Audit - data storage

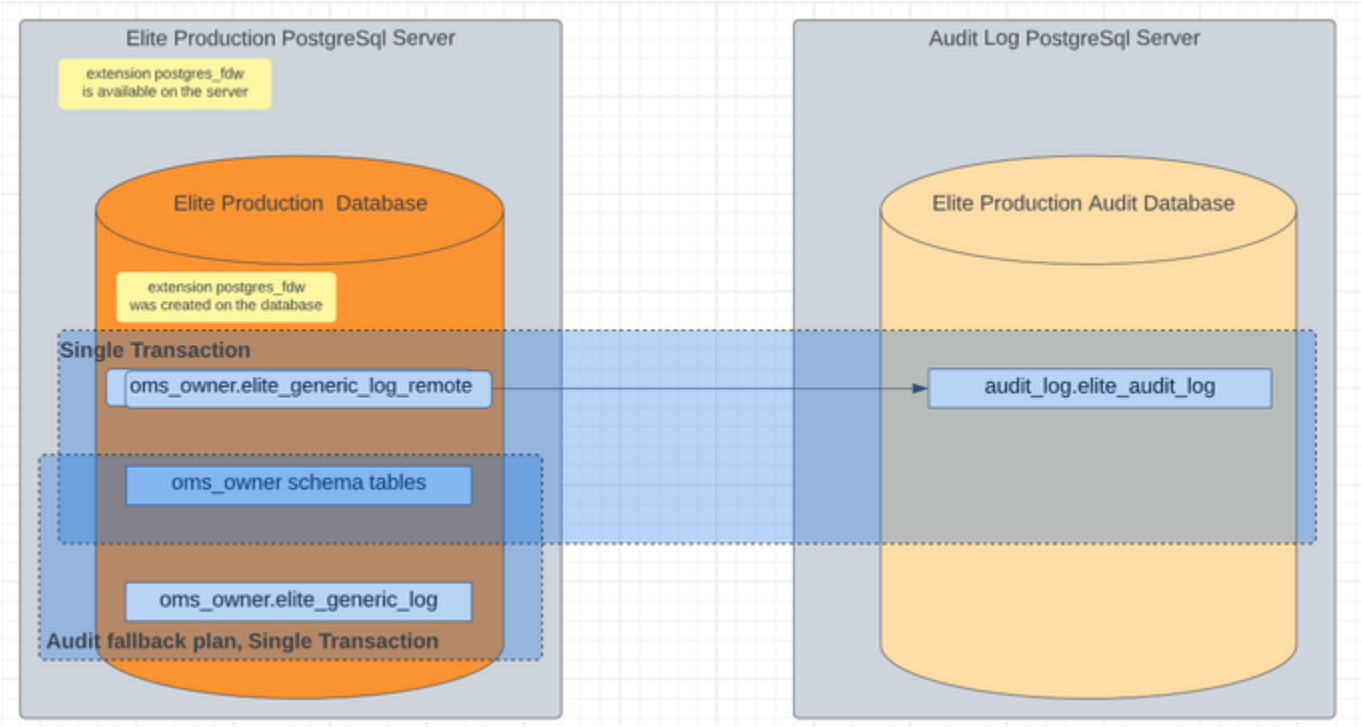
Introduction

The audit functionality allows capturing the V5 Elite application user, time and action of every database INSERT/UPDATE/DELETE operation triggered by the Elite application. It is implemented using a generic function, a generic database trigger and an audit table. The audit trigger can be created/enabled via the Audit module in the Elite application for any tables that requires auditing. The audit triggers/function will persist the audit information in the audit table.

Due to the possible high volume of data that might accumulate in the audit table, there is a need to manage the storage of audit data.

Configuration

The proposed solution for managing the audit data is to create it and store it outside of the Elite database in a separate database reserved for audit data. The audit database may reside on the same PostgreSql server as the Elite database or on a separate PostgreSql server within the same network. This allows for searching the audit data without affecting the Elite databases and/or the Elite server. It also eliminates the need to constantly have to prune audit data out of the Elite databases and archiving it somewhere else. One audit database is proposed for each Elite database.



When the Elite application creates or modifies data in the Elite database on tables configured for auditing, the Elite application creates and stores audit data at the same time. The audit data is persisted directly in the remote audit database (elite_generic_log_remote aka remote table elite_audit_log) and the Elite database transaction succeeds only if the persisting of the audit data succeeds also. No loss of audit data can occur since a single database transaction takes place.

The remote table elite_audit_log is created partitions by datetime column to allow for efficient searches based on datetime.

