Audit Logging in YugabyteDB

Valerie Parham-Thompson Friday, December 2, 2022





YugabyteDB security features

YugabyteDB offers several security features:

- Identity and access management
- Role-based access control
- Database access logging
- Audit logging



Purpose of logging

Logging can tell you the "who, what, when, where" of actions on your systems.

On a database system, advanced logging allows you to capture this information for any changes to or access to the data.



First, configure log_line_prefix

The ysql_pg_conf_csv gflag is used to configure a comma-separated list of PostgreSQL server parameters that is appended to the postgresql.conf file. (Default is %m [%p])

This is how you collect a record of "who, what, when, and where."

Example:

```
--ysql_pg_conf_csv="log_line_prefix
='%m [%p %l %c] %q[%C %R %Z %H] [%r
%a %u %d] '"
```

%m timestamp with milliseconds %p process ID (PID) %q stops the line entry for system processes : YuqabyteDB adds %H current hostname the highlighted %C cloud name ones %R region / data center name %Z availability zone / rack name %r client IP address %a application name %c session identifier % line number within the session %u database user %d database name

Next, pgaudit extension: improvement over basic logging

Standard logging via log_statement provides the statements sent to the database.

The open source pgaudit extension improves the information collected by recording what was *executed* on the database.

Additional pieces of information are stored in the logs for each statement to facilitate log analysis.

pgaudit provides a more complete "what."



Remember community Postgres tools work for YugabyteDB.

Configuration



Configuration

Create the extension.

```
yugabyte=# CREATE EXTENSION IF NOT EXISTS pgaudit;

CREATE EXTENSION

yugabyte=# \dx

List of installed extensions

Name | Version | Schema | Description

pg_stat_statements | 1.6 | pg_catalog | track execution statistics of all SQL statements executed

pgaudit | 1.3.2 | public | provides auditing functionality

plpgsql | 1.0 | pg_catalog | PL/pgSQL procedural language
```



The necessary

Recommended pgaudit configuration

```
pgaudit.log = 'all, -misc'
set pgaudit.log_parameter=on;
set pgaudit.log_relation=on;
set pgaudit.log_catalog=off;
```

pgaudit.log records the statement type: READ, WRITE, FUNCTION, ROLE, DDL, MISC, MISC_SET, or ALL. If using ALL, can exclude with -.

pgaudit.log_parameter records any parameters sent with prepared statements.

pgaudit.log_relation records each table in a multi-join query.

pgaudit.log_catalog can be turned off toavoid noise from system catalog access.

Several more are available. These are key settings.



Object-level logging

Why?

- Reduce load
- Reduce size of log files

Overhead of full session audit logging is about 5% - typical for logging/monitoring tools.

Use pgaudit.role to define roles used for logging.

Analyzing Log Output



Reading pgaudit output

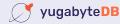
2022-11-28 16:02:29.491 UTC [30832 13 6384d90c.7870] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(56986) ysqlsh yugabyte yugabyte] LOG: AUDIT:

SESSION, 3, 1, READ, SELECT, TABLE, public.milliontable, select * from limit 10;, <none>

Notice the fully qualified schema.table name.

- AUDIT_TYPE SESSION or OBJECT.
- STATEMENT_ID Unique statement ID for this session. Each statement ID represents a backend call. Statement IDs are sequential even if some statements are not logged. There may be multiple entries for a statement ID when more than one relation is logged.
- SUBSTATEMENT_ID Sequential ID for each sub-statement
 within the main statement (e.g., calling a function from a
 query). Sub-statement IDs are continuous even if some
 sub-statements are not logged. There may be multiple entries
 for a sub-statement ID when more than one relation is logged.
- CLASS e.g. READ, ROLE (see pgaudit.log).

- **COMMAND** e.g., DDL, SELECT.
- OBJECT_TYPE (SELECT, DML, DDL) TABLE, INDEX, VIEW, etc.
- OBJECT_NAME (SELECT, DML, DDL) The fully-qualified object name (e.g., public.account).
- STATEMENT Statement executed on the backend.
- PARAMETER If pgaudit.log_parameter is set, then this field will contain the statement parameters as quoted CSV or <none> if there are no parameters. Otherwise, the output is <not logged>.



