# Universidad ORT Uruguay

Facultad de Ingeniería Bernard Wand Polak
Entregado como requisito para la obtención del crédito
de la materia Base de Datos 2


**Estudiantes:**
**Diego Erviti - 240305**
**Matias Olivera - 267919**
**Gonzalo Gularte - 270959**

**Docente: Cecilia Belleti y Gonzalo Camano**

**Junio 2023**

# PARTE 1

## 1.1) Implementar las restricciones no estructurales de los componentes de Registro y perfil de usuarios y Gestión de servidores y canales.

***--Registrar la fecha de amistad al aceptar una solicitud de amistad***

```
create or replace TRIGGER Trg_RegistrarAmistad
BEFORE INSERT ON Amistad
FOR EACH ROW
BEGIN
   IF :new.fecha_amistad IS NULL THEN
      :new.fecha_amistad := SYSDATE; -- Usa la fecha actual si no se proporciona una fecha
   END IF;
END;
/
```

***--Asegurarse que los usuarios tengan al menos 13 años de edad***

```
create or replace TRIGGER Trg_VerificarEdadUsuario
BEFORE INSERT OR UPDATE ON Usuario
FOR EACH ROW
DECLARE
   years INT;
BEGIN
   SELECT EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM
         :NEW.fecha_nacimiento) INTO years FROM dual;
   IF years < 13 THEN
      RAISE_APPLICATION_ERROR(-20001, 'El usuario debe tener al menos 13 años de
         edad');
   END IF;
END;
/
```

***-- Agregar como miembro al creador del servidor, asignarle permisos de Owner, crear rol "Everyone" y crear canales por defecto***

```
CREATE OR REPLACE TRIGGER Trg_CrearServidor
AFTER INSERT ON Servidor
FOR EACH ROW
DECLARE
   v_owner_rol_id NUMBER;
BEGIN
   -- Insertar una nueva fila en la tabla Rol para el rol "Everyone"
   INSERT INTO Rol (nombre_rol, servidor_id)
   VALUES ('Everyone', :new.servidor_id);

   -- Agregar al dueño como miembro del servidor
   INSERT INTO MiembroServidor (usuario_id, servidor_id, fecha_ingreso)
   VALUES (:new.usuario_owner_id, :new.servidor_id, :new.fecha_creacion);
```

```
    -- Crear canal de voz "General"
    INSERT INTO Canal (nombre_canal, tipo, descripcion, privado, servidor_id)
    VALUES ('General', 'voz', '', 0, :new.servidor_id);

    -- Crear canal de texto "General"
    INSERT INTO Canal (nombre_canal, tipo, descripcion, privado, servidor_id)
    VALUES ('General', 'texto', '', 0, :new.servidor_id);

    -- Crear rol "Owner"
    INSERT INTO Rol (nombre_rol, servidor_id)
    VALUES ('Owner', :new.servidor_id)
    RETURNING rol_id INTO v_owner_rol_id;

    -- Asignar todos los permisos al rol "Owner"
    UPDATE PermisosRol
    SET ver_canales = 1, gestionar_canales = 1, gestionar_roles = 1
    WHERE rol_id = v_owner_rol_id;

    -- Asignar el rol "Owner" al propietario del servidor
    INSERT INTO MiembroRol (rol_id, miembro_servidor_id, servidor_id)
    VALUES (v_owner_rol_id, :new.usuario_owner_id, :new.servidor_id);
END;
/
```

***--Cada vez que se cree un rol asignarle permisos por defecto***

```
create or replace TRIGGER Trg_CreacionRolPermisos
AFTER INSERT ON Rol
FOR EACH ROW
BEGIN
    -- Insertar una nueva fila en la tabla PermisosRol con los permisos por defecto
    INSERT INTO PermisosRol (rol_id, ver_canales, gestionar_canales, gestionar_roles)
    VALUES (:NEW.rol_id, 1, 0, 0);
END;
/
```

