Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

INSTITUTO TECNOLOGICO DE COSTA RICA

UNIDAD DE COMPUTACION

**SISTEMA DE GENERACION AUTOMÁTICA DE CRUDS**

INTEGRANTES

RANDY ESPINOZA DOWNING

JENNIFER LÓPEZ MIRANDA

SEDE SAN CARLOS

I SEMESTRE 2025

**INTRODUCCIÓN**

En la administración de bases de datos relacionales como PostgreSQL y SQL Server, los procedimientos almacenados son fundamentales para encapsular operaciones CRUD (Crear, Leer, Actualizar y Eliminar), mejorando la eficiencia y seguridad en la manipulación de datos. Sin embargo, la creación manual de estos procedimientos suele ser repetitiva y propensa a errores, especialmente en sistemas con múltiples tablas o estructuras complejas. Por lo que para abordar estas dificultades, han surgido herramientas que automatizan la generación de procedimientos almacenados. Por ejemplo, Appsmith permite crear interfaces CRUD completas a partir de bases de datos existentes con solo unos clics, reduciendo significativamente el esfuerzo manual requerido (Appsmith, 2023). Asimismo, otra empresa como Cloving CLI utiliza inteligencia artificial para generar procedimientos almacenados en SQL, optimizando el flujo de trabajo y mejorando la calidad del código, esto evidencia que el problema existe y la demanda por su solución es real (Cloving, 2025), podemos también observar el caso de Metatype (Metatype, n.d.) que es una herramienta que automatiza la creación de APIs CRUD con validación de datos para diversas bases de datos, incluyendo PostgreSQL y SQL Server, lo cual ayuda a inspirar grandemente el proyecto en mano, por lo que se pretende en el mismo tomar un camino similar, pero con otra perpectiva en cuanto a la solución de su problema.

Estas soluciones mencionadas con brevedad no solo aceleran el desarrollo, sino que también promueven la estandarización y reducen la posibilidad de errores humanos.

**ANÁLISIS DEL PROBLEMA**

En los entornos donde se utilizan bases de datos relacionales, como PostgreSQL y SQL Server, los procedimientos almacenados toman el papel de herramienta fundamental para encapsular operaciones CRUD (Medic, 2014) y garantizar consistencia, seguridad y eficiencia en la manipulación de datos. Sin embargo, la generación manual de estos procedimientos representa una actividad repetitiva, sujeta a errores humanos y con un alto costo en tiempo, especialmente cuando se trabaja con múltiples tablas o bases de datos extensas.

El problema principal identificado radica en la ausencia de una herramienta que permita automatizar de manera flexible, segura y eficiente la generación de estos procedimientos. Sin una solución a la vista, los administradores de bases de datos deberían de escribir cada procedimiento de forma manual, adaptándolo a la estructura particular de cada tabla, a los tipos de datos que contiene y a las políticas de acceso definidas por la organización. Esto no solo aumenta el riesgo de inconsistencias en la codificación, sino que también ralentiza procesos clave en el desarrollo y mantenimiento de sistemas (Axial SQL, 2010).

Se tiene entendido que para tratar con el problema, se debe proponer el desarrollo de una herramienta que permita:

1. Conectar a bases de datos PostgreSQL o SQL Server de manera segura.
2. Detectar las tablas disponibles en la base de datos conectada.
3. Evaluar los permisos de acceso para el usuario autenticado, a nivel de tabla y atributos.
4. Generar de manera automática procedimientos almacenados CRUD con base en la estructura de las tablas seleccionadas.
5. Ofrecer dos modos de operación: solo generación del código, o generación y ejecución directa en la base de datos.

La herramienta se apoyará en técnicas de análisis dinámico de metadatos, así como en estructuras de generación de código parametrizable que se ajusten a las necesidades del usuario por lo que se piensa que en teoría debería de ser totalmente funcional para los motores de bases de datos a tratar y a cualquier estructura de dichas bases de datos.

**SOLUCIÓN DEL PROBLEMA**

1. **Arquitectura de la solución**

**1.1 Enfoque General**

La solución implementa una arquitectura cliente-servidor que permite la generación automatizada de procedimientos almacenados a través de una interfaz web intuitiva. El sistema se divide en tres capas principales, cada una con responsabilidades específicas y bien definidas:

La arquitectura cliente-servidor fue elegida por su capacidad para separar claramente las responsabilidades y permitir una escalabilidad eficiente. El frontend maneja la interacción con el usuario y la presentación de datos, mientras que el backend se encarga de la lógica de negocio y la comunicacion con la base de datos, esta separación permite una mayor flexibilidad en el desarrollo así como el mantenimiento del sistema.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Figura [1]**

*Arquitectura cliente-servidor de la solución*. Elaboración propia.

