Analisis Perbandingan Aspek Keamanan Basis Data Menggunakan Schema dan Tanpa Schema Pada Postgresql.

Muhammad Rizqy Ath-Thaariq*[1], Amir Hamzah[2], and Suwanto Raharjo[3]

1-3 Universitas AKPRIND Indonesia Yogyakarta
Jl. Kalisahak No. 28, Kompleks Balapan, Yogyakarta 55222
mrizqy.thariq@gmail.com; amir@akprind.ac.id; wa2n@akprind.ac.id

Abstrak

Pengolahan data menjadi informasi pada saat ini sangatlah mudah dengan memperoleh data yang tersimpan pada sebuah sistem basis data, data yang tersimpan haruslah tersimpan dengan aman sehingga nantinya informasi yang dihasilkan haruslah informasi yang benar dan tidak memiliki kerancuan yang dapat mengakibatkan terjadinya kesalahan informasi. Pada PostgreSQL terdapat sebuah objek sistem basis data yang bernama schema yang pertama kali dikenalkan pada tahun 2002 silam yang memiliki fungsi seperti direktori atau folder pada sebuah sistem operasi yang mana schema ini tidak dapat saling membungkus satu sama lain. Terdapat perbedaan penerapan menggunakan schema dan tanpa schema, yang pada dasarnya tanpa schema pada PostgreSQL merujuk pada schema public. Hasil penelitian perbandingan penggunaan schema atau tidak dalam keamanan data menunjukkan bahwa penerapan schema dapat meningkatkan keamanan dibandingkan tidak menerapkan schema dikarenakan ketika Pengguna ingin mengakses tabel di dalam schema memiliki izin yang terbatas berdasarkan kriteria pemetaan hak akses sesuai kebutuhannya. Hal ini berbeda dengan pemberian hak akses tanpa schema yang mana schema public memiliki sifat publik sehingga dapat diakses oleh Pengguna siapapun, hal ini mempengaruhi berkurangnya keamanan terhadap data yang disimpan.

Kata Kunci keamanan basis data, hak akses, schema, postgresql

Digital Object Identifier 10.36802/jnanaloka.2024.v5-no2-51-61

1 Pendahuluan

Pengolahan data menjadi informasi pada saat ini sangatlah mudah dengan memperoleh data yang tersimpan pada sebuah sistem basis data, data yang tersimpan haruslah tersimpan dengan aman sehingga nantinya informasi yang dihasilkan haruslah informasi yang benar dan tidak memiliki kerancuan yang dapat mengakibatkan terjadinya kesalahan informasi [1–3]. Berbagai penelitian telah dilakukan untuk menawarkan teknik pengamanan data pada basis data, salah satunya penelitian yang dilakukan oleh Triyono (2023) mengenai penerapan hak akses basis data yang menyebutkan bahwa pengguna yang mengakses dapat dikelompokkan sesuai akun dan perannya, sehingga memungkinkan hanya akun tersebut yang mengakses data sesuai hak aksesnya [4].

Penelitian lain yang dilakukan untuk meningkatkan keamanan data juga dilakukan dengan menerapkan *Programming Language* SQL (Pl-SQL). Penggunaan Pl-SQL seperti *function, trigger, view* dan *store procedure* mempermudah dalam pengembangan aplikasi dan

* Corresponding author.

Jnanaloka Jurnal Open Access
Yayasan Lentera Dua Indonesia

semua kejadian pada *rule* bisnis telah diikutsertakan dalam proses sistem basis data [5, 6]. Arif dkk. (2019) dan Jaelani (2022) menyebutkan dalam penelitian mereka bahwa penerapan hak akses yang tepat dapat mengurangi kemungkinan terjadinya kerusakan integritas suatu data yang disebabkan oleh serangan hacker, serta jumlah pengguna dan hak akses yang diberikan dapat disesuaikan dengan kebutuhan pada suatu organisasi [7, 8].

PostgreSQL merupakan sistem basis yang bersifat sumber terbuka, dukungan komunitas yang besar, dukungan penuh untuk lintas platform dan dapat digunakan untuk standar SQL seperti complex queries, transactional integrity, multiversion concurrency control dan masih banyak lainnya [9,10]. Selain itu, PostgreSQL juga lebih efisien dan cepat dibandingkan dengan MySQL untuk pengelolaan data sebanyak 1.000 hingga 1.000.000 baris [11, 12]. Pengembang PostgreSQL selalu berupaya untuk memprioritaskan data yang berintegritas, sehingga apabila ada sesuatu fitur yang beresiko terhadapnya integritas suatu data, maka pengembang PostgreSQL tidak mengizinkannya masuk sampai fitur tersebut dapat berjalan dengan benar dan dapat menjaga integritasnya data tadi [13].