***--Asignar automáticamente el rol "Everyone" a los usuarios que se unen a un servidor e incrementar la cantidad de miembros del mismo***

```
create or replace TRIGGER Trg_AsignarRolEveryone
AFTER INSERT ON MiembroServidor
FOR EACH ROW
BEGIN
    -- Insertar una nueva fila en la tabla MiembroRol asignando el rol "Everyone"
    INSERT INTO MiembroRol (rol_id, miembro_servidor_id, servidor_id)
    SELECT r.rol_id, :NEW.usuario_id, :NEW.servidor_id
    FROM Rol r
    WHERE r.nombre_rol = 'Everyone'
    AND r.servidor_id = :NEW.servidor_id;
```

```
END;
/

--Para crear una invitación se necesita ser miembro del servidor
CREATE OR REPLACE TRIGGER Trg_VerificarMiembroInvitacion
BEFORE INSERT ON Invitacion
FOR EACH ROW
DECLARE
  v_count NUMBER;
BEGIN
  -- Verificar si el usuario es miembro del servidor
  SELECT COUNT(*) INTO v_count
  FROM MiembroServidor
  WHERE usuario_id = :NEW.remitente_id
  AND servidor_id = :NEW.servidor_id;

  -- Si no es miembro, lanzar un error
  IF v_count = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Debe ser miembro del servidor para crear una
invitación.');
  END IF;
END;
/

--Cada vez que entre se intente agregar un miembro a un servidor se verifica que haya
lugar disponible
CREATE OR REPLACE TRIGGER Trg_CapacidadMaximaServidor
BEFORE INSERT OR UPDATE ON MiembroServidor
FOR EACH ROW
DECLARE
    total_miembros NUMBER;
BEGIN
    SELECT COUNT(*) INTO total_miembros
    FROM MiembroServidor
    WHERE servidor_id = :NEW.servidor_id;

    IF total_miembros >= 250000 THEN
        RAISE_APPLICATION_ERROR(-20001, 'No se pueden agregar más miembros a este
servidor. La cantidad máxima de miembros ha sido alcanzada (250.000).');
    END IF;
END;
/

--Servidores privados son solo accesibles con invitación
-- Este trigger se trató de realizar pero surgieron problemas de mutación en la tabla
“Servidor”, debido a que cada vez que se crea un servidor se inserta como miembro
al owner, entonces se dispara este trigger y ahí es donde trata de consultar datos de
la tabla “Servidor” que está siendo modificada, por lo cual se procedió a quitar.
```

```
--CREATE OR REPLACE TRIGGER trg_acceso_servidor_privado
--BEFORE INSERT ON MiembroServidor
--FOR EACH ROW
--DECLARE
  --v_publico NUMBER(1);
  --v_invitacion_existente NUMBER(1);
  --v_owner_id NUMBER;
--BEGIN
  --SELECT publico, usuario_owner_id INTO v_publico, v_owner_id
  --FROM Servidor
  --WHERE servidor_id = :NEW.servidor_id;

  --IF v_publico = 0 AND :NEW.usuario_id <> v_owner_id THEN -- El servidor es privado y el
usuario no es el propietario
    -- Verificar si el usuario tiene una invitación válida
    --SELECT COUNT(*) INTO v_invitacion_existente
    --FROM Invitacion
    --WHERE destinatario_id = :NEW.usuario_id
      --AND servidor_id = :NEW.servidor_id;

   -- IF v_invitacion_existente = 0 THEN
      --RAISE_APPLICATION_ERROR(-20001, 'No tienes acceso a este servidor privado.');
    --END IF;
  --END IF;
--EXCEPTION
  --WHEN NO_DATA_FOUND THEN
   -- RAISE_APPLICATION_ERROR(-20002, 'El servidor no existe.');
--END;
--/
```

***NOTAS: Se realizaron pequeños cambios en el modelo al crear un nuevo servidor y al crear una invitación, en este nuevo modelo se eliminó la columna que almacenaba la cantidad de miembros totales en un servidor, para esta versión ese atributo ya no es necesario y se verifica a través de un trigger que el servidor no exceda el límite de usuarios establecido (250.000), también en la tabla Invitación se agregó el atributo destinatario_id, con el fin de comprobar si un usuario tiene una invitación.***

# PARTE 2

## 2.1) Diseñar el modelo de datos que permita almacenar y consultar los datos relativos al componente de Mensajería con MongoDB.