* 1. **Componentes principales**

**Frontend**

* **Interfaz de Usuario**: Implementada en React que ofrece al usuario una experiencia simple e inuitiva que permite:
  + Conexión a bases de datos: Permite a los usuarios establecer conexiones seguras con diferentes tipos de bases de datos mediante un formulario de configuración intuitivo.
  + Visualización de tablas: Muestra la estructura y contenido de las tablas de manera organizada y fácil de entender.
  + Generación de procedimientos: Ofrece una interfaz amigable para seleccionar las tablas y operaciones para las que se generarán procedimientos.
  + Modo preview/ejecución: Permite a los usuarios verificar el código generado antes de su ejecución en la base de datos o ejecutarlo directamente.

**Backend**

* **API:** Hecha en Node.js, esta maneja lo siguiente:
  + Conexiones a bases de datos: Gestiona de forma segura las conexiones a diferentes motores de base de datos, implementando patrones de conexión y desconexión.
  + Operaciones CRUD: Proporciona endpoints para realizar operaciones básicas de creación, lectura, actualización y eliminación de datos.
  + Generación de procedimientos: Implementa la lógica para analizar la estructura de las tablas y generar código SQL optimizado.
  + Gestión de permisos: Controla el acceso a los recursos y operaciones basado en roles y permisos configurados.

**Base de Datos**

* **Motores Soportados(Por limites del proyecto)**:
  + PostgreSQL: Se utiliza para bases de datos relacionales con soporte avanzado para procedimientos almacenados y tipos de datos complejos.
  + SQL Server: Se integra con sistemas empresariales que utilizan la plataforma Microsoft.

**2. Etapas de la solución**

* 1. **Conexión y Autenticación**

El proceso de conexión y autenticación es necesario para poder hacer uso de la herramienta correctamente, también por temas de seguridad, esta fase se encargaría de la validación de las credenciales proporcionadas, de establecer conexiones seguras utilizando protocolos encriptados, del maneje de errores de conexión y finalmente de implementar timeouts y reconexiónes automáticas

A continuacion un snippet del código utilizado en el proyecto.

// Ejemplo de manejo de conexión  
const conectarBaseDatos = async (config) => {  
 try {  
 const { dbType, host, username, password, dbname } = config;  
 let connection;  
   
 if (dbType === 'SQL') {  
 connection = await connectSQL(config);  
 } else if (dbType === 'Postgre') {  
 connection = await connectPostgres(config);  
 }  
   
 return { success: true, connection };  
 } catch (error) {  
 return { success: false, error: error.message };  
 }  
};

* 1. **Detección de Estructura**

La detección de la estructura de las tablas es un proceso crítico que analiza por decirlo así, la “metadata” de las tablas o incluso de la base de datos lo que permite identificar todas las columnas y sus tipos de datos, también determinar las restricciones y valores por defecto, analizar las relaciones entre tablas y detectar claves primarias y foráneas, todo solo mediante su metadata.

Este proceso se realiza mediante consultas específicas para cada motor de base de datos que puede ver a continuacion:

-- PostgreSQL  
SELECT   
 column\_name,  
 data\_type,  
 is\_nullable,  
 column\_default  
FROM information\_schema.columns   
WHERE table\_name = :nombreTabla;  
  
-- SQL Server  
SELECT   
 c.name,  
 t.name,  
 c.is\_nullable,  
 dc.definition  
FROM sys.columns c  
JOIN sys.types t ON c.user\_type\_id = t.user\_type\_id  
LEFT JOIN sys.default\_constraints dc ON c.default\_object\_id = dc.object\_id  
WHERE OBJECT\_NAME(c.object\_id) = :nombreTabla;

**2.3 Generación de Procedimientos**

La generación de procedimientos seria el núcleo de la solución, mediante un proceso completo que analiza la estructura de la tabla, genera código SQL optimizado adaptado a los requerimientos específicos, implementa buenas prácticas de programación para garantizar mantenibilidad y rendimiento, e incluye procesos robustos de manejo de errores y validaciones que aseguran la integridad y consistencia de los datos.