Pada PostgreSQL terdapat sebuah objek sistem basis data yang bernama schema yang pertama kali dikenalkan pada tahun 2002 silam yang memiliki fungsi seperti direktori atau folder pada sebuah sistem operasi yang mana schema ini tidak dapat saling membungkus satu sama lain. Schema memiliki kedudukan level di bawah basis data dan sebelum tabel, sehingga setiap schema dapat memiliki banyak tabel. Penerapan schema dapat menjadikan sistem basis data menjadi lebih terstruktur dan sistematis serta teratur untuk dikelola, tabel-tabel tidak bercampur karena telah dikelompokkan berdasarkan schema nya dan dapat digunakan banyak pengguna dalam satu basis data tanpa takut terganggu satu dengan yang lain [10]. Objek basis data schema ini dapat diatur izin hak aksesnya, sehingga tidak semua pengguna yang terdaftar pada sistem basis data dapat mengaksesnya [2, 7]. Hal ini dapat mencegah kegiatan-kegiatan yang dapat merusak dan mengurangi integritas pada suatu informasi. Penelitian ini bertujuan untuk menguji perbedaan penggunaan schema dan tanpa schema dalam pengamanan data pada PostgreSQL dengan studi kasus basis data untuk menyimpan data sistem informasi silsilah keluarga (SISKA).

2 Metodologi

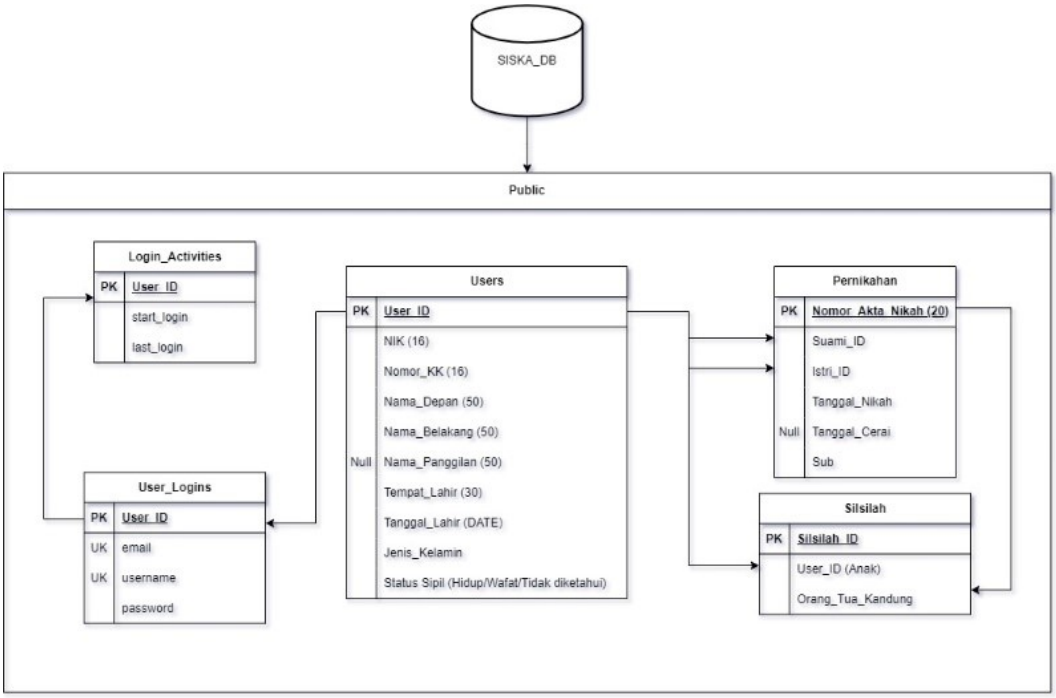
Pada penelitian ini digunakan laptop dengan merek dan tipe Acer Aspire 5 A515-56G dengan CPU Intel i5-gen 11 dan RAM 24 GB. Pengujian penelitian menggunakan sistem operasi Windows 11 dengan Windows Support Linux (WSL) Ubuntu 22.04.3 LTS, serta perangkat lunak sistem basis data relasional, PostgreSQL dengan versi 16.2.

Penelitian yang dilakukan oleh Jaelani (2022) menerangkan bahwa alur penelitian yang dilakukan untuk mengimplementasikan keamanan basis data terdapat tujuh tahapan yang dimulai dari: (1) Perencanaan berdasarkan studi literatur; (2) Perumusan dan pendefinisian masalah yang diteliti yang dilanjutkan dengan pengumpulan data; (3) Melakukan studi lapangan secara langsung terhadap masalah yang diteliti; (4) Design rancangan keamanan basis data; (5) Menerapkan konsep keamanan basis data; (6) Pengujian terhadap keamanan basis data; (7) Perumusan hasil akhir yang didapat [8].