El diseño del modelo de datos se mantendría de forma muy similar a lo trabajado en la entrega anterior con SQL en lo que a las tablas anteriores (ahora colecciones) se refiere.

La principal novedad se encuentra relacionada con la colección que se encarga de almacenar los datos de mensajería en sí. Se manejaron distintas opciones para la implementación del sistema; al tener dos mecanismos de mensajes lo suficientemente diferenciados, es decir, los mensajes directos entre usuarios y los mensajes generados en los canales, en primer lugar se pensó en crear dos colecciones distintas que se diferenciaran por el tipo de mensajes. Luego decidimos descartar esta opción, y aprovechando la flexibilidad que nos brinda MongoDB por ser una base de datos NoSQL, decidimos crear una colección sola llamada “mensajes”.

Diseñamos la colección "mensajes" para manejar millones de mensajes generados en canales y a través de mensajes directos. Realizamos un análisis de las diferentes alternativas y se tomaron las siguientes decisiones de diseño:

- Se definió una estructura básica para los mensajes con los campos obligatorios "usuarioEmisorId" y "fecha".

- Se incluyeron atributos opcionales como "enlaces" (Array de Strings) y "hashtags" (Array de Strings) para representar enlaces y hashtags relacionados en el mensaje.
- Se agregó el atributo "fijado" (Boolean) para indicar si un mensaje está fijado en un canal.

- Se consideraron los archivos adjuntos y se creó el atributo "archivos" (Array de objetos) para almacenar información sobre cada archivo adjunto, como nombre, tipo, tamaño, fecha de creación y el ID del usuario propietario.

- Para permitir mensajes directos y mensajes en canales, se agregaron los atributos "usuarioReceptorId" en caso de mensajes directos, y "canalId" y "servidorId" en caso de mensajes en canales.

- Se establecieron validaciones en el esquema para garantizar la presencia de los campos obligatorios y su tipo correcto.

- Se buscó la flexibilidad y escalabilidad de la colección para adaptarse a cambios futuros sin afectar la funcionalidad existente.

NOTA: Los atributos como ID’s deberían de ser de tipo objectId, pero al no tener las otras colecciones se tomó la decisión de realizarlas de tipo “string” así facilita el ingreso de los datos.

Las colección creada para almacenar y consultar los datos relativos al componente de Mensajería serían la siguiente:

```
db.createCollection("mensajes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["texto", "fecha", "usuarioEmisorId"],
      properties: {
        texto: {
          bsonType: "string",
          description: "El texto del mensaje"
        },
        fecha: {
          bsonType: "date",
          description: "La fecha de creación del mensaje"
        },
        enlaces: {
          bsonType: "array",
          items: {
            bsonType: "string",
            description: "Enlaces relacionados en el mensaje"
          }
        },
        hashtags: {
          bsonType: "array",
          items: {
            bsonType: "string",
            description: "Hashtags relacionados en el mensaje"
          }
        },
        fijado: {
          bsonType: "bool",
          description: "Indica si el mensaje está fijado en un canal"
        },
        archivos: {
          bsonType: "array",
          items: {
            bsonType: "object",
            required: ["nombreArchivo", "tipoArchivo", "tamano", "fechaCreacion",
"usuarioPropietarioId"],
            properties: {
              nombreArchivo: {
                bsonType: "string",
                description: "Nombre del archivo adjunto"
              },
              tipoArchivo: {
                bsonType: "string",
```

```
            description: "Tipo del archivo adjunto"
          },
          tamano: {
            bsonType: "number",
            description: "Tamaño del archivo adjunto"
          },
          fechaCreacion: {
            bsonType: "date",
            description: "Fecha de creación del archivo adjunto"
          },
          usuarioPropietarioId: {
            bsonType: "objectId",
            description: "ID del usuario propietario del archivo"
          }
        }
      }
    },
    usuarioEmisorId: {
      bsonType: "string",
      description: "ID del usuario emisor del mensaje"
    },
    usuarioReceptorId: {
      bsonType: "string",
      description: "ID del usuario receptor del mensaje (en caso de mensaje directo)"
    },
    canalId: {
      bsonType: "string",
      description: "ID del canal al que pertenece el mensaje (en caso de ser mensaje a
canal)"
    },
    servidorId: {
      bsonType: "string",
      description: "ID del servidor al que pertenece el mensaje (en caso de ser mensaje a
canal)"
    }
  }
 }
});
```

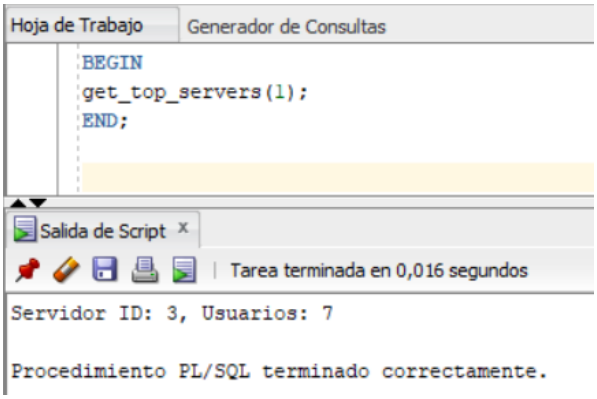
# PARTE 3

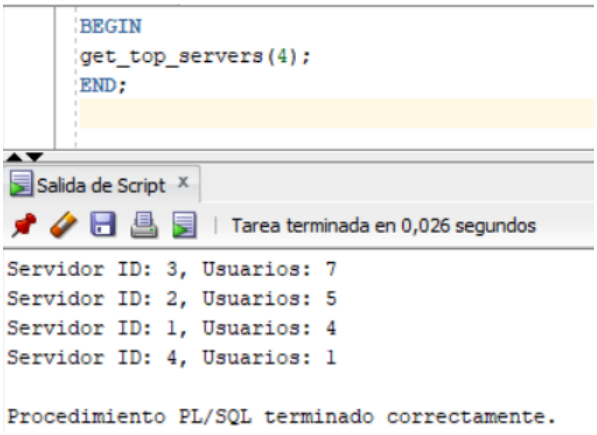
## 3.1) Implementar requerimientos para los componentes de Registro y perfil de usuario y Gestión de servidores y canales:

Es necesario ejecutar antes de las consultas:
SET SERVEROUTPUT ON;

***--Requerimiento 1: Proveer un servicio que retorne los servidores con más usuarios registrados. El servicio deberá recibir un parámetro para listar solo los N primeros servidores***

```
CREATE OR REPLACE PROCEDURE get_top_servers (p_top IN NUMBER) IS
BEGIN
  FOR r IN (
    SELECT servidor_id, COUNT(usuario_id) as total_usuarios
    FROM MiembroServidor
    GROUP BY servidor_id
    ORDER BY COUNT(usuario_id) DESC
    FETCH FIRST p_top ROWS ONLY
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Servidor ID: ' || r.servidor_id || ', Usuarios: ' ||
r.total_usuarios);
  END LOOP;
END;
```



***--Requerimiento 2: Proveer un servicio que revise las solicitudes de amistad pendientes. En caso de que la solicitud no se haya confirmado en los últimos 10 días el servicio cancelará automáticamente la solicitud. La cantidad de días puede variar a futuro, ese servicio debe permitir el cambio con costo mínimo***

*--NOTA: PARA QUE ESTE REQUERIMIENTO FUNCIONE DE MANERA CORRECTA SE DEBE UTILIZAR EL DDL CON LAS MODIFICACIONES REALIZADAS PARA ESTE OBLIGATORIO 2 (TABLA "SolicitudAmistad" agregada)*

```
create or replace PROCEDURE update_friend_request(p_days IN NUMBER) AS
   CURSOR cur_solicitud IS
   SELECT usuario_id_1, usuario_id_2
   FROM SolicitudAmistad
   WHERE fecha_solicitud < SYSDATE - p_days AND estado = 0;
BEGIN
   FOR cur IN cur_solicitud LOOP
      BEGIN
         DELETE FROM SolicitudAmistad
         WHERE usuario_id_1 = cur.usuario_id_1
         AND usuario_id_2 = cur.usuario_id_2;
         COMMIT;
         EXCEPTION
         WHEN OTHERS THEN
         ROLLBACK;
         RAISE;
      END;
   END LOOP;
END;
```

