Programación 3

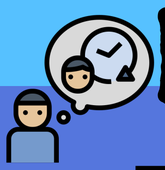
Procedimientos Almacenados (back-end en Java)

Dr. Andrés Melgar

Objetivos de la Sesión

- Incorporar soporte al framework para invocar **procedimientos almacenados** con y sin parámetros.

- Invocar una **consulta SQL que contenga varios *joins*** y con parámetros que podrían ser variables.



Recordando

Arquitectura de SW (estilo arquitectónico monolítico por capas)

Presentación

Front-End

Microsoft .NET

Lógica de Negocio

Java

BussinessObject

+regla_de_negocio()

Patrón DAO
(patrón de interacción con BD Data Mapper)

Dominio

Back-End

Model

-campo1
-campo2

Patrones:

- Data Transfer Object (diseño)
- Identity Field (persistencia)
- Sigle Table Inherence (persistencia)

Persistencia

DataAccessObject

+insertar()
+actualizar()
+eliminar()
+consultar()

DataAccessObjetImplementation

-conexion
-resultSet

+insertar()
+actualizar()
+eliminar()
+consultar()

Base de datos

MySQL

Microsoft® SQL Server®

Procedimientos Almacenados

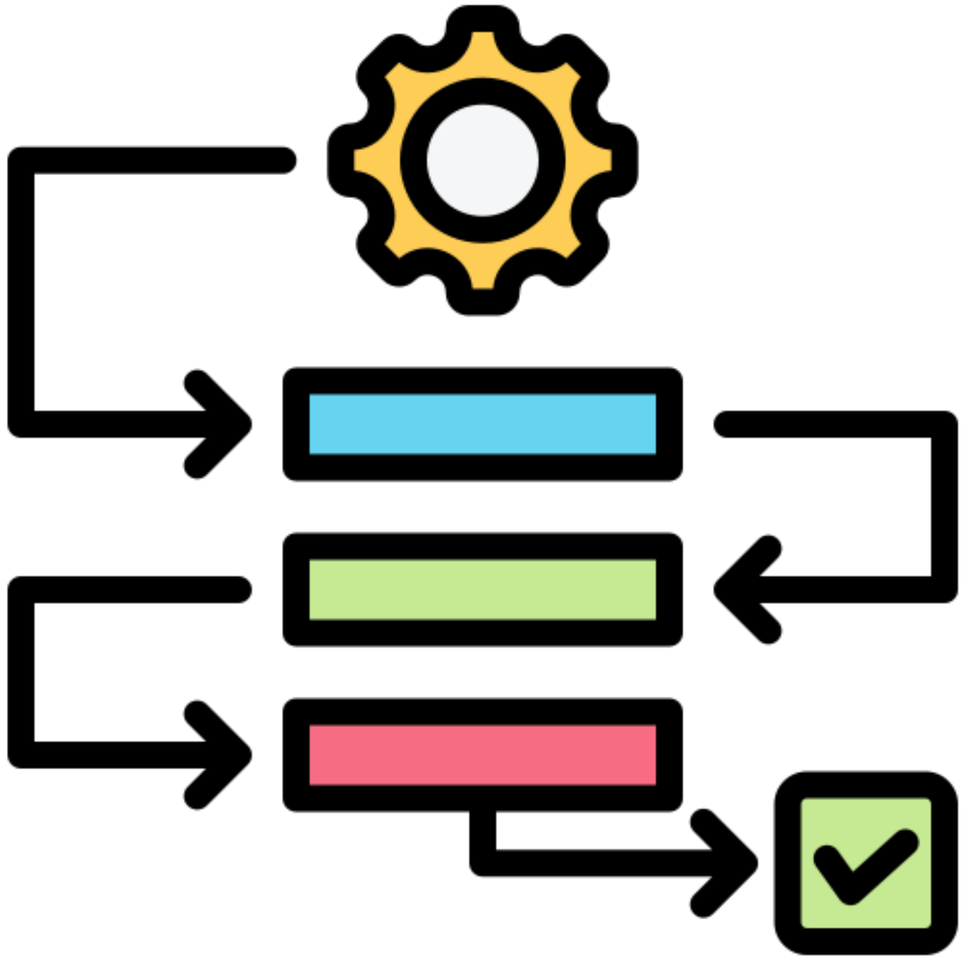
Procedimiento Almacenado

Definición

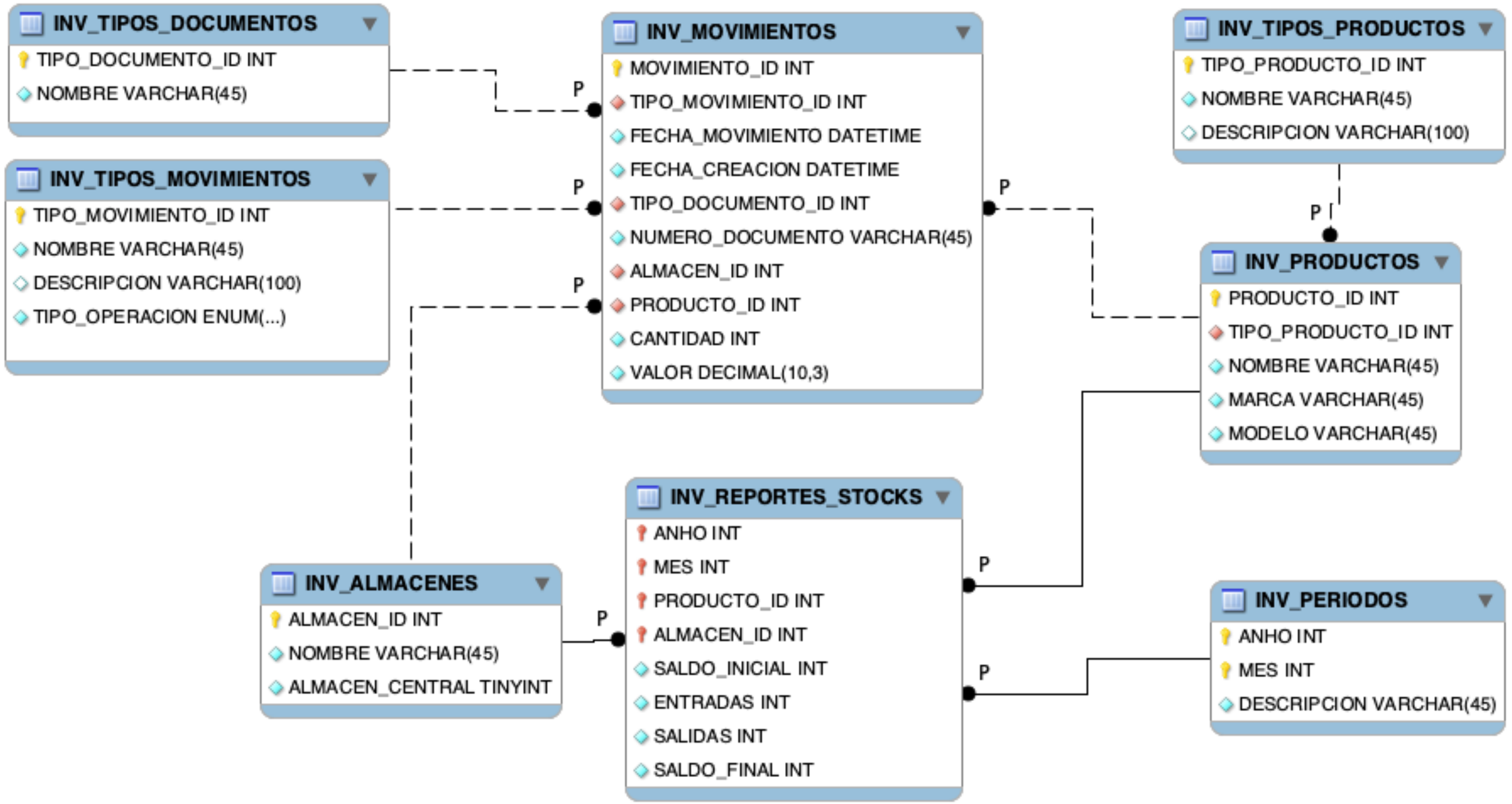
- Un procedimiento almacenado (*stored procedure*, en inglés) es un conjunto de **instrucciones SQL** que se guarda y se ejecuta en el servidor de una base de datos relacional.