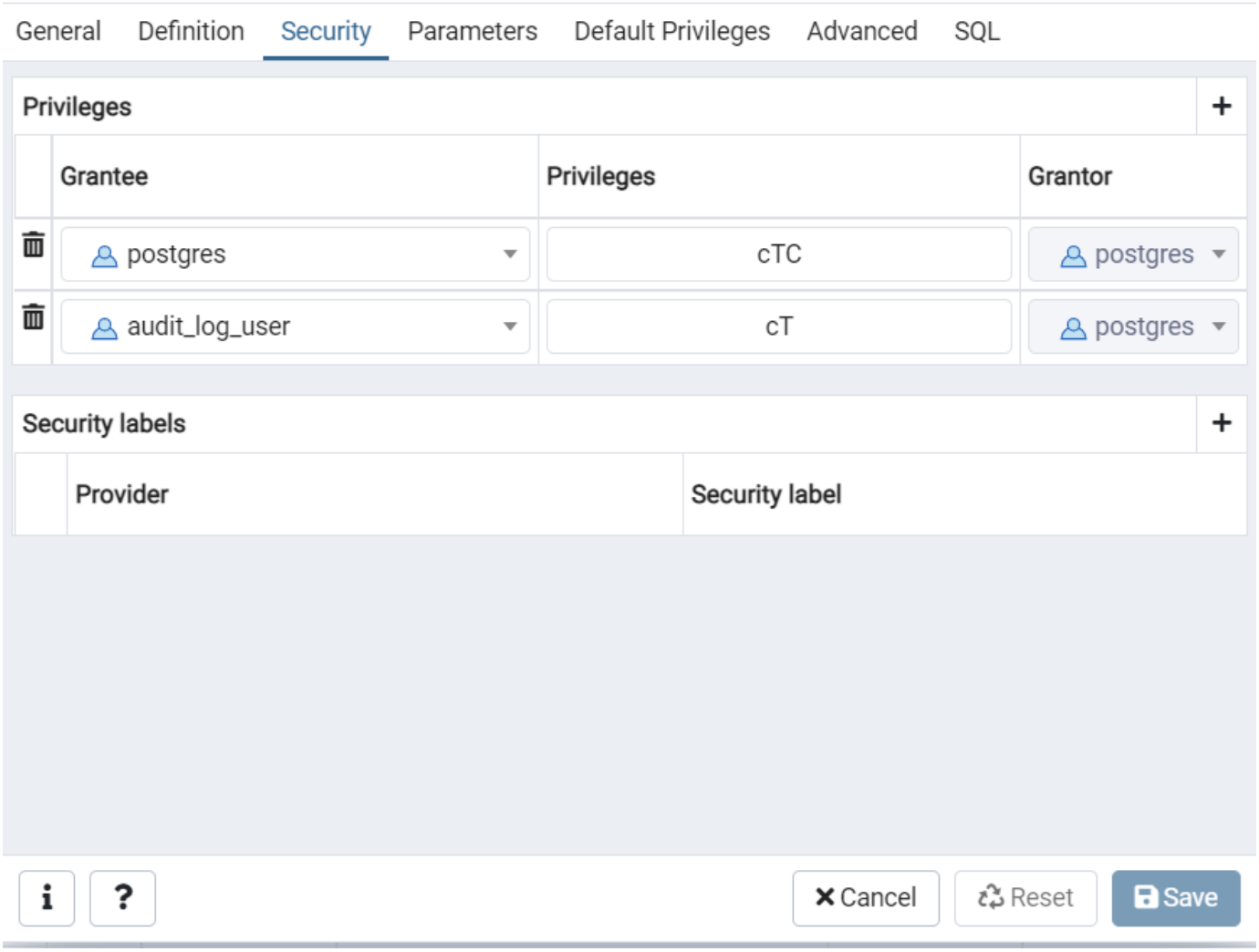
A fallback plan is in place in case the audit server/audit database fail/are not available. The Elite application will try to persist the audit data in a local table within the Elite database (elite_generic_log). The audit data in the Elite database table elite_generic_log will have to be manually moved to the audit database once the issue is resolved.

If the fallback plan fails, the Elite application cannot proceed. If the Elite application must proceed, the audit generic function can be replaced with an audit generic function that does nothing, thus allowing the Elite application to be available without the audit data being captured.