Tabla SolicitudAmistad antes de invocar al procedimiento

| | USUARIO_ID_1 | USUARIO_ID_2 | FECHA_SOLICITUD | ESTADO |
|---|---|---|---|---|
| 1 | 1 | 2 | 02/02/2019 00:00:00 | 0 |
| 2 | 1 | 3 | 01/03/2023 00:00:00 | 0 |
| 3 | 1 | 4 | 11/04/2018 00:00:00 | 1 |
| 4 | 1 | 7 | 07/05/2022 00:00:00 | 1 |
| 5 | 1 | 8 | 17/01/2019 00:00:00 | 1 |
| 6 | 1 | 9 | 18/02/2020 00:00:00 | 0 |
| 7 | 1 | 10 | 21/09/2018 00:00:00 | 0 |
| 8 | 2 | 5 | 22/12/2017 00:00:00 | 0 |
| 9 | 2 | 6 | 30/11/2021 00:00:00 | 1 |
| 10 | 2 | 9 | 21/07/2019 00:00:00 | 0 |
| 11 | 3 | 7 | 25/09/2018 00:00:00 | 1 |
| 12 | 3 | 5 | 01/01/2019 00:00:00 | 0 |
| 13 | 3 | 10 | 01/02/2022 00:00:00 | 1 |
| 14 | 4 | 5 | 11/02/2023 00:00:00 | 0 |
| 15 | 5 | 6 | 21/01/2020 00:00:00 | 1 |
| 16 | 5 | 9 | 27/03/2021 00:00:00 | 0 |
| 17 | 6 | 7 | 28/09/2022 00:00:00 | 1 |
| 18 | 6 | 8 | 25/08/2021 00:00:00 | 0 |
| 19 | 7 | 10 | 16/07/2020 00:00:00 | 1 |
| 20 | 8 | 9 | 18/06/2021 00:00:00 | 0 |
| 21 | 8 | 10 | 01/05/2019 00:00:00 | 1 |
| 22 | 9 | 10 | 01/04/2020 00:00:00 | 0 |
| 23 | 11 | 10 | 01/04/2023 00:00:00 | 0 |
| 24 | 11 | 9 | 20/06/2023 00:00:00 | 0 |
| 25 | 11 | 7 | 11/06/2023 00:00:00 | 0 |
| 26 | 11 | 6 | 01/05/2023 00:00:00 | 0 |
| 27 | 11 | 1 | 01/04/2020 00:00:00 | 0 |
| 28 | 11 | 3 | 01/04/2022 00:00:00 | 0 |

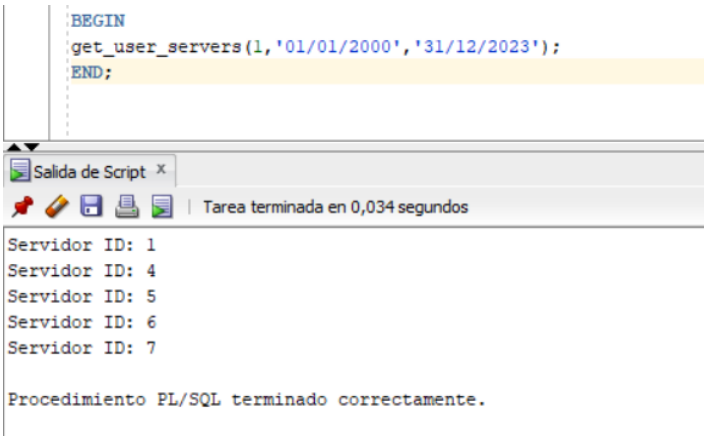
Tabla SolicitudAmistad luego de invocar al procedimiento:

```
BEGIN
        update_friend_request(10);
END;
```