- Funciona como una **función o subrutina precompilada** que puede ser llamada múltiples veces desde aplicaciones cliente o desde la misma base de datos.

- Puede recibir **parámetros de entrada y de salida**, lo que lo hace flexible para distintos contextos.



Modelo Relacional (inventarios)



Inventario: Procedimientos Almacenados creados

MySQL_SP_INV_INSERTAR_DATOS_PRUEBA_REPORTE_STOCK.sql
MSSQL_SP_INV_INSERTAR_DATOS_PRUEBA_REPORTE_STOCK.sql

- Genera los datos para probar el reporte de stocks.
- Antes de generar los datos, elimina los datos anteriores.

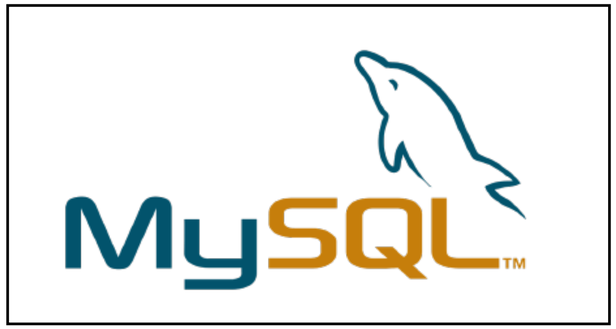
- Genera el reporte de stock por almacén y por producto.
- Para cada almacén, recorre todos los productos que existen en la base de datos. Se calcula el saldo inicial, las entradas, las salidas y el saldo final.
- El reporte se genera por periodo: año y mes.

MySQL_SP_INV_GENERAR_REPORTE_STOCK.sql
MSSQL_SP_INV_GENERAR_REPORTE_STOCK.sql

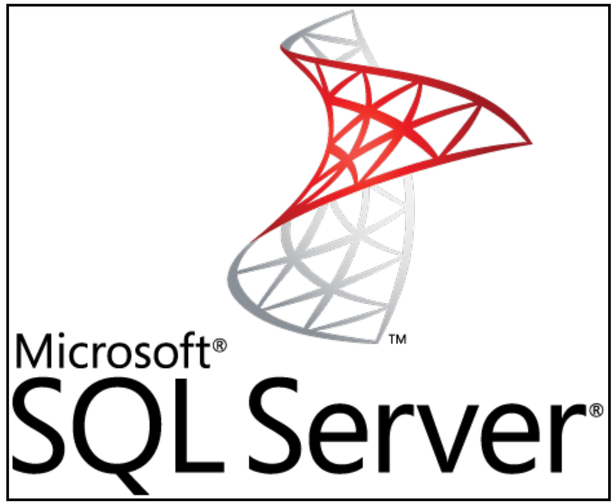
MySQL_SP_INV_ELIMINAR_DATOS_PRUEBA_REPORTE_STOCK.sql
MSSQL_SP_INV_ELIMINAR_DATOS_PRUEBA_REPORTE_STOCK.sql

- Se eliminan los datos de las tablas de la base de datos.
- La base de datos queda lista para hacer otra prueba.