Using audit logging output

- Save to cloud storage for future audit needs
- Log analysis tools: splunk, etc.
- Alerts (e.g., on DDL)
- Manual filtering

Demo (Session Level)



Create table

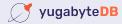
```
create table milliontable(name varchar(10), age integer,
joindate date);
```

```
2022-11-28 16:49:05.691 UTC [30832 21 6384d90c.7870] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(56986) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 6, 1, DDL, CREATE TABLE, TABLE, public.milliontable, "create table milliontable (name varchar(10), age integer, joindate date); ", <none>
```

If you try to create a table that already exists, you will see a log entry in the standard log, but not the audit log. This illustrates the difference between what the user <u>sent</u> and what was actually <u>executed</u>.

Create user with grants

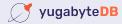
```
CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
GRANT CONNECT ON DATABASE yugabyte TO user1;
GRANT USAGE ON SCHEMA public TO user1;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO user1;
2022-11-28 16:56:22.514 UTC [30832 26 6384d90c.7870] [gcp us-east1 us-east1-c vb-demo-parham-audit8-n1]
[10.204.0.60(56986) ysqlsh yuqabyte yuqabyte] LOG: AUDIT: SESSION, 7, ROLE, CREATE ROLE, , , CREATE ROLE user1
WITH LOGIN PASSWORD <REDACTED>, <none>
2022-11-28 16:56:28.917 UTC [30832 28 6384d90c.7870] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(56986) ysqlsh yuqabyte yuqabyte] LOG: AUDIT: SESSION, 8, ROLE, GRANT, , , GRANT CONNECT ON
DATABASE vugabyte TO user1;, <none>
2022-11-28 16:56:33.488 UTC [30832 30 6384d90c.7870] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(56986) ysqlsh yuqabyte yuqabyte] LOG: AUDIT: SESSION, 9, ROLE, GRANT, SCHEMA, GRANT USAGE ON
SCHEMA public TO user1;, <none>
2022-11-28 16:56:37.286 UTC [30832 32 6384d90c.7870] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(56986) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 10, ROLE, GRANT, TABLE, GRANT ALL
PRIVILEGES ON ALL TABLES IN SCHEMA public TO user1;, <none>
```



Change user password

```
ALTER ROLE user1 PASSWORD 'password2';
```

```
2022-11-28 19:10:57.875 UTC [7198 16 6384f6d9.1cle] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(60012) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 4, ROLE, ALTER ROLE, , , ALTER ROLE user1 PASSWORD PASSWORD
```



Simple insert

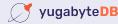
```
INSERT INTO milliontable (name, age, joindate) SELECT substr(md5(random()::text), 1, 10), (random() * 70 +
10)::integer, DATE '2018-01-01' + (random() * 700)::integer FROM generate_series(1, 1000);

2022-11-28 19:14:37.823 UTC [31980 8 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT:
SESSION,1,1,WRITE,INSERT,TABLE,public.milliontable "INSERT INTO milliontable (name, age, joindate) SELECT substr(md5(random()::text), 1, 10), (random() * 70 + 10)::integer, DATE '2018-01-01' + (random() * 700)::integer FROM generate series(1, 1000);",<none>
```

Notice the audit log doesn't record all of the values inserted in this case. Different from CDC.

Simple read

```
select * from milliontable limit 5;
     name | age | joindate
41ab60e791 | 38 | 2019-02-21
166bb6b35c | 65 | 2019-07-28
 75b3397992 | 13 | 2019-10-19
 692f1fc721 | 70 | 2019-07-22
 8cf51def6e | 70 | 2018-09-22
(5 rows)
2022-11-28 19:16:11.033 UTC [31980 12 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(33654) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 3, 1, READ, SELECT, TABLE, public.milliontable select * from milliontable limit 5;, <none>
```



Create table as select

create table milliontable_detail as select * from milliontable;

The audit logs are much more granular than the standard logs. You can easily parse out the individual read and change statements.