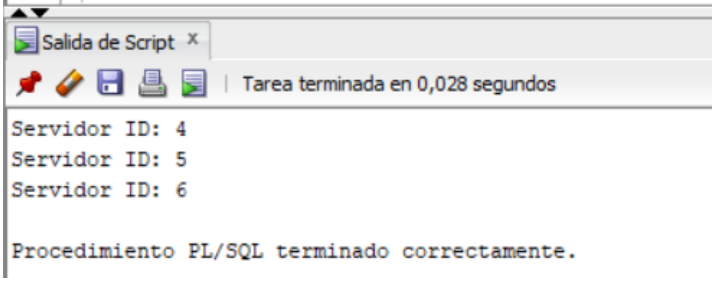
| | USUARIO_ID_1 | USUARIO_ID_2 | FECHA_SOLICITUD | ESTADO |
|---|---|---|---|---|
| 1 | 1 | 4 | 11/04/2018 00:00:00 | 1 |
| 2 | 1 | 7 | 07/05/2022 00:00:00 | 1 |
| 3 | 1 | 8 | 17/01/2019 00:00:00 | 1 |
| 4 | 2 | 6 | 30/11/2021 00:00:00 | 1 |
| 5 | 3 | 7 | 25/09/2018 00:00:00 | 1 |
| 6 | 3 | 10 | 01/02/2022 00:00:00 | 1 |
| 7 | 5 | 6 | 21/01/2020 00:00:00 | 1 |
| 8 | 6 | 7 | 28/09/2022 00:00:00 | 1 |
| 9 | 7 | 10 | 16/07/2020 00:00:00 | 1 |
| 10 | 8 | 10 | 01/05/2019 00:00:00 | 1 |
| 11 | 11 | 9 | 20/06/2023 00:00:00 | 0 |
| 12 | 11 | 7 | 11/06/2023 00:00:00 | 0 |

***--Requerimiento 3: Proveer un servicio que, dado un rango de fechas y un usuario, retorna la lista de servidores públicos y privados a los que se unió***

```
CREATE OR REPLACE PROCEDURE get_user_servers (p_user_id IN NUMBER,
p_start_date IN DATE, p_end_date IN DATE) AS
   CURSOR cur_ingreso IS
   SELECT servidor_id
   FROM MiembroServidor
   WHERE usuario_id = p_user_id AND fecha_ingreso BETWEEN p_start_date AND
p_end_date;
BEGIN
      FOR ingreso IN cur_ingreso LOOP
      DBMS_OUTPUT.PUT_LINE('Servidor ID: ' || ingreso.servidor_id);
      END LOOP;
END;
```



```
BEGIN
get_user_servers(1,'01/01/2019','31/12/2022');
END;
```



### 3.2) Para el componente de Mensajería:

***//Requerimiento 4: Proveer una consulta que dado un usuario, un servidor y un rango de fechas devuelva la cantidad de mensajes que envió el usuario en cada canal en los que participó, ordenado de mayor a menor según la cantidad de mensajes***

```
var usuarioId = "1";//Cambiar ID del Usuario
var servidorId = "1";//Cambiar ID del Servidor
var fechaInicio = new Date("01/01/2023");//Cambiar fecha de inicio (Formato MM/DD/YYYY)
var fechaFin = new Date("01/01/2023");//Cambiar fecha de final (Formato MM/DD/YYYY)

db.mensajes.aggregate([
 {
  $match: {
   usuarioEmisorId: usuarioId,
   servidorId: servidorId,
   fecha: {
    $gte: fechaInicio,
    $lte: fechaFin
   }
  }
 },
 {
  $group: {
   _id: "$canalId",
   cantidadMensajes: { $sum: 1 }
  }
 },
 {
  $sort: {
   cantidadMensajes: -1
  }
 }
]);
```

***//Requerimento 5: Proveer una consulta que dado un canal, muestre los últimos 100 mensajes (de cualquier tipo) enviados al canal. En caso de ser archivos, contener hashtags, etc. deben también mostrarse***

```
var idCanal= "1";//Cambiar ID del Canal
db.Mensajes.find({ canalId: idCanal })
 .sort({ fecha: -1 })
 .limit(100);
```

# PARTE 4

## 4.1) Buscar información y explicar los problemas que tuvieron en el uso de MongoDB, con qué base de datos la sustituyeron y por qué.