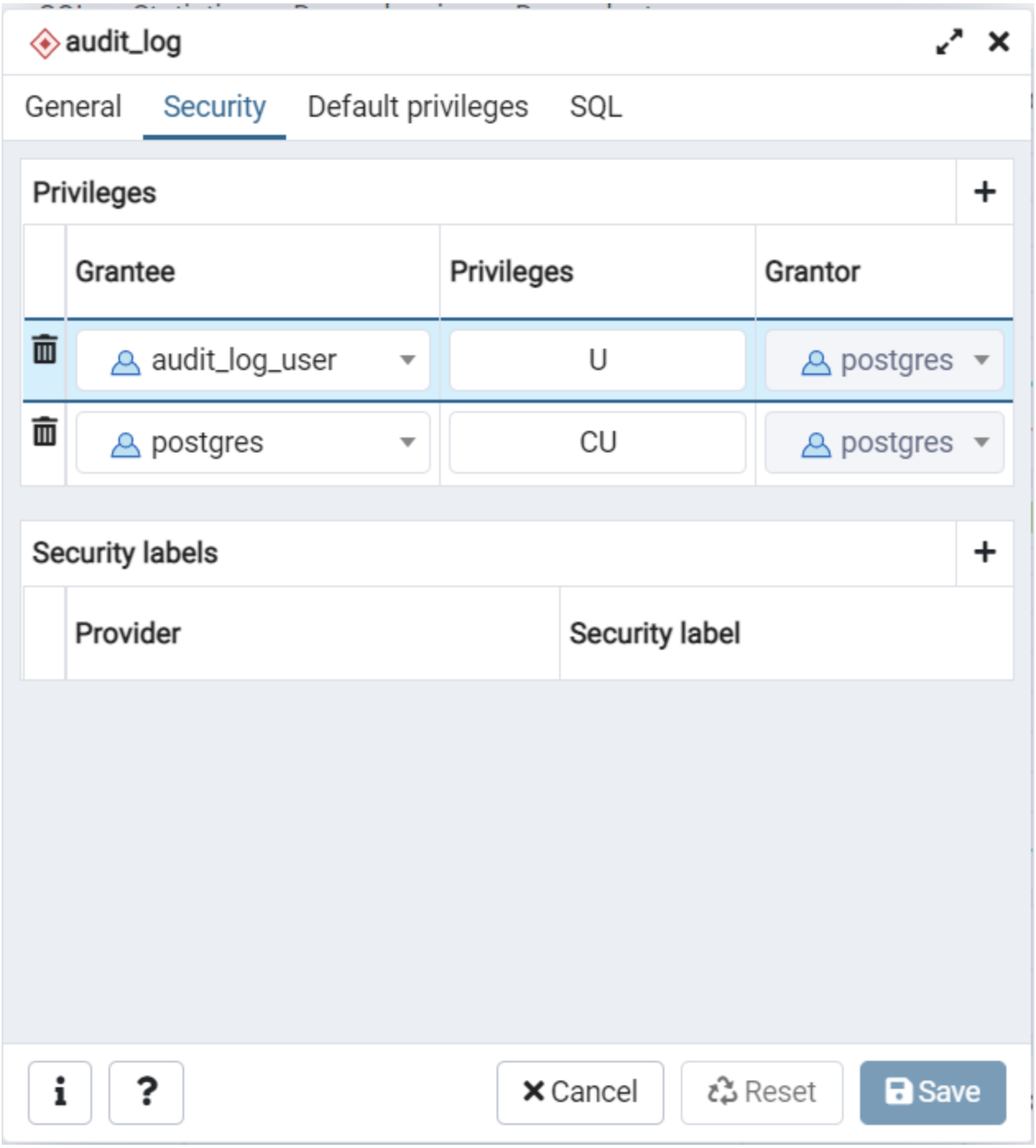
Implementation

Audit Server/Audit Database

- create an audit database owned by postgres user
- create a schema called: audit_log
- created a user called: audit_log_user
- grant Connect and Temporary privileges on the audit database to user audit_log_user:



- grant usage privileges on schema audit_log to user audit_log_user:



- test user audit_log_user:

  psql -U audit_log_user -d **auditDatabase** -h **auditServer**

- create the audit log table in the audit schema, the table is owned by postgres:

