-- 1. Views & Materialized Views

-- a. Create a VIEW that displays:
-- i. user_id
-- ii. full_name (first_name + last_name)
-- iii. Email
-- iv. role_name

```
CREATE OR REPLACE VIEW hc.vw_user_role AS
SELECT u.user_id, CONCAT(u.first_name,' ',u.last_name) as "full_name", u.email, r.role_name
FROM hc.users as u
JOIN hc.roles as r ON u.role_id = r.role_id;

SELECT * FROM hc.vw_user_role;
```

-- b. Create a VIEW that shows only active users created in the last 30 days.

```
CREATE OR REPLACE VIEW hc.vw_active_users AS
SELECT user_id,  CONCAT(first_name,' ',last_name) as "full_name",created_date
FROM hc.users
WHERE created_date >= now() - interval '30 days';

SELECT * FROM hc.vw_active_users;
```

-- c. Create a MATERIALIZED VIEW that stores: i. role_name
ii. total_users per role

```
CREATE MATERIALIZED VIEW hc.mv_total_users_per_role AS
SELECT r.role_name, COUNT(r.role_id)
FROM hc.users as u
JOIN hc.roles as r ON u.role_id = r.role_id
GROUP BY r.role_id;

SELECT * FROM hc.mv_total_users_per_role;
```

-- d. Write a query to refresh the materialized view.

```
--update in user table
UPDATE hc.users SET role_id = (SELECT role_id FROM hc.roles WHERE role_name =
'service_provider')WHERE hc.users.email = 'aarsi@gmail.com';
select * from hc.users;

SELECT * FROM hc.mv_total_users_per_role;

--refresh the materialized view
REFRESH MATERIALIZED VIEW hc.mv_total_users_per_role;

SELECT * FROM hc.mv_total_users_per_role;
```

-- 2. Functions
-- a. Create a SQL function that:
-- i. Accepts a role_id as input
-- ii. Returns the total number of users for that role

```
CREATE OR REPLACE FUNCTION hc.total_users_per_role(id UUID)
RETURNS Integer
LANGUAGE sql
AS $$
	SELECT COUNT(*) FROM hc.users WHERE role_id = id;
$$;

SELECT * FROM hc.total_users_per_role('b1e91a70-9720-4136-91e8-1633f2a4adef');
```

-- b. Create a PL/pgSQL function that:
-- i. Accepts a user_id
-- ii. Returns the user’s full name

```
CREATE OR REPLACE FUNCTION hc.user_full_name(id UUID)
RETURNS varchar
LANGUAGE plpgsql
AS $$
DECLARE user_name varchar;
BEGIN
	SELECT CONCAT(first_name,' ',last_name) INTO user_name FROM hc.users WHERE user_id = id;
	RETURN user_name;
END;
$$;

SELECT * FROM  hc.user_full_name('46187b76-f718-4bcb-a4e0-86a72f243dee');

select * from hc.users;


-- c. Create a function that:
-- i. Accepts birth_date
-- ii. Returns calculated age

CREATE OR REPLACE FUNCTION hc.calculated_age(user_birth_date DATE)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
DECLARE age INTEGER;
BEGIN
	SELECT EXTRACT(YEAR FROM age(user_birth_date)) into age ;
	RETURN age;
END;
$$;

SELECT hc.calculated_age('2005-02-06');
```

-- d. Create a function that:
-- i. Returns all users created today

```
INSERT INTO hc.users (first_name, last_name, email, password, birth_date, address,
mobile_number,role_id)
VALUES ('keval','vora','keval@gmail.com','keval123','2005-02-06', 'junagadh',
6532748956,'b1e91a70-9720-4136-91e8-1633f2a4adef');

CREATE OR REPLACE FUNCTION hc.all_users_created_today()
RETURNS TABLE (user_id UUID, first_name VARCHAR, last_name VARCHAR,
created_date TIMESTAMP)
LANGUAGE plpgsql
AS $$
BEGIN
   RETURN QUERY
   SELECT u.user_id, u.first_name, u.last_name, u.created_date
   FROM hc.users u
   WHERE u.created_date::DATE = CURRENT_DATE;
END;
$$;

SELECT * FROM hc.all_users_created_today();
```

