Database Triggers and Functions

1. After a RentRequest is accepted then reject all other requests for that property during that tenure

```
CREATE OR REPLACE FUNCTION updateRentRequests(
  i INT,
  sd TIMESTAMP WITH TIME ZONE,
  ed TIMESTAMP WITH TIME ZONE
) RETURNS VOID AS $$
BEGIN
  UPDATE "RentRequest"
  SET "adminStatus" = 'REJECTED', "ownerStatus" = 'REJECTED'
  WHERE "itemId" = i
    AND "startDate" >= sd
    AND "endDate" <= ed;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION request_accepted()
RETURNS TRIGGER AS $$
BEGIN
  -- Insert the accepted request into ActiveRent
  INSERT INTO "ActiveRent" ("itemId", "userId", "startDate", "endDate", "price")
  VALUES (NEW."itemId", NEW."userId", NEW."startDate", NEW."endDate", NEW.price);

  -- Delete the accepted request from RentRequest
  -- DELETE FROM "RentRequest" WHERE id = NEW.id;

  -- Update other overlapping requests to be rejected
  PERFORM updateRentRequests(NEW."itemId", NEW."startDate", NEW."endDate");

  RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER request_accepted
AFTER UPDATE ON "RentRequest"
FOR EACH ROW
WHEN (NEW."ownerStatus" = 'ACCEPTED' AND NEW."adminStatus" = 'ACCEPTED')
EXECUTE FUNCTION request_accepted();
```

2. If an Item is deleted i.e. userId is set to 0, then 'REJECT' all the requests for that item and update user to 0 which is pointing to NULL user to avoid foreign key constraint and deleting user renting history

```sql
CREATE OR REPLACE FUNCTION rejectRentRequestsOnUserUpdate()
RETURNS TRIGGER AS $$
BEGIN
  -- Check if the userId is updated to 0
  IF NEW."userId" = 0 AND OLD."userId" IS DISTINCT FROM 0 THEN
    -- Reject all RentRequests with the same itemId
    UPDATE "RentRequest"
    SET adminStatus = 'REJECTED', ownerStatus = 'REJECTED'
    WHERE "itemId" = NEW.id;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create a trigger to execute the function before each update on Item
CREATE TRIGGER reject_rent_requests_trigger
BEFORE UPDATE ON "Item"
FOR EACH ROW
EXECUTE FUNCTION rejectRentRequestsOnUserUpdate();
```

3. If a property is made unavailable then you can't create any more new requests for that property

```sql
CREATE OR REPLACE FUNCTION checkItemAvailability()
RETURNS TRIGGER AS $$
BEGIN
  IF NOT EXISTS (
    SELECT 1
    FROM "Item"
    WHERE "Item"."id" = NEW."itemId"
      AND "Item"."isAvailable" = true
  ) THEN
    RAISE EXCEPTION 'The requested item is not available';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_item_availability_trigger
BEFORE INSERT ON "RentRequest"
FOR EACH ROW
EXECUTE FUNCTION checkItemAvailability();
```

4. As soon as User Created we need to initialize empty cart

```
CREATE OR REPLACE FUNCTION createCartForNewUser()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO "Cart" ("userId", "value")
  VALUES (NEW.id, 0);

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER create_cart_after_user_insert
AFTER INSERT ON "User"
FOR EACH ROW
EXECUTE FUNCTION createCartForNewUser();
```

5. On Update, Add or Delete of CartItem, update the cart value

```
CREATE OR REPLACE FUNCTION updateCartOnCartItemInsert()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE "Cart"
  SET "value" = "value" + NEW."price"
  WHERE "id" = NEW."cartId";

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION updateCartOnCartItemDelete()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE "Cart"
  SET "value" = "value" - OLD."price"
  WHERE "id" = OLD."cartId";

  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION updateCartOnCartItemUpdate()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE "Cart"
  SET "value" = "value" - OLD."price" + NEW."price"
  WHERE "id" = NEW."cartId";

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_cart_on_cartitem_insert
AFTER INSERT ON "CartItem"
```

```
FOR EACH ROW
EXECUTE FUNCTION updateCartOnCartItemInsert();

CREATE TRIGGER update_cart_on_cartitem_delete
AFTER DELETE ON "CartItem"
FOR EACH ROW
EXECUTE FUNCTION updateCartOnCartItemDelete();

CREATE TRIGGER update_cart_on_cartitem_update
AFTER UPDATE ON "CartItem"
FOR EACH ROW
EXECUTE FUNCTION updateCartOnCartItemUpdate();
```

5. After user create, automatically update his profile pic

```
CREATE OR REPLACE FUNCTION updateUserProfilePic()
RETURNS TRIGGER AS $$
BEGIN
  UPDATE "User"
  SET "profilePic" =
'https://aniaodrkdkwrtfkhpjgp.supabase.co/storage/v1/object/public/profile-photos/'
|| NEW."id" || '/profile'
  WHERE "id" = NEW."id";

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_user_profile_pic
AFTER INSERT ON "User"
FOR EACH ROW
EXECUTE FUNCTION updateUserProfilePic();
```

6. Procedure to migrate ActiveRent to OverRent Need to enable `pgcron` extension and schedule the procedure to run daily

```
select cron.schedule(
  're',
  '0 0 * * *',
  $$ CALL moveactiverenttooverrent(); $$
);
```

```
CREATE
OR REPLACE PROCEDURE moveactiverenttooverrent () AS $$
BEGIN

  INSERT INTO "OverRent" ("itemId", "userId", "startDate", "endDate", "isPaid",
"price")
  SELECT
    "itemId",
    "userId",
    "startDate",
    "endDate",
    "isPaid",
    "price"
  FROM "ActiveRent"
  WHERE  ("endDate"::TIMESTAMP AT TIME ZONE 'Asia/Kolkata') < (CURRENT_TIMESTAMP AT
TIME ZONE 'Asia/Kolkata')::TIMESTAMