**ANEXO IA – ETAPA 4: Seguridad e Integridad**

**Tema: Creación de usuario con privilegios mínimos en MySQL**

**Pregunta al asistente:**

“Necesito crear un usuario en MySQL con privilegios mínimos para realizar pruebas de acceso restringido en mi base de datos. ¿Cuáles son los puntos más importantes a tener en cuenta?”

**Respuesta de IA:**

Crear un usuario con privilegios mínimos para pruebas es una muy buena práctica de seguridad.

Puntos más importantes a tener en cuenta, tanto conceptuales como prácticos:

**1. Principio de privilegio mínimo**

* **Solo otorgá los permisos estrictamente necesarios.**  
  Si el usuario solo debe consultar datos, usá SELECT.  
  Si necesita insertar o actualizar, agregá INSERT o UPDATE.  
  Evitá privilegios globales como ALL PRIVILEGES o acceso a \*.\*.

Ejemplo:

*GRANT SELECT, INSERT ON Pedido\_Envio.Pedido TO 'tester'@'localhost';*

**2. Crear el usuario correctamente**

Usá CREATE USER para crear el usuario de forma explícita (no con GRANT, como se hacía en versiones viejas):

*CREATE USER 'tester'@'localhost' IDENTIFIED BY 'contraseña\_segura';*

Recomendaciones:

* Usá una **contraseña fuerte** (larga, con letras, números y símbolos).
* Limitá el **host de conexión** (localhost si las pruebas son locales, o una IP específica si es remota).

**3. Asignar privilegios específicos**

Definí los permisos por **base de datos o tabla** según lo que el usuario deba probar.

Ejemplos:

**Solo lectura en toda la base:**

*GRANT SELECT ON Pedido\_Envio.\* TO 'tester'@'localhost';*

**Sin posibilidad de borrar datos**

No incluyas DELETE.

**4. Evitá privilegios administrativos**

Nunca le des permisos como:

* GRANT OPTION
* DROP, ALTER, CREATE
* SUPER, FILE, PROCESS, SHUTDOWN
* USAGE global con acceso a \*.\*

Estos pueden comprometer la integridad del sistema o de otras bases.

**5. Refrescá privilegios y verificá**

Después de asignar permisos:

*FLUSH PRIVILEGES;*

Y para verificar:

*SHOW GRANTS FOR 'tester'@'localhost';*

**6. Probá el acceso con ese usuario**

Conectate usando ese usuario para confirmar qué puede y qué no puede hacer. Intentá ejecutar:

*USE Pedido\_Envio;*

*SELECT \* FROM Pedido;*

*DELETE FROM Pedido WHERE id = 1; -- Debería fallar si no tiene permiso*

**7. Revocar o eliminar permisos cuando termines**

Cuando termines las pruebas:

*REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'tester'@'localhost';*

*DROP USER 'tester'@'localhost';*

**Tema: Vistas que oculten información sensible**

**Pregunta al asistente:**

¿Qué tipo de información o datos pueden ser considerados como sensibles en base de datos?

**Respuesta del IA:**

Los tipos más comunes son:

**1. Datos personales identificables (PII - Personally Identifiable Information)**

Son los que permiten **identificar directa o indirectamente a una persona**.

🔸 **Ejemplos directos:**

* Nombre completo
* DNI / CUIT / CUIL
* Dirección física
* Teléfono
* Correo electrónico personal
* Fecha de nacimiento

🔸 **Ejemplos indirectos:**

* Género
* Edad
* Código postal
* Ubicación aproximada
* IP si se asocia a un usuario concreto

📌 **Protección recomendada:**  
Cifrado o enmascaramiento; acceso restringido solo a personal autorizado.

2. **Datos financieros o bancarios**

Relacionados con pagos o recursos económicos.

🔸 Ejemplos:

* Números de tarjeta de crédito/débito
* Códigos CVV o PIN
* Números de cuenta bancaria o CBU
* Información sobre préstamos o ingresos

📌 **Protección recomendada:**  
Cifrado fuerte (AES, RSA), nunca guardar CVV, usar tokenización si es posible.

**3. Datos de salud**

Muy sensibles, especialmente si identifican a una persona.

🔸 Ejemplos:

* Diagnósticos médicos
* Tratamientos o medicaciones
* Resultados de estudios
* Información sobre discapacidades, embarazo, etc.

