Universidad Privada Boliviana

Facultad de Ingeniería y Arquitectura

Proyecto Final

Cowork de cocina

Bases De Datos Avanzadas

Rebeca Navarro

Zein Tonconi

La paz- Bolivia- 26 junio de 2025

ÍNDICE

Resumen **1**
1. Introducción 1
2. Objetivos **1**
2.1. Objetivo General 1
2.2. Objetivos Específicos 1
3. Requisitos **2**
3.1. Requisitos funcionales 2
3.2. Requisitos no funcionales 2
4. Diseño conceptual **3**
4.1. Modelo Conceptual 3
4.2. Modelo Relacional 3
4.2.1. PostgreSQL 3
4.2.2. MariaDB 4
4.3. Normalización 4
4.4. Diseño Documental 4
4.5. Diagrama de flujo del ETL 5
4.6. Diseño de la base de Datos SnowFlake 6
5. Diseño de la Base de Datos **6**
5.1. Modelo NoSQL 6
5.2. Estructura de documentos y colecciones 7
6. Arquitectura y Justificación Técnica **7**
7. Implementación **9**
7.1. Vistas 9
7.2. Stored Procedures SP 9
7.3. Funciones 10
7.4. Triggers 10
7.5. Particiones 11
7.6. Ofuscamiento 11
7.7. Consultas Optimizadas 11
7.8. Transacciones ACID 12
7.9. Roles, usuarios y permisos 12
7.10. Base de datos distribuida 13
7.11. Caché 14
7.12. Consultas Mongo 14
7.13. Big Data, ETL, Diagrama Snowflake 16
7.14. Herramientas y Tecnologías 17
8. Pruebas y Evaluación **17**
8.1. Vistas 17
8.2. Stored Procedures SP 17
8.3. Funciones 19
8.4. Triggers 20
8.5. Particiones 21
8.6. Ofuscamiento 21

8.7. Consultas Optimizadas..... 22
8.8. Roles, usuarios y permisos..... 22
8.9. Sharding..... 23
8.10. Caché..... 24
8.11. Consultas Mongo..... 24
8.12. Big Data, ETL, Diagrama Snowflake..... 26
9. Conclusiones y Recomendaciones..... 27
10. Referencias..... 27
11. Anexos..... 28

Resumen
El presente trabajo presenta la gestión de un cowork de cocinas, la cual es una idea que surge con el fin de alquilar ambientes equipados a emprendedores y personas que no cuenten con los recursos suficientes para tener todo lo necesario por su cuenta. Para lograr la administración eficiente se llevó a cabo la implementación de algunos conceptos de bases de datos, relacionales y no relacionales, que nos sirven principalmente para realizar consultas más rápidas y fáciles de usar.

1. Introducción

La idea de un cowork de cocinas surge como una alternativa para ayudar a que los emprendedores puedan iniciar sus negocios sin la necesidad de obtener un espacio adecuado que muchas veces requiere una gran inversión que muchas veces no es una opción si recién están empezando.para evitando la complejidad de obtener un espacio adecuado y la gran inversión que esto conlleva. El proyecto llamado El Mortero, fue tan bien recibido por los ciudadanos de La Paz, que decidieron abrir una nueva sucursal en la ciudad de Cochabamba y Santa Cruz.

Debido al éxito de El Mortero, el número de clientes y reservas de espacios aumentó considerablemente. Sin embargo, el sistema antiguo de gestión de datos resultó insuficiente. Con frecuencia se realizaban reservas duplicadas para el mismo espacio, lo que provocaba conflictos dejando a sus clientes sin lugar. Esta situación representaba un problema serio, ya que su público objetivo son personas de escasos recursos que dependen de estos espacios para trabajar. Cancelarles una reserva significaba quitarles un día de trabajo. Además como los equipos son utilizados sin descansar, deben tener un registro del estado de los equipos para saber cuándo llevarlos a mantenimiento y que la capacidad de sus clientes no se vea afectada.

La expansión de El Mortero trajo problemas como choque de reservas, uso de equipos de mal estado, pérdida de contacto con los proveedores, compra excesiva de un solo tipo de equipos, falta de información para la toma de decisiones, etc. Es por eso que se debe hallar una mejor manera de manejar los datos para mitigar los riesgos que puedan afectar negativamente a los clientes y al negocio.

2. Objetivos

2.1. Objetivo General

Utilizar todas las herramientas de bases de datos tanto relacionales como no relacionales para ofrecer una gestión eficiente del cowork El Mortero.

2.2. Objetivos Específicos

- Implementar una base de datos relacional que nos permita tener todos los datos de forma que sea más fácil gestionarlos, y una NoSQL que nos permita tener datos flexibles.
- Optimizar el rendimiento con el uso de Stored procedures (SP), funciones, triggers, vistas, índices y caché, además de otras opciones que tengan nuestras bases de datos.

3. Requisitos

3.1. Requisitos funcionales

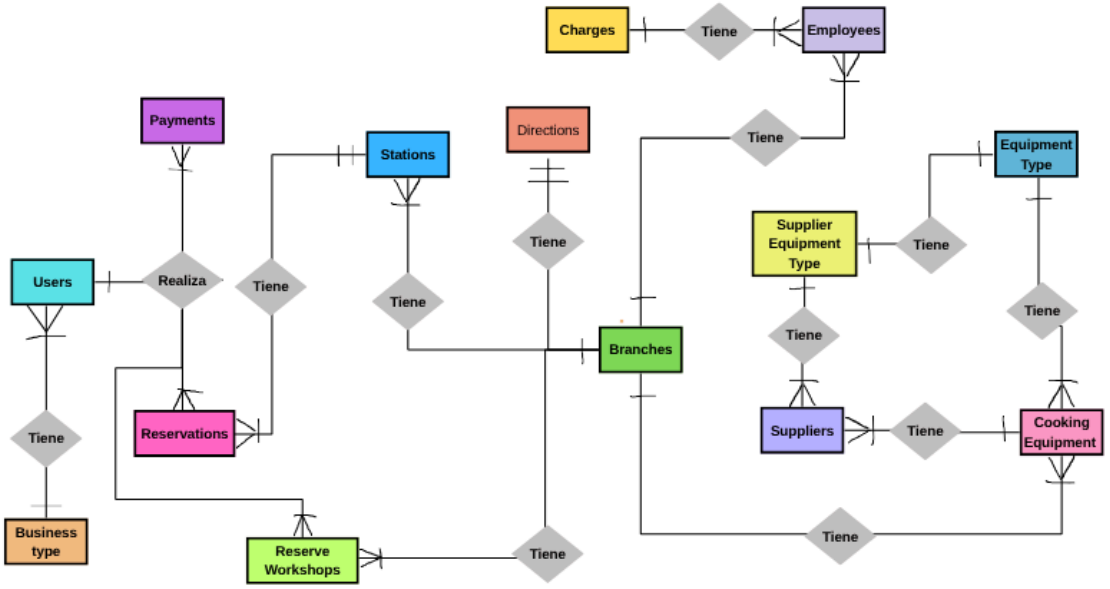
- El sistemas debe mostrar la información de los clientes y qué negocios tienen, esto para realizar estudios de mercado
- El sistema debe mostrar todas las estaciones y también su ubicación, esto con el fin de que el cliente pueda escoger estaciones disponibles de otras sucursales.
- El sistema debe mostrar todos los equipos que necesitan mantenimiento para agilizar la reparación.
- El sistema debe mostrar estadísticas sobre los equipos: cuantos equipos existen de cierto tipo y si valió la pena comprar un equipo.
- El sistema debe ser capaz de insertar un nuevo usuario.
- El sistema debe verificar que el pago se realice solo una vez por reserva y que el pago no sea negativo.
- El sistema debe registrar las transferencias de los equipos entre las sucursales, además de llevar un historial de transferencias.
- El sistema debe mostrar las transferencias hechas en un año específico.
- El sistema debe ser capaz de recibir una cantidad de horas y marcar todos los equipos con un tiempo de uso mayor a las horas dadas, que necesitan mantenimiento.
- El sistema debe calcular la cantidad total pagada por un cliente y el número de reservas que realizó.
- Debe existir un historial sobre las reparaciones de los equipos.
- El sistema debe registrar los comentarios y sugerencias de los clientes sobre el negocio.
- El sistema debe guardar las notificaciones mandadas a un usuario.

3.2. Requisitos no funcionales

- La información sensible de los clientes y proveedores deben estar cifrados para la protección de datos, no obstante que aún se pueda realizar análisis.
- Un cliente no debe tener más de 3 reservas activas, esto para dar oportunidad a otros emprendimientos para usar los espacios y el equipamiento.
- Si la fecha de una reserva ha pasado, el sistema debe marcar automáticamente dicha reserva como vencida.
- Cada vez que se realice el mantenimiento de un equipo se debe marcar automáticamente que el equipo fue reparado y no necesita mantenimiento
- Los datos de las sucursales de La Paz y Cochabamba deben mantenerse separados, evitando cualquier cruce de información entre ellas.