A continuación, un snippet del código como ejemplo:

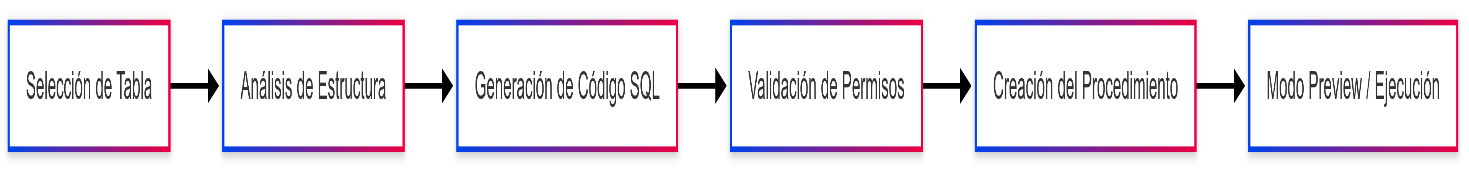
const generarProcedimiento = async (tabla, operacion, config) => {  
 // 1. Obtener estructura de la tabla  
 const estructura = await obtenerEstructuraTabla(tabla);  
   
 // 2. Generar código SQL según operación  
 const codigoSQL = generarCodigoSQL(estructura, operacion);  
   
 // 3. Crear procedimiento almacenado  
 const procedimiento = await crearProcedimiento(  
 config.schema,  
 config.prefijo,  
 operacion,  
 codigoSQL  
 );  
   
 return procedimiento;  
};

**3. Diagramas de Flujo**

**3.1 Idea general del funcionamiento de generación de procedimientos**

En cuanto al flujo de generación de procedimientos sigue un proceso secuencial que comienza con la selección de la tabla objetivo por parte del usuario, continúa con el análisis automático de la estructura de dicha tabla, prosigue con la generación de código SQL optimizado, verifica que el usuario cuente con los permisos necesarios para realizar las operaciones, crea el procedimiento dinámico según sea la acción a realizar(primero pregunta por el schema y prefijo) y finalmente ofrece un modo de preview/ejecución donde el usuario puede revisar el procedimiento generado antes de ejecutarlo o proceder directamente con su ejecución sin necesidad de ver el código a ejecutar, solo con tener la idea de que se hace.

En la siguiente figura, se puede observar proceso de manera simple:



**Figura [2]**  
*Flujo de generación de procedimientos.* Elaboración propia.

**3.2 Flujo de permisos**

Este sistema de permisos realizado implementa un modelo de seguridad basado en roles que autentica al usuario mediante credenciales verificables, comprueba sus permisos específicos dentro del sistema, determina el nivel de acceso apropiado según su perfil y responsabilidades, y controla los permisos tanto a nivel de tabla como de columna para garantizar que cada usuario solo pueda acceder y modificar la información para la cual está explícitamente autorizado.

En la siguiente figura se evidencia este flujo de mejor manera:

Diagrama

El contenido generado por IA puede ser incorrecto.

**Figura [3]**  
*Flujo de verificación de permisos en el sistema.* Elaboración propia.

**4. Ciertas consideraciones a tener en cuenta**

**4.1 Seguridad**

La seguridad es una prioridad, por lo menos en niveles básicos en el diseño del sistema, implementándose múltiples capas de protección que incluyen autenticación segura mediante protocolos y almacenamiento protegido de credenciales, validación de permisos con control granular de acceso tanto a nivel de tabla como de columna, así como manejo seguro de conexiones mediante encriptación y gestión adecuada de sesiones para garantizar la integridad y confidencialidad de todos los datos procesados.

**4.2 Flexibilidad**

El sistema está diseñado para ser altamente flexible y adaptable, ofreciendo una arquitectura modular que permite la integración sencilla de diferentes motores de base de datos, proporcionando opciones de configuración que pueden personalizarse según las necesidades específicas de cada implementación, brindando flexibilidad operativa mediante modos de preview o ejecución directa, y demostrando una notable capacidad para adaptarse a diversas estructuras de tablas y relaciones complejas.

**4.3 Eficiencia**

La eficiencia definitivamente es un aspecto clave en el diseño del sistema ya que no se sabe el tamaño de las bases de datos con las que el sistema eventualmente va a tratar , manifestándose a través de la generación dinámica que optimiza el código SQL producido para cada operación específica, la implementación inteligente de un sistema de caché que permite la reutilización de procedimientos frecuentemente utilizados, el análisis continuo y optimización de las consultas generadas para hacer uso correcto del rendimiento y la gestión eficiente de las conexiones y recurso del sistema que garantiza una funcionalidad ideal.

**5. Ejemplo de implementación**

A continuación, se demuestra detenidamente la plantilla base de los procedimientos almacenados dinámicamente generados para los CRUD’s, los cuales reciben del usuario el schema(public es el default) donde se guardan y un prefijo para su nombre, también se muestra que es lo que estos incluyen.