  psql -U postgres -d **auditDatabase** -h **auditServer**

  \i elite_audit_log.tab

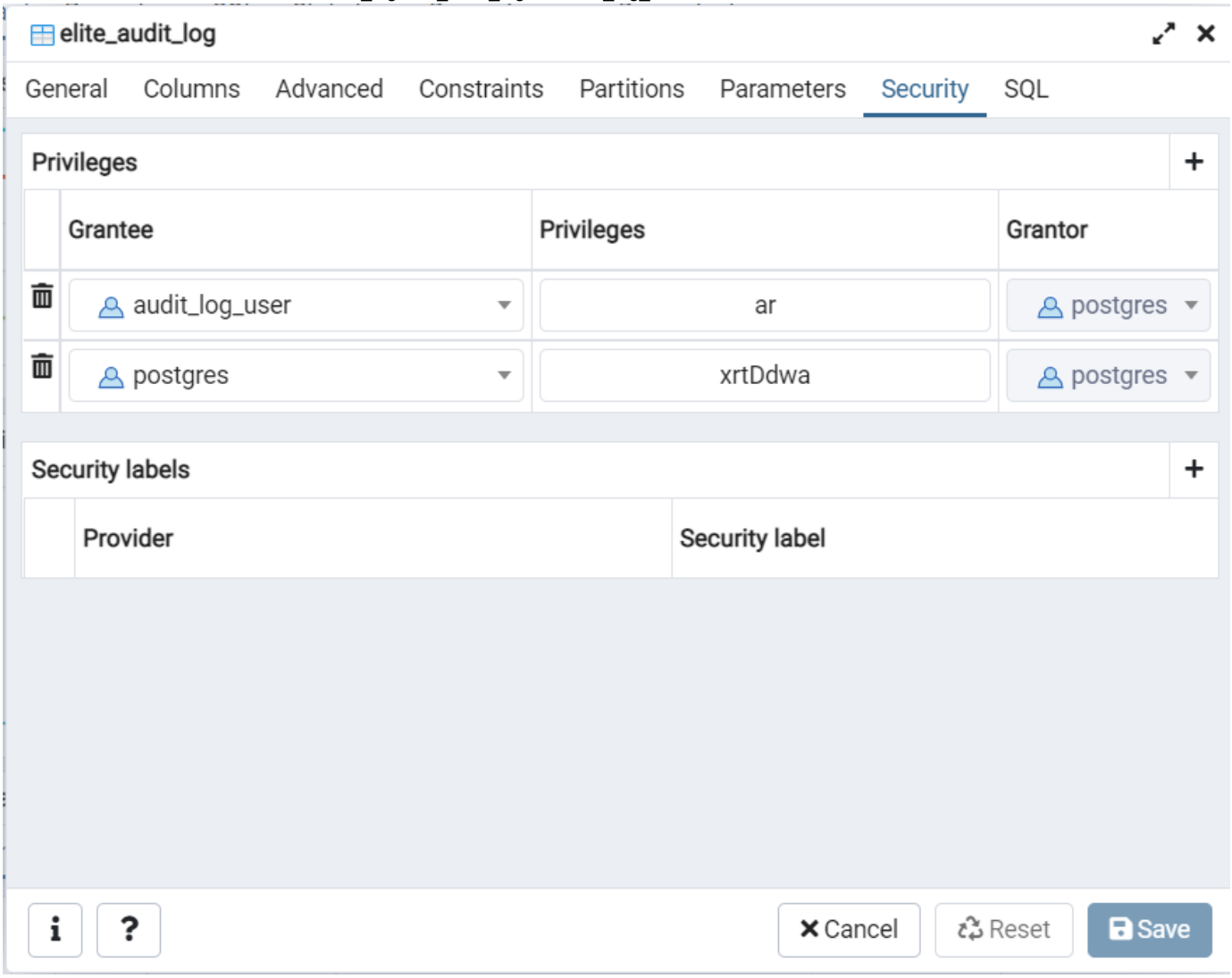
Please modify the script to add more partitions as needed.

```
CREATE TABLE audit_log.elite_audit_log (
audittime     TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP NOT NULL ,
action        bpchar(1) NOT NULL,
username      text NOT NULL,
table_name    text NOT NULL,
row_data_new  jsonb NULL,
row_data_old  jsonb NULL,
CONSTRAINT elite_audit_log_action_chk1 CHECK ((action = ANY (ARRAY['I'::
bpchar, 'U'::bpchar, 'D'::bpchar, 'T'::bpchar])))
) PARTITION BY RANGE (audittime);

CREATE TABLE audit_log.elite_audit_log_part_2023_jan PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-01-01') TO ('2023-02-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_feb PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-02-01') TO ('2023-03-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_mar PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-03-01') TO ('2023-04-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_apr PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-04-01') TO ('2023-05-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_may PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-05-01') TO ('2023-06-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_jun PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-06-01') TO ('2023-07-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_jul PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-07-01') TO ('2023-08-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_aug PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-08-01') TO ('2023-09-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_sep PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-09-01') TO ('2023-10-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_oct PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-10-01') TO ('2023-11-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_nov PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-11-01') TO ('2023-12-01');
CREATE TABLE audit_log.elite_audit_log_part_2023_dec PARTITION OF
audit_log.elite_audit_log
 FOR VALUES FROM ('2023-12-01') TO ('2024-01-01');

 CREATE INDEX ON audit_log.elite_audit_log (audittime);
```

- grant only SELECT and INSERT to user audit_log_user:

  psql -U postgres -d **auditDatabase** -h **auditServer**

  GRANT INSERT, SELECT ON audit_log.elite_audit_log TO audit_log_user;