Di sisi lain, Arif dkk (2019) dalam penelitiannya tentang implementasi keamanan basis data dengan PostgreSQL menggunakan metode eksperimen yakni dengan melakukan manipulasi satu atau lebih variabel dengan sengaja untuk dicari efeknya terhadap satu atau variabel lain yang diteliti. Eksperimen dilakukan dengan uji coba terhadap data yang dibuat dengan melakukan pembuatan basis data kemudian penerapan konsep keamanan basis datanya [7]. Alur penelitian yang dilakukan pada penelitian ini dilakukan dengan empat

tahapan: (1) Perancangan basis data yang meliputi perancangan schema dan tanpa schema, tabel, kolom dan pembagian hak akses; (2) Pembuatan basis data yang meliputi pembuatan schema dan tanpa schema, tabel, kolom dan pembuatan peran yang akan digunakan. (3) Pengisian data dummy untuk memudahkan dalam transaksi SQL dan memberikan hak akses schema, tabel dan kolom kepada peran yang telah dibuat. (4) Melakukan uji coba terhadap peran dengan hak akses yang diberikan untuk melakukan transaksi terhadap tabel yang menggunakan schema dan tanpa schema. (5) Perumusan hasil akhir yang diperoleh berdasarkan alur penelitian yang telah dirancang.

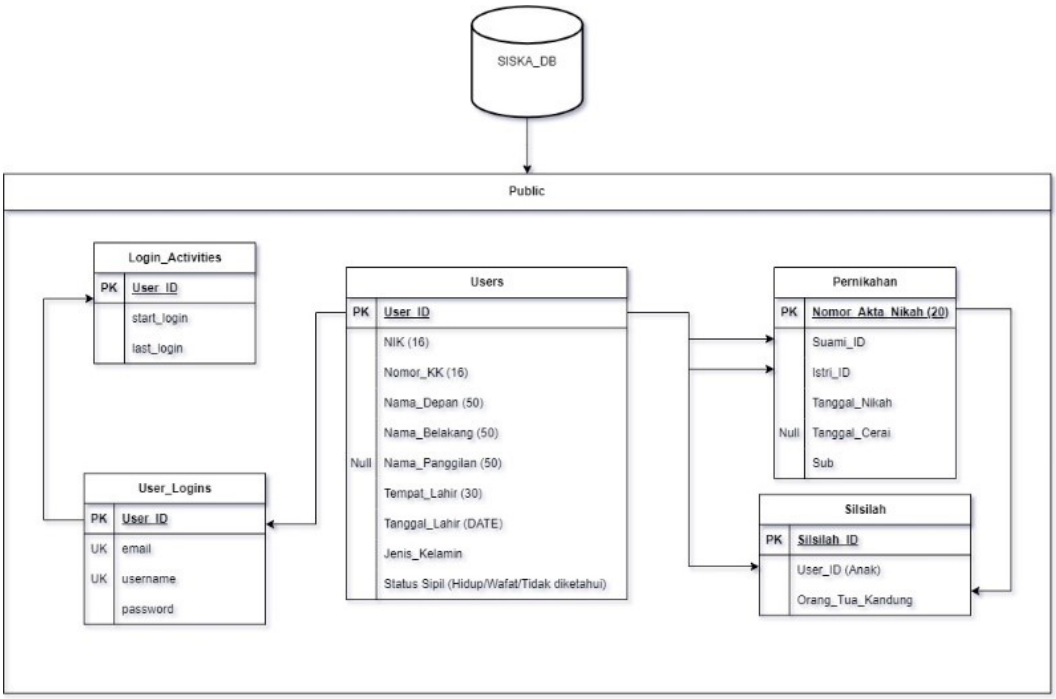
Pada saat pembuatan basis data menggunakan perintah CREATE DATABASE, secara bawaan PostgreSQL memberikan akses *schema public* kepada peran `PUBLIC`. Hal ini menjadi masalah tersendiri dikarenakan konfigurasi `SEARCH_PATH` di PostgreSQL secara bawaan memulai perintah-perintah dari *schema* `$user` ke schema *public* yang mana apabila tidak disebutkan secara spesifik nama schemanya [2]. Berbagai penelitian yang dilakukan tentang perancangan basis data menggunakan PostgreSQL menunjukkan bahwa perancangan tanpa *schema* dapat dilakukan dan rancangan tabel-tabel tadi disimpan di dalam schema *public* [12–14]. Rancangan basis data tanpa menyebutkan *schema* secara terperinci, alias *schema public* berdasarkan prototype Genealogy dirangkum menjadi seperti Gambar 6 [15].