Discord se enfrentó a varios desafíos al utilizar MongoDB como base de datos para su sistema de mensajería. A medida que la plataforma iba creciendo, surgieron problemas de escalabilidad y rendimiento. MongoDB tenía dificultades para mantenerse al día con el creciente volumen de datos y la carga del sistema. Esto generaba problemas de rendimiento y disponibilidad.

Además, Discord necesitaba realizar consultas complejas y avanzadas en su sistema de mensajería. Sin embargo, MongoDB no proporcionaba la flexibilidad necesaria para manejar eficientemente estas consultas. La falta de índices múltiples y la complejidad en la estructura de las consultas dificultaban la implementación de las funcionalidades requeridas.

Otro desafío importante fue la falta de soporte transaccional completo en MongoDB. Discord necesitaba garantizar la consistencia de los datos en todas las operaciones, pero MongoDB carecía de transacciones ACID completas en ese momento. Esto generaba problemas para mantener la integridad de los datos y asegurar la coherencia en todas las transacciones.

Debido a estos problemas, Discord tomó la decisión de buscar una alternativa a MongoDB. Después de investigar y evaluar varias opciones, optaron por migrar a Cassandra. Cassandra es una base de datos distribuida NoSQL (Base de datos no relacional) y altamente escalable, diseñada para manejar grandes volúmenes de datos y cargas de trabajo intensivas.

Cassandra ofrece varias ventajas importantes para este caso. En primer lugar, tiene una capacidad de escalabilidad masiva. Puede manejar grandes volúmenes de datos y soportar un alto rendimiento incluso en clústeres distribuidos en múltiples nodos. Esto brinda la confianza de que la base de datos podrá crecer junto con Discord sin comprometer el rendimiento.

Además, Cassandra garantiza alta disponibilidad y tolerancia a fallos. Utiliza un modelo de replicación descentralizado que asegura que los datos estén redundantes en diferentes nodos. Esto significa que incluso si hay fallos en hardware o nodos individuales, el sistema seguirá funcionando sin interrupciones.

Otra ventaja de Cassandra es su flexibilidad en el esquema de datos. A diferencia de las bases de datos relacionales, no requiere un esquema fijo. Esto facilita la evolución del esquema con el tiempo y permite adaptarlo según las necesidades de Discord.

Sin embargo, también es importante mencionar algunas desventajas de Cassandra. A diferencia de las bases de datos relacionales, su modelo de consulta se basa en clave-valor y no admite consultas complejas. Esto requiere un diseño cuidadoso de los datos para satisfacer las necesidades específicas de las consultas.

Además, Cassandra sigue un modelo de consistencia eventual, lo que significa que puede haber un período de tiempo en el que los datos no estén completamente consistentes. Esto requiere una gestión adecuada de la coherencia de los datos en aplicaciones que requieren una consistencia estricta, pero para este caso, como en otras redes sociales la prioridad no es que los datos estén completamente consistentes.

No obstante, a principios de 2022 Discord se enfrentaba con ciertos problemas significativos ante Cassandra. A medida que su clúster de mensajes crecía, con 177 nodos y billones de mensajes, surgieron desafíos importantes. El equipo de guardia de Discord se encontraba constantemente ocupado, ya que la base de datos presentaba problemas de latencia impredecible y requería operaciones de mantenimiento costosas de ejecutar.

La causa principal de estos problemas estaba relacionada con la partición de los datos en Cassandra. Discord utilizaba un esquema de mensajes que los organizaba por canal y "bucket". Sin embargo, esta partición provocaba desequilibrios en el rendimiento. Los canales con un alto volumen de mensajes generaban particiones "calientes", donde las lecturas concurrentes creaban cuellos de botella y aumentaban la latencia en todo el clúster.

Además, las tareas de mantenimiento, como las compactaciones y la gestión del recolector de basura, también afectaban el rendimiento. Discord se veía obligado a realizar operaciones de mantenimiento periódicas, como el "gossip dance", para mantener la base de datos en funcionamiento. Estas tareas consumían tiempo y recursos, lo que agravaba aún más los problemas de latencia.