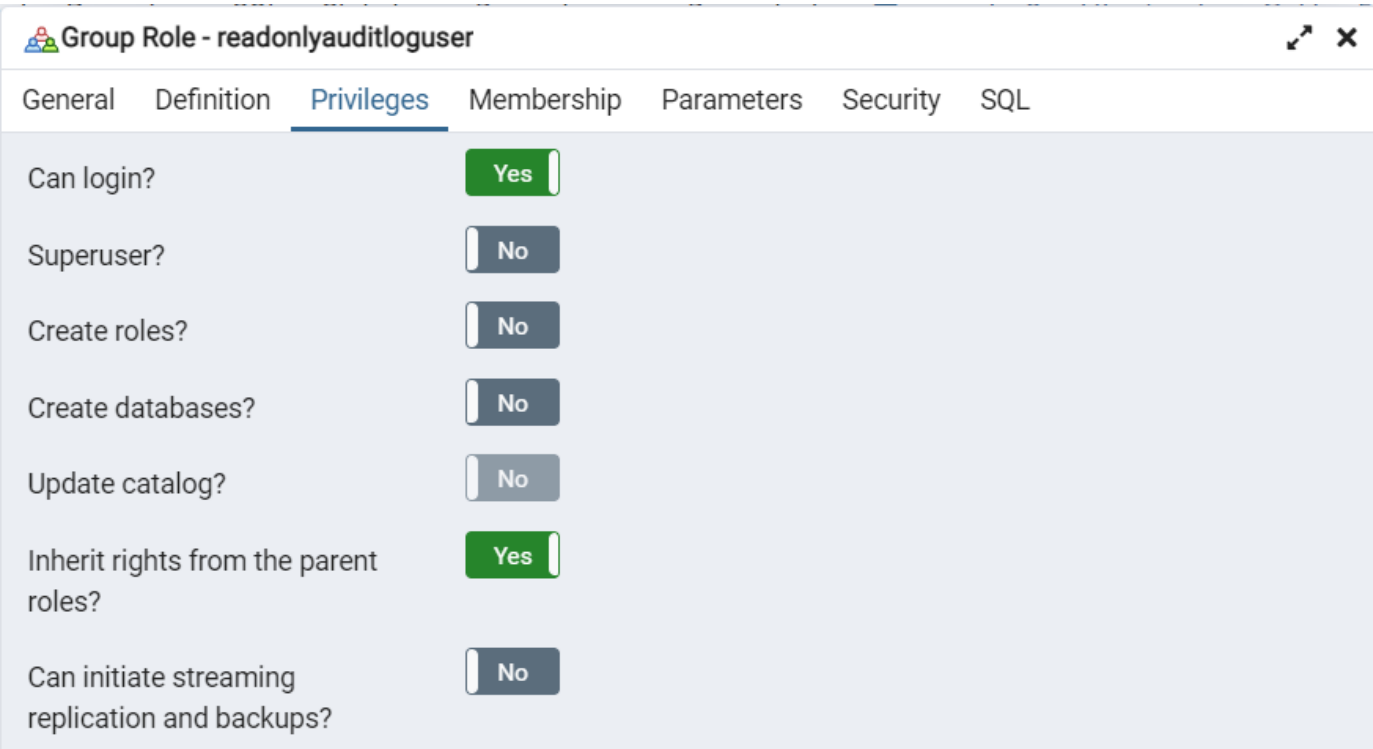


- test that user audit_log_user can insert but cannot delete !

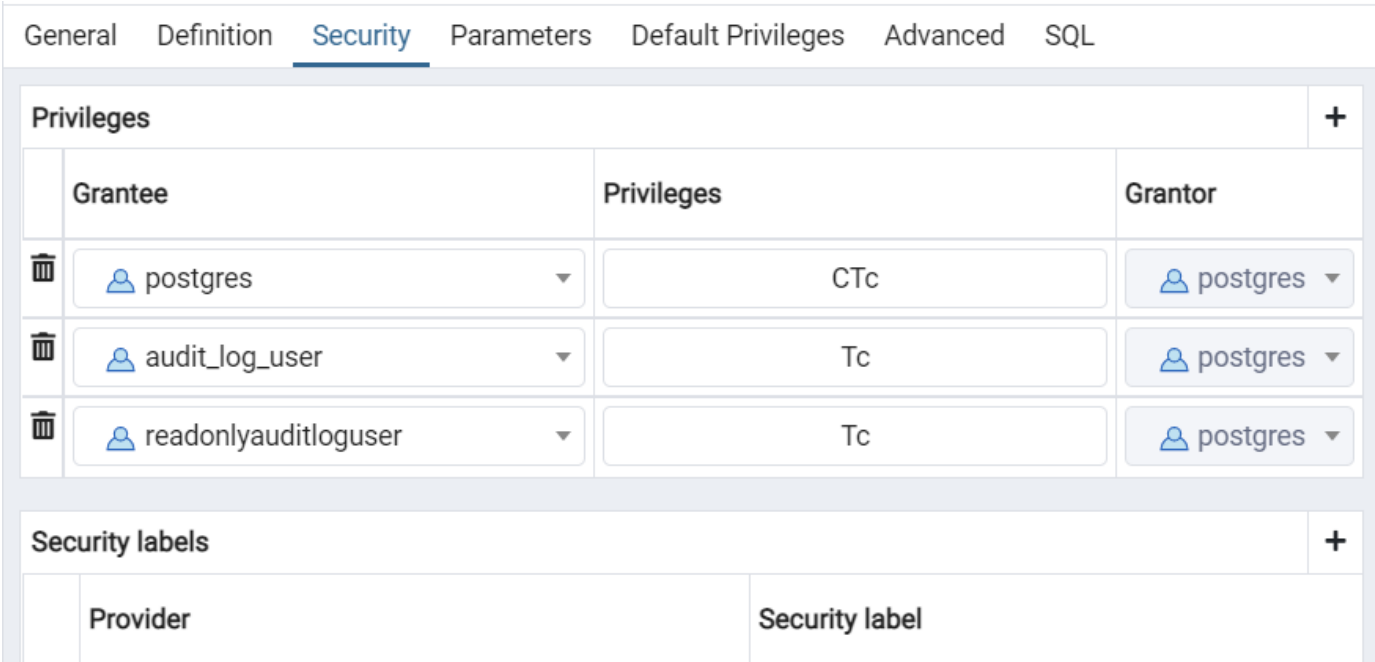
  psql -U audit_log_user -d **auditDatabase** -h **auditServer**

```
INSERT INTO audit_log.elite_audit_log (audittime,"action",username,
table_name,row_data_new,row_data_old) VALUES
         ('2023-01-01 10:58:26.266893-
07','I','oms_owner','offender_alerts','{"row_id": 191, "alert_seq": 13,
"seal_flag": null, "alert_code": "NO_CS", "alert_date": "2022-12-14T01:
58:05", "alert_type": "U", "caseload_id": null, "create_date": "2022-09-
14T01:58:23", "expiry_date": null, "alert_status": "ACTIVE",
"comment_text": null, "caseload_type": "INST", "verified_flag": "N",
"create_user_id": "TESTUSER", "modify_user_id": null,
"create_datetime": "2022-09-14T01:58:26.266893", "modify_datetime":
"2022-09-14T01:58:26.266893", "offender_book_id": 8701,
"root_offender_id": 8961, "authorize_person_text": null}',NULL);

delete from audit_log.elite_audit_log where audittime >= current_date ;
```