-- e. Demonstrate usage of:
-- i. IN parameters

```
CREATE OR REPLACE FUNCTION hc.total_users_per_role_in(id UUID)
RETURNS Integer
LANGUAGE plpgsql
AS $$
DECLARE total_users integer;
BEGIN
        SELECT COUNT(*) INTO total_users FROM hc.users WHERE role_id = id;
END;
$$;
```

-- ii. OUT parameters

```
CREATE OR REPLACE FUNCTION hc.total_users(OUT total Integer)
RETURNS Integer
LANGUAGE plpgsql
AS $$
BEGIN
        SELECT COUNT(*) INTO total FROM hc.users;
END
$$;

select hc.total_users();
```

-- iii. RETURN TABLE

```
CREATE OR REPLACE FUNCTION hc.all_users()
RETURNS TABLE (user_id UUID, first_name VARCHAR, last_name VARCHAR,
created_date TIMESTAMP)
LANGUAGE plpgsql
AS $$
BEGIN
   RETURN QUERY
   SELECT u.user_id, u.first_name, u.last_name, u.created_date
   FROM hc.users u
   WHERE u.created_date::DATE = CURRENT_DATE;
END;
$$;

select * from hc.all_users();
```

-- 3. Stored Procedures
-- a. Create a stored procedure to:
-- i. Insert a new user
-- ii. Validate email uniqueness
-- iii. Set created_date automatically

Assignment-3

```
CREATE OR REPLACE PROCEDURE hc.insert_user(p_first_name VARCHAR, p_last_name
VARCHAR, p_email VARCHAR, p_password VARCHAR, p_birth_date DATE, p_address
VARCHAR, p_mobile_number VARCHAR, p_role_id UUID)
LANGUAGE plpgsql
AS $$
BEGIN
	IF EXISTS (
		SELECT 1 FROM hc.users as u
		WHERE u.email = p_email
	) THEN
	RAISE EXCEPTION 'email is exists';
	END IF;

INSERT INTO hc.users
(first_name,last_name,email,password,birth_date,address,mobile_number,role_id)
VALUES(p_first_name,p_last_name,p_email,p_password,p_birth_date,p_address,p_mobile_nu
mber,p_role_id);
END;
$$;

CALL hc.insert_user('om','patel','om@gmail.com','om123','2005-01-
17','bhavnagar','3214569875','8f6ae340-be81-48ae-81b3-ada4d87653b6');

select * from hc.users;
```

-- b. Create a stored procedure that:
-- i. Soft deletes a user (is_deleted = true)

```
CREATE OR REPLACE PROCEDURE hc.soft_user_delete(id UUID)
LANGUAGE plpgsql
AS $$
BEGIN

UPDATE hc.users
SET is_deleted = true
WHERE user_id = id;

END;
$$;
```

```
CALL hc.soft_user_delete('472bf73e-9a95-4dd7-816b-2fcdf1d48367');

-- c. Create a stored procedure that:
-- i. Updates user role
-- d. Logs old role and new role into an audit table

CREATE TABLE hc.user_role_audit (
   audit_id UUID DEFAULT gen_random_uuid(),
   user_id UUID,
   old_role_id UUID,
   new_role_id UUID,
   changed_at TIMESTAMPTZ DEFAULT NOW()
);


CREATE OR REPLACE PROCEDURE hc.update_user_role(id UUID,p_new_role_id UUID)
LANGUAGE plpgsql
AS $$
DECLARE v_old_role_id UUID;
BEGIN

SELECT role_id INTO v_old_role_id FROM hc.users WHERE user_id = id;

INSERT INTO hc.user_role_audit (user_id,old_role_id, new_role_id)
values (id , v_old_role_id , p_new_role_id);

UPDATE hc.users
SET role_id = p_new_role_id
WHERE user_id = id;

END;
$$;

CALL hc.update_user_role('472bf73e-9a95-4dd7-816b-2fcdf1d48367','5aed1713-d347-473e-
b223-9be3991ec12e');
   --OLD_id =>  "8f6ae340-be81-48ae-81b3-ada4d87653b6"

SELECT * FROM hc.user_role_audit;
```

-- 4. Triggers
-- a. Create a trigger that:
-- i. Automatically updates modified_date on UPDATE of users
-- table

```
CREATE OR REPLACE FUNCTION hc.update_modified_date()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
        NEW.modified_date = CURRENT_DATE;
        RETURN NEW;
END;
$$;

CREATE TRIGGER trg_update_modified_date
BEFORE UPDATE
ON hc.users
FOR EACH ROW
EXECUTE FUNCTION hc.update_modified_date();

UPDATE hc.users
SET is_active = false
WHERE user_id = '472bf73e-9a95-4dd7-816b-2fcdf1d48367';

SELECT * FROM hc.users;
```