Inventario: Ejecución de Procedimientos Almacenados

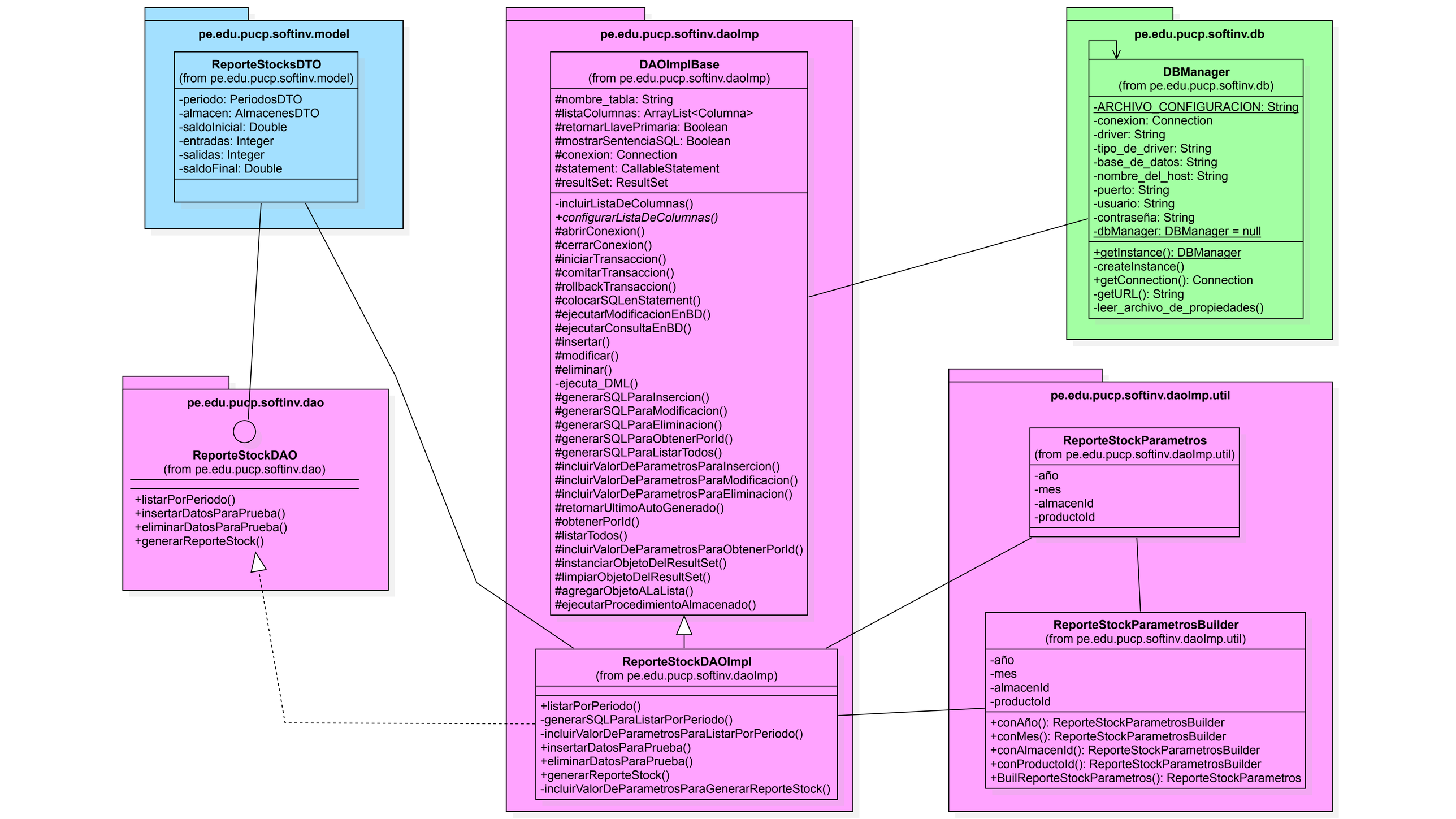



```mysql
CALL SP_INV_INSERTAR_DATOS_PRUEBA_REPORTE_STOCK();
CALL SP_INV_GENERAR_REPORTE_STOCK(2025, 01);
CALL SP_INV_GENERAR_REPORTE_STOCK(2025, 02);
CALL SP_INV_GENERAR_REPORTE_STOCK(2025, 03);
CALL SP_INV_ELIMINAR_DATOS_PRUEBA_REPORTE_STOCK();
```




```sql
EXEC SP_INV_INSERTAR_DATOS_PRUEBA_REPORTE_STOCK;
EXEC SP_INV_GENERAR_REPORTE_STOCK 2025, 1;
EXEC SP_INV_GENERAR_REPORTE_STOCK 2025, 2;
EXEC SP_INV_GENERAR_REPORTE_STOCK 2025, 3;
EXEC SP_INV_ELIMINAR_DATOS_PRUEBA_REPORTE_STOCK;
```