**5.1.1 Inserción**

Los procedimientos de inserción siguen un patrón estándar que incluye:

* Validación de parámetros
* Manejo de errores
* Optimización de la operación
* Retorno de resultados

-- Ejemplo de procedimiento de inserción  
CREATE OR REPLACE PROCEDURE public.tabla\_insert(  
 p\_columna1 tipo1,  
 p\_columna2 tipo2,  
 ...  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
 INSERT INTO tabla (columna1, columna2, ...)  
 VALUES (p\_columna1, p\_columna2, ...);  
END;  
$$;

**5.1.2 Consulta**

Importante mencionar que la consulta no es una acción aparte en este sistema, la consulta se da directamente al seleccionar la tabla a ver, donde aparecen todos los registros de la misma.

Esta contiene:

* Filtrado de datos
* Paginación
* Ordenamiento
* Retorno de resultados estructurados

-- Ejemplo de procedimiento de consulta  
CREATE OR REPLACE PROCEDURE public.tabla\_select(  
 p\_filtro VARCHAR DEFAULT NULL,  
 p\_pagina INT DEFAULT 1,  
 p\_tamano\_pagina INT DEFAULT 10,  
 p\_ordenar\_por VARCHAR DEFAULT 'id'  
)  
LANGUAGE plpgsql  
AS $$  
DECLARE  
 v\_offset INT;  
BEGIN  
 v\_offset := (p\_pagina - 1) \* p\_tamano\_pagina;  
   
 RETURN QUERY  
 SELECT \*  
 FROM tabla  
 WHERE (p\_filtro IS NULL OR columna LIKE '%' || p\_filtro || '%')  
 ORDER BY   
 CASE p\_ordenar\_por  
 WHEN 'columna1' THEN columna1  
 WHEN 'columna2' THEN columna2  
 ELSE id  
 END  
 LIMIT p\_tamano\_pagina  
 OFFSET v\_offset;  
END;  
$$;

**5.1.3 Actualización**

Los procedimientos de actualización aplican:

* Validación de existencia
* Actualización selectiva
* Manejo de versiones
* Registro de cambios

-- Ejemplo de procedimiento de actualización  
CREATE OR REPLACE PROCEDURE public.tabla\_update(  
 p\_id INT,  
 p\_columna1 tipo1 DEFAULT NULL,  
 p\_columna2 tipo2 DEFAULT NULL,  
 ...  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
 UPDATE tabla  
 SET   
 columna1 = COALESCE(p\_columna1, columna1),  
 columna2 = COALESCE(p\_columna2, columna2),  
 ...  
 WHERE id = p\_id;  
   
 IF NOT FOUND THEN  
 RAISE EXCEPTION 'Registro no encontrado';  
 END IF;  
END;  
$$;

**5.1.4 Eliminación**

Los procedimientos de eliminación realizan:

* Verificación de dependencias
* Eliminación lógica o física
* Registro de auditoría
* Manejo de transacciones

-- Ejemplo de procedimiento de eliminación  
CREATE OR REPLACE PROCEDURE public.tabla\_delete(  
 p\_id INT,  
 p\_eliminacion\_logica BOOLEAN DEFAULT TRUE  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
 IF p\_eliminacion\_logica THEN  
 UPDATE tabla  
 SET   
 eliminado = TRUE,  
 fecha\_eliminacion = CURRENT\_TIMESTAMP,  
 usuario\_eliminacion = CURRENT\_USER  
 WHERE id = p\_id;  
 ELSE  
 DELETE FROM tabla  
 WHERE id = p\_id;  
 END IF;  
   
 IF NOT FOUND THEN  
 RAISE EXCEPTION 'Registro no encontrado';  
 END IF;  
END;  
$$;

**ANALISIS DE RESULTADOS**

|  |  |  |
| --- | --- | --- |
| Requerimiento | Estado | Observaciones |
| Conexión a las bases de datos tipo PostgreSQL | 100% |  |
| Conexión a las bases de datos tipo SQL Server | 100% |  |
| Mostrar todas las tablas de la base de datos para selección | 100% |  |
| Mostrar nivel de seguridad para el usuario actual a nivel de tabla y atributo | 100% |  |
| Modo de generación de Código | 100% | Muestra el procedimiento almacenado generado y como se utilizaría en la base de datos |
| Modo de generación y ejecución de código | 100% | Lo ejecuta de inmediato y los cambios pueden verse directamente en el GUI sin necesidad de recargar la pagina. |
| Generación de código dinámico para los procedimientos CRUD | 100% |  |
| Interfaz intuitiva | 100% |  |

**CONCLUSIONES**

El sistema propuesto demostró ser eficiente y funcional para la generación automatizada de procedimientos almacenados, en comparación con el enfoque manual tradicional. La interfaz intuitiva permite invertir menos tiempo para producir código SQL funcional y bien estructurado, lo cual mejora la productividad y disminuye la posibilidad de errores humanos.