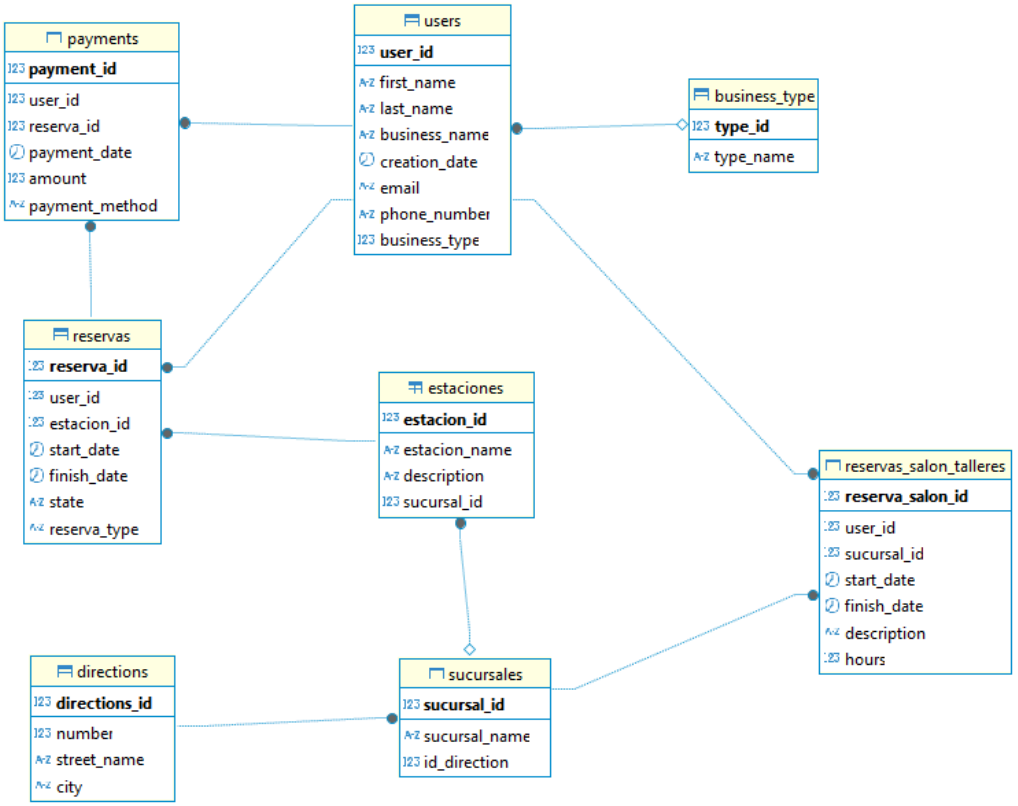
4. Diseño conceptual

4.1. Modelo Conceptual

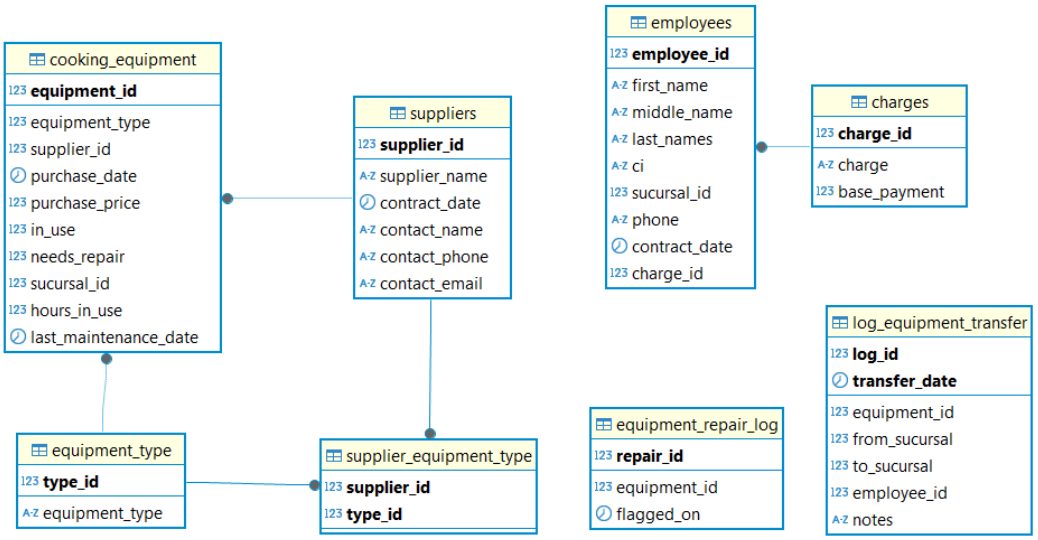


4.2. Modelo Relacional

4.2.1. PostgreSQL



4.2.2. MariaDB



4.3. Normalización

Primera Forma Normal 1FN:

Una base de datos se encuentra en la primera forma normal si todos los atributos son atómicos y simples, y no hay duplicados.

Teniendo en cuenta esto, tanto en PostgreSQL como en MariaDB todos nuestros atributos tienen valores simples y atómicos, y no hay atributos con varios valores.

Segunda Forma Normal 2FN:

Para que se encuentre en la segunda forma normal, primero que nada debe estar en la primera forma normal y los atributos son dependientes de la llave primaria.

En nuestro caso, ambos gestores de bases de datos relacionales tienen PK (Primary Key) definidas.

Tercera Forma Normal 3FN:

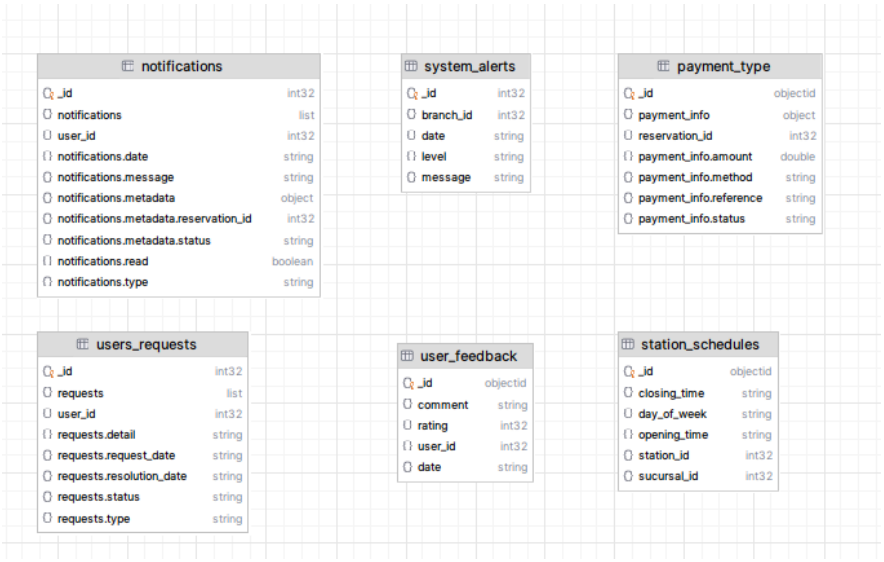
Por último, para que esté en la tercera forma normal, no debe haber columnas que no dependan de la clave primaria.

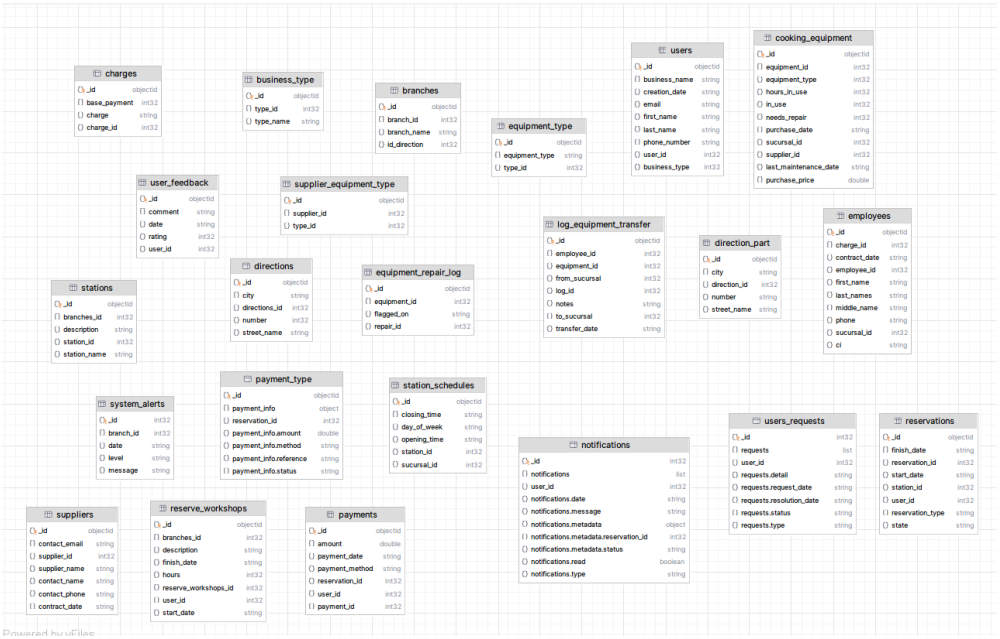
En este proyecto, no hay dependencias transitivas, es decir que dependen de uno y este de otro, y los atributos dependen de la clave.