Implementación en la capa de persistencia


pe.edu.pucp.softinv.model
ReporteStocksDTO
(from pe.edu.pucp.softinv.model)
-periodo: PeriodosDTO
-almacen: AlmacenesDTO
-saldoInicial: Double
-entradas: Integer
-salidas: Integer
-saldoFinal: Double
pe.edu.pucp.softinv.daoImp
DAOImplBase
(from pe.edu.pucp.softinv.daoImp)
#nombre_tabla: String
#listaColumnas: ArrayList<Columna>
#retornarLlavePrimaria: Boolean
#mostrarSentenciaSQL: Boolean
#conexion: Connection
#statement: CallableStatement
#resultSet: ResultSet
-incluirListaDeColumnas()
+configurarListaDeColumnas()
#abrirConexion()
#cerrarConexion()
#iniciarTransaccion()
#comitarTransaccion()
#rollbackTransaccion()
#colocarSQLenStatement()
#ejecutarModificacionEnBD()
#ejecutarConsultaEnBD()
#insertar()
#modificar()
#eliminar()
-ejecuta_DML()
#generarSQLParaInsercion()
#generarSQLParaModificacion()
#generarSQLParaEliminacion()
#generarSQLParaObtenerPorId()
#generarSQLParaListarTodos()
#incluirValorDeParametrosParaInsercion()
#incluirValorDeParametrosParaModificacion()
#incluirValorDeParametrosParaEliminacion()
#retornarUltimoAutoGenerado()
#obtenerPorId()
#listarTodos()
#incluirValorDeParametrosParaObtenerPorId()
#instanciarObjetoDelResultSet()
#limpiarObjetoDelResultSet()
#agregarObjetoALaLista()
#ejecutarProcedimientoAlmacenado()
pe.edu.pucp.softinv.db
DBManager
(from pe.edu.pucp.softinv.db)
-ARCHIVO_CONFIGURACION: String
-conexion: Connection
-driver: String
-tipo_de_driver: String
-base_de_datos: String
-nombre_del_host: String
-puerto: String
-usuario: String
-contraseña: String
-dbManager: DBManager = null
+getInstance(): DBManager
-createInstance()
+getConnection(): Connection
-getURL(): String
-leer_archivo_de_propiedades()
pe.edu.pucp.softinv.dao
ReporteStockDAO
(from pe.edu.pucp.softinv.dao)
+listarPorPeriodo()
+insertarDatosParaPrueba()
+eliminarDatosParaPrueba()
+generarReporteStock()
pe.edu.pucp.softinv.daoImp.util
ReporteStockParametros
(from pe.edu.pucp.softinv.daoImp.util)
-año
-mes
-almacenId
-productoId
ReporteStockParametrosBuilder
(from pe.edu.pucp.softinv.daoImp.util)
-año
-mes
-almacenId
-productoId
+conAño(): ReporteStockParametrosBuilder
+conMes(): ReporteStockParametrosBuilder
+conAlmacenId(): ReporteStockParametrosBuilder
+conProductoId(): ReporteStockParametrosBuilder
+BuilReporteStockParametros(): ReporteStockParametros
ReporteStockDAOImpl
(from pe.edu.pucp.softinv.daoImp)
+listarPorPeriodo()
-generarSQLParaListarPorPeriodo()
-incluirValorDeParametrosParaListarPorPeriodo()
+insertarDatosParaPrueba()
+eliminarDatosParaPrueba()
+generarReporteStock()
-incluirValorDeParametrosParaGenerarReporteStock()

Interfaces funcionales

Definición

- Las **interfaces funcionales** en Java son interfaces que tienen un solo método abstracto.
- Están diseñadas para ser utilizadas con expresiones lambda **o referencias a métodos**, lo que facilita la programación funcional en Java.
- Permiten pasar **funciones como parámetros** (funciones de primera clase).
- Permiten usar **métodos como argumentos** (referencias a métodos).

Tipos de Interfaces funcionales

Interfaz	Método abstracto	Uso común
`Consumer<T>`	`void accept(T t)`	Ejecutar una acción con un argumento.
`Supplier<T>`	`T get()`	Proveer un valor.
`Function<T, R>`	`R apply(T t)`	Transformar un valor.
`Predicate<T>`	`boolean test(T t)`	Evaluar una condición.
`UnaryOperator<T>`	`T apply(T t)`	Operación sobre un único valor.
`BinaryOperator<T>`	`T apply(T t1, T t2)`	Combinar dos valores del mismo tipo.

