To summarize, this approach involves setting up triggers directly on the individual shards (placements) of distributed tables in Citus. Here’s a breakdown of the main concepts and steps:

**Explanation of Triggers on Distributed Tables in Citus**

1. **Issue with Triggers on Distributed Tables:** Citus, a distributed database extension for PostgreSQL, does not natively support creating triggers on distributed tables. This is because triggers in PostgreSQL are usually local to the node where the table resides, while distributed tables in Citus span multiple worker nodes.
2. **Workaround via Shard-Level Triggers:**
   * To create triggers, you have to manually add them on the individual shards of the distributed table.
   * This is done by executing the trigger setup directly on the worker nodes where the shards (placements) of the distributed tables exist.
3. **Colocated Distributed Tables:**
   * When two distributed tables are **colocated** (i.e., their shards reside on the same worker nodes and are partitioned by the same key), it’s possible to create a trigger on one table that affects the other.
   * Citus provides a helper function, run\_command\_on\_colocated\_placements, which allows you to create triggers between pairs of colocated placements.

**Example Code Breakdown**

1. **Create Tables and Distribute:**

sql

Copy code

CREATE TABLE little\_vals (key int, val int);

CREATE TABLE big\_vals (key int, val int);

SELECT create\_distributed\_table('little\_vals', 'key');

SELECT create\_distributed\_table('big\_vals', 'key');

* + This creates two tables, little\_vals and big\_vals, which are distributed based on the key column.

1. **Create a Trigger Function on Worker Nodes:**

sql

Copy code

SELECT run\_command\_on\_workers($cmd$

CREATE OR REPLACE FUNCTION embiggen() RETURNS TRIGGER AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

EXECUTE format(

'INSERT INTO %s (key, val) SELECT ($1).key, ($1).val\*2;',

TG\_ARGV[0]

) USING NEW;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

$cmd$);

* + This creates a trigger function called embiggen() on the worker nodes.
  + The function doubles the value of the val column from little\_vals and inserts it into big\_vals.

1. **Attach Trigger to Colocated Placements:**

sql

Copy code

SELECT run\_command\_on\_colocated\_placements(

'little\_vals',

'big\_vals',

$cmd$

CREATE TRIGGER after\_insert AFTER INSERT ON %s

FOR EACH ROW EXECUTE PROCEDURE embiggen(%s)

$cmd$

);

* + This command adds the trigger after\_insert to the colocated shards.
  + It ensures that whenever a new row is inserted into little\_vals, a row is also inserted into big\_vals with the val column doubled.

**Key Functions Used**

* **run\_command\_on\_workers(cmd)**: Executes the given command on all worker nodes.
* **run\_command\_on\_colocated\_placements(table1, table2, cmd)**: Executes the given command on colocated shard pairs of two distributed tables.

**Important Considerations**

* **Colocation Requirement:** For this method to work, the tables must be properly colocated, meaning they should share the same distribution key and reside on the same worker nodes.
* **Trigger Behavior:** Because the trigger operates directly on shards, it doesn’t apply to distributed tables as a whole but rather to their individual shards.

This setup allows distributed systems like Citus to leverage PostgreSQL's triggers in a distributed environment, albeit with some manual configuration at the shard level.

========================

To adapt the creation of the smartiq.account and smartiq.account\_hist tables for a Citus distributed environment, I’ll focus on distributing these tables based on the client\_id column, which is a logical choice for distribution, as it is likely to group related data effectively.

**Step 1: Create and Distribute the Tables**

First, I'll create the two tables as distributed tables in Citus, using client\_id as the distribution column:

-- Create the smartiq.account table

CREATE TABLE IF NOT EXISTS smartiq.account

(

account\_id bigint NOT NULL DEFAULT nextval('smartiq.account\_account\_id\_seq'::regclass),

entered timestamp without time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

entered\_by character varying(100) COLLATE pg\_catalog."default" NOT NULL DEFAULT CURRENT\_USER,

client\_id bigint NOT NULL,

account\_name character varying(1000) COLLATE pg\_catalog."default" NOT NULL,

account\_code character varying(1000) COLLATE pg\_catalog."default",

account\_desc character varying(9000) COLLATE pg\_catalog."default",

account\_type smallint NOT NULL,

status smallint NOT NULL,

version integer NOT NULL,

effective\_date timestamp without time zone,

CONSTRAINT account\_pkey PRIMARY KEY (account\_id)

) TABLESPACE pg\_default;

-- Distribute the account table on client\_id

SELECT create\_distributed\_table('smartiq.account', 'client\_id');