4.4. Diseño Documental

La base de datos information de la parte no relacional MongoDB, no tiene colecciones que se conecten entre sí, ya que las tablas, que se explicaran posteriormente, son para temas separados, pero que se une con las bases de datos relacionales. Para que esta conexión exista, lo que se hizo fue crear una segunda base de datos en Mongodb llamada cowork_central donde

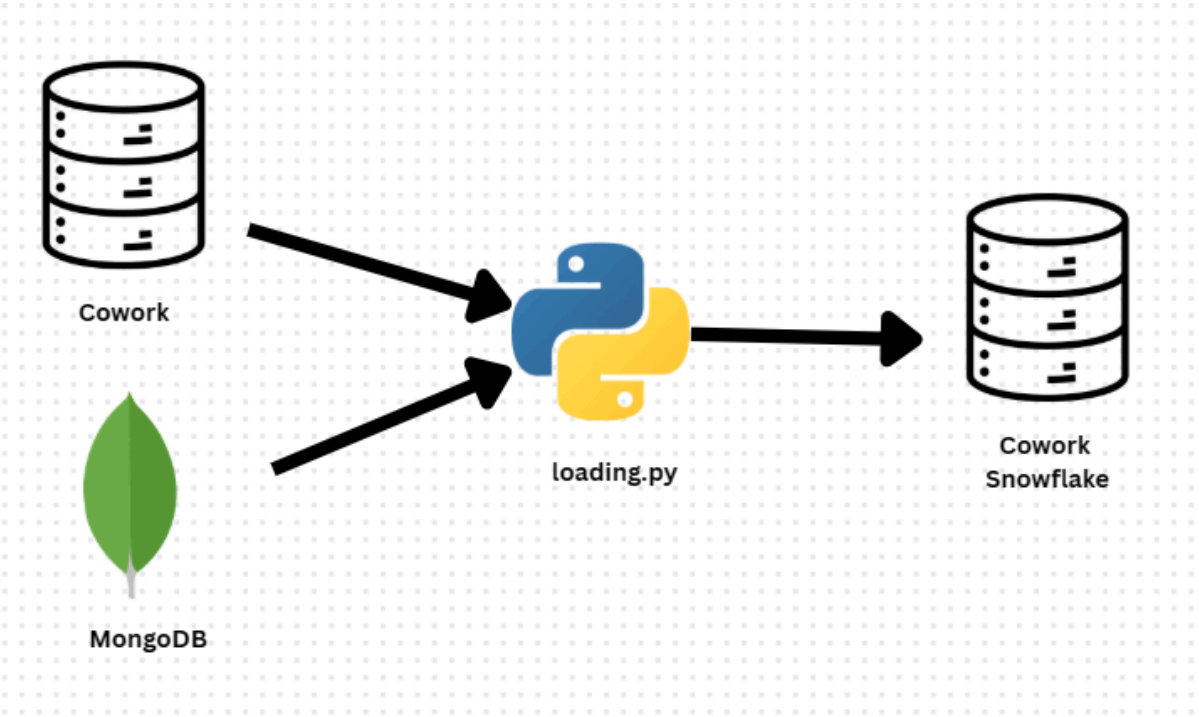
están los datos relacionales y no relacionales como se puede observar en la segunda imagen.





4.5. Diagrama de flujo del ETL

Como primer paso se identificó las bases de datos y las entidades relevantes para responder las preguntas de El Mortero que están presentadas en el siguiente punto. Se implemento un programa python *loading.py* que extrae la información de las bases de datos, para luego transformar y filtrar la información incompleta o incoherente para luego llevarlo a la base de datos cowork_snowflake.



4.6. Diseño de la base de Datos SnowFlake

Se definieron las siguientes preguntas que son vitales para la toma de decisiones para El Mortero:

- ¿Cuáles son los horarios de reserva más frecuentes, segmentados por sucursal y tipo de negocio? Esta pregunta permitirá conocer mejor el comportamiento de los clientes, según la ubicación y que es lo que produce.
- ¿Qué tipos de negocios reservan más horas? ¿Cuáles son los que generan mayores ingresos? Con esto se podrá realizar toma de decisión en el marketing.
- ¿En qué horarios reservan más los diferentes tipos de negocios? Este análisis ayuda a optimizar la asignación de recursos y horarios según la demanda del tipo de negocio.
- ¿Qué tipo de pago utilizan más?

A partir de estas preguntas se definieron los hechos y dimensiones para la base de datos SnowFlake.

5. Diseño de la Base de Datos

5.1. Modelo NoSQL

Para el sistema de cowork de cocina, se optó por utilizar MongoDB como base de datos para almacenar información como horarios, feedback de usuarios, tipo de pago, alertas del sistema, peticiones y notificaciones.
Se escogió MongoDB porque es de rápida lectura, lo cual es esencial para este sistema, donde múltiples usuarios (clientes, cocineros, administradores) deben acceder en tiempo real a datos como horarios de estaciones,

notificaciones y solicitudes. Además, muchos de estos datos no cambian de manera frecuente.

5.2. Estructura de documentos y colecciones.

- **station_schedules (Horarios de las estaciones):** Se guardará los horarios en los que una estación está lista para operar, cabe recalcar que esta colección no almacena los horarios disponibles.
  Esta colección es referencial ya que se trata de una relación de muchos a muchos (M:N) entre las estaciones y las sucursales.

- **user_feedback (Comentarios):** En esta colección se guardaran los comentarios de los clientes que utilizaron el servicio de cowork. Es una colección referencial debido a que utiliza el id del usuario para saber de quien es, además al ser una relación 1 a N, un usuarios podría colocar muchos comentarios. Otra razón para que sea referencial es porque los datos del usuario están en PostgreSQL.

- **payment_type (Tipo de pago):** Se debe tener registro de cuál fue el método de pago de cada reserva realizada, no obstante este puede cambiar con el paso del tiempo y requerir información adicional. El detalle del pago está embebido, mientras que la reserva es referencial.

- **system_alerts (Alertas de sistema):** Es una colección que almacena los eventos generados por fallos dentro de una sucursal, como fallos eléctricos, fugas o alertas de seguridad. Estas alertas son clave para prevenir accidentes, programar mantenimientos y garantizar un entorno seguro para los usuarios. Tiene un modelo referencial porque una misma sucursal puede generar múltiples alertas a lo largo del tiempo (relación 1:N)

- **users_requests (Peticiones de los Usuarios):** En esta colección se tendrá registro de todas las peticiones que los usuarios hayan realizado. Tiene una referencia al usuario que pertenece la petición y la petición como tal está embebido dentro de la colección, debido a que un usuario no realizará tantas peticiones, es decir que tendrá la relación de 1:N pero N no será muy grande.

- **notifications (Notifcaciones):** Almacena mensajes importantes, actualizaciones relevantes relacionados con sus reservas, pagos o cambios en el sistema. Se eligió un modelo embebido porque las notificaciones están directamente relacionadas con un solo usuario, es decir que tienen una relación 1:N y además son pocas.

6. Arquitectura y Justificación Técnica

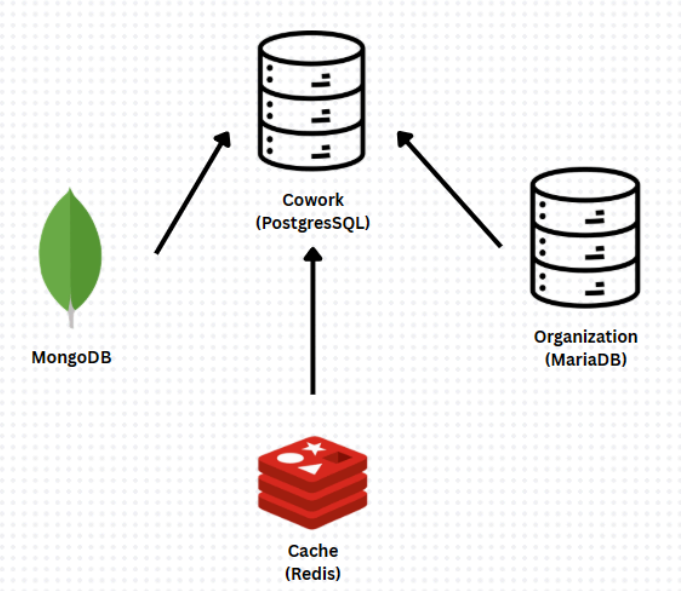
Para el proyecto se utilizaron 2 base de datos relacionales y 2 dos no relacionales:

- **Relacionales**
  - **PostgreSQL (cowork):** En esta base de datos se almacenan los datos mínimos y fundamentales para el funcionamiento del cowork, tales como: usuarios, sucursales, estaciones de trabajo, reservas y

pagos. Estos datos presentan una estructura fija y bien definida, y están fuertemente relacionados entre sí. Por este motivo, se decidió utilizar PostgreSQL, ya que permite modelar relaciones, y garantiza la integridad de datos entre las entidades. Además, al tratarse de información crítica del negocio, es importante asegurar la consistencia de las operaciones, por ejemplo, al registrar una reserva y su pago de forma simultánea.