DAOImplBase.java

```java
public void ejecutarProcedimientoAlmacenado(String sql, Consumer incluirValorDeParametros, Object parametros, Boolean conTransaccion) {
    try {
        if (conTransaccion) {
            this.iniciarTransaccion();
        }
        this.colocarSQLenStatement(sql);
        if (incluirValorDeParametros != null) {
            incluirValorDeParametros.accept(parametros);
        }
        this.ejecutarModificacionEnBD();
        if (conTransaccion) {
            this.comitarTransaccion();
        }
    } catch (SQLException ex) {
        System.err.println("Error al intentar ejecutar procedimiento almacenado: " + ex);
        try {
            if (conTransaccion) {
                this.rollbackTransaccion();
            }
        } catch (SQLException ex1) {
            System.err.println("Error al hacer rollback - " + ex);
        }
    } finally {
        try {
            this.cerrarConexion();
        } catch (SQLException ex) {
            System.err.println("Error al cerrar la conexión - " + ex);
        }
    }
}
```

ReporteStockDAOImpl.java

```
@Override
public void generarReporteStock(Integer año, Integer mes) {
    Object parametros = new ReporteStockParametrosBuilder().
            conAño(año).
            conMes(mes).
            BuilReporteStockParametros();
    String sql = "{call SP_INV_GENERAR_REPORTE_STOCK (?, ?)}";
    Boolean conTransacion = true;
    this.ejecutarProcedimientoAlmacenado(sql, this::incluirValorDeParametrosParaGenerarReporteStock, parametros, conTransacion);
}
```

Consumer

Object

¿Cómo implementar el reporte de stocks?

Inventarios: Consulta SQL

```
SELECT r.ANHO,
       r.MES,
       a.ALMACEN_ID,
       a.NOMBRE AS NOMBRE_ALMACEN,
       a.ALMACEN_CENTRAL,
       p.PRODUCTO_ID,
       t.TIPO_PRODUCTO_ID,
       t.NOMBRE AS NOMBRE_TIPO_PRODUCTO,
       t.DESCRIPCION,
       p.NOMBRE AS NOMBRE_PRODUCTO,
       p.MARCA,
       p.MODELO,
       r.SALDO_INICIAL, r.ENTRADAS, r.SALIDAS, r.SALDO_FINAL
FROM INV_REPORTES_STOCKS r
JOIN INV_PRODUCTOS p ON p.PRODUCTO_ID = r.PRODUCTO_ID
JOIN INV_TIPOS_PRODUCTOS t on t.TIPO_PRODUCTO_ID = p.TIPO_PRODUCTO_ID
JOIN INV_ALMACENES a on a.ALMACEN_ID = r.ALMACEN_ID
WHERE r.ANHO = 2025 AND r.MES = 1
```

DAOImplBase.java

```
public List listarTodos(String sql, Consumer incluirValorDeParametros, Object parametros) {
    List lista = new ArrayList<>();
    try {
        this.abrirConexion();
        if (sql == null) {
            sql = this.generarSQLParaListarTodos();
        }
        this.colocarSQLenStatement(sql);
        if (incluirValorDeParametros != null) {
            incluirValorDeParametros.accept(parametros);
        }
        this.ejecutarConsultaEnBD();
        while (this.resultSet.next()) {
            agregarObjetoALaLista(lista);
        }
    } catch (SQLException ex) {
        System.err.println("Error al intentar listarTodos - " + ex);
    } finally {
        try {
            this.cerrarConexion();
        } catch (SQLException ex) {
            System.err.println("Error al cerrar la conexión - " + ex);
        }
    }
    return lista;
}
```

ReporteStockDAOImpl.java

```
@Override
    public ArrayList<ReportesStocksDTO> listarPorPeriodo(Integer año,
    Integer mes, Integer almacenId, Integer productoId) {
        Object parametros = new ReporteStockParametrosBuilder().
                conAño(año).
                conMes(mes).
                conAlmacenId(almacenId).
                conProductoId(productoId).
                BuilReporteStockParametros();
        String sql = this.generarSQLParaListarPorPeriodo();
        return (ArrayList<ReportesStocksDTO>) super.listarTodos(sql,
            this::incluirValorDeParametrosParaListarPorPeriodo, parametros);
    }
```

Patrón Builder

Sobreescritura