Gambar 1 Desain rancangan basis data tanpa *schema*.

Di sisi lain, rancangan basis data menggunakan *schema* yang digunakan dalam penelitian tertampil dalam Gambar 2.

Basis data SISKA memiliki objek data schema yang berjumlah tiga, dan setiap schema memiliki tabel-tabel yang dapat berelasi antar schema seperti yang diilustrasikan pada Gambar 2. Penggunaan *schema* pada PostgreSQL ini membuat basis data SISKA dengan tabel-tabel memiliki level atau pemisah, sehingga basis data tidak membawahi secara langsung ke tabel melainkan melalui *schema*.



Gambar 2 Desain rancangan basis data dengan *schema*.

Pembagian tabel dan *schema* dalam penelitian ini tertampil dalam Tabel 1. Penerapan schema dapat mengelompokkan tabel-tabel sesuai fungsi dan tujuannya seperti yang terlihat pada Tabel 1. Pemisahan tabel user_logins dengan tabel users yang dikelompokkan pada dua schema yang berbeda. Pembagian ini ditujukan untuk dapat membantu tim *Back-End Engineer* dalam membuat autentikasi pada aplikasi sistem informasi. Dengan demikian, pada saat pengguna aplikasi melakukan login, pengecekan hanya berfokus pada tabel user_logins. Data-data sensitif yang ada pada tabel users seperti NIK, Nomor KK dan lain sebagainya tidak perlu tercampur dengan kegiatan autentikasi.

Tabel 1 Rancangan tabel dalam *schema*.

No.	Nama schema	Nama Tabel	Keterangan
1	`authentication_schema`	login_ativities, user_logins	sebagai penyimpanan informasi otentikasi pada sistem aplikasi
2	`person_schema`	users	Menyimpan data sensitif seperti data personal seseorang
3	`relation_schema`	silsilah, pernikahan	Menyimpan informasi dan transaksi silsilah dan keluarga

Perancangan hak akses pada penelitian ini mengacu penelitian Joko Triyono [4, 6] yang menggunakan Programming Language SQL (PLSQL) berupa *trigger*, *view*, *function* dan *store procedure*. Pemberian hak akses berdampak besar terhadap transaksi yang terjadi pada tabel seperti adanya perubahan tabel dapat berpengaruh dengan tabel lain, atau bahkan

tabel itu sendiri. Pembagian hak akses pada Tabel 2 menunjukkan bahwa peran yang dibuat dan diberikan hak akses dapat memiliki wilayah batasannya masing-masing. Peran yang satu dengan yang lainnya dapat sangat terbatas, sebagai contoh peran user_login hanya berfokus untuk melakukan `SELECT` pada tabel user_logins. Peran ini tidak dapat melakukan `INSERT`, `UPDATE` dan `DELETE` pada tabel user_logins. Jika dibutuhkan untuk mengubah data pada tabel maka pengguna harus berganti perannya menjadi person_update yang memiliki akses untuk mengubah data.

Tabel 2 Hak akses terhadap *schema*.

No.	Nama Peran	Nama Tabel	Hak Akses
1	`user_register`	users user_logins login_activities	`SELECT`, `INSERT`
2	`user_login`	user_logins login_activities	`SELECT` `SELECT`, `UPDATE`(last login)
3	`person_update`	user_logins users	`SELECT` `UPDATE` (mail, username, password) `SELECT`, `UPDATE` (semua, kecuali: user_id)
4	`relation_activity`	pernikahan silsilah	`SELECT`, `INSERT`, `UPDATE`, `DELETE` `SELECT`, `INSERT`, `DELETE`, `UPDATE`(semua, kecuali: user_id)
5	`keluarga_update`	users pernikahan silsilah	`SELECT`, `UPDATE`(semua, kecuali: user_id) `SELECT`, `UPDATE` `SELECT`, `UPDATE` (semua, kecuali: silsilah_id)

3 Hasil dan pembahasan

Penerapan schema dan pemberian hak akses pada SISKA dilakukan dengan menjalankan query pada terminal postgres, seperti tertampil dalam Listing 1 dan 2.

```
CREATE SCHEMA authentication_schema;
CREATE SCHEMA person_schema;
CREATE SCHEMA relation_schema;
```

Listing 1 Pembuatan *schema*

```
CREATE TABLE person_schema.users (
    user_id CHAR(5) PRIMARY KEY,
    ...
);

CREATE TABLE authentication_schema.user_logins (
    user_id CHAR(5) PRIMARY KEY ,
    ...
    FOREIGN KEY (user_id) REFERENCES person_schema.users(user_id)
);
```

Listing 2 pembuatan tabel di dalam *schema*.