📌 **Protección recomendada:**  
Altos niveles de encriptación, control de acceso estricto y anonimización si se usan con fines estadísticos.

**4. Datos de autenticación o seguridad**

Relacionados con el acceso al sistema.

🔸 Ejemplos:

* Contraseñas (jamás en texto plano)
* Tokens de sesión o API
* Preguntas/respuestas de seguridad
* Hashes o claves privadas

📌 **Protección recomendada:**  
Almacenar contraseñas con hash seguro (bcrypt, Argon2), nunca reversible.

**5. Datos empresariales o confidenciales**

Propiedad intelectual o información estratégica de una organización.

🔸 Ejemplos:

* Estrategias comerciales
* Datos de proveedores o precios internos
* Información de proyectos o contratos
* Datos de empleados

📌 **Protección recomendada:**  
Control de permisos, auditoría de accesos, cifrado de backups.

**6. Datos sensibles según leyes locales**

En Argentina y en muchos países, se consideran sensibles los datos que revelen:

* Origen racial o étnico
* Opiniones políticas
* Convicciones religiosas o filosóficas
* Afiliación sindical
* Datos genéticos o biométricos
* Vida sexual o salud

👉 Según la **Ley 25.326 (Protección de Datos Personales)**, estos requieren consentimiento expreso y tratamiento especial.

**Tema: Prueba de restricciones PK, FK, UNIQUE, CHECK**

**Pregunta al asistente:**

¿Cuáles son las buenas prácticas de las pruebas de restricciones PK, FK, UNIQUE, CHECK?

**Respuesta del IA:**

**Primary Key (PK)**

**Qué valida:** unicidad de la columna y que no sea NULL.

**Buenas prácticas de prueba:**

* Intentar insertar **dos registros con la misma PK** → debe fallar.
* Intentar insertar **un registro con NULL en la PK** → debe fallar.
* Insertar varios registros con **PK únicas** → debe pasar.
* Verificar que **la PK se use correctamente como referencia en FK**.

**Ejemplo:**

INSERT INTO alumnos (id, nombre) VALUES (1, 'Ana'); -- correcto

INSERT INTO alumnos (id, nombre) VALUES (1, 'Juan'); -- debe fallar (PK duplicada)

INSERT INTO alumnos (id, nombre) VALUES (NULL, 'Luis'); -- debe fallar (PK no NULL)

**Foreign Key (FK)**

**Qué valida:** que el valor exista en la tabla referenciada y respeta integridad referencial.

**Buenas prácticas de prueba:**

* Insertar un registro con **FK inexistente** → debe fallar.
* Insertar un registro con **FK existente** → debe pasar.
* Probar **ON DELETE / ON UPDATE** si se definieron acciones (CASCADE, SET NULL, RESTRICT).

**Ejemplo:**

-- fk id\_carrera debe existir en tabla carreras

INSERT INTO alumnos (id, nombre, id\_carrera) VALUES (2, 'Luis', 99); -- falla si 99 no existe

INSERT INTO alumnos (id, nombre, id\_carrera) VALUES (3, 'Ana', 1); -- pasa si 1 existe en carreras

**Unique**

**Qué valida:** que un valor no se repita en una o varias columnas.

**Buenas prácticas de prueba:**

* Insertar **valores duplicados** → debe fallar.
* Insertar **valores únicos** → debe pasar.
* Probar en **columnas combinadas (unique compuesto)** si corresponde.

**Ejemplo:**

INSERT INTO usuarios (email) VALUES ('a@x.com'); -- correcto

INSERT INTO usuarios (email) VALUES ('a@x.com'); -- falla (duplicado)

**Check**

**Qué valida:** condiciones lógicas o reglas de negocio en los datos.

**Buenas prácticas de prueba:**

* Insertar valores **que cumplan la condición** → debe pasar.
* Insertar valores **que violen la condición** → debe fallar.
* Probar límites y valores extremos si aplica.

