'Hamburguesa', 'Pasta', 'Taco', 'Sushi', 'Sandwich',
    'Helado', 'Galleta', 'Batido', 'Smoothie', 'Café', 'Té', 'Jugo']
    [number + 1] AS nombre
FROM numbers(20);
```

```
INSERT INTO comida_db.comida
SELECT
    number + 1 AS id,
    concat('Comida_', number + 1) AS nombre,
    number + 1 AS tipo_id,
    toFloat32(rand() % 10000 + 1) AS precio_unitario,
    concat('Descripción de la comida ', number + 1) AS descripcion
FROM numbers(20);
```

Como se puede revisar en la documentación, el *INSERT* también permite insertar en lote datos mediante archivos en formatos como *JSON*, *CSV*, entre otros (ClickHouse, Inc., 2025b). También es posible realizar inserciones de una única fila como se realiza en SQL:

```
INSERT INTO comida_db.tipo_comida (id, nombre)
    VALUES(21, 'Guarnición');
FROM comida_db.tipo_comida;
```

Sin embargo, es importante considerar que ClickHouse no está pensada para manejar transacciones, por lo que siempre es preferible la inserción en lote.

Borrado de datos:

Tanto el *DELETE* como *UPDATE* presentan diferencias importantes con SQL, ya que en ClickHouse no se eliminan ni actualizan las filas de forma inmediata. En su lugar, se generan *mutaciones*, que son operaciones asíncronas aplicadas en segundo plano por el motor (ClickHouse, Inc., 2025b). En el siguiente ejemplo se realiza un borrado condicional de una fila específica:

```
ALTER TABLE comida_db.tipo_comida
    DELETE WHERE id = 21 AND nombre = 'Guarnición';
```

Desde versiones más recientes, también se permite el uso de la sentencia *DELETE FROM*, el cual se considera un borrado ligero (implica una menor carga de trabajo y aumenta el

rendimiento), ya que las filas se marcan como borradas de manera lógica y se omiten en futuras consultas, sin realizar el borrado físico inmediatamente. Este borrado está disponible en los motores de la familia *MergeTree* (ClickHouse, Inc., 2025b).

```
DELETE FROM comida_db.tipo_comida
WHERE id = 21;
```

Actualización de datos:

La sentencia *UPDATE* también se comporta de forma distinta a SQL tradicional. En ClickHouse, las actualizaciones no se aplican de manera inmediata, sino que se gestionan como *mutaciones*, las cuales se ejecutan en segundo plano por el motor (ClickHouse, Inc., 2025b). En el siguiente ejemplo se actualiza el valor de una fila específica:

```
ALTER TABLE comida_db.tipo_comida
    UPDATE nombre = 'Emparedado'
WHERE id = 13;
```

Es importante notar que, si se trata de borrar o actualizar un dato para el cual no hay coincidencia, la base de datos no devuelve error; por ejemplo, si se usa un *WHERE* con un *id* que no existe.

Selección de datos:

La sintaxis de la selección de datos en ClickHouse sigue el estándar de SQL, permitiendo recuperar información mediante la sentencia *SELECT*. Esta puede incluir cláusulas como *WHERE*, *JOIN*, *GROUP BY* y *ORDER BY*, entre otras (ClickHouse, Inc., 2025b). En el siguiente ejemplo se realiza una consulta simple que filtra las comidas cuyo precio unitario supera los 5000:

```
SELECT * from comida_db.comida
where precio_unitario > 5000;
```

En la siguiente consulta se unen ambas tablas mediante *JOIN*:

```
SELECT
    c.id, c.nombre AS comida,
    t.nombre AS tipo, c.descripcion
FROM comida_db.comida AS c
INNER JOIN comida_db.tipo_comida AS t
ON c.tipo_id = t.id;
```

Como último ejemplo, se realiza una operación de agregación para el cálculo del mínimo, máximo, suma y promedio del precio unitario de las comidas según el tipo de comida, ordenado por el nombre de la comina de manera ascendente.

```
SELECT
    t.nombre AS tipo_comida,
    min(c.precio_unitario) AS precio_minimo,
    max(c.precio_unitario) AS precio_maximo,
    sum(c.precio_unitario) AS suma_precios,
    avg(c.precio_unitario) AS precio_promedio
FROM comida_db.comida AS c
INNER JOIN comida_db.tipo_comida AS t ON c.tipo_id = t.id
GROUP BY t.nombre
ORDER BY t.nombre ASC;
```

Implementación de la API REST

La API REST se implementó usando Node.js 18 (ver en el manual de usuario el proceso de instalación y ejecución) (OpenJS Foundation, 2025), en especifico usando las bibliotecas *Express* para la creación de la API y *@clickhouse/client* para la conexión directa con la base de datos ClickHouse. La API REST implementada permite realizar las operaciones *GET*, *PUT*, *PATCH* y *DELETE* para modificar los datos correspondientes a la tabla

comidas. Se puede utilizar Postman (u otro cliente HTTP) para realizar los siguientes ejemplos representativos:

Consulta de datos (GET). Permite recuperar los registros completos o agregados:

```
GET http://localhost:3001/comidas
GET http://localhost:3001/comidas/resumen
```

Inserción de datos (POST). Permite registrar una nueva comida en la base de datos:

```
POST http://localhost:3001/comidas
Body:
{
  "nombre": "Hamburguesa Clásica",
  "tipo_id": 9,
  "precio_unitario": 3000,
  "descripcion": "Hamburguesa de carne de res, lechuga, tomate,
  cebolla y queso."
}
```

Actualización de datos (PATCH):. Permite modificar una o varias columnas de un registro existente, identificado por su *id*:

```
PATCH http://localhost:3001/comidas/21
Body:
{
  "nombre": "Ensalada César"
}
```

Eliminación de datos (DELETE):. Permite eliminar un registro específico. Si el *id* no existe, la operación no genera error, simplemente no afecta ningún registro:

```
DELETE http://localhost:3001/comidas/10000
DELETE http://localhost:3001/comidas/22
```

Para más detalles del funcionamiento de la API, se recomienda revisar el código fuente de la misma.

Lecciones aprendidas

- La instalación de ClickHouse mediante Docker simplifica el despliegue y evita problemas de dependencias.
- La API REST permite exponer la base de datos para consumo de aplicaciones externas.
- El SQL de ClickHouse está enfocado en la velocidad de las consulta para grandes volúmenes de datos. Las principales optimizaciones están enfocadas en *SELECT* y la inserción por lotes. Se sacrificando la existencia de transacciones en el sentido tradicional y la eficiencia de operaciones unitarias de escritura es baja.
- Las operaciones como *UPDATE* y *DELETE* se gestionan mediante mutaciones asíncronas.
- No existe soporte nativo para transacciones multi-etapa o reversibles.
- ClickHouse no soporta claves foráneas, existen claves de ordenamiento que presentan similitudes con las claves primarias, pero no son iguales.
- La elección del motor de tablas en ClickHouse afecta el rendimiento de las consultas, las capacidades disponibles para la manipulación de datos y ciertas características relacionadas con el funcionamiento distribuido.

Referencias

ClickHouse, Inc. (2025a). *ClickHouse Docker Hub* [Accedido el 27 de septiembre de 2025]. https://hub.docker.com/_/clickhouse

ClickHouse, Inc. (2025b). *ClickHouse SQL Reference* [Consultado el 29 de septiembre de 2025]. https://clickhouse.com/docs/sql-reference

ClickHouse, Inc. (2025c). *Connect DBeaver to ClickHouse* [Consultado el 28 de septiembre de 2025]. https://clickhouse.com/docs/integrations/dbeaver

ClickHouse, Inc. (2025d). *Connecting DbVisualizer to ClickHouse* [Consultado el 28 de septiembre de 2025]. https://clickhouse.com/docs/integrations/dbvisualizer

ClickHouse, Inc. (2025e). *MergeTree Table Engine Documentation* [Consultado el 29 de septiembre de 2025]. https://clickhouse.com/docs/engines/table-engines/mergetree-family/mergetree

ClickHouse, Inc. (2025f). *What is ClickHouse?* [Consultado el 28 de septiembre de 2025]. https://clickhouse.com/docs/intro

OpenJS Foundation. (2025). *Node.js Docker Official Image* [Accedido el 27 de septiembre de 2025]. https://hub.docker.com/_/node

Spoonest. (2025). *Tabix Web Client Docker Hub* [Accedido el 27 de septiembre de 2025]. https://hub.docker.com/r/spoonest/clickhouse-tabix-web-client