Por otro lado, la aplicación de una arquitectura cliente-servidor, utilizando React para el frontend, Node.js para el backend y PostgreSQL/SQL Server como motores de base de datos, resultó ser pertinente en escenarios donde se requiere flexibilidad, escalabilidad y facilidad de mantenimiento ya que el problema planteado puede ser perfectamente uno real por lo que las herramientas fueron las ideales.

Finalmente, se logró establecer un control efectivo sobre los permisos de usuario, asegurando que solo administradores puedan generar y aplicar procedimientos almacenados. Esto refuerza la seguridad del sistema y garantiza un manejo responsable de las bases de datos, lo cual es importante en entornos reales.

**RECOMENDACIONES**

En base a los resultados obtenidos se entienden ciertas direcciones que el proyecto podría tomar para una mejora incremental del mismo, con el objetivo de ser una herramienta un poco más competente y fiable en el contexto de la vida real, estas direcciones o ideas son las siguientes:

**Ampliación del alcance funcional del sistema**

Se recomienda extender la compatibilidad del sistema hacia otros motores de bases de datos ampliamente utilizados en entornos empresariales y académicos, tales como MySQL y Oracle. Esto permitiría que la herramienta sea adoptada en una mayor diversidad de contextos técnicos y organizacionales.

**Optimización del rendimiento del sistema**

La eficiencia del sistema podría beneficiarse de estrategias de optimización a nivel de consultas SQL generadas, especialmente en entornos con volúmenes masivos de datos.

**Fortalecimiento de la seguridad y el control de accesos**

Dado que el sistema interactúa directamente con bases de datos, es fundamental garantizar la integridad y confidencialidad de la información. Se recomienda la implementación de un módulo de auditoría que registre las acciones realizadas por cada usuario, permitiendo el rastreo de operaciones sensibles.

REFERENCIAS

Appsmith. (2023). *Generate and build a CRUD app from any database with one click*. <https://www.appsmith.com/blog/generate-a-crud-app-from-any-database-with-one-click>​

Axial SQL. (2010, June 20). *Automatización de la creación de procedimientos almacenados en SQL Server*. <https://axial-sql.com/es/automatizacion-de-la-creacion-de-procedimientos-almacenados-en-sql-server/>​

Beekeeper Studio. (2023). *PostgreSQL information\_schema guide for database management*. <https://www.beekeeperstudio.io/blog/postgresql-information-schema>​

Cloving. (2025). *Utilizing GPT to automate SQL stored procedure generation*. <https://cloving.ai/tutorials/utilizing-gpt-to-automate-sql-stored-procedure-generation>​

Dhull, P. (2025, January 24). *Implementing role-based access control (RBAC) in Node.js with PostgreSQL*. Medium. <https://medium.com/@er.pwndhull07/implementing-role-based-access-control-rbac-in-node-js-with-postgresql-c1073ba23ee2>​

Medic, M. (2014, April 7). *Creating and using CRUD stored procedures*. SQLShack. <https://www.sqlshack.com/creating-using-crud-stored-procedures/>​

Metatype. (n.d.). *Instant APIs on your database*. <https://metatype.dev/use-cases/automatic-crud-validation>​

Microsoft. (2023). *Step 3: Proof of concept connecting to SQL using Node.js*. Microsoft Learn. <https://learn.microsoft.com/en-us/sql/connect/node-js/step-3-proof-of-concept-connecting-to-sql-using-node-js?view=sql-server-ver16>​

Microsoft. (2023). *System information schema views (Transact-SQL)*. Microsoft Learn. <https://learn.microsoft.com/en-us/sql/relational-databases/system-information-schema-views/system-information-schema-views-transact-sql?view=sql-server-ver16>​

ScaleGrid. (2023). *How to connect PostgreSQL® database in Node.js*. <https://scalegrid.io/blog/how-to-connect-postgresql-database-in-node-js/>​

SystemWeakness. (2023). *Secure database connection in NodeJS application*. Medium. <https://systemweakness.com/secure-database-connection-in-nodejs-application-3c3979566741>