Perintah yang ditampilkan pada Listing 2, menyebutkan nama_schema.nama_tabel sesudah perintah `CREATE TABLE`. Pemanggilan nama_schema.nama_tabel selalu dilakukan

setiap kali operasi dilakukan pada suatu tabel yang bertujuan agar pemetaan tabel dapat terstruktur dan rapi.

Peran yang mengakses terhadap basis data dibuat dengan perintah yang tertampil dalam Listing 3. Pembuatan tabel tanpa *schema* di PostgreSQL merujuk pada *schema* yang bernama *public*. Pembuatan tabel pada *schema public* ini dapat ditulis dengan public.nama_tabel ataupun langsung menuliskan nama tabelnya [10]. Perintah untuk membuat tabel tanpa *schema* (dengan *schema public*) tertampil pada Listing 4.

```
CREATE USER user_register WITH ENCRYPTED PASSWORD '123';
CREATE USER user_login WITH ENCRYPTED PASSWORD '123';
CREATE USER person_update WITH ENCRYPTED PASSWORD '123';
CREATE USER relation_activity WITH ENCRYPTED PASSWORD '123';
CREATE USER keluarga_update WITH ENCRYPTED PASSWORD '123';
```

Listing 3 Pembuatan peran yang mengakses basis data.

```
CREATE TABLE public.users (
    user_id CHAR(5) PRIMARY KEY,
    ...
);

CREATE TABLE user_logins (
    user_id CHAR(5) PRIMARY KEY ,
    ...
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

Listing 4 Pembuatan tabel tanpa schema.

Pada Listing 4, terdapat dua penyebutan public dan tidak menggunakan public. PostgreSQL secara bawaan menganggap penyebutan tanpa public langsung mengarah pada schema public [2, 10]. Penelitian yang dilakukan oleh Firdaus dkk. (2020) menunjukkan bahwa pemanggilan Schema public haruslah dilakukan setiap kali melakukan transaksi pada PostgreSQL ataupun perangkat lunak PgAdmin [16]. Di sisi lain, penulisan public pada query di atas dilakukan untuk menegaskan bahwa tabel Users berada di dalam Schema public. Penulisan public juga dilakukan untuk menghindari tabel yang kita definisikan bertabrakan dengan function atau operator yang terdaftar pada PostgreSQL, sebagai contoh pada PostgreSQL terdapat function bernama user yang mengarah pada peran yang memakai basis data saat ini, dapat mengakibatkan kesalahan informasi dalam mengaksesnya seperti yang ditampilkan pada hasil Gambar 3.

Pada hasil Gambar 3 menunjukkan bahwa terdapat tabel bernama User yang terdaftar pada Schema public, kesalahan informasi terjadi pada saat menjalankan *query* `SELECT` pada tabel User tanpa menyebutkan *schema public*, hasil yang didapatkan menampilkan nama peran saat ini yang mengakses basis data.

Penerapan pembatasan hak akses peran pada *schema* dilakukan dengan perintah yang tertampil di Listing 5.

```
GRANT USAGE ON SCHEMA authentication_schema TO user_register;
GRANT USAGE ON SCHEMA person_schema TO user_register;
GRANT USAGE ON SCHEMA relation_schema TO relation_activity;
```

Listing 5 Pemberian penggunaan terhadap schema kepada peran.

```
siska_db=⇒ SELECT * FROM user;
     user
--------------
 user_register
(1 row)

siska_db=⇒ \dt user;
        List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+----------
 public | user | table | postgres
(1 row)
```

Gambar 3 Hasil pemanggilan yang salah terhadap tabel user

Perintah pada Listing 5 menunjukkan bahwa PostgreSQL memberikan akses penggunaan (`USAGE`) kepada peran terhadap suatu schema. Pemberian `USAGE` terhadap schema haruslah dilakukan, dikarenakan tabel yang dibuat pada Gambar 3 berinduk pada schema yang telah ditentukan di Tabel 2. Penulisan query ini dibutuhkan apabila pengguna memiliki sebuah transaksi pada schema, apabila tidak menerapkan perintah `GRANT USAGE ON SCHEMA`, pesan gagal yang ditampilkan pada Gambar 4.

```
siska_db=⇒ SELECT * FROM authentication_schema.user_logins;
ERROR:  permission denied for schema authentication_schema
LINE 1: SELECT * FROM authentication_schema.user_logins;
                      ^
```

Gambar 4 Hasil kegagalan saat mengakses schema.

Berbeda dengan schema public, pemberian USAGE tidak diperlukan karena schema public itu sudah bersifat publik yang mana dapat digunakan oleh pengguna siapa saja. Peningkatan keamanan pada schema public lebih berfokus pada pengamanan tabel dan pembatasan tabel perlu ditentukan dengan baik terhadap tabel atau kolom tertentu mana saja yang dapat diakses peran. Pembatasan hak akses peran pada tabel di dalam *schema* dilakukan dengan perintah di Listing 6.