-- b. Create a trigger that:
-- i. Prevents deletion of users
-- ii. Raises an exception if DELETE is attempted

```
CREATE OR REPLACE FUNCTION hc.prevent_deletion()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
   RAISE EXCEPTION 'Deletion is not allowed';
   RETURN OLD;
```

```
END;
$$;


CREATE TRIGGER trg_prevent_deletion
BEFORE DELETE
ON hc.users
FOR EACH ROW
EXECUTE FUNCTION hc.prevent_deletion();

DELETE FROM hc.users
WHERE user_id ='472bf73e-9a95-4dd7-816b-2fcdf1d48367'

-- c. Create an AFTER INSERT trigger to:
-- i. Log user creation into an audit table

CREATE TABLE hc.user_audit (
   audit_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
   user_id UUID NOT NULL,
   created_at TIMESTAMP DEFAULT NOW()
);

CREATE OR REPLACE FUNCTION hc.user_log()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
   INSERT INTO hc.user_audit (user_id)
   VALUES (NEW.user_id);

   RETURN NEW;
END;
$$;

CREATE TRIGGER trg_user_creation
AFTER INSERT
ON hc.users
FOR EACH ROW
EXECUTE FUNCTION hc.user_log();
```

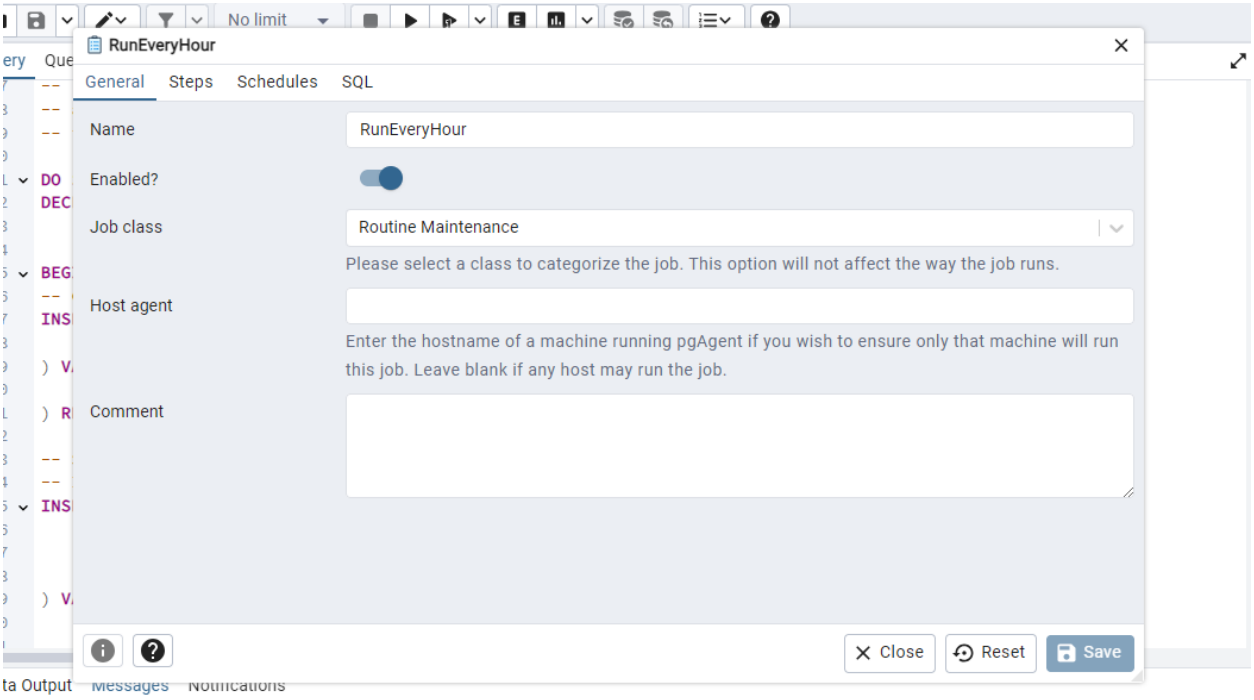
```
INSERT INTO hc.users
(first_name,last_name,email,password,birth_date,address,mobile_number,role_id)
VALUES('nitesh','kukreja','nitesh@gmail.com','nitesh123','2000-02-
10','MP','1236547895','b1e91a70-9720-4136-91e8-1633f2a4adef');

select * from hc.user_audit;
```