- **MariaDB (organization):** En esta base de datos se gestionan los aspectos administrativos del cowork, como el personal, el inventario de equipamiento de cocina, los proveedores y las categorías de equipamiento. La naturaleza de estos datos implica múltiples relaciones entre entidades; por ejemplo, un empleado está asociado a una sucursal y a un cargo, mientras que un equipo pertenece a una categoría específica y fue adquirido a través de un proveedor. Por este motivo se optó por utilizar MariaDB, que permite garantizar la integridad de los datos y modelar de forma adecuada las relaciones entre ellos.

- **No relacionales**
  - **MongoDB:** En esta base de datos se almacenan componentes complementarios del sistema que no requieren una estructura rígida, como el feedback de los usuarios, las notificaciones internas, las solicitudes de usuarios, las alertas del sistema y los métodos de pago usados en reservas. Este tipo de información varía en estructura, con el tiempo puede llegar a cambiar, agregando nuevos campos de información o quitandolos; y generalmente se consulta más de lo que se modifica. Por estas razones, se optó por utilizar MongoDB, que permite modelar los datos de forma flexible y acceder a estos datos de forma rápida.
  - **Redis (cache):** Se utiliza Redis como sistema de caché para mejorar el rendimiento de lectura en aquellas entidades del sistema que son consultadas de manera frecuente, como usuarios, sucursales y horarios. Como esta información no cambia con frecuencia, puede almacenarse temporalmente en Redis usando estructuras como hashes y con tiempos de expiración (TTLs) definidos según el tipo de dato. Esto reduce la carga en las bases de datos principales (PostgreSQL y MariaDB), mejorando los tiempos de respuesta del sistema.



7. Implementación

7.1. Vistas

En total se crearon 4 vistas, dos para cada gestor de base de datos.
PostgreSQL:

- **business_type_and_users**: Ésta es una vista que tiene como objetivo centralizar el nombre del negocio, el tipo al que pertenece y la información de contacto. Todo esto facilita en caso de que buscamos por negocio y no por usuario.
- **direction_station**: Ésta vista sirve para tener la dirección de cada estación, teniendo la ciudad, la calle y su descripción.

MariaDB

- **equipment_maintenance_summary**: La vista nos permite ver todos los equipos de cocinas que necesitan mantenimiento y contacto del proveedor, esto es relevante ya que facilita el mandar a mantenimiento con su respectivo proveedor.
- **equipment_costs_by_type**: Vista que muestra el gasto que se realizó por equipo y la cantidad que se tiene, esto ayuda en el stock y a la hora de analizar nuestras ganancias y comparar precios.

7.2. Stored Procedures SP

En total se crearon 4 SP, dos para cada gestor de base de datos.
PostgreSQL:

- **add_user**: Este SP cumple la idea de que el insertar usuarios sea más fácil y rápido, ya que agrega un usuario mediante los datos que recibe.
- **payment_register**: Este SP es útil en el momento de realizar un pago, ya que ayuda en el registro y en la verificación de datos correctos, en este caso que no sean negativos.

MariaDB

- **organization.sp_transfer_equipment**: Este SP consiste en transferir un equipo a otro sucursal además de realizar el log de dicha acción. En caso de que se intente transferir el mismo equipo a dos diferentes sucursales se utilizara manejo de excepciones.
- **sp_mark_high_usage_equipment_for_maintenance**: Este SP marca todos los equipos que tengan un tiempo mínimo de uso como que necesitan mantenimiento, evitando que nuestro equipo sufra daños por uso excesivo.

7.3. Funciones

En total se crearon 4 funciones, dos para cada gestor de base de datos.
PostgreSQL:

- **rental_history**: Es una función que devuelva el historial de rentas hechas por un emprendimiento, en qué sucursal y cuantas veces para poder realizar estadísticas sobre las preferencias de los usuarios.
- **user_general_summary**: Es una función que hace un resumen por cliente, tanto del total de lo que pagó y la cantidad de reservas que hizo.

MariaDB

- **fn_equipment_utilization_score**: Se creó una función para darle un puntaje al equipo que nos dirá cuanto tiempo se usó desde la última vez que se realizó mantenimiento. Con el puntaje de tiempo podemos conseguir cuales son los equipos que están más nuevos o más usados, para tomar decisiones. Por ejemplo, utilizar los más usados para llevarlos a mantenimiento lo más pronto posible.
- **fn_equipment_cost_efficiency**: Relacionada con la función anterior, con esta se puede calcular la relación costo eficiencia con el tiempo de uso del equipo.

7.4. Triggers

En total se crearon 4 triggers que vienen con sus funciones, dos para cada gestor de base de datos.
PostgreSQL:

- **max_active_reservations**: Sirve para validar que un mismo usuario no esté con más de 3 reservas activas, esto para que otros usuarios puedan acceder al servicio.
- **update_reservations**: Es un trigger que cada que se actualice una reserva buscará en la tabla para ver si no hay reservas que ya acabaron pero aparecen como activas, esto porque puede haber un error de insertado de datos y generaría inconsistencias luego.

MariaDB

- **tr_log_repair_flag**: Es un trigger que se encarga de que cada vez que se marca que un equipo necesita reparación se guardará en una tabla de logs.
- **tr_set_repair**: Cuando se realice mantenimiento a un equipo, este ya no requiere reparaciones por lo que cada vez que se actualice la fecha de mantenimiento la columna needs_repair irá a false.

7.5. Particiones

En total se crearon 2 particiones, una para cada gestor de base de datos.
PostgreSQL:

- **direction_part**: Esta partición es según la ciudad de las sucursales para tener todo más ordenado.
  - directions_lpz
  - directions_scz
  - directions_cbb
  - directions_sucre

MariaDB

- **log_equipment_transfer**: Se creó una partición en la tabla de logs de transferencia de equipos entre sucursales según el año.
  - p2018, p2019, …, p2025

7.6. Ofuscamiento

Con el fin de proteger la información sensible de los clientes y del personal del sistema, se realizó el ofuscamiento sobre los datos de las entidades que contienen información personal. Esto incluye:

- **(Proveedores) suppliers:** contact_name, contact_email, contact_phone
- **(Empleados) employees:** first_name, middle_name, last_names, ci, phone
- **(Usuarios) users:** first_name, last_name, email, phone_number

El proceso consistió en reemplazar los valores reales por el nombre de la columna, esto para ocultar la información personal y no alterar los datos importantes para posterior análisis. De esta forma, aún se pueden realizar estudios sin comprometer la privacidad de las personas.

7.7. Consultas Optimizadas

Para optimizar consultas, se hizo uso de los índices únicamente en PostgreSQL, ya que era la base de datos con consultas más pesadas por su cantidad de tablas y de datos.
Para llevar a cabo esto, primero realizamos consultas usando las tablas de users, reservations y payments para ver cuánto tiempo nos tomará realizar cada consulta. Aplicamos para esto el Planning Time y Execution Time, que

nos daban valores referenciales sobre el tiempo que tomaba ya sea el execution como el planning. Con esto, se crearon los índices para el nombre del tipo de negocio (business_type type_name), para la ciudad de la dirección (directions city), para algunos ids como el de sucursal en estaciones, estación en reservas, usuario en reservas, el tipo de negocio en usuarios y dirección en sucursales.
Así se pasó de Planning Time: 7.839 ms y Execution Time: 65.727 ms a Planning Time: 0.509 ms y Execution Time: 56.851 ms. Esto con una consulta más grande y teniendo en cuenta la cantidad de datos usados.

7.8. Transacciones ACID

Se tienen dos transacciones:

- **Transferencia de equipos (sp_transfer_equipment):**
  - **Atomicidad:** Toda la operación de transferencia tanto la actualización de la ubicación del equipo y el registro en el historial se realiza como una sola operación, si ocurre algún error se revierten todos los cambios.
  - **Consistencia:** Al verificar que los el destino sea diferente al origen se evita que existan registros erróneos dentro del historial, manteniendo la consistencia en los datos.
  - **Aislamiento:** Se utiliza el nivel de aislamiento SERIALIZABLE, por lo que si se realiza dos transferencias al mismo tiempo con el mismo equipo, solo se ejecute una.
  - **Durabilidad:** Luego de realizar la transacción sin ningún error, se guardan todos los datos.
- **Marcar equipos para mantenimiento (sp_mark_high_usage_equipment_for_maintenance):**
  - **Atomicidad:** Todos los equipos que cumplen el tiempo de uso son marcados en una sola transacción. Si ocurre un error, ninguno se actualiza.
  - **Consistencia:** Se garantiza que sólo equipos que no están ya en reparación y que superan cierto número de horas de uso sean actualizados. Y no marcar equipos que no necesiten mantenimiento.
  - **Aislamiento:** Este no se verá afectado por otras transacciones y si existen otras transacciones ocurriendo de forma paralela no se verá afectado el funcionamiento del sp.
  - **Durabilidad:** Se actualizan los equipos una vez verificado que no existe ningún error.