Habiendo experimentado problemas con Cassandra, se mostraron intrigados por ScyllaDB, una base de datos compatible con Cassandra escrita en C++. ScyllaDB prometía un mejor rendimiento, reparaciones más rápidas, una mayor aislación de cargas de trabajo a través de su arquitectura de fragmentos por núcleo y la ausencia de recolección de basura, ya que está escrita en C++ en lugar de Java.

El equipo había tenido problemas significativos con el recolector de basura en Cassandra, lo cual afectaba la latencia y causaba largas pausas de recolección de basura que requerían reiniciar manualmente los nodos afectados. Estos problemas generaban una carga de trabajo adicional y eran la causa de numerosos problemas de estabilidad en el clúster de mensajes de Discord.

Después de experimentar con ScyllaDB y observar mejoras en las pruebas, tomaron la decisión de migrar todas sus bases de datos a ScyllaDB. Aunque esta decisión requería un esfuerzo considerable debido al tamaño del clúster de mensajes de Cassandra, decidieron migrarlo también. Querían asegurarse de que su nueva base de datos funcionará de la mejor manera posible, optimizando su rendimiento. También deseaban obtener más experiencia con ScyllaDB en producción y comprender sus desafíos.

Además, trabajaron en mejorar el rendimiento de ScyllaDB para sus casos de uso específicos. Durante las pruebas, descubrieron que el rendimiento de las consultas inversas no era suficiente para sus necesidades. Sin embargo, el equipo de ScyllaDB priorizó las mejoras y logró implementar consultas inversas eficientes, eliminando el último obstáculo para la migración del plan de bases de datos.

Sin embargo, el equipo de Discord reconoció que simplemente reemplazar la base de datos no solucionaría automáticamente todos los problemas. Aún podrían surgir problemas de particiones calientes en ScyllaDB. Por lo tanto, también trabajaron en mejorar sus sistemas en general para proteger y facilitar un mejor rendimiento de la base de datos.

**Ventajas de ScyllaDB:**

1. Mejor rendimiento: ScyllaDB promete un rendimiento mejorado en comparación con Cassandra. Su arquitectura basada en fragmentos por núcleo (shard-per-core) y su implementación en C++ permiten un procesamiento más eficiente y una menor latencia en las operaciones de lectura y escritura.

2. Reparaciones más rápidas: ScyllaDB ofrece reparaciones más rápidas en comparación con Cassandra. Esto es posible gracias a su enfoque en la reducción de la sobrecarga de red y algoritmos optimizados que aceleran el proceso de reparación de los datos.

3. Sin recolector de basura: A diferencia de Cassandra, ScyllaDB está escrito en C++ y no utiliza un recolector de basura. Esto evita las pausas prolongadas en el rendimiento causadas por la gestión de la memoria y reduce la variabilidad de la latencia.

4. Mayor estabilidad: La migración a ScyllaDB ayudó a abordar problemas de estabilidad en el clúster de mensajes de Discord. Las mejoras en el rendimiento y la eliminación de la necesidad de recolección de basura contribuyeron a una mayor estabilidad operativa.

**Desventajas de ScyllaDB:**

1. Curva de aprendizaje: Al ser una base de datos diferente de Cassandra, la adopción de ScyllaDB implicó una curva de aprendizaje para el equipo de Discord. Se requería tiempo y recursos adicionales para familiarizarse con las particularidades y la administración de ScyllaDB.

2. Menor comunidad y soporte: Aunque ScyllaDB ha ganado popularidad, su comunidad y soporte aún son más limitados en comparación con Cassandra. Esto puede dificultar la obtención de documentación, recursos y soluciones a problemas específicos.

3. Adaptación de herramientas y ecosistema: Al cambiar a ScyllaDB, Discord tuvo que adaptar su conjunto de herramientas y su ecosistema existente. Esto implicó la evaluación y posible reemplazo de herramientas y bibliotecas que podrían no ser directamente compatibles con ScyllaDB.

# ANEXOS

## Fuentes de información

[How Discord Stores Trillions of Messages]
How Discord Stores Trillions of Messages | Deep Dive
[Discord Chooses ScyllaDB as Its Core Storage Layer]
Why Discord Moved from MongoDB to Apache Cassandra, Let us Discuss
[Discord Went From MongoDB to Cassandra Then ScyllaDB - Why? | HackerNoon]