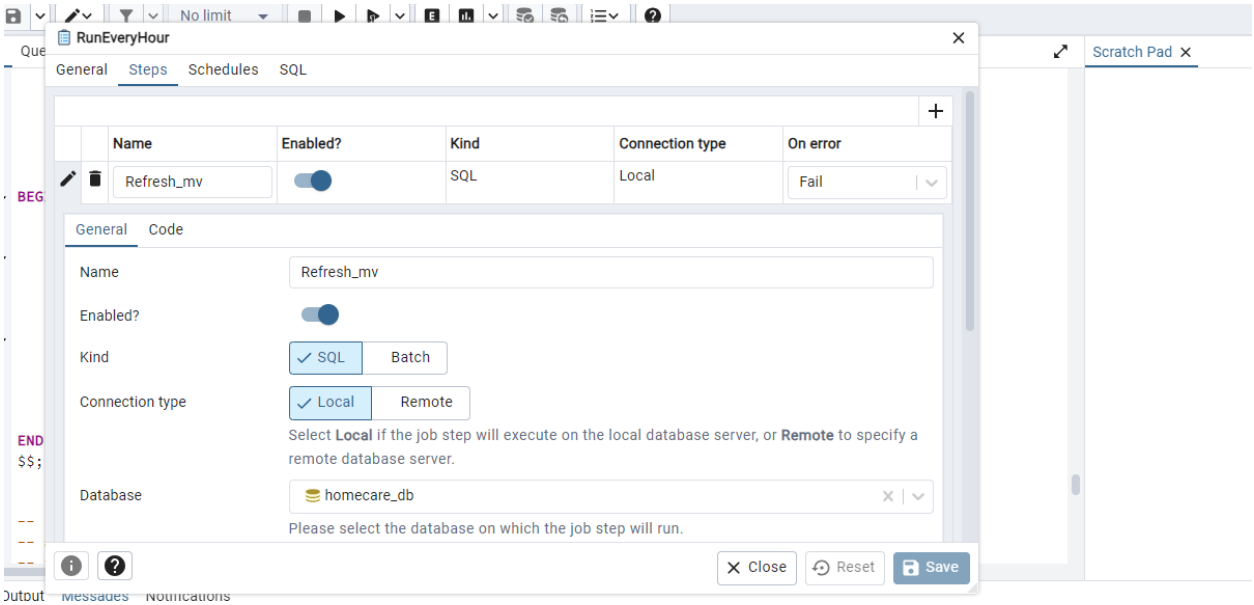
-- 5. Cursors
-- a. Write a PL/pgSQL block using a cursor to:
-- i. Loop through all users
-- ii. Print user_id and email using RAISE NOTICE

```
DO $$
DECLARE
   user_cursor CURSOR FOR
      SELECT user_id, email
      FROM hc.users;

         id UUID;
         user_email VARCHAR;

BEGIN
   OPEN user_cursor;

   LOOP
      FETCH user_cursor INTO id,user_email;
      EXIT WHEN NOT FOUND;

      RAISE NOTICE 'User ID: %, Email: %',id,
                  user_email;
   END LOOP;

   CLOSE user_cursor;
END;
$$;
```

