

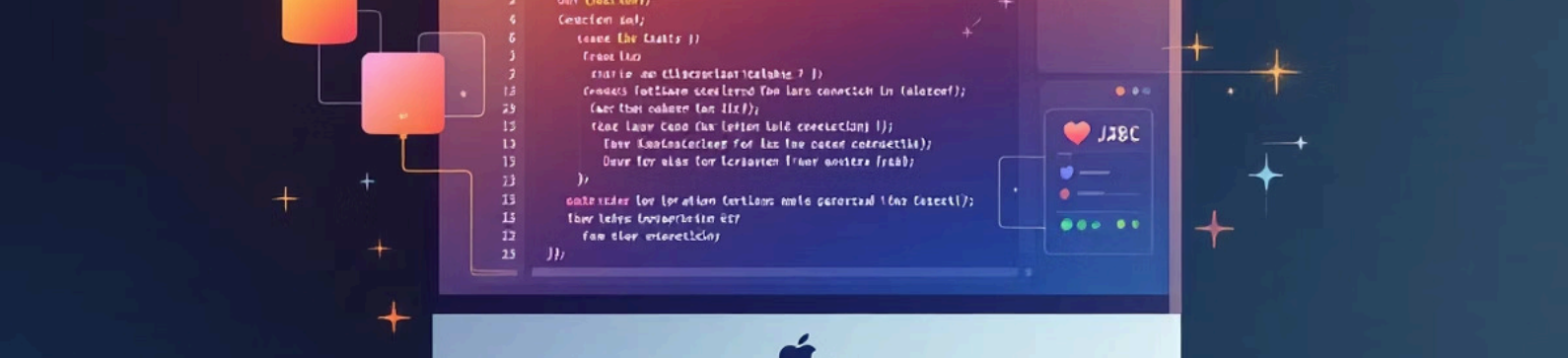
PROMETEO

Unidad 9:

Acceso a Bases de Datos con JDBC

Programación

Técnico Superior de DAM / DAW



Sesión 22 – Conexión básica a BD con JDBC.

PreparedStatement

Cuando trabajas con aplicaciones Java que gestionan datos, necesitas una forma estandarizada y segura de conectarte a motores de base de datos como MySQL, PostgreSQL, Oracle o SQL Server. Aquí es donde entra en juego JDBC (Java Database Connectivity), la API oficial de Java para el acceso a bases de datos relacionales.

Su mayor ventaja es que te permite escribir código independiente del motor de BD: cambias la URL y el driver, pero el resto del programa permanece igual. La conexión con la BD se establece mediante el método `DriverManager.getConnection()`, que recibe la URL de conexión (`jdbc:mysql://...`, `jdbc:postgresql://...`), el usuario y la contraseña.

Estas conexiones consumen recursos, por lo que es fundamental cerrarlas correctamente para evitar fugas de memoria o conexiones huérfanas que saturan el servidor. La forma moderna y recomendada de gestionar este ciclo de vida es `try-with-resources`, que garantiza que `Connection`, `PreparedStatement` y `ResultSet` se cierren automáticamente incluso si ocurre un error.

Un aspecto esencial del desarrollo profesional es evitar vulnerabilidades. Muchos fallos de seguridad en aplicaciones Java vienen por el mal uso de `Statement`, que concatena cadenas y abre la puerta a la inyección SQL. Para evitarlo, debes utilizar `PreparedStatement`, que separa la consulta SQL de los valores que introduces. Los parámetros se expresan con "?" y asignas valores mediante métodos como `setString()`, `setInt()`, `setDouble()`, etc. Esto asegura el escape correcto de datos y protege tu aplicación.

`PreparedStatement` también mejora el rendimiento. El motor de la base de datos compila la consulta una vez y la reutiliza aunque cambien los valores de los parámetros. Esto es especialmente relevante en aplicaciones con alta carga, donde pequeñas optimizaciones marcan una gran diferencia.

Por último, las operaciones básicas siguen el patrón CRUD:

- Create (INSERT)
- Read (SELECT)
- Update (UPDATE)
- Delete (DELETE)