LOG

```
2022-11-28 19:17:58.922 UTC [31980 13 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: statement: create table milliontable_detail as select * from milliontable;
```

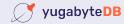
AUDIT

```
2022-11-28 19:17:58.929 UTC [31980 14 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 4, 1, READ, SELECT, TABLE, public.milliontable, create table milliontable_detail as select * from milliontable;, <none>

2022-11-28 19:17:59.769 UTC [31980 15 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 4, 1, WRITE, INSERT, TABLE, public.milliontable_detail, create table milliontable_detail as select * from milliontable;, <none>

2022-11-28 19:18:53.007 UTC [31980 16 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 4, 2, DDL, CREATE TABLE

AS, TABLE, public.milliontable_detail, create table milliontable_detail as select * from milliontable;, <none>
```



Add indexes

Notice when running this index creation that the audit log entry is not created until the index has completed backfilling.

```
create index nameage_idx1 on milliontable(name,age);

create index namepop_idx on milliontable_detail(name,population);

2022-11-28 20:25:15.761 UTC [31980 94 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION,22, DpL,CREATE
INDEX,INDEX,public.nameage_idx" create index nameage_idx on milliontable(name,age);",<none>

2022-11-28 20:26:13.339 UTC [31980 100 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION,23, DpL,CREATE
INDEX,INDEX,public.namepop_idx" create index namepop_idx on milliontable_detail(name,population);",<none>
```

Read with joins

You get two distinct log entries, one for each table that is read.

select milliontable_detail.population from milliontable_detail join milliontable on milliontable.name=milliontable_detail.name where milliontable.age>65 limit 5;

```
2022-11-28 20:27:22.719 UTC [31980 102 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT:
SESSION,24,1,READ,SELECT,TABLE,public.milliontable_detailselect milliontable_detail.population from milliontable_detail join milliontable on milliontable.name=milliontable_detail.name where milliontable.age>65 limit 5;,<none>
2022-11-28 20:27:22.719 UTC [31980 103 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT:
SESSION,24,1,READ,SELECT,TABLE,public.milliontable_select milliontable_detail.population from milliontable_detail join milliontable on milliontable.name=milliontable_detail.name where milliontable.age>65 limit 5;,<none>
```

Prepared statement

More information is passed in the audit log, allowing you to tell exactly which table is changed.

```
prepare milliontablestmt(character varying(10), integer, date)
as insert into milliontable (name, age, joindate) values ($1, $2, $3);
execute milliontablestmt ('75b33939a1', 77, '2020-10-01');
2022-11-28 20:37:32.506 UTC [31980 127 63850890.7cec] [qcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(33654) ysqlsh yuqabyte yuqabyte] LOG: AUDIT: SESSION, 27, WRITE, PREPARE, , , "prepare
milliontablestmt (character varying (10), integer, date) as insert into milliontable (name, age, joindate)
values ($1, $2, $3);", <none>
2022-11-28 20:37:42.388 UTC [31980 130 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(33654) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 28, 1, WRITE, INSERT, TABLE, public.milliontable prepare milliontable stmt (character varying (10),
integer, date) as insert into milliontable (name, age, joindate) values ($1, $2,
$3);","75b33939a1,77,2020-10-01"
```



CTE

Again, distinct entries for each table accessed or changed.

```
WITH CTE AS
(UPDATE milliontable
SET age = 888
WHERE name = 'asdf6789'
RETURNING name)
INSERT INTO milliontable detail
SELECT name
  FROM cte;
2022-11-29 17:46:01.299 UTC [13533 75 6386425b.34dd] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(37530) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 32, 1, WRITE, INSERT, TABLE, public.milliontable detailWITH CTE AS (UPDATE milliontable SET age = 888
WHERE name = 'asdf6789' RETURNING name) INSERT INTO milliontable detail SELECT name FROM cte;,<none>
2022-11-29 17:46:01.299 UTC [13533 76 6386425b.34dd] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(37530) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 32, 1, WRITE, UPDATE, TABLE, public.milliontable WITH CTE AS (UPDATE milliontable SET age = 888 WHERE
name = 'asdf6789' RETURNING name) INSERT INTO milliontable detail SELECT name FROM cte;,<none>
```



CTE - compare to log_statement

```
WITH CTE AS

(UPDATE milliontable

SET age = 888

WHERE name = 'asdf6789'

RETURNING name)

INSERT INTO milliontable_detail

SELECT name

FROM cte;
```

Much harder to parse out accesses or changes to a particular table with standard logging. What if you wanted to look only for updates to milliontable_detail?