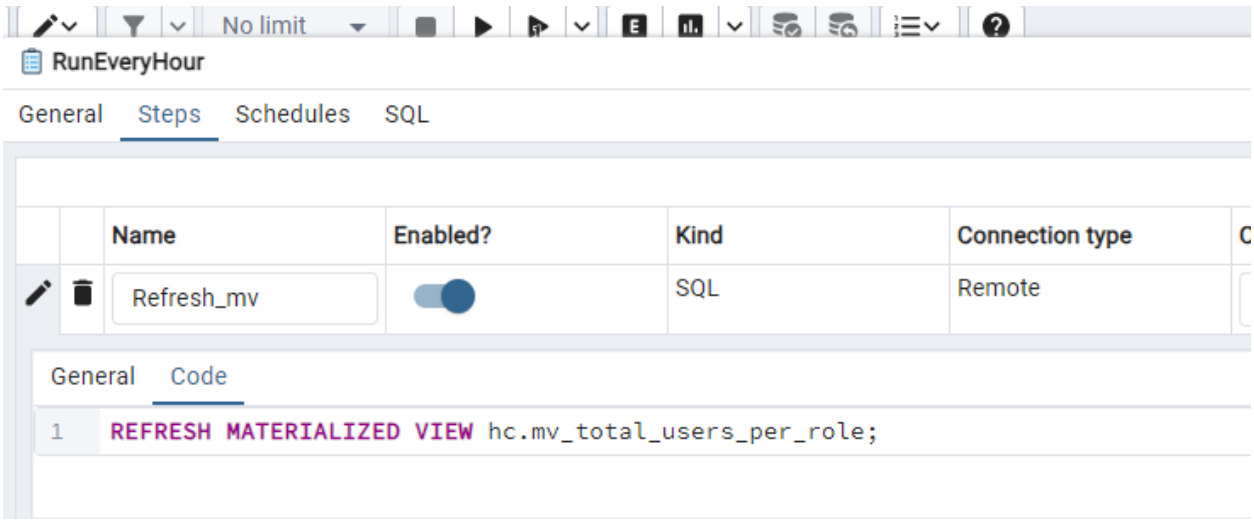
Assignment-3

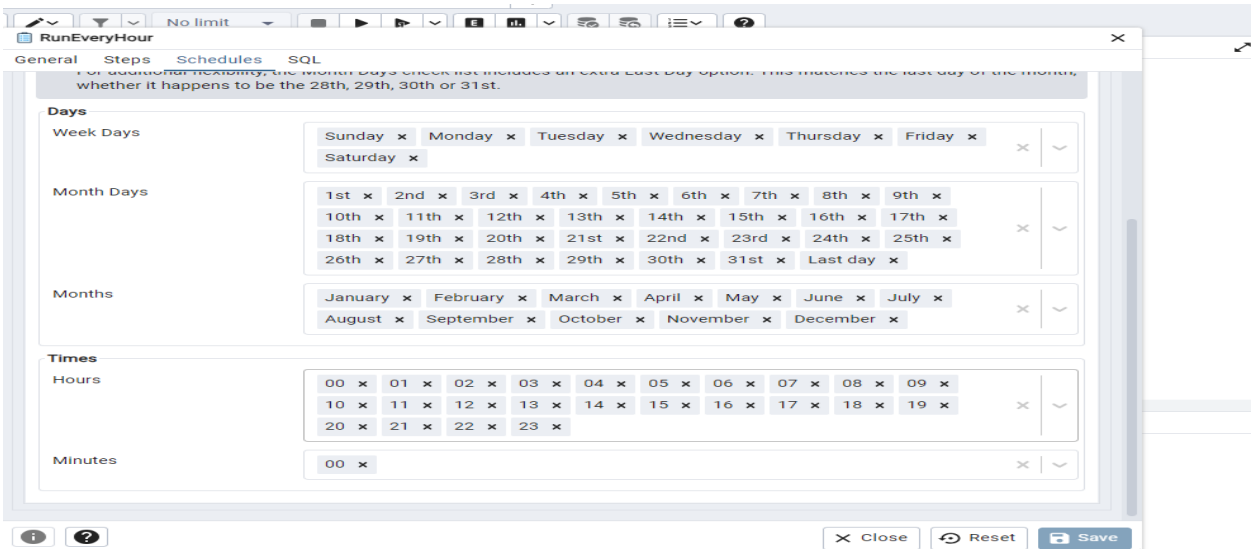
-- Jobs / Scheduling
-- a. Write a job to:
-- i. Refresh a materialized view every hour



