Learning Objectives

Learners will be able to…

- Describe what monitoring is and why it is needed
- Describe what is audit
- List audit types
- Use trigger based audit

info

Make Sure You Know

A basic understanding of triggers in databases is essential. Triggers are special types of stored procedures that are executed or triggered in response to certain events on a particular table or view in a database.

Learners should be comfortable with the syntax of SQL commands. This includes knowledge of how to structure a query, use WHERE clauses, and understand basic operators.

Monitoring

Monitoring the state of the database helps keep the database performance and availability high. Database monitoring involves checking for free space in all table spaces and checking for database and server errors.

info

Databases Monitoring

Databases Monitoring is a set of processes and tools to be able to determine how well/fast databases are running.

You need to constantly monitor the operation of the database to ensure that it is in working order and its work is not suspended by a complex task that reduces performance.

What needs to be monitored

1. **Availability** - One of the main tasks is to check the availability, because the base is a service that provides access to these clients and if a failure occurs, this can lead to the loss of users.
2. **Connected clients and their activity** - We also need to monitor the clients that connect to our database, because they can be both normal clients and harmful clients that can harm the database. Their activities need to be tracked and logged.
3. **Queries and performance** - Clients access the database using queries, so it’s important to evaluate what queries are coming up, keep track of their adequacy, that they are not problematic written, that some options need to be rewritten and made so that they work faster and with better performance.
4. **System Metrics** - Since the database is a piece of software that runs on some kind of environment, it is also necessary to check its metrics, such as hard disk load and CPU usage.

Typical Monitoring Tasks

It is recommended to perform tasks from the monitoring task checklist. These tasks include the following tasks:

- Checking for error messages in Log Files
- Checking free disk space and disk usage

- Monitor server resource usage and errors
- Checking the status of a periodic backup
- Monitoring the use of rollbacks
- Configuring Alerts for Certain Situations

Audit

A database audit is a procedure for tracing data flows in a database to check the operation of components, detect and eliminate errors. Based on the results of the audit, it is possible to form a list of measures aimed at improving the stability of the system and protection against unauthorized access, cyber attacks.

info

Database audit

A database audit is a set of activities that results in an assessment of two parameters that determine the effectiveness: rationality and security.

Depending on the frequency of application, the audit of database systems can be divided into:

- **Primary**, which is carried out in such cases as, for example, starting a new project or building and setting up IT processes and communications in a company;
- **Regular**, the purpose of which is to periodically check the structure of the database, to find errors and ways to optimize in a constantly changing information environment.

Database audit tasks

Comprehensive verification includes the following steps:

- Formation of a list of actions to improve the efficiency of the database elements.
- Updating databases to the state relevant at the time of the audit.
- Auditing errors in the system, due to which all users or individual groups are unable to obtain the necessary data.
- Complete elimination of errors that cause unstable database operation.
- Optimizing the time to gain access, increasing the speed of the database response to user actions.
- Investigate suspicious activity.
- Checking the actions of an unauthorized user. For example, an unauthorized user may modify or delete data, or the user may have more privileges than expected, which may cause user permissions to be reevaluated.

- Tracking and collecting data on specific actions with the database.

Based on an audit of SQL data, you can:

- give an independent assessment of the level of safety;
- control all the data that users have involved, make sure their integrity;
- compile a list of weaknesses, the improvement of which will increase the level of database security;
- develop a list of recommendations for the further operation of the database with maximum protection of stored information

Audit types

Auditing tools will vary greatly from the DBMS used, but the following general types can be distinguished:

- **SQL transaction logs**. The transaction log in SQL Server is like the black box of an airplane. It will record everything that happens, which is good for purposes like auditing. No additional cost is required as this is already a built-in DBMS process.
- **SQL Triggers** - This has been the go-to tool for many years. They can be easily configured and track various information. Triggers are fully customizable, allowing users to create their own audit information stores. Triggers are an intrusive technology that can cause errors in client applications if they fail. They are not recommended for high throughput operations or bulk table/operation inserts, and trigger based tier maintenance can take a long time.
- **Manual auditing** - this could include a set of queries and possibly reports to track the activity of each table, user transactions, keep track of recent changes to sensitive tables, etc. But besides being time consuming, it will be nearly impossible to scale this is for all possible audit events.

PostgreSQL allow you to use this options:

Transactions logs

The `log_statement` parameter in PostgreSQL configuration specifies which SQL statements are logged. Only superusers can change it. It has the following values:
- **none** – no logging. This is the default option.
- **ddl** – logging all data definition operations like CREATE, DROP, and ALTER
- **mod** – same as ddl, and also logs operations that modify data, like INSERT, UPDATE and DELETE
- **all** – logs all SQL queries.
For auditing purposes, it is advised to set `log_statement` to all.

PostgreSQL Audit Extension

The PostgreSQL Audit Extension (pgAudit) provides detailed session and/or object audit logging via the standard PostgreSQL logging facility.

It is a native PostgreSQL extension that uses the standard PostgreSQL logging facility to provide detailed object and session audit logging. pgAudit was designed to extend native PostgreSQL capabilities, providing users with the ability to produce audit logs. It is often used to supply information for financial or regulatory compliance, as well as for ISO certifications.