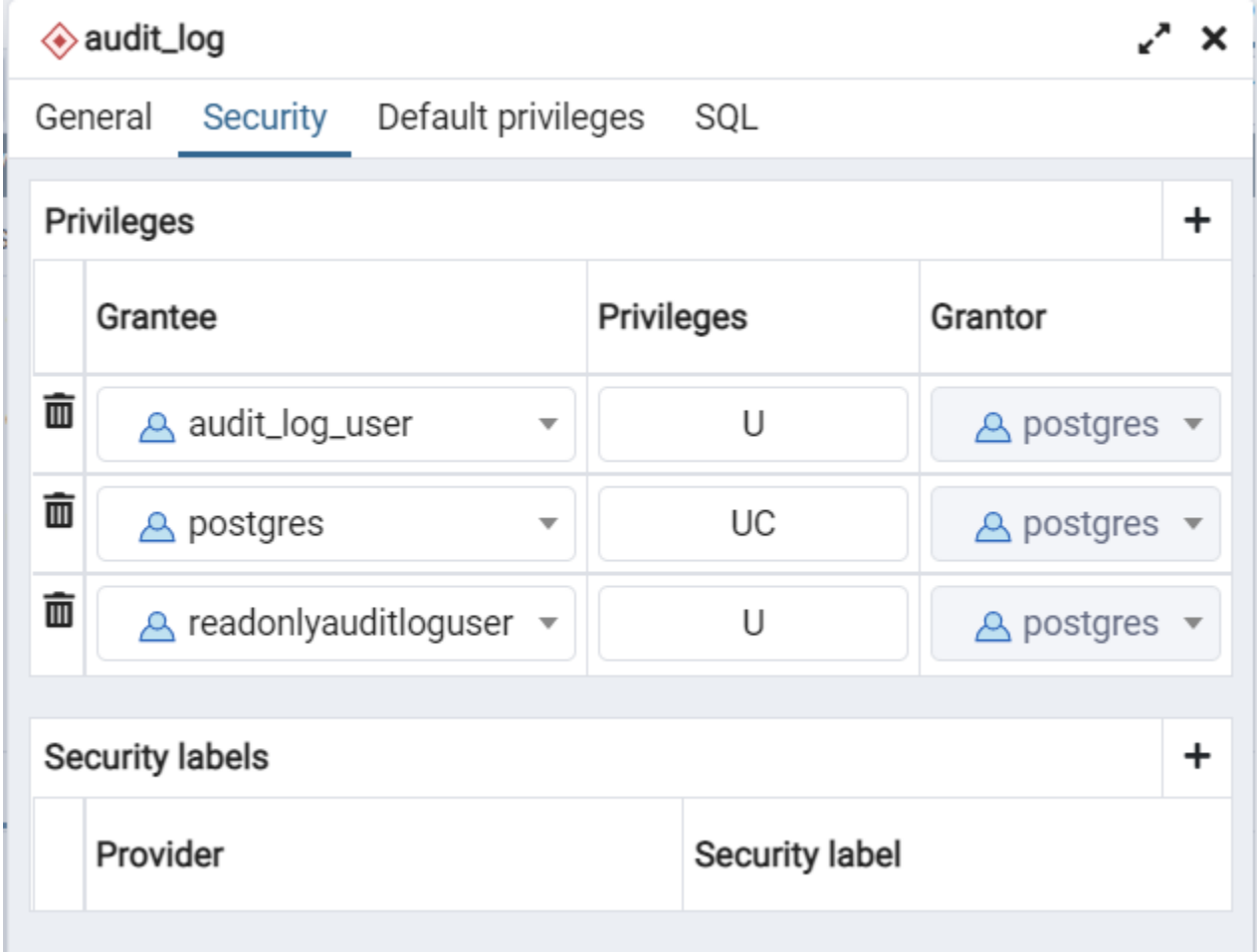
- as postgres delete the one record inserted for test
- create an additional user -ex. **readonlyauditloguser**, a read only user with Login rights who can only select from table `audit_log.elite_audit_log`; this user is to be used for searching / investigating audit logs for Insights or reporting purpose