```
GRANT SELECT, INSERT
ON ALL TABLES IN SCHEMA "public"
TO user_register;
GRANT SELECT, INSERT, DELETE
ON ALL TABLES IN SCHEMA relation_schema
TO relation_activity;
```

Listing 6 Pembatasan hak akses tabel di dalam schema kepada peran.

Perintah pada Listing 6, menunjukkan bahwa pembatasan hak akses pengguna pada schema dilakukan dengan memberikan izin hak akses kepada semua tabel yang ada di dalam schema. peran `user_register` tidak dapat melakukan perubahan dan penghapusan data pada semua tabel yang ada di dalam schema `authentication_schema`, sesuai dengan ketentuan pada Tabel 2. Hal ini berlaku pada pengguna `relation_activity` yang mana tidak memiliki akses untuk ubah dan hapus data pada tabel-tabel yang ada di dalam

schema `relation_schema`. Di sisi lain, penerapan hak akses tanpa schema dilakukan dengan menyebutkan tabel-tabel yang hendak diberikan akses, seperti yang ditampilkan pada Listing 7.

```
GRANT SELECT, INSERT
    ON TABLE users, user_logins, public.login_activites
    TO user_register;
```

Listing 7 Pemberian hak akses tabel tanpa schema

Pada perintah di atas, dilakukan penyebutan tabel-tabel secara spesifik yang hendak diberikan izin. Hal ini dapat mengakibatkan tidak efisiensi dan tidak efektif dalam pengembangan basis data. Berbeda dengan perintah penerapan pembatasan hak akses pengguna pada tabel di dalam schema yang telah menempatkan tabel-tabel berdasarkan tujuannya ke dalam schema, sehingga pengembang dapat dengan mudah menuliskan perintah `ON ALL TABLES IN SCHEMA nama_schema` tanpa perlu khawatir tabel lain yang terdampak sama dengan tabel-tabel spesifik yang hendak diberikan izin. Selain pemberian hak akses terhadap schema dan tabel, pemberian hak akses dapat diperinci bahwa peran hanya dapat melakukan transaksi UPDATE pada kolom-kolom tertentu, seperti yang ditampilkan pada Listing 8.

```
GRANT UPDATE(anak_id, orang_tua_kandung)
ON TABLE relation_schema.silsilah
TO relation_activity;
```

Listing 8 Pemberian hak akses kolom tertentu kepada peran.

Pesan gagal ditampilkan ketika peran tidak diberikan akses terhadap tabel di dalam schema dan melakukan transaksi `UPDATE` diluar kolom yang diizinkan, seperti yang ditampilkan pada Gambar 5. Pengujian berikutnya dilakukan dengan melakukan login ke basis data

```
siska_db=> SELECT silsilah_id, nama_depan AS anak
FROM relation_schema.silsilah
JOIN person_schema.users
ON silsilah.anak_id = users.user_id;
          silsilah_id           |  anak
--------------------------------+--------
 SIL01                          | Tukino
(1 row)

siska_db=> UPDATE relation_schema.silsilah
siska_db-> SET silsilah_id = 'FAM01'
siska_db-> WHERE silsilah_id = 'SIL01';
ERROR:  permission denied for table silsilah
```

Gambar 5 Melakukan UPDATE pada kolom yang tidak diizinkan kepada peran

melalui terminal menggunakan pengguna `user_register`. Kemudian dilakukan pengujian untuk melakukan transaksi `INSERT` dan `SELECT` pada tabel Users dan didapatkan hasil yang ditampilkan pada Gambar **??**. Pada hasil yang ditampilkan Gambar **??**, didapatkan bahwa TRIGGER telah berjalan dikarenakan terdapat transaksi INSERT pada tabel Users, sehingga secara automatis `user_id` pada tabel Users terisikan pada tabel `user_logins`. Hal ini dapat membantu memudahkan dan mengamankan dalam pengembangan sistem informasi dikarenakan tidak perlu menuliskan logika pemrograman untuk menyimpan data users pada `user_logins`.