Custom Triggers for Auditing

The basic idea behind custom trigger auditing is to create a shadow audit table for the database tables you need to track. Each change to the main table should be logged to the shadow table, using triggers.

It is important to only implement triggers using database operations and stored procedures, not via application code. If triggers are created as part of application code, the code can break when ad-hoc changes are made to the data.

Trigger Based Audit Example

Let’s try implement trigger based audit.

1. Create table from write audit data

   ```
   sql     CREATE TABLE logged_actions (       table_name TEXT NOT NULL,       user_name TEXT,       action_tstamp TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,       action TEXT NOT NULL CHECK (action IN ('I','D','U')),       original_data TEXT,       new_data TEXT,       query TEXT     );
   ```

2. Create audit trigger

   ~~~
   ```sql
   CREATE OR REPLACE FUNCTION if_modified_func() RETURNS trigger
   AS body
   DECLARE
   v_old_data TEXT;
   v_new_data TEXT;
   BEGIN
   /* If this actually for real auditing (where you need to log EVERY action),
   then you would need to use something like dblink or plperl that could
   log outside the transaction,
   regardless of whether the transaction committed or rolled back.
   */
   ~~~

```
 if (TG_OP = 'UPDATE') then
     v_old_data := ROW(OLD.*);
     v_new_data := ROW(NEW.*);
     insert into logged_actions
(table_name,user_name,action,original_data,new_data,query)
     values
(TG_TABLE_NAME::TEXT,session_user::TEXT,substring(TG_OP,1,1),
v_old_data,v_new_data, current_query());
     RETURN NEW;
 elsif (TG_OP = 'DELETE') then
     v_old_data := ROW(OLD.*);
     insert into logged_actions
(table_name,user_name,action,original_data,query)
     values
(TG_TABLE_NAME::TEXT,session_user::TEXT,substring(TG_OP,1,1),
v_old_data, current_query());
     RETURN OLD;
 elsif (TG_OP = 'INSERT') then
     v_new_data := ROW(NEW.*);
     insert into logged_actions
(table_name,user_name,action,new_data,query)
     values
(TG_TABLE_NAME::TEXT,session_user::TEXT,substring(TG_OP,1,1),
v_new_data, current_query());
     RETURN NEW;
 else
     RAISE WARNING '[AUDIT.IF_MODIFIED_FUNC] - Other action
occurred: %, at %',TG_OP,now();
     RETURN NULL;
 end if;
```

EXCEPTION
WHEN data_exception THEN
RAISE WARNING ‘[AUDIT.IF_MODIFIED_FUNC] - UDF ERROR [DATA EXCEPTION] - SQLSTATE: %, SQLERRM: %’,SQLSTATE,SQLERRM;
RETURN NULL;
WHEN unique_violation THEN
RAISE WARNING ‘[AUDIT.IF_MODIFIED_FUNC] - UDF ERROR [UNIQUE] - SQLSTATE: %, SQLERRM: %’,SQLSTATE,SQLERRM;
RETURN NULL;
WHEN others THEN
RAISE WARNING ‘[AUDIT.IF_MODIFIED_FUNC] - UDF ERROR [OTHER] - SQLSTATE: %, SQLERRM: %’,SQLSTATE,SQLERRM;
RETURN NULL;
END;

body
LANGUAGE plpgsql;
```

3. After creating a table and a trigger, we need to select a table to be audited. Let it be a table `actor`.
   `sql     CREATE TRIGGER actor_if_modified_trg      AFTER INSERT OR UPDATE OR DELETE ON actor      FOR EACH ROW EXECUTE PROCEDURE if_modified_func();`

Now we can try adding, changing and deleting data.

```
INSERT INTO actor (first_name, last_name)
VALUES ('JACKIE', 'CHAN');

UPDATE actor
SET first_name = 'First', last_name = 'Last'
WHERE actor_id = 201;

DELETE FROM actor WHERE actor_id = 201;
```

The audit table will display the operations that were performed on the data.

```
SELECT * FROM logged_actions LIMIT 3;
```
```

```
 table_name | user_name |         action_tstamp         | action
|                 original_data                 |
new_data                     |                    query
------------+-----------+-------------------------------+-------
-+---------------------------------------------+--------------
---------------------------------+----------------------------
-----------------
 actor      | codio     | 2023-08-22 15:11:46.462556+00 | I
|                                               |
(202,JACKIE,CHAN,"2023-08-22 15:11:46.462556") | INSERT INTO
actor (first_name, last_name)   +
            |           |                               |
|                                               |
| VALUES ('JACKIE', 'CHAN');

 actor      | codio     | 2023-08-22 15:28:46.805185+00 | U
| (201,JACKIE,CHAN,"2023-07-01 04:34:33")       |
(201,First,Last,"2023-08-22 15:28:46.805185")  | UPDATE actor
+
            |           |                               |
|                                               |
| SET first_name = 'First', last_name = 'Last'+
            |           |                               |
|                                               |
| WHERE actor_id = 201;

 actor      | codio     | 2023-08-22 15:29:47.09555+00  | D
| (201,First,Last,"2023-08-22 15:28:46.805185") |
| DELETE FROM actor WHERE actor_id = 201;
(3 rows)
```