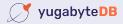
2022-11-30 12:43:45.017 UTC [26600 55 63874eba.67e8] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(36808) ysqlsh yugabyte yugabyte] LOG: statement: WITH CTE AS (UPDATE milliontable SET age = 888 WHERE name = 'asdf6789' RETURNING name) INSERT INTO milliontable_detail SELECT name FROM cte;

Function

```
DO $$
DECLARE
   result RECORD;
BEGIN
    FOR result IN
       SELECT name, age
         FROM milliontable limit 5
   LOOP
       INSERT INTO milliontable_detail (name, population)
              VALUES (result.name, result.age * 100);
    END LOOP;
END $$;
```

Function Logging - audit log

```
2022-11-29 20:57:01.262 UTC [12343 68961 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yuqabyte yuqabyte] LOG: AUDIT: SESSION, 24,1, FUNCTION, DO,,,, "DO $$
2022-11-29 20:57:01.263 UTC [12343 68962 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yuqabyte yuqabyte] LOG: AUDIT: SESSION, 24,2, READ, SELECT, TABLE, public.milliontable, "SELECT
name, age
2022-11-29 20:57:01.266 UTC [12343 68963 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 24, 3, WRITE, INSERT, TABLE, public.milliontable detail, "INSERT INTO milliontable detail (name, population)
2022-11-29 20:57:01.267 UTC [12343 68964 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yugabyte yugabyte] LOG: AUDIT:
SESSION, 24, 4, WRITE, INSERT, TABLE, public.milliontable detail, "INSERT INTO milliontable detail (name, population)
2022-11-29 20:57:01.267 UTC [12343 68965 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yugabyte yugabyte] LOG: AUDIT:
SESSION, 24,5, WRITE, INSERT, TABLE, public.milliontable detail, "INSERT INTO milliontable detail (name, population)
2022-11-29 20:57:01.267 UTC [12343 68966 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 24,6, WRITE, INSERT, TABLE, public.milliontable detail, "INSERT INTO milliontable detail (name, population)
2022-11-29 20:57:01.267 UTC [12343 68967 638670ad.3037] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(42242) ysqlsh yuqabyte yuqabyte] LOG: AUDIT:
SESSION, 24,7, WRITE, INSERT, TABLE, public.milliontable detail, "INSERT INTO milliontable detail (name, population)
```



Function Logging - standard logging

2022-11-30 12:46:23.498 UTC [26600 58 63874eba.67e8] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(36808) ysqlsh yugabyte yugabyte] LOG: statement: DO \$\$ DECLARE result RECORD; BEGIN FOR result IN SELECT name, age FROM milliontable limit 5 LOOP INSERT INTO milliontable_detail (name, population) VALUES (result.name, result.age * 100); END LOOP; END \$\$;

Just one entry, compared to several distinct entries in the audit log lines.

Create view and select from view

The standard log only shows a read on an object named "lowage." The audit log shows that this was a read on a view, and shows the read on the table underlying that view. What if view was changed or removed?

Audit:

```
2022-11-28 20:51:24.522 UTC [31980 152 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 37, 1, READ, SELECT, VIEW, public.lowage select * from lowage limit 5;, <none> 2022-11-28 20:51:24.522 UTC [31980 153 63850890.7cec] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(33654) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 37, 1, READ, SELECT, TABLE, public.milliontable select * from lowage limit 5;, <none>
```

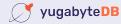
Standard Log:

```
2022-11-30 12:48:21.791 UTC [26600 68 63874eba.67e8] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(36808) ysqlsh yugabyte yugabyte] LOG: statement select * from lowage limit 5;
```

Truncate table

```
truncate account;
```

```
2022-11-29 00:14:27.533 UTC [31682 9 63854e51.7bc2] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1] [10.204.0.60(40796) ysqlsh yugabyte yugabyte] LOG: AUDIT: SESSION, 1, 1, WRITE, TRUNCATE TABLE, , truncate account; , <none>
```



Transaction!

```
select * from milliontable where name='41ab60e791';

begin;

update milliontable set age=65 where name='41ab60e791';

select * from milliontable where name='41ab60e791';

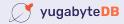
rollback;

select * from milliontable where name='41ab60e791';

2022-11-29 16:37:08.892 UTC [26810 25 6386347b.68ba] [gcp us-east1 us-east1-c yb-demo-parham-audit8-n1]
[10.204.0.60(36106) ysqlsh yugabyte yugabyte] LOG: AUDIT:

SESSION,10,1,WRITE,UPDATE,TABLE,public.milliontableupdate milliontable set age=65 where

name='41ab60e791';<none>
```



Be aware that statements inside transactions are



Thank You

Join us on Slack: yugabyte.com/slack (#yftt channel)

Star us on Github: github.com/yugabyte/yugabyte-db