```
postgres@kang:~$ psql -h localhost -U user_register -d siska_db -p 5432 -W
Password:
psql (16.2 (Ubuntu 16.2-1.pgdg22.04+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

siska_db=> INSERT INTO person_schema.users (user_id, nama_depan, jenis_kelamin)
siska_db-> VALUES ('17', 'Azzam', 'L');
INSERT 0 1
siska_db=> SELECT user_id, nama_depan, jenis_kelamin
siska_db-> FROM person_schema.users WHERE user_id = '17';
 user_id | nama_depan | jenis_kelamin
---------+------------+---------------
 17      | Azzam      | L
(1 row)

siska_db=> SELECT * FROM authentication_schema.user_logins WHERE user_id = '17';
 user_id | email | username | password
---------+-------+----------+----------
 17      |       |          |
(1 row)
```

Gambar 6 Proses login dan melakukan transaksi menggunakan peran `user_register`

Pengujian berikutnya adalah pembatasan hak akses terhadap kolom. Pada dokumentasi PostgreSQL menyebutkan bahwa keamanan tingkat kolom (*Column-level security*) dapat diimplementasikan dengan memberikan hak akses GRANT ke peran dengan menyebutkan nama kolom yang diberikan izin [10]. Terdapat artikel web mengenai *Column-level security* yang menerangkan bahwa keamanan tingkat kolom mengizinkan peran untuk melihat kolom tertentu sehingga membuat kolom lain menjadi privat dengan memblokir akses peran ke kolom yang tidak ditentukan, pembatasan hak akses terhadap kolom dapat dilakukan dengan tiga cara: (1) Menggunakan view; (2) Membuat izin akses peran terhadap kolom; (3) Mengenkripsi kolom [17]. Pada penelitian ini dilakukan pembatasan hak akses terhadap kolom dengan melakukan pengujian transaksi `UPDATE` dan `DELETE` menggunakan pengguna `user_register` pada tabel Users dan tabel yang terdapat di dalam schema `authentication_schema`, dan didapatkan hasil yang ditampilkan pada Gambar 7. Pembatasan hak akses pada `user_register` seperti hasil Gambar 7 menunjukkan bahwa