**Ejemplo:**

CREATE TABLE empleados (

id INT PRIMARY KEY,

nombre VARCHAR(50),

edad INT CHECK (edad >= 18)

);

INSERT INTO empleados (id, nombre, edad) VALUES (1, 'Ana', 25); -- pasa

INSERT INTO empleados (id, nombre, edad) VALUES (2, 'Juan', 16); -- falla (edad < 18)

**✅ Buenas prácticas generales**

1. **Probar casos positivos y negativos:** no solo los que deben pasar, sino los que deben fallar.
2. **Probar valores límite:** 0, NULL, valores máximos/minimos.
3. **Registrar resultados:** anotar qué insert, update o delete pasó o falló.
4. **Revertir cambios después de la prueba** (rollback o usar transacciones).
5. **Probar combinaciones:** PK + FK + UNIQUE + CHECK juntas, porque a veces hay interdependencias.

**Tema: Uso de PreparedStatement en Java**

**Pregunta al asistente:**

¿Qué es un PreparedStatement?

**Respuesta del IA:**

En Java (JDBC), un PreparedStatement es una sentencia SQL precompilada que permite enviar consultas o comandos a la base de datos de forma segura y eficiente, separando la estructura de la consulta de los valores que se envían como parámetros.

Esto significa que la sentencia SQL se define una sola vez (con marcadores “?” donde irán los valores), y luego el programa reemplaza esos marcadores con los datos reales sin alterar la estructura del SQL.  
El motor de base de datos interpreta los parámetros como datos, no como código, lo cual previene ataques de inyección SQL.

Ejemplo con PreparedStatement (✅ seguro)

*String sql = "SELECT \* FROM Pedido WHERE numero = ?";*

*PreparedStatement ps = conn.prepareStatement(sql);*

*ps.setString(1, numero);*

*ResultSet rs = ps.executeQuery();*

Aquí el valor de numero (aunque sea ' OR '1'='1) se envía como dato, no como parte del SQL.  
El servidor lo interpreta literalmente, buscando un pedido cuyo número sea "' OR '1'='1" (que no existe), y no ejecuta código malicioso.

**Ventajas principales de PreparedStatement**

1. ✅ **Seguridad:** previene **SQL Injection** al separar código SQL y datos.
2. ⚡ **Rendimiento:** la consulta se **precompila** en la base de datos y puede ejecutarse muchas veces con distintos valores.
3. 🧩 **Legibilidad y mantenimiento:** el código es más claro y evita errores de concatenación.
4. 🧠 **Validación automática de tipos:** los métodos setInt, setString, etc., garantizan que los valores coincidan con los tipos esperados.

**Pregunta al asistente:**

¿Qué es un procedimiento almacenado y un callableStatement en Java?

**Respuesta del IA:**

**Procedimiento almacenado (Procedure / Stored Procedure)**

Un **procedimiento almacenado** es básicamente **una función que se guarda dentro de la base de datos** y que puede hacer operaciones SQL.

* **Qué hace:** ejecuta un conjunto de instrucciones SQL (SELECT, INSERT, UPDATE, DELETE, etc.) de manera predefinida.
* **Dónde vive:** dentro de la base de datos.
* **Para qué sirve:**
  + Reutilizar lógica SQL en varios programas o aplicaciones.
  + Ejecutar operaciones complejas en un solo llamado.
  + Mejorar seguridad: los parámetros se pasan sin concatenar SQL (reduce inyección).
* **Sintaxis MySQL ejemplo:**

*DELIMITER $$*

*CREATE PROCEDURE sp\_get\_pedido\_por\_numero(IN p\_numero VARCHAR(50))*

*BEGIN*

*SELECT id, numero, fecha, clienteNombre, total, estado*

*FROM Pedido*

*WHERE numero = p\_numero AND eliminado = FALSE;*

*END$$*

*DELIMITER ;*

**CallableStatement en Java**

Un **CallableStatement** es la clase de Java que **permite llamar a procedimientos almacenados** desde tu aplicación Java.

* **Equivalente a PreparedStatement, pero para SP.**
* **Cómo se usa:**

*String sql = "{ CALL sp\_get\_pedido\_por\_numero(?) }";*

*try (CallableStatement cs = conn.prepareCall(sql)) {*

*cs.setString(1, "PED-TEST-001"); // pasar parámetro*

*try (ResultSet rs = cs.executeQuery()) {*

*while(rs.next()) {*

*System.out.println(rs.getString("numero"));*

*}*

*}*

*}*

* Protege contra inyecciones porque el parámetro no se concatena directamente en SQL.