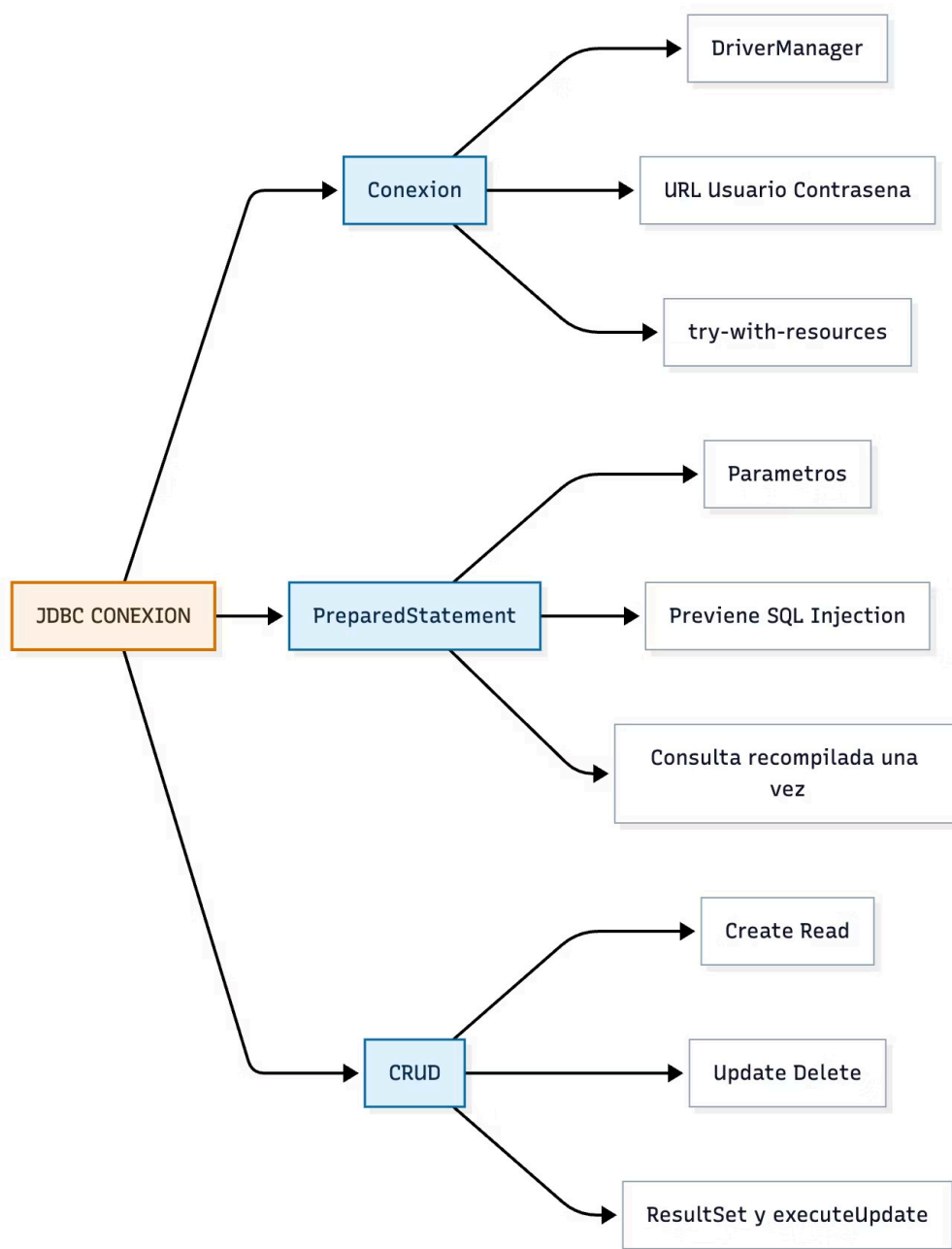
En JDBC se ejecutan con dos métodos clave:

- `executeUpdate()` para INSERT, UPDATE y DELETE (devuelve número de filas afectadas)
- `executeQuery()` para SELECT (devuelve un `ResultSet` que debes recorrer)

Dominar estas bases es imprescindible antes de pasar a técnicas avanzadas como transacciones, batch processing o connection pooling.



Esquema Visual



Explicación del esquema

JDBC Conexión es el nodo central que agrupa los tres pilares del acceso a datos: conexión, ejecución segura y operaciones CRUD.

En el bloque Conexión, DriverManager representa el punto de entrada a la BD, mientras que URL/credenciales y try-with-resources aseguran un ciclo de vida correcto.

En el bloque PreparedStatement, los parámetros indican el uso de "?", la prevención de inyección SQL destaca la seguridad y la reutilización muestra la mejora de rendimiento.

En el bloque CRUD, se representan las cuatro operaciones estándar y los métodos executeQuery/executeUpdate que permiten recuperar o modificar datos.



Caso de Estudio – Sistema de Inventario de Amazon

Amazon gestiona uno de los sistemas de inventario más complejos del mundo. Maneja millones de referencias activas que cambian continuamente: entradas de producto, salidas por venta, devoluciones, transferencias entre almacenes o reposiciones automáticas basadas en predicciones de demanda. Aunque la arquitectura global es extremadamente sofisticada, una parte importante de sus microservicios Java sigue utilizando JDBC para operaciones directas en bases de datos distribuidas.