-- Create the smartiq.account\_hist table

CREATE TABLE IF NOT EXISTS smartiq.account\_hist

(

account\_hist\_id bigint NOT NULL DEFAULT nextval('smartiq.account\_hist\_account\_hist\_id\_seq'::regclass),

entered timestamp without time zone NOT NULL DEFAULT CURRENT\_TIMESTAMP,

entered\_by character varying(100) COLLATE pg\_catalog."default" NOT NULL DEFAULT CURRENT\_USER,

replaced timestamp without time zone NOT NULL,

replaced\_by character varying(100) COLLATE pg\_catalog."default",

account\_id bigint NOT NULL,

client\_id bigint NOT NULL,

account\_name character varying(1000) COLLATE pg\_catalog."default" NOT NULL,

account\_code character varying(1000) COLLATE pg\_catalog."default",

account\_desc character varying(9000) COLLATE pg\_catalog."default",

account\_type smallint NOT NULL,

status smallint NOT NULL,

version integer NOT NULL,

effective\_date timestamp without time zone,

CONSTRAINT account\_hist\_pkey PRIMARY KEY (account\_hist\_id)

) TABLESPACE pg\_default;

-- Distribute the account\_hist table on client\_id

SELECT create\_distributed\_table('smartiq.account\_hist', 'client\_id');

**Step 2: Create the Trigger Function for Shard-Level Operations**

The trigger function needs to be adjusted to work at the shard level, inserting records into the smartiq.account\_hist table when changes are made to smartiq.account. Here’s the modification:

**Trigger Function Creation on Worker Nodes**

First, create the trigger function on each worker node:

SELECT run\_command\_on\_workers($cmd$

CREATE OR REPLACE FUNCTION smartiq.tfn\_account\_hist() RETURNS TRIGGER AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

EXECUTE format(

'INSERT INTO %s (entered, entered\_by, replaced, replaced\_by, account\_id, client\_id, account\_name, account\_code, account\_desc, account\_type, status, version, effective\_date)

VALUES ($1.entered, $1.entered\_by, CURRENT\_TIMESTAMP, NULL, $1.account\_id, $1.client\_id, $1.account\_name, $1.account\_code, $1.account\_desc, $1.account\_type, $1.status, $1.version, $1.effective\_date);',

TG\_ARGV[0]

) USING NEW;

ELSIF (TG\_OP = 'UPDATE') THEN

EXECUTE format(

'INSERT INTO %s (entered, entered\_by, replaced, replaced\_by, account\_id, client\_id, account\_name, account\_code, account\_desc, account\_type, status, version, effective\_date)

VALUES ($1.entered, $1.entered\_by, CURRENT\_TIMESTAMP, $2.entered\_by, $1.account\_id, $1.client\_id, $1.account\_name, $1.account\_code, $1.account\_desc, $1.account\_type, $1.status, $1.version, $1.effective\_date);',

TG\_ARGV[0]

) USING NEW, OLD;

ELSIF (TG\_OP = 'DELETE') THEN

EXECUTE format(

'INSERT INTO %s (entered, entered\_by, replaced, replaced\_by, account\_id, client\_id, account\_name, account\_code, account\_desc, account\_type, status, version, effective\_date)

VALUES (CURRENT\_TIMESTAMP, OLD.entered\_by, CURRENT\_TIMESTAMP, NULL, OLD.account\_id, OLD.client\_id, OLD.account\_name, OLD.account\_code, OLD.account\_desc, OLD.account\_type, OLD.status, OLD.version, OLD.effective\_date);',

TG\_ARGV[0]

) USING OLD;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

$cmd$);

**Step 3: Attach the Trigger to the Colocated Shards**

The final step is to add the trigger to the colocated shards of smartiq.account and smartiq.account\_hist:

SELECT run\_command\_on\_colocated\_placements(

'smartiq.account',

'smartiq.account\_hist',

$cmd$

CREATE TRIGGER trg\_account\_hist

AFTER INSERT OR UPDATE OR DELETE ON %s

FOR EACH ROW EXECUTE PROCEDURE smartiq.tfn\_account\_hist(%s)

$cmd$

);

**Key Notes**

* **Distribution Column (client\_id):** The choice of client\_id as the distribution column ensures that related data is colocated, facilitating efficient operations between the smartiq.account and smartiq.account\_hist tables.
* **Using run\_command\_on\_workers and run\_command\_on\_colocated\_placements:** These functions help propagate the creation of functions and triggers across the shards of distributed tables.
* **Shard-Level Triggers:** Since the trigger operates at the shard level, this setup ensures data consistency for the history table in a distributed manner.