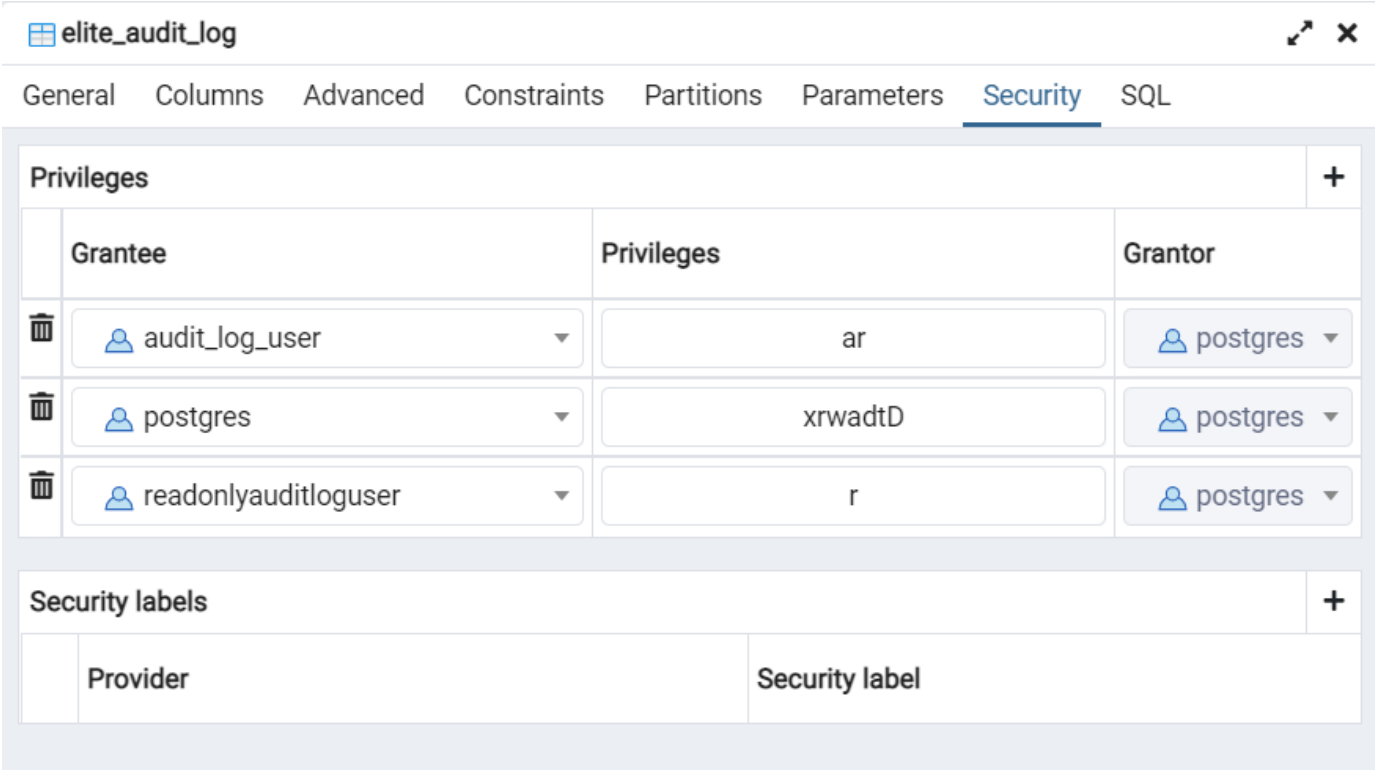
1. grant Connect and Temporary privileges on the audit database to the read only user



2. grant usage privileges on schema audit_log to the read only user



3. grant SELECT only on table audit_log.elite_audit_log to the read only user



Elite Server/Elite Database

- check that extension postgres_fdw is available (both AWS RDS and Azure Pass):

  psql -U postgres -d **EliteDatabase** -h **EliteServer**

  SELECT * FROM pg_available_extensions order by name;
  SELECT * FROM pg_available_extension_versions order by name;

- check extension postgres_fdw is installed on the EliteDatabase:

  psql -U postgres -d **EliteDatabase** -h **EliteServer**

  SELECT * FROM pg_extension;

  SELECT e.extname AS "Name", e.extversion AS "Version", n.nspname AS "Schema", c.description AS "Description"
  FROM pg_catalog.pg_extension e
  LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
  LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
  ORDER BY 1;