7.9. Roles, usuarios y permisos

Se crearon 3 roles que son los principales para poder asignarlos a las diferentes personas del negocio, además que se dió los permisos correspondientes.

- **rol_lecture**: A este rol se le otorgaron los permisos de lectura sobre las tablas reservations, branches, reserve_workshops y station, ya que este rol está destinado a los clientes, los cuales sólo deberán ver las estaciones que tenemos y cuándo pueden hacer sus reservas.
- **rol_admin**: Este rol tiene todos los permisos en la base de datos, ya que éste se encargará de todo lo importante, cómo agregar usuarios o editar las sucursales por ejemplo, por lo que debe tener acceso a todo lo necesario en el schema.
- **rol_escritura**: Este rol tiene permisos de actualización y creación en las tablas reservations, users, reserve_workshops y payments, ya que éste rol está destinado para los trabajadores del cowork, que deberán realizar las reservas de acuerdo a lo que les indique el cliente y de igual manera con los pagos podrán guardar si un cliente paga.

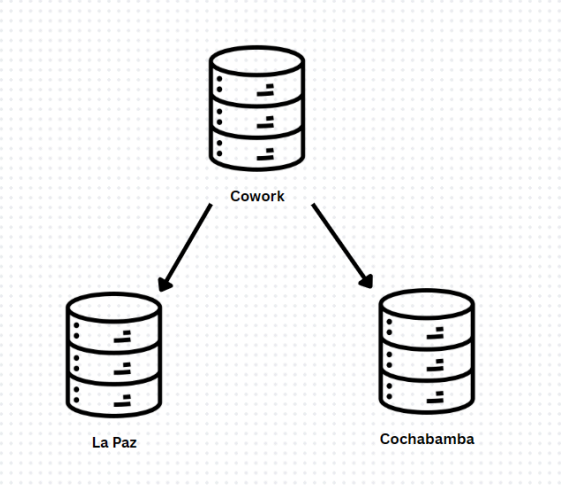
Se crearon de igual manera 3 usuarios para otorgarle un rol a cada uno:

- administrador_admin el cual es un administrador que tiene el rol_admin.
- zein_encargado es un encargado de la tienda que tiene el rol de escritura.
- rebeca_cliente es un cliente que tiene el rol de lectura.

7.10. Base de datos distribuida

- **Sharding**

  Dado que El Mortero cuenta con sucursales en todo el país, principalmente en las ciudades de Cochabamba y La Paz, se aplicó sharding a la base de datos cowork, que contiene toda la lógica del negocio. Esto permite separar los datos correspondientes a cada ciudad en distintos shards. Cada shard almacena únicamente la información de su ciudad, ya que las operaciones y consultas no requieren acceso a los datos de otras regiones. Esta división reduce la carga en cada base de datos y mejora el rendimiento de las consultas.



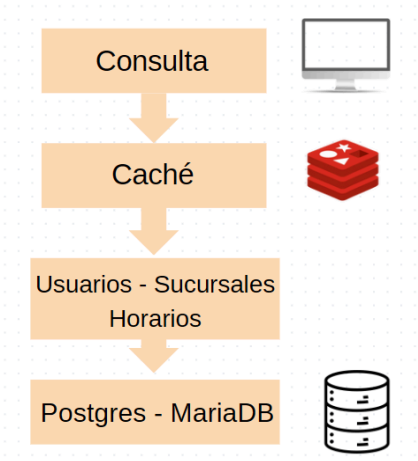
- **Master-slave**

Dado que las reservas es un aspecto muy importante para el negocio como también los usuarios se optó por hacer backups de la base de datos cowork, donde están todos los datos de las reservas y de lo clientes, como también existe información de los pagos, no debe perderse estos datos.
Ahora dado que los equipos de cocina son la materia prima del negocio y el personal es fundamental para su funcionamiento del negocio, se implementó replicación Master-Slave en la base de datos organization. Esto permite contar con copias en tiempo real de los datos más críticos, reduciendo el riesgo de pérdida de información sensible. Además, al delegar las consultas de lectura a los servidores esclavos, se optimiza el rendimiento de las operaciones, especialmente en la lectura de datos relacionados con los equipos de cocina.

7.11. Caché

Para mejorar el rendimiento y reducir la carga sobre la base de datos, se implementó un sistema de caché utilizando Redis, enfocado principalmente en usuarios, sucursales y horarios, datos de acceso frecuente pero que no cambian con frecuencia.

- **Usuarios:** la información del usuario se utiliza de forma constante para mostrar reservas, pagos y notificaciones. Como estos datos no cambian con frecuencia, se decidió almacenar en caché por una hora.
- **Sucursales y horarios:** se usan constantemente para mostrar información como ubicación, disponibilidad y estaciones. Esta información es consultada con alta frecuencia pero rara vez se modifica, por lo que permanecerá en caché por una hora.



7.12. Consultas Mongo

En total se realizaron 13 consultas en la base de datos de Mongo, estas consultas se realizaron con el objetivo de tener sobre todo para evaluar la percepción de los clientes sobre nuestro servicio y facilitar información como los horarios y otros.

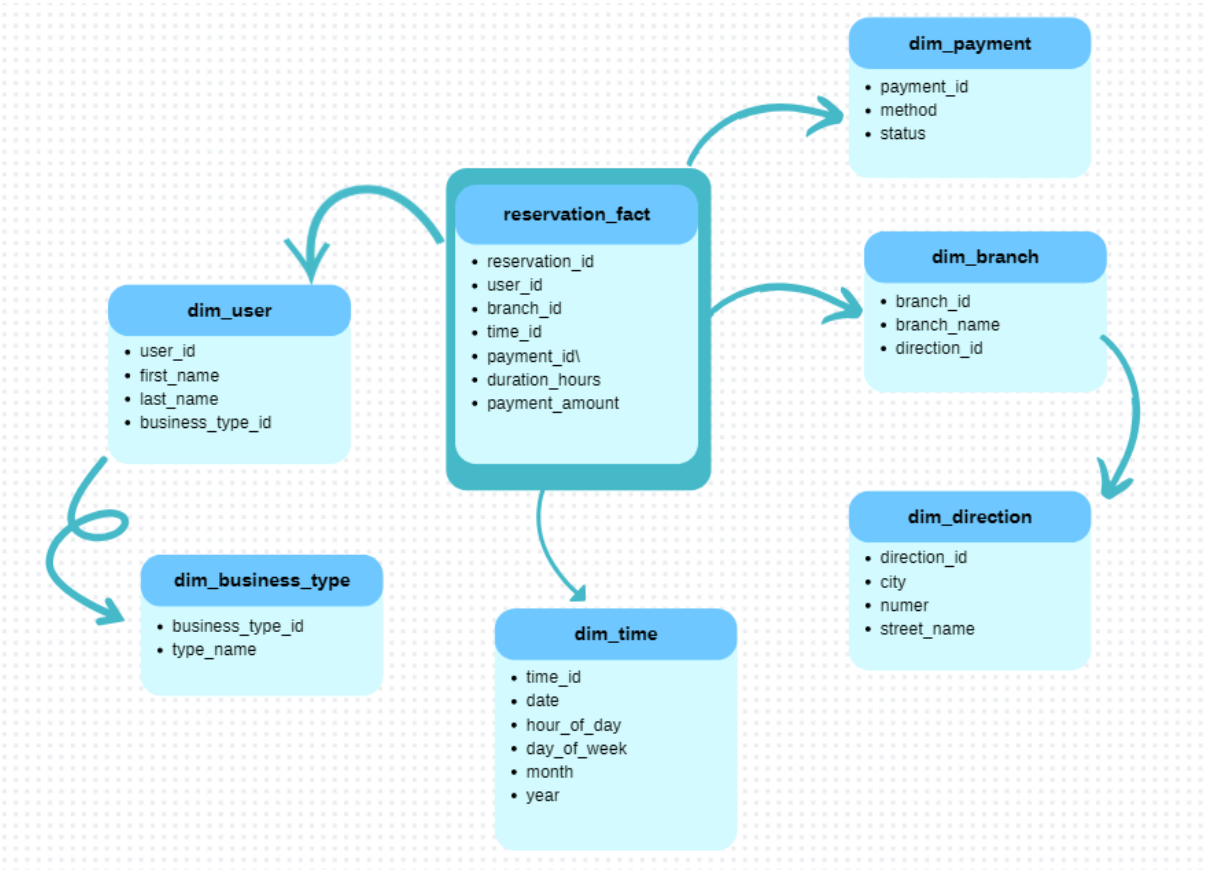
1. La consulta lo que hace es devolver el id de la sucursal y los días en los que atiende, esto con el fin de que el cliente pueda ver a qué sucursal ir de acuerdo a los horarios de atención.
2. La consulta devuelve el promedio de la puntuación que los usuarios dan, esto lo tomamos como si fueran estrellas, en este caso 5 estrellas contaría como una excelente atención y 0 lo contrario. Esta consulta nos sirve para analizar cómo está funcionando el cowork, ya que si los clientes no están satisfechos podría generar muchos problemas.
3. La consulta lo que verifica es todas aquellas alertas de una sucursal que sean de de nivel “high” o “critical” se muestren, ya que estas podrían ser de resolución inmediata., además de mostrar qué es lo que hay que resolver.
4. La consulta lo que hace es mostrar todas las notificaciones que no han sido leídas, ya que cada notificación que un usuario recibe puede este leerla o no, esto es relevante ya que si es de un cambio de fecha en la reserva por ejemplo y no la lee, lo que se podría hacer es notificar nuevamente para evitar que el cliente pierda su reserva.
5. La consulta lo que hace es devolver las solicitudes y cuántas se hicieron de estas, esto porque un usuario puede querer cambiar de horario, que no es tan grave, pero por ejemplo si tenemos muchas solicitudes de tipo queja ya debemos tomar acciones.
6. Esta consulta está relacionada con la anterior, la idea es que esta devuelva cuántas solicitudes se resolvieron, si es nueva y cosas así para que si en un futuro surge un reclamo podamos ver si fue culpa de la empresa o del cliente.
7. La consulta lo que muestra es los tipos de pago que existen y cuánto ha sido pagado por ese medio, esto sobre todo para la parte de finanzas de la empresa, ya que así podemos ver dónde falta dinero o si todo cuadra.
8. Esta consulta se relaciona con las consultas 5 y 6, ya que esta lo que hace es decir cuántas solicitudes tiene un cliente, ya que por ejemplo, puede llegar a ser sospechoso que siempre se arruine algo para el mismo cliente o para ofrecer otros servicios a clientes en específico, digamos que hay un cliente que siempre necesita cambio de horario, se podría ver de ser más accesible.
9. Esta consulta está relacionada a la consulta 3, ya que lo que hace es calcular cuántas alertas hay de cada tipo para ver si hay muchas críticas significa que no se está haciendo un buen trabajo, o si son más informativas deberíamos buscar cómo explicar a los clientes de otras formas.
10. De igual forma, esta consulta está relacionada con la consulta 1, ya que la idea es ver todos los horarios de atención que se tiene y cuántas estaciones hay en ese horario, porque si ya están todas ocupadas a las 09:00 podría tomarse en cuenta cambiar una estación más a ese horario si tenemos muchas solicitudes de este horario.
11. La consulta lo que hace es mostrar los estados de los pagos, cuántos pagos han sido completados y cuántos faltan, entre otros, pero así