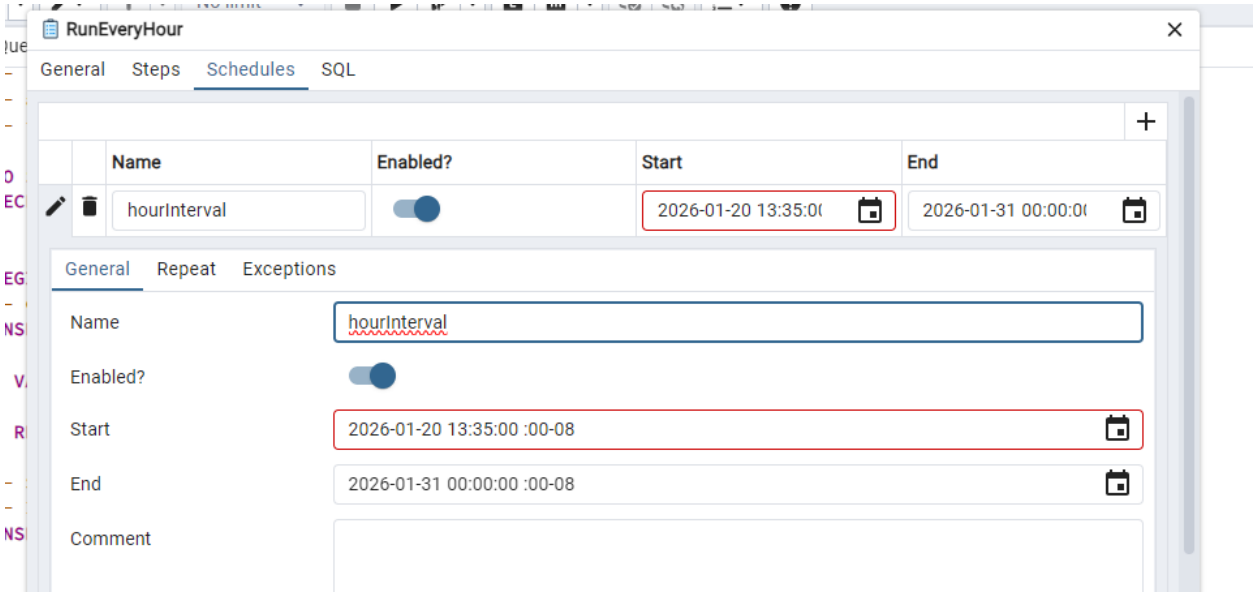


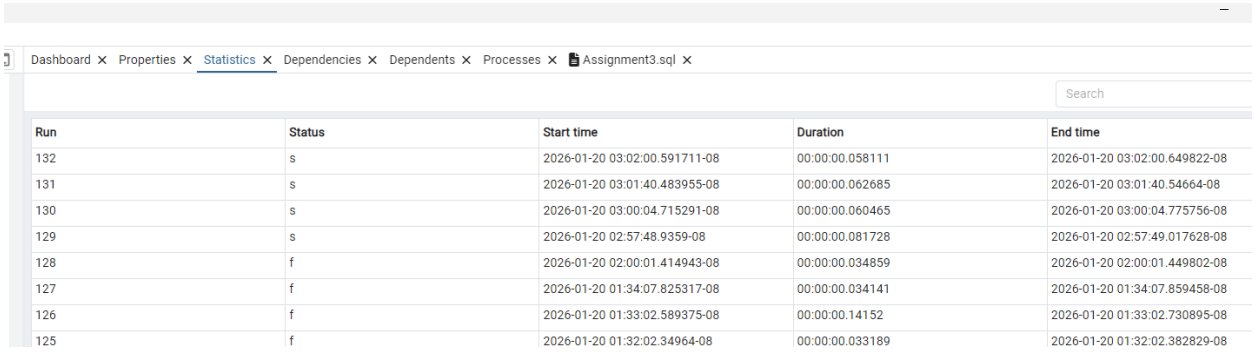
Assignment-3





Assignment-3





Run	Status	Start time	Duration	End time
132	s	2026-01-20 03:02:00.591711-08	00:00:00.058111	2026-01-20 03:02:00.649822-08
131	s	2026-01-20 03:01:40.483955-08	00:00:00.062685	2026-01-20 03:01:40.54664-08
130	s	2026-01-20 03:00:04.715291-08	00:00:00.060465	2026-01-20 03:00:04.775756-08
129	s	2026-01-20 02:57:48.9359-08	00:00:00.081728	2026-01-20 02:57:49.017628-08
128	f	2026-01-20 02:00:01.414943-08	00:00:00.034859	2026-01-20 02:00:01.449802-08
127	f	2026-01-20 01:34:07.825317-08	00:00:00.034141	2026-01-20 01:34:07.859458-08
126	f	2026-01-20 01:33:02.589375-08	00:00:00.14152	2026-01-20 01:33:02.730895-08
125	f	2026-01-20 01:32:02.34964-08	00:00:00.033189	2026-01-20 01:32:02.382829-08