```
siska_db=> DELETE FROM person_schema.users WHERE user_id = '15';
ERROR:  permission denied for table users
siska_db=> UPDATE person_schema.users
siska_db-> SET nama_depan = 'Muhammad Azzam'
siska_db-> WHERE user_id = '17';
ERROR:  permission denied for table users
siska_db=> UPDATE authentication_schema.user_logins
siska_db-> SET email = 'dummy@siska.db'
siska_db-> WHERE user_id = '17';
ERROR:  permission denied for table user_logins
siska_db=> DELETE FROM authentication_schema.user_logins
siska_db-> WHERE user_id = '17';
ERROR:  permission denied for table user_logins
```

Gambar 7 Pesan gagal melakukan transaksi diluar hak akses yang diberikan.

`user_register` tertolak untuk melakukan transaksi `UPDATE` dan `DELETE`, dan telah memenuhi kriteria yang ditentukan.

Dari empat pembahasan di atas, dihimpun bahwa perbanding penggunaan schema dan tanpa schema ke dalam Tabel 3. Pemberian hak akses terhadap schema, tabel dan kolom tertentu kepada peran dapat meningkatkan keamanan untuk menghindari terjadinya kegiatan yang tidak diinginkan. Penerapan keamanan ini dilakukan dengan pemetaan dan pendefinisian Hak Akses serta peran yang baik dan matang.

Tabel 3 Hasil dan pembahasan terhadap perbandingan penggunaan schema dan tanpa schema.

No.	Tanpa Schema	Menggunakan Schema
Dominasi	2	
Stabilitas	8	
Kualitas	15	
Kecerahan	4	
Total data	29	

4 Kesimpulan dan saran

Dari hasil rancangan, penerapan dan pengujian pada basis data sistem informasi silsilah keluarga diperoleh hasil bahwa penerapan schema dapat meningkatkan keamanan dibandingkan tidak menerapkan schema dikarenakan ketika peran ingin mengakses tabel di dalam schema memiliki izin yang terbatas berdasarkan kriteria pemetaan hak akses sesuai kebutuhannya. Hal ini berbeda dengan pemberian hak akses tanpa schema yang mana schema public memiliki sifat publik sehingga dapat diakses oleh peran siapapun, hal ini dapat mempengaruhi berkurangnya keamanan terhadap data yang disimpan.

Penerapan hak akses peran terhadap schema, tabel dan kolom tertentu yang tepat selain diterapkan pada basis data juga dapat meningkatkan keamanan di sisi Back-End dengan melakukan login terhadap pengguna tertentu berdasarkan service atau transaksi yang diperlukan, sehingga kecil kemungkinan terjadinya klien menggunakan akun superuser seperti root atau postgres dalam melakukan transaksi. Hal ini dapat menghindari resiko terjadinya serangan sql injection yang mana hacker tidak dapat mengeksploitasi transaksi basis data secara bebas karena terkendala hak akses.

Di sisi lain, hasil dari penerapan keamanan hak akses dan schema selain menjaga keamanan tabel dan kolom tertentu, juga dapat ditingkatkan dengan melakukan keamanan pada tingkat baris (Row-level security) sehingga peran hanya dapat melakukan transaksi pada baris tertentu untuk menjamin keabsahan dan integritas informasi yang disajikan.

Berdasarkan kesimpulan di atas, diperlukan adanya penelitian berikutnya untuk menguji praktik penerapan keamanan hak akses penggunaan schema pada sisi Back-end dalam menghadapi serangan sql injection. Pengamanan basis data selain menerapkan pada schema, tabel dan kolom, dapat memungkinkan adanya penelitian lebih lanjut mengenai kemanan pada Row-level security sehingga data yang tersimpan telah terkuci dengan aman dari tingkat schema, tabel, kolom dan barisnya berdasarkan hak aksesnya.

Pustaka

1 E. Sutanta, *Basis data dalam tinjauan konseptual*. Yogyakarta: Andi, 2011.

2 S. Raharjo dan E. Utami, *Keamanan basis data relasional*. Penerbit Andi, 2022.

3 G. Susilo, "Keamanan basis data pada sistem informasi di era global," *Transformasi*, vol. 12, no. 2, 2017.

4 T. Joko, "Penerapan hak akses pada perancangan database akademik untuk meningkatkan keamanan data," *Indonesian Journal of Machine Learning and Computer Science*, vol. 3, no. 1, pp. 50–59, 2023.

5 M. N. Fauzy, R. Nuari, Y. D. Pambudi, H. Nurmawan, M. F. Noor, dan Y. F. Andriani, "Keamanan basis data pada validasi data sistem informasi kepakaran," *Jurnal Informa: Jurnal Penelitian dan Pengabdian Masyarakat*, vol. 4, no. 3, pp. 32–41, 2018.

6 J. Triyono dan P. Akbar, "Implementasi rancangan database akademik menggunakan function, store procedure, trigger dan view," *FAHMA*, vol. 21, no. 1, pp. 45–59, 2023.

7 A. Y. Arif, E. Utami, dan S. Raharjo, "Implementasi database security menggunakan konsep role-based access control (rbac) dalam rancangan database sistem informasi manajemen sekolah dengan postgresql," *Informasi Interaktif*, vol. 4, no. 1, pp. 51–55, 2019.

8 W. L. Jaelani, "Implementasi model role base access control dan teknik replikasi basis data untuk keamanan basis data smk maarif terpadu cicalengka," *Naratif: Jurnal Nasional Riset, Aplikasi dan Teknik Informatika*, vol. 4, no. 2, pp. 147–154, 2022.

9 D. A. B. Prasetyo, "Implementasi information schema database pada postgresql untuk pembuatan tabel informasi dengan menggunakan python di pt xyz," *JATISI (Jurnal Teknik Informatika dan Sistem Informasi)*, vol. 9, no. 3, pp. 1961–1972, 2022.

10 T. P. G. D. Group, "Postgresql 16.2 documentation," 2024, online, 2024. [Online]. Available: https://www.postgresql.org/docs/current/index.html

11 R. Wodyk dan M. Skublewska-Paszkowska, "Performance comparison of relational databases sql server, mysql and postgresql using a web application and the laravel framework," *Journal of Computer Sciences Institute*, vol. 17, pp. 358–364, 2020.

12 A. D. Praba dan M. Safitri, "Studi perbandingan performansi antara mysql dan postgresql," *jurnal khatulistiwa informatika*, vol. 8, no. 2, 2020.

13 R. Stones dan N. Matthew, *Beginning databases with postgreSQL: From novice to professional*. Apress, 2005.

14 M. Barokah dan S. Sauda, "Penerapan nodejs dan postgresql sebagai backend pada aplikasi ecommerce localla," *INFOTECH journal*, vol. 8, no. 2, pp. 101–105, 2022.

15 M. Geurts, "Genealogy," 2024, online, 2024. [Online]. Available: https://github.com/MGeurts/genealogy

16 S. S. Firdaus, Y. Budisusanto, dan U. W. Deviantari, "Pembuatan basis data pajak bumi dan bangunan (pbb)(studi kasus: Desa bener, kecamatan saradan, kabupaten madiun)," *Jurnal Teknik ITS*, vol. 9, no. 2, pp. A182–A188, 2021.

17 A. Raghuwanshi, "How to implement column and row level security in postgresql," 2024, online, 2024. [Online]. Available: https://www.enterprisedb.com/postgres-tutorials/how-implement-column-and-row-level-security-postgresql