tenemos en cuenta que como pueden ser pagos por día, hora o mes, asi evitamos pérdidas por olvidar un pago, además del total de cada estado porque si dice que ganamos un monto debe coincidir con los montos en el area de finanzas.

12. Esta consulta se relaciona con la consulta 2, ya que lo que hace esta es devolver los comentarios de aquellas valoraciones menor a dos estrellas, para entender qué es lo que falló para tener esa puntuación.
13. Esta consulta está relacionada con la consulta 4, ya que la idea de esta es mostrar aquellas notificaciones que fueron confirmadas, para que en caso de quejas tener un respaldo de que el cliente confirmó por lo que no nos compete.

7.13. Big Data, ETL, Diagrama Snowflake

Para empezar con el análisis y responder con las consultas planteadas, primero se realizó el diagrama snowflake. Se identificó la tabla de hechos (reservation_fact) y las tablas de dimension (dim_branch, dim_direction, dim_time, dim_user, dim_business_type, dim_payment)



7.14. Herramientas y Tecnologías

Para lograr compartir todas las bases de datos de PostgreSQL, MariaDB y MongoDB, lo que se hizo es un docker-compose.yml en la carpeta db el cual se encarga de crear y levantar los contenedores para trabajar en un mismo entorno. En conjunto, se implementaron scripts para cada gestor, PostgreSQL y MongoDB, que nos permiten restaurar los datos de manera más rápida, los scripts fueron: import_database.sh para PostgreSQL e import_mongo.sh para MongoDB.
Para la parte de master-slave y sharding, se crearon sus propios docker-compose.yml para levantarlos independientemente.

8. Pruebas y Evaluación

8.1. Vistas

PostgreSQL:

- **business_type_and_users**

	business_name	type_name	first_name	last_name	email
1	Del Meléndez Comida Italiana	Comida Italiana	Jose	Barco	boadalupita@example.com
2	Mi Landa Catering	Catering	Belén	Rocha	leyremesa@example.com
3	Comida Vegana Gourmet	Comida Vegana	Alfonso	Grau	beldalaura@example.net

- **direction_station**

WHERE ORDER BY

	station_name	city	branch_name	street_name	description
1	Estación de Repostería Avanzada #1	La Paz	Cocina Central 4	Pasaje Dolores Lobo	Área para 3 reposteros. Cuenta
2	Área de Fermentación Controlada #2	Santa Cruz	Cocina Creativa 10	Pasaje de Fabiola Patiño	Cabina profesional para 1-2 per

MariaDB

- **equipment_maintenance_summary**

equipment_id	equipment_type	purchase_date	purchase_price	in_use	sucursal_id	supplier_name	contact_name
20002	Parrilla	2023-09-22	3684.62	0	2	Cocinas Logística S.A.	Víctor Fernández
20033	Parrilla	2024-02-18	4280.86	0	1	Maquinaria Profesiona…	Andrea Valencia
20053	Parrilla	2021-04-18	2143.73	1	4	Cocinas Industrial LT…	Karen Quispe
20076	Parrilla	2023-11-29	6462.95	1	2	Maquinaria Logística …	Gabriela González
20105	Parrilla	2021-09-23	7074.41	0	1	Suministros Bolivia S…	Patricia Alcázar

- **equipment_costs_by_type**

	equipment_type	total_units	avg_price	total_spent
1	Batidora	2922	4244.305462	12401860.56
2	Freidora	2851	4214.969411	12016877.79
3	Horno	2803	4293.429536	12034482.99
4	Microondas	2785	4246.054223	11825261.01
5	Parrilla	2952	4225.289631	12473054.99
6	Plancha	2861	4213.981961	12056202.39
7	Refrigerador	2826	4246.159048	11999645.47

8.2. Stored Procedures SP

PostgreSQL:

- **add_user**

```sql
CREATE OR REPLACE PROCEDURE add_user(
    p_first_name VARCHAR,
    p_last_name VARCHAR,
    p_business_name VARCHAR,
    p_creation_date DATE,
    p_email VARCHAR,
    p_phone VARCHAR,
    p_business_type INT,
    p_user_id INT
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO users(first_name, last_name, business_name, creation_date, email, phone_number, business_type, user_id)
    VALUES ( first_name p_first_name, last_name p_last_name, business_name p_business_name,
             creation_date p_creation_date, email p_email, phone_number p_phone, business_type p_business_type, user_id p_user_id);
END;
$$;
CALL add_user( p_first_name 'Rebeca', p_last_name 'Navarro', p_business_name 'Glugluglut en',
               p_creation_date current_date, p_email 'beca@gmail.com', p_phone '+591 78810003', p_business_type 1, p_user_id 97685);
```

Output | cowork.public.users

1 row | Tx: Auto | DDL | CSV

	first_name	last_name	business_name	creation_date	email	phone_number	user_id
1	Rebeca	Navarro	Glugluglut en	2025-06-25	beca@gmail.com	+591 78810003	5987705

- **payment_register**
  - Antes de ejecutar se ve de la siguiente manera con el filtro de reservation_id=45:

WHERE reservation_id=45 | ORDER BY

	payment_id	user_id	reservation_id	payment_date	amount	payment_method
1	45	12959	45	2025-06-01	0	transferencia

  - Después de ejecutar se ve de la siguiente manera si no había un pago previo y el valor es positivo.

WHERE reservation_id=45 | ORDER BY

	payment_id	user_id	reservation_id	payment_date	amount	payment_method
1	45	12959	45	2025-06-25	500	tarjeta

  - En caso contrario, si ya hay un valor nos da un error.

```sql
CALL payment_register( userId 123, reservationId 456, pr_payment_date current_date, pr_amount 500.00, pr_payment_method 'tarjeta');
```

[P0001] ERROR: La reserva 456 ya tiene un pago registrado de 319
Where: función PL/pgSQL payment_register(integer,integer,timestamp without time zone,numeric,character varying) en la línea 11 en RAISE

Stop Retry Ignore

MariaDB

- **organization.sp_transfer_equipment**
  Para demostrar el funcionamiento del SP tomaremos como ejemplo el equipo con id 20002 y se mostrará la tabla de logs de transferencias.

	log_id	equipment_id	from_sucursal	to_sucursal	transfer_date	employee_id	notes
1	5	20002	2	6	2025-06-25 13:14:24	27	Mantenimiento en Suc. 5

Si se intenta transferir a la misma sucursal del equipo muestra el siguiente error.

```sql
15  CALL sp_transfer_equipment( p_equipment_id 20002, p_new_sucursal_id 6, p_employee_id 27,
```

[45000][1644] (conn=8) Error: Destination branch is the same as the current one.