El proceso comienza cuando un usuario realiza una compra. El sistema debe:

1. Verificar stock disponible.
2. Reservar inventario.
3. Actualizar el nivel de existencias del almacén correspondiente.
4. Registrar el movimiento para trazabilidad.

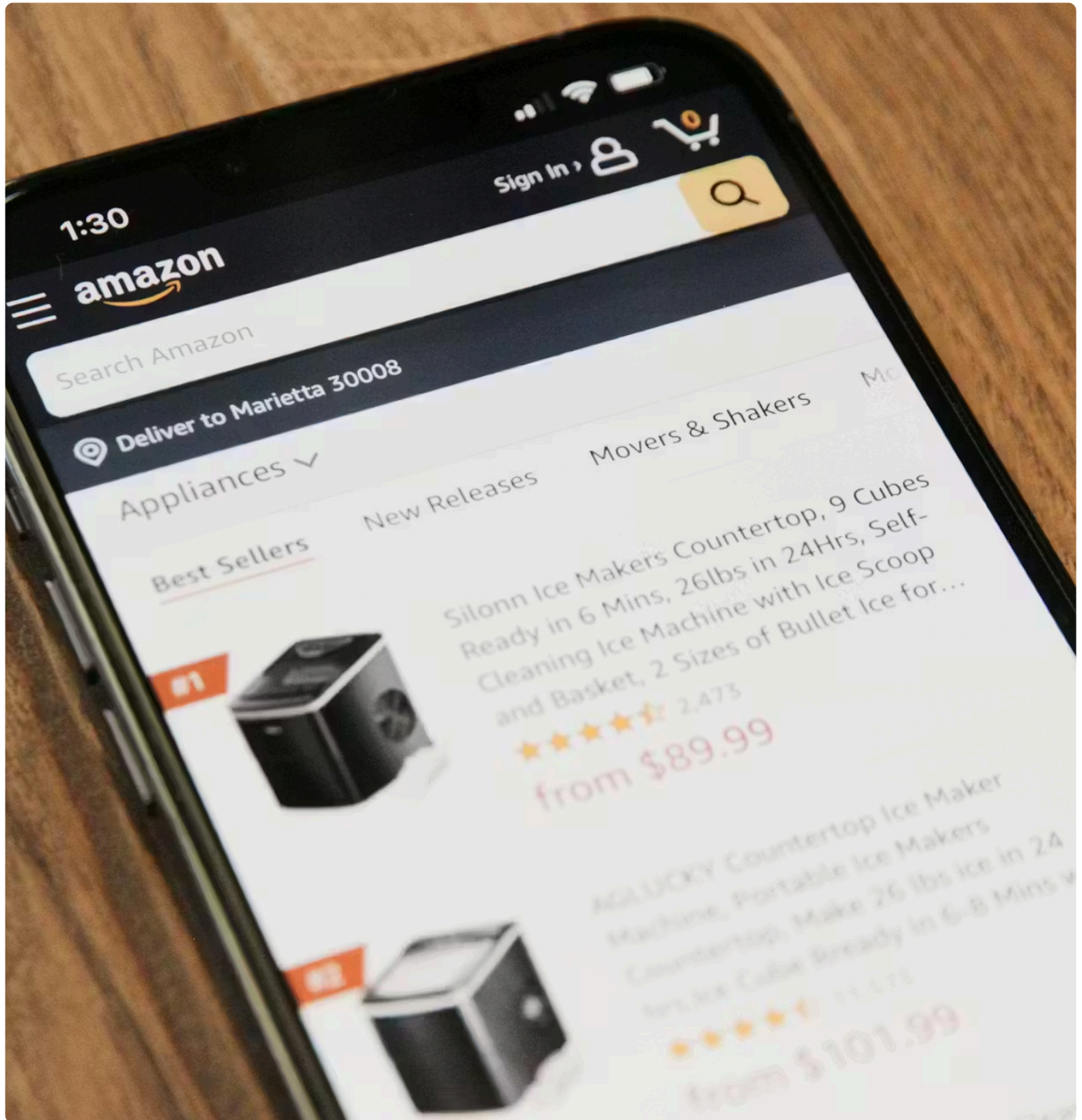
Cada uno de estos pasos es una operación que afecta tablas distintas, y el rendimiento debe ser milimétrico. Aquí `PreparedStatement` es crítico porque permite ejecutar consultas precompiladas millones de veces al día sin el coste de compilarlas continuamente. Además, protege frente a cualquier entrada manipulada, algo esencial en un entorno expuesto a ataques.

Para escalar, Amazon utiliza `connection pooling`, que mantiene un conjunto de conexiones abiertas y reutilizables. Abrir una conexión nueva cada vez sería inviable en términos de latencia. También utilizan `PreparedStatement caching`, que almacena consultas precompiladas en memoria para evitar su recompilación a nivel de driver o motor de BD.

Otra característica clave es el manejo simultáneo de inventarios en múltiples regiones. Amazon procesa aproximadamente 1.6 millones de paquetes diarios, lo que implica un número aún mayor de modificaciones de inventario. JDBC aporta una API estándar que permite agregar servidores de aplicación sin modificar la lógica de acceso a datos: solo se ajusta la configuración del pool de conexiones.

El éxito de Amazon ilustra lo que deberías aplicar en tu propio código:

- PreparedStatement por defecto.
- Pooling para eficiencia.
- Validación de parámetros.
- Manejo cuidadoso de excepciones.
- Código de acceso a datos claro, seguro y mantenible.



Herramientas y Consejos

✓ Usa un connection pooling profesional (HikariCP)

Crear conexiones es una de las operaciones más costosas. HikariCP proporciona un pool muy rápido y estable, utilizado por empresas como Netflix o Red Hat. Reduce la latencia de acceso a datos de forma notable.

✓ Implementa try-with-resources siempre

Usa: `try (Connection conn = ...; PreparedStatement ps = ...; ResultSet rs = ...) { ... }`

Este patrón te asegura que todos los recursos se cierran incluso si ocurre un error.

✓ Valida los parámetros antes de enviarlos a la BD

Nunca asignes valores a un PreparedStatement sin validarlos. Ejemplo: `if (id > 0) ps.setInt(1, id);`

Esto evita errores de datos y protege la integridad de la BD.

✓ Usa DBeaver o DataGrip para probar tus consultas antes de llevarlas al código

Reducirás errores y ahorrarás tiempo de debugging.



Mitos y Realidades

❌ Mito: "Statement y PreparedStatement son prácticamente iguales."

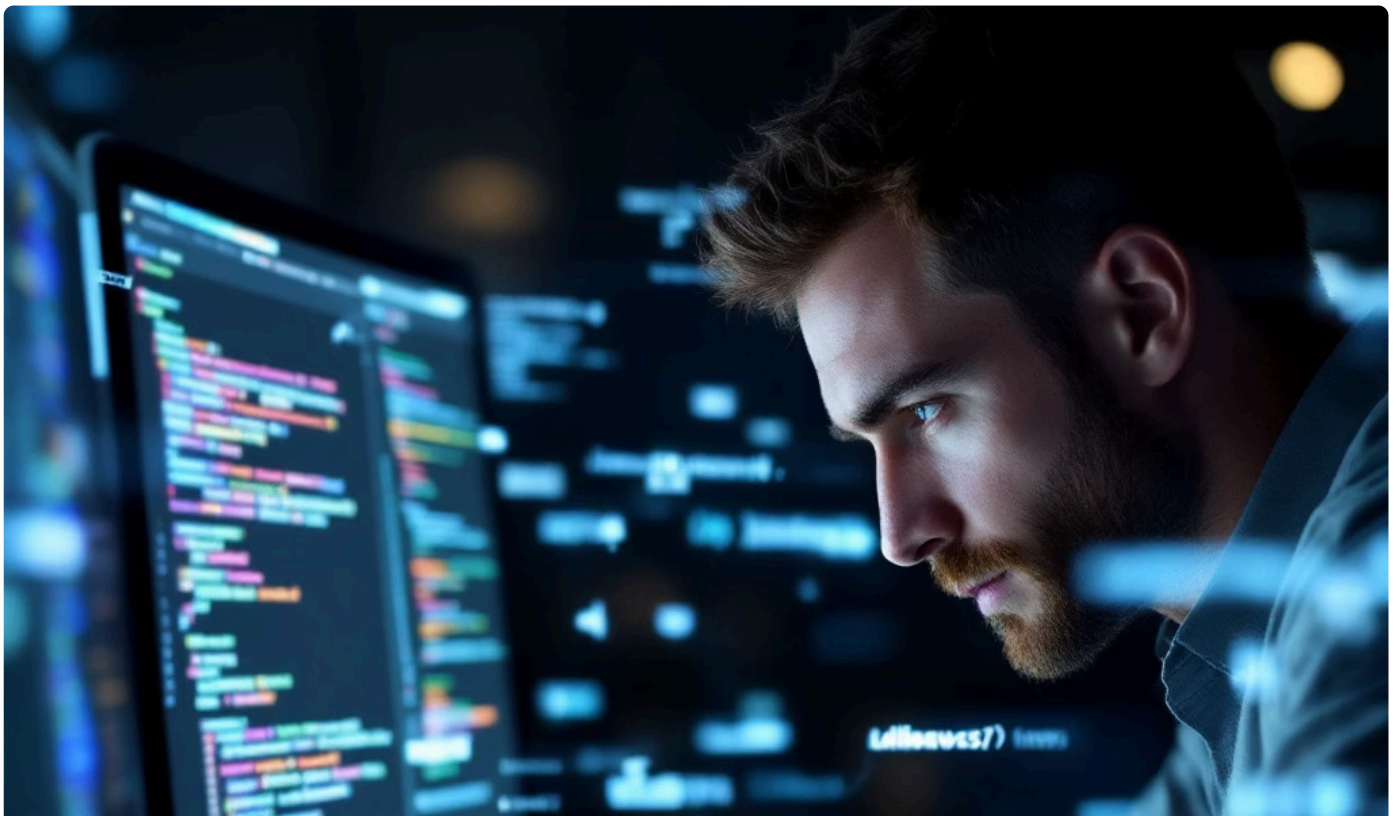
→ FALSO. PreparedStatement previene inyección SQL aislando los parámetros del código SQL, mejora el rendimiento al permitir reutilización de consultas y garantiza la validación de tipos. Statement solo debería usarse en casos muy concretos donde no exista entrada del usuario.

❌ Mito: "JDBC está obsoleto porque ahora existen frameworks como Spring."

→ FALSO. Spring JDBC y JPA funcionan sobre JDBC. Comprender JDBC es indispensable para depurar problemas reales, optimizar consultas y resolver cuellos de botella en proyectos profesionales.

📄 Resumen Final

- JDBC conecta Java con bases de datos mediante DriverManager y URLs específicas.
- PreparedStatement evita inyección SQL y mejora rendimiento gracias a consultas precompiladas.
- CRUD se gestiona con executeQuery y executeUpdate.
- Buenas prácticas: try-with-resources, pooling, validación de parámetros.





Sesión 23 – Transacciones, batch processing y procedimientos almacenados

Cuando tu aplicación necesita ejecutar varias operaciones que dependen unas de otras, no basta con lanzar consultas aisladas. Si una falla, el conjunto completo debe revertirse para evitar inconsistencias. Aquí entran en juego las transacciones, un concepto fundamental en cualquier sistema que maneje datos críticos: pagos, inventario, reservas, procesos contables, etc.

Una transacción garantiza que todas las operaciones se ejecuten con atomicidad, consistencia, aislamiento y durabilidad (ACID). En términos prácticos: o se realizan todas, o no se realiza ninguna. En JDBC, la gestión transaccional se controla principalmente desactivando el modo automático: `conn.setAutoCommit(false);`

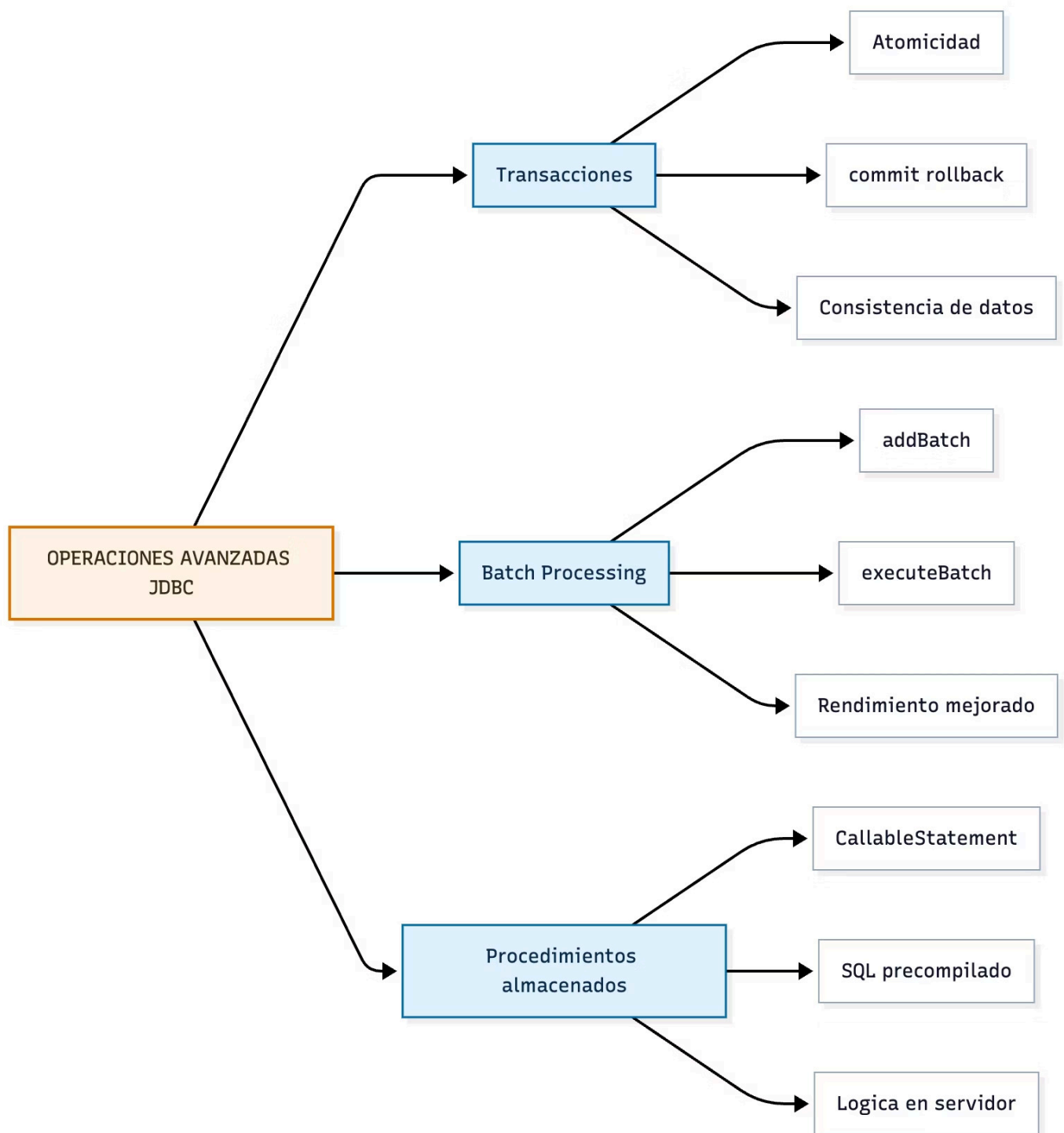
A partir de ese momento, cada operación forma parte de la transacción. Si todo va bien, confirmas cambios con `commit()`. Si ocurre un error —una validación fallida, un dato duplicado, un fallo de red— debes llamar a `rollback()`, lo que devuelve la base de datos a su estado inicial. Esta mecánica evita estados "a medias" que podrían comprometer la integridad del sistema.

Cuando necesitas procesar un gran volumen de operaciones, como miles de inserciones o actualizaciones, ejecutar cada una por separado genera un coste elevado: cada operación implica un viaje de ida y vuelta a la base de datos. La solución es el batch processing, una técnica que permite agrupar instrucciones y enviarlas como un solo lote usando `addBatch()` y `executeBatch()`. Esto reduce la latencia y la carga sobre el servidor, y puede multiplicar por diez el rendimiento en operaciones masivas.

El tercer pilar de esta sesión son los procedimientos almacenados, que consisten en lógica SQL avanzada ejecutada directamente en el servidor de BD. Son útiles cuando necesitas cálculos complejos, reglas de negocio centralizadas o coordinación entre múltiples tablas. Desde Java, los invocas mediante `CallableStatement`, que permite pasar parámetros de entrada o recibir parámetros de salida, algo imposible con consultas normales.

Utilizar transacciones, batch processing y procedimientos almacenados te permite construir sistemas más robustos, eficientes y preparados para manejar carga real. No se trata solo de ejecutar consultas, sino de controlar el proceso de extremo a extremo con garantías profesionales de integridad y rendimiento.

Esquema Visual



Transacciones agrupan operaciones críticas, asegurando atomicidad y revertiendo cambios con rollback cuando algo falla.

Batch processing permite enviar cientos o miles de operaciones como un paquete, reduciendo viajes a la BD y aumentando el rendimiento.

Procedimientos almacenados trasladan la lógica compleja al servidor, optimizando cálculos, validaciones y operaciones encadenadas.

Caso de Estudio – Sistema de Pagos de PayPal

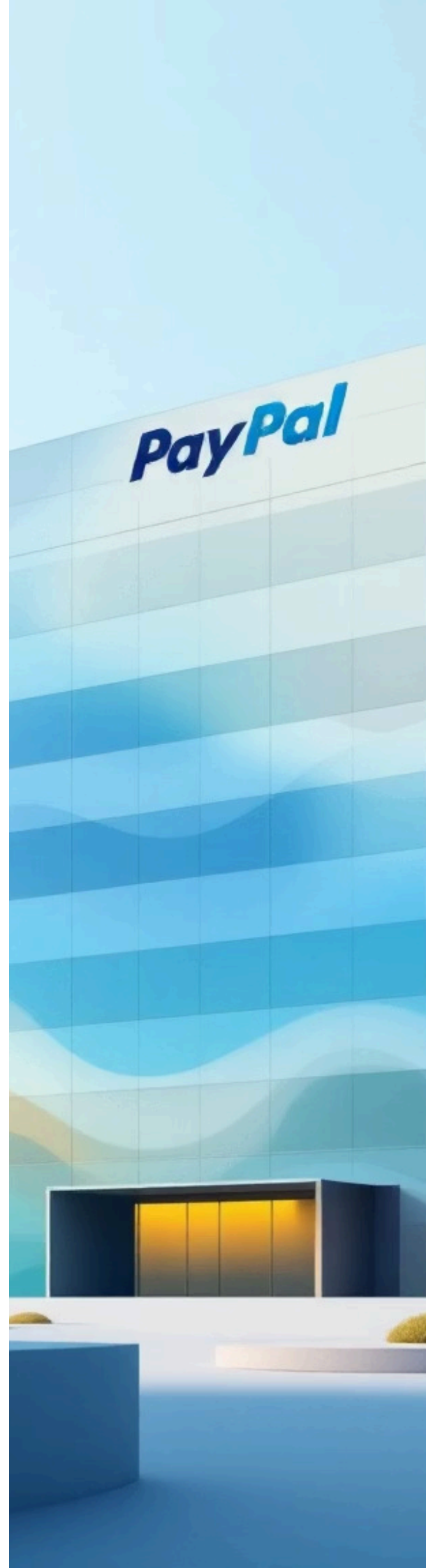
PayPal gestiona millones de transacciones económicas cada día, y para mantener la integridad del dinero de los usuarios necesita un sistema transaccional extremadamente robusto. Cada vez que se realiza un pago, múltiples operaciones deben ejecutarse de forma coordinada:

1. Verificar el saldo de la cuenta origen.
2. Debitar el importe correspondiente.
3. Acreditar la cuenta destino.
4. Registrar el movimiento en el historial transaccional.
5. Actualizar balances consolidados.

Si cualquiera de estos pasos falla —desde un problema de comunicación hasta un conflicto de concurrencia— el sistema debe revertir absolutamente todo. Aquí, las transacciones JDBC tienen un papel esencial: `conn.setAutoCommit(false);`
`// operaciones... conn.commit();`

Este patrón asegura que los movimientos financieros nunca queden en estados inconsistentes. Es preferible un rollback completo a reflejar un pago parcialmente procesado. PayPal cuenta con mecanismos automáticos de rollback y reintentos para garantizar que los fondos de los usuarios estén totalmente protegidos.

Además, PayPal procesa miles de millones de microtransacciones internas cada noche: ajustes, liquidaciones entre países, conversiones de divisas, cálculo de comisiones y auditorías financieras. Ejecutar estas operaciones una por una sería inviable. Por eso emplean batch processing, agrupando operaciones en lotes de entre 5.000 y 10.000 instrucciones.

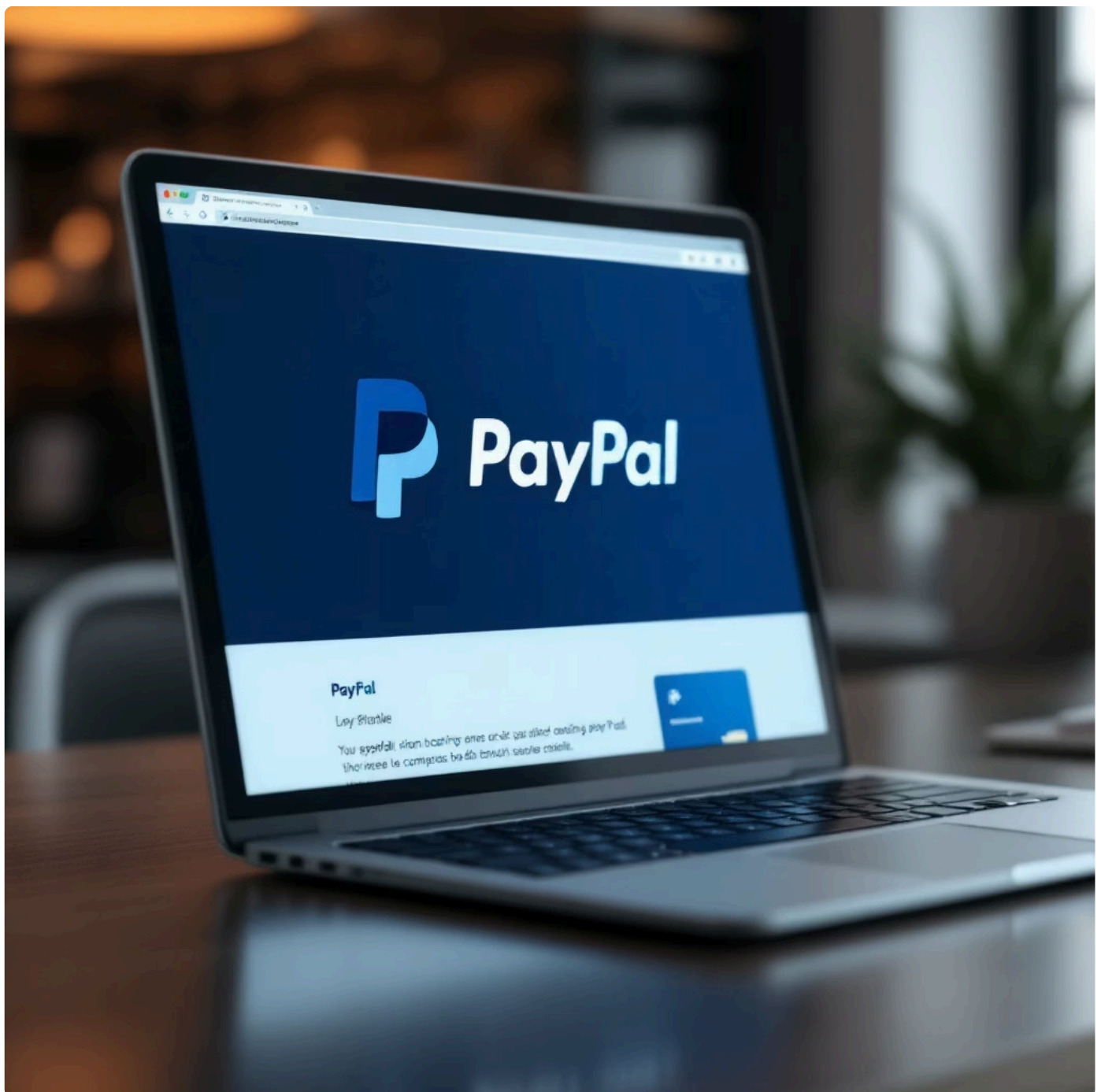


Esto reduce el tiempo de ejecución de horas a minutos y disminuye drásticamente la carga sobre la red y los motores de BD.


Los procedimientos almacenados se utilizan para cálculos complejos, como: aplicación de tarifas según país y tipo de operación, detección de patrones sospechosos para alertas de fraude, validación de límites diarios y reglas de cumplimiento normativo.

Al estar precompilados y ejecutarse en el servidor, evitan latencia, disminuyen el tráfico y aseguran resultados consistentes en todas las regiones donde opera la compañía.

La combinación de transacciones, batch processing y procedimientos almacenados permite a PayPal ofrecer un servicio fiable y seguro en un entorno donde un error mínimo puede tener un impacto millonario.




Herramientas y Consejos

-  Usa savepoints para controlar transacciones complejas

Permiten hacer rollback a un punto intermedio sin deshacer toda la transacción, ideal para operaciones por etapas. Savepoint sp =
`conn.setSavepoint(); // rollback parcial`
`conn.rollback(sp);`

-  Configura timeouts para evitar bloqueos

Las transacciones largas pueden retener bloqueos sobre tablas críticas.
`ps.setQueryTimeout(30); // evita esperas indefinidas`


-  Ajusta el tamaño del batch para rendimiento óptimo

En sistemas reales, lotes entre 1000 y 10000 operaciones suelen ofrecer el mejor equilibrio entre uso de memoria, velocidad y estabilidad.


-  Herramientas recomendadas

JProfiler: análisis profundo de rendimiento
JDBC. DBeaver: testeo de procedimientos y revisiones de triggers. HikariCP (pooling): evita cuellos de botella con conexiones repetidas.

Mitos y Realidades

 Mito: "Las transacciones siempre hacen que la aplicación vaya más lenta."

→ FALSO. El pequeño coste añadido se compensa con creces gracias a la protección contra inconsistencias. La degradación de rendimiento suele venir por un mal diseño, no por usar transacciones.

 Mito: "El batch processing solo sirve para millones de registros."

→ FALSO. Incluso lotes pequeños (100–1000 operaciones) reducen el número de viajes a la base de datos y mejoran el rendimiento, especialmente en redes lentas o arquitecturas distribuidas.

Resumen Final

- Transacciones: atomicidad, commit/rollback, integridad garantizada.
- Batch processing: `addBatch()` + `executeBatch()`, alto rendimiento.
- Procedimientos almacenados: lógica compleja en BD con `CallableStatement`.
- Caso PayPal: pagos seguros, lotes masivos y reglas críticas ejecutadas en servidor.