- if the extension is not installed on the EliteDatabase, create the extension:

  psql -U postgres -d **EliteDatabase** -h **EliteServer**

  EliteDatabase => create extension postgres_fdw;

- create the foreign wrapper server pointing to the auditServer and auditDatabase

  psql -U postgres -d **EliteDatabase** -h **EliteServer**

  CREATE SERVER remote_audit_log_srv FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host '**auditServer**', dbname '**auditDatabase**');

  --verify
  select * from pg_foreign_server;

- grant usage of the foreign server to oms_owner

  psql -U postgres -d **EliteDatabase** -h **EliteServer**

  GRANT USAGE ON FOREIGN SERVER remote_audit_log_srv TO oms_owner;

- create the user mapping

  psql -U postgres -d **EliteDatabase** -h **EliteServer**

  CREATE USER MAPPING FOR oms_owner SERVER remote_audit_log_srv OPTIONS (user 'audit_log_user', password '**auditloguserpassword**');

  --verify
  select * from pg_user_mapping;
  select * from pg_user_mappings;

- create the foreign wrapper table pointing to the remote audit table (as oms_owner)

  psql -U oms_owner -d **EliteDatabase** -h **EliteServer**

  \i elite_generic_log_remote.tab

```
create foreign table elite_generic_log_remote (audittime TIMESTAMP(6)
DEFAULT CURRENT_TIMESTAMP NOT NULL , action bpchar(1) NOT NULL,
username text NOT NULL, table_name text NOT NULL, row_data_new jsonb
NULL, row_data_old jsonb NULL)
  SERVER remote_audit_log_srv OPTIONS (schema_name 'audit_log',
table_name 'elite_audit_log');
```

  --test oms_owner can insert

```
INSERT INTO elite_generic_log_remote (audittime,"action",username,
table_name,row_data_new,row_data_old) VALUES
         ('2023-01-01 20:00:00.266893-
07','I','oms_owner','offender_alerts','{"row_id": 191, "alert_seq": 13,
"seal_flag": null, "alert_code": "NO_CS", "alert_date": "2022-12-14T01:
58:05", "alert_type": "U", "caseload_id": null, "create_date": "2022-09-
14T01:58:23", "expiry_date": null, "alert_status": "ACTIVE",
"comment_text": null, "caseload_type": "INST", "verified_flag": "N",
"create_user_id": "TESTUSER", "modify_user_id": null,
"create_datetime": "2022-09-14T01:58:26.266893", "modify_datetime":
"2022-09-14T01:58:26.266893", "offender_book_id": 8701,
"root_offender_id": 8961, "authorize_person_text": null}',NULL);
```

  --test oms_owner cannot delete

```
delete from elite_generic_log_remote where audittime >= current_date ;
```

  --delete the one record inserted for test

- apply the generic audit function (as oms_owner)

  psql -U oms_owner -d **EliteDatabase** -h **EliteServer**

  \i elite_generic_log_20230210.fun

```
CREATE OR REPLACE FUNCTION oms_owner.elite_generic_log()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
BEGIN
  IF TG_OP = 'DELETE' THEN
     INSERT INTO oms_owner.elite_generic_log_remote VALUES (now(), 'D',
session_user, TG_TABLE_NAME, NULL , to_jsonb(OLD));
  ELSIF TG_OP = 'UPDATE' THEN
     INSERT INTO oms_owner.elite_generic_log_remote VALUES (now(), 'U',
session_user, TG_TABLE_NAME, to_jsonb(NEW),to_jsonb(OLD));
  ELSIF TG_OP = 'INSERT' THEN
     INSERT INTO oms_owner.elite_generic_log_remote VALUES (now(), 'I',
session_user, TG_TABLE_NAME, to_jsonb(NEW), NULL);
  ELSIF TG_OP = 'TRUNCATE' THEN
     INSERT INTO oms_owner.elite_generic_log_remote VALUES (now(), 'T',
session_user, TG_TABLE_NAME, NULL, NULL);
  ELSE
     NULL;
  END IF;

  RETURN NULL;

EXCEPTION
  WHEN OTHERS THEN
          IF TG_OP = 'DELETE' THEN
                 INSERT INTO oms_owner.elite_generic_log VALUES (now(),
'D', session_user, TG_TABLE_NAME, NULL , to_jsonb(OLD));
          ELSIF TG_OP = 'UPDATE' THEN
                 INSERT INTO oms_owner.elite_generic_log VALUES (now(),
'U', session_user, TG_TABLE_NAME,to_jsonb(NEW),to_jsonb(OLD));
      ELSIF TG_OP = 'INSERT' THEN
                 INSERT INTO oms_owner.elite_generic_log VALUES (now(),
'I', session_user, TG_TABLE_NAME,to_jsonb(NEW), NULL);
      ELSIF TG_OP = 'TRUNCATE' THEN
         INSERT INTO oms_owner.elite_generic_log VALUES (now(), 'T',
session_user, TG_TABLE_NAME, NULL, NULL);
      ELSE
         NULL;
          END IF;

          RETURN NULL;


END;
$function$
;
```