Si se intenta transferir un equipo que no existe muestra el error.

```
15  CALL sp_transfer_equipment( p_equipment_id 1,  p_new_sucursal_id 6,  p_employee_id 27,
```

[45000][1644] (conn=8) Error: The equipment does not exist.

- **sp_mark_high_usage_equipment_for_maintenance**
  Para mostrar el funcionamiento se marcó todos los equipos que tienen más de 17 horas en uso. Primero se realizó un select de todos los equipos con mas de 17 horas:

	equipment_id	needs_repair	hours_in_use
1	20001	0	20
2	20003	0	18

Luego se llamó al SP para marcarlos que necesitan reparación.

	equipment_id	needs_repair	hours_in_use
1	20001	1	20
2	20003	1	18

8.3. Funciones

PostgreSQL:

- **rental_history**

```
SELECT rental_history( business 'Comida Peruana Gourmet');
```

	rental_history
2	("Comida Peruana Gourmet","Comida Peruana","Cocina Central 3",7)
3	("Comida Peruana Gourmet","Comida Peruana","Cocina Central 4",22)
4	("Comida Peruana Gourmet","Comida Peruana","Cocina Central 9",10)
5	("Comida Peruana Gourmet","Comida Peruana","Cocina Creativa 10",6)
6	("Comida Peruana Gourmet","Comida Peruana","Cocina Creativa 7",16)
7	("Comida Peruana Gourmet","Comida Peruana","Cocina Creativa 8",7)
8	("Comida Peruana Gourmet","Comida Peruana","Laboratorio de Sabores 1",4)
9	("Comida Peruana Gourmet","Comida Peruana","Taller Gastronómico 6",11)

- **user_general_summary**

```
SELECT user_general_summary( userId 216);
```

	user_general_summary
1	(Flor,Marquez,1,0,618.53)

MariaDB

- **fn_equipment_utilization_score**

```
SELECT fn_equipment_utilization_score( p_equipment_id 20001);
```

	`fn_equipment_utilization_score(20001)`
1	0.11

- **fn_equipment_cost_efficiency**

```
SELECT fn_equipment_cost_efficiency( p_equipment_id 20001);
```

	`fn_equipment_cost_efficiency(20001)`
1	126.85

8.4. Triggers

PostgreSQL:

- **max_active_reservations**
  Se puede observar que si un usuario tiene 3 reservas activas, una vez que intentemos insertar una más sale un error.

```
INSERT INTO reservations(reservation_id,user_id, station_i
VALUES ( reservation_id 32463,
     user_id 97685, station_id 5, start_date current_date, finish_d
```

[P0001] ERROR: El usuario 97685 ya tiene 3 reservas activas.
Where: función PL/pgSQL maximum_active_reservations() en la línea 10 en RAISE

- **update_reservations**
  Vemos que se actualiza a vencida si insertamos o actualizamos una reserva con el estado activo y las fechas son menores a la actual.

```
UPDATE reservations SET state = 'active' WHERE reservation_id= 45;
```

```
WHERE reservation_id=45   ORDER BY
```

	d	user_id	station_id	start_date	finish_date	state
1	45	3887	7	2025-06-01	2025-06-01	vencida

MariaDB

- **tr_log_repair_flag**
  Para este ejemplo se utilizó el equipo con id 20004 y se marcó que necesita reparación.

```
update cooking_equipment ce set needs_repair = true where equipment_id = 20004;
select * from equipment_repair_log erl;
```

	repair_id	equipment_id	flagged_on
1	7	20004	2025-06-25 13:49:50

- **tr_set_repair**
  Para la demostración se utilizó el equipo 20005, que nunca fue llevado a reparación:

	equipment_id	needs_repair	last_maintenance_date
1	20005	1	<null>

Luego de ser llevado a mantenimiento y actualizado la fecha de ultimo vez en mantenimiento, el trigger actualizo el campo needs_repair a falso.

	equipment_id	needs_repair	last_maintenance_date
1	20005	0	2025-06-25

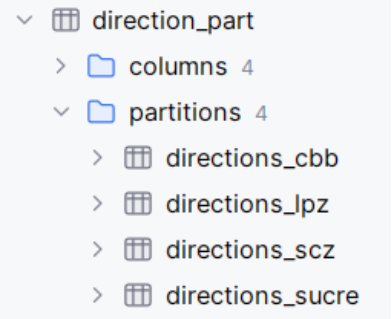
8.5. Particiones

En total se crearon 2 particiones, una para cada gestor de base de datos.

PostgreSQL:

- **direction_part**

  Podemos ver que se crearon las tablas de cada partición y si vemos una, se ve sólo los datos que deberían.

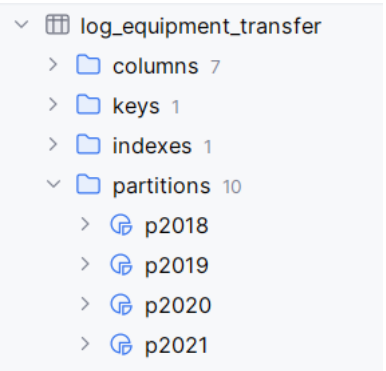


WHERE ORDER BY

	direction_id	number	street_name	city
1	9	1	Cañada Cebrián Guerrero	Cochabamba
2	7	59	Pasaje Brunilda Casanovas	Cochabamba
3	1	1	Urbanización de Jacinto C	Cochabamba
4	2	9	Camino Telmo Antón	Cochabamba

MariaDB

- **log_equipment_transfer**



Dependiendo de la fecha en la que se realizó la transferencia de equipo se mostrará en respectiva partición.

	log_id	equipment_id	from_sucursal	to_sucursal	transfer_date
1	5	20002	2	6	2025-06-25 13:14:24

8.6. Ofuscamiento

Con el fin de proteger la información sensible de los clientes y del personal del sistema, se realizó el ofuscamiento sobre los datos de las entidades que contienen información personal. Esto incluye:

- **(Proveedores) suppliers:** contact_name, contact_email, contact_phone

	supplier_id	supplier_name	contract_date	contact_name	contact_phone	contact_email
1	1	Maquinaria Logistica SAC	2023-07-27	name	phone	email
2	2	Insumos Logistica SRL	2020-11-18	name	phone	email
3	3	Maquinaria Industrial S.A.	2019-07-18	name	phone	email
4	4	Insumos Gastronómica S.A.	2020-02-01	name	phone	email
5	5	Maquinaria Gastronómica S.A.	2019-09-12	name	phone	email

- **(Empleados) employees:** first_name, middle_name, last_names, ci, phone

	employee_id	first_name	middle_name	last_names	ci	sucursal_id	phone
1	1	first_name	middle_name	last_names	ci	3	phone_number
2	2	first_name	middle_name	last_names	ci	14	phone_number
3	3	first_name	middle_name	last_names	ci	2	phone_number
4	4	first_name	middle_name	last_names	ci	5	phone_number

- **(Usuarios) users:** first_name, last_name, email, phone_number

	first_name	last_name	business_name	creation_date	email	phone_number	user_id	business_type
1	first_name	last_name	Catering Original	2022-09-08	email	phone_number	231	12
2	first_name	last_name	Empanadas Gourmet	2023-08-10	email	phone_number	458	20
3	first_name	last_name	Catering Premium	2024-05-11	email	phone_number	912	12
4	first_name	last_name	Panadería Premium	2024-08-04	email	phone_number	1,138	2

El proceso consistió en reemplazar los valores reales por el nombre de la columna, esto para ocultar la información personal y no alterar los datos importantes para posterior análisis. De esta forma, aún se pueden realizar estudios sin comprometer la privacidad de las personas.

8.7. Consultas Optimizadas

- Explain analise antes de la creación de los índices:

```
Planning Time: 7.839 ms
Execution Time: 65.727 ms
```

- Explain analise después de la creación de los índices:

```
Planning Time: 0.509 ms
Execution Time: 56.851 ms
```

8.8. Roles, usuarios y permisos

Para el usuario zein que es encargado, se ve de la siguiente manera, lo que nos dice que no tiene permiso y en caso de tenerlo se ejecuta la consulta:

```
insert into reservations (reservation_id, user_id, station_id, start_date, finish_date, state, reservation_type)
 values ( reservation_id 902878,  user_id 4,  station_id 3,  start_date current_date,
         finish_date current_date + INTERVAL '2 hours', state 'activa',  reservation_type 'hora' )

insert into business_type (type_id, type_name) VALUES ( type_id 3,  type_name 'Asiatica')
```

[42501] ERROR: permission denied for table business_type

Para el usuario rebeca que es cliente se ve de la siguiente manera, si no tiene acceso a una tabla directamente sale un error, tanto para insert, lectura o cualquier acción que intente:

```
select *
from reservations;

select *
from direction_part;
```

[42501] ERROR: permission denied for table direction_part

```
INSERT INTO reservations(reservation_id,user_id, station_id, start_date, finish_date, state, reservation_type)
VALUES ( reservation_id 32463,
    user_id 97685, station_id 5, start_date current_date, finish_date current_date + INTERVAL '2 hours', state 'activa',  reserv
```

[42501] ERROR: permission denied for table reservations

8.9. Sharding

Se hizo la prueba con 3 usuarios, uno de La Paz, uno de Cochabamba y otro de Santa Cruz.

```
La Paz: Result(1) [
  { user_id: 1000002, first_name: 'Rebeca', last_name: 'Navarro' }
]
Cbba: Result(2) [
  { user_id: 1000003, first_name: 'Hade', last_name: 'Villegas' },
  { user_id: 1000005, first_name: 'Monserrat', last_name: 'Del Pilar' }
]
```

Como se puede ver realiza la distribución de datos dependiendo a la ciudad y si no se existe el sharding para esa ciudad utiliza un algoritmo simple de redistribución.

8.10. Caché

Ejecutando el archivo cache.js se observa que primero consigue el dato de la BD y luego de redis

```
Usuario:
{
  first_name: 'Jose',
  last_name: 'Barco',
  business_name: 'Del Meléndez Comida Italiana',
  creation_date: 2023-02-19T00:00:00.000Z,
  email: 'boadalupita@example.com',
  phone_number: '+34820822782',
  user_id: 1,
  business_type: 10
}
From Redis
[Object: null prototype] {
  first_name: 'Jose',
  last_name: 'Barco',
  business_name: 'Del Meléndez Comida Italiana',
  creation_date: 'Sat Feb 18 2023 20:00:00 GMT-0400 (Bolivia Time)',
  email: 'boadalupita@example.com',
  phone_number: '+34820822782',
  user_id: '1',
  business_type: '10'
}
```

8.11. Consultas Mongo

1.

	{} _id	{} dias_apertura
1	2	["Lunes", "Domingo", "Miércoles"]
2	3	["Jueves", "Lunes", "Viernes", "Martes"]
3	1	["Lunes", "Martes", "Sábado"]

2.

	{} _id	{} promedio_rating
1	0	3.7

3.

		{} level	{} message	{} sucursal_id
1	Z	high	Temperatura elevada en …	1
2	Z	critical	Fallo en sistema de ven…	2
3	Z	high	Alerta de seguridad: ac…	3

4.

	{} _id	{} mensajes
1	2001	Descuento especial del 20% en talleres e…
2	2002	Su factura del mes de mayo está disponib…
3	2003	Alerta: su estación de trabajo será mant…
4	2004	Recordatorio: reserva activa para hoy a …
5	2004	Nuevos equipos disponibles - reserve aho…
6	2004	¡Gracias por su evaluación! ¿Cómo califi…
7	2005	Pago rechazado - por favor actualice su …

5.

	_id	total
1	Consulta general	1
2	Renovación membresía	1
3	Reclamo técnico	1
4	Reembolso	1
5	Mantenimiento	1
6	Queja	1
7	Cambio de horario	1
8	Facturación	1

6.

	_id	total
1	en_proceso	2
2	nuevo	2
3	resuelto	2
4	pendiente	3

7.

	_id	monto_total
1	app	85.3
2	bizum	90
3	efectivo	135.5
4	transferencia	150.75
5	paypal	180.25
6	dividido	350
7	tarjeta	365.9
8	contrato	1200

8.

	total_solicitudes	user_id
1	3	103
2	2	101
3	2	105
4	1	102
5	1	104

9.

	_id	total
1	high	2
2	medium	2
3	critical	1
4	informational	2
5	low	3

10.

	_id	total_estaciones
1	09:00	2
2	07:30	2
3	08:30	2
4	10:00	2
5	08:00	2

11.

	_id	monto_total	total
1	completado	1951.95	7
2	pendiente	75.5	1
3	rechazado	180.25	1
4	parcial	350	1

12.

	comentario	rating
1	Las estaciones estaban un poco suci…	2
2	El salón estaba muy ruidoso y no pu…	2

13.

	date	message	status	user_id
1	2025-06-03T20:20:27Z	Su reserva para el 15/0…	confirmed	101

8.12. Big Data, ETL, Diagrama Snowflake

Se respondió las preguntas planteadas:

- ¿Cuáles son los horarios de reserva más frecuentes, segmentados por sucursal y tipo de negocio?

	branch_name	business_type	hour_of_day	reservation_count	total_reserved_hours
1	Cocina Central 4	Comida Italiana	17	26	70
2	Cocina Central 4	Repostería	14	27	66
3	Cocina Central 4	Comida Rápida	20	25	66
4	Cocina Central 4	Hamburguesas Gourmet	8	22	66
5	Cocina Central 4	Comida Saludable	16	27	65
6	Cocina Central 4	Pizzas Artesanales	15	22	62
7	Cocina Central 4	Hamburguesas Gourmet	15	25	61
8	Cocina Central 4	Heladería	13	22	61
9	Cocina Central 4	Comida Rápida	17	24	59
10	Cocina Central 4	Catering	20	23	59
11	Cocina Central 4	Comida Saludable	17	18	58
12	Laboratorio de Sabores 1	Comida Rápida	8	20	58
13	Cocina Central 4	Sushi Bar	1-500 of 501+	22	57
14	Cocina Central 4	Cocina Asiática	12	22	57

- ¿Qué tipos de negocios reservan más horas? ¿Cuáles son los que generan mayores ingresos?

	business_type	total_hours_reserved
1	Hamburguesas Gourmet	2770
2	Comida Rápida	2678
3	Pastelería	2658
4	Panadería	2627
5	Heladería	2620
6	Comida Saludable	2548
7	Comida Mexicana	2535
8	Empanadas	2531
9	Comida Italiana	2528
10	Repostería	2507
11	Comida Vegana	2479

4

	user_segment	payment_method	total_revenue
1	Hamburguesas Gourmet	transferencia	5801.37
2	Cafetería	efectivo	4543.61
3	Comida Rápida	transferencia	4460.28
4	Hamburguesas Gourmet	tarjeta	4435.03
5	Repostería	QR	4394.51
6	Catering	paypal	4355.19
7	Postres	paypal	4351.46
8	Empanadas	paypal	4062.88
9	Comida Saludable	QR	4037.57
10	Panadería	paypal	4036.82

- ¿En qué horarios reservan más los diferentes tipos de negocios?

	business_type	hour_of_day	reservation_count
1	Hamburguesas Gourmet	17	109
2	Comida Mexicana	15	105
3	Comida Saludable	20	103
4	Cafetería	14	100
5	Pastelería	14	99
6	Comida Rápida	8	98
7	Cocina Asiática	9	96
8	Pizzas Artesanales	17	95
9	Heladería	15	94
10	Repostería	14	93
11	Comida Italiana	10	
12	Comida Vegana	12	

20 rows

- ¿Qué tipo de pago utilizan más?

	payment_method	total_revenue
1	QR	61372.12
2	paypal	59173.62
3	transferencia	56844.49
4	tarjeta	50719.35
5	efectivo	48991.53

9. Conclusiones y Recomendaciones

Con la finalización de este proyecto, nos dimos cuenta que logramos entender mejor los conceptos de bases de datos relacionales y no relacionales. Entendimos más que sólo la teoría, sino cómo y cuándo aplicar cada idea, ya que la aplicación en un contexto cotidiano, el resolver problemas reales y verlo en tiempo real nos ayudaba a pensar fuera de la caja.

Si bien el proyecto era pequeño, logramos implementar exitosamente todo lo solicitado, lo que nos permitió comprender que no importa el tamaño del proyecto, las bases de datos son muy útiles para ayudar en cualquier todo de negocio.

10. Referencias

- https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion

- https://blog.saleslayer.com/es/por-que-es-importante-la-normalizacion-de-base-de-datos#:~:text=Tipos%20de%20normalizaci%C3%B3n%20de%20bases%20de%20datos&text=1NF%3A%20Elimina%20duplicados%20y%20crea,dependen%20de%20la%20clave%20principal.
- https://bookdown.org/paranedagarcia/database/normalizacion.html
- https://dev.to/denisakp/backup-and-restore-mongodb-in-a-docker-environment-1ebb

11. Anexos

- Anexo 1: Consulta utilizada para comparar la optimización.

```
EXPLAIN ANALYSE
SELECT
        u.business_name,
        bt.type_name,
        b.branch_name,
        count(r.reservation_id)::INTEGER
    FROM branches b
    INNER JOIN stations s  1<->1..n: ON b.branch_id = s.branches_id
    INNER JOIN reservations r  1<->1..n: ON s.station_id = r.station_id
    INNER JOIN users u  1..n<->1: ON r.user_id = u.user_id
    INNER JOIN business_type bt  1..n<->1: ON u.business_type = bt.type_id
    GROUP BY b.branch_name, u.business_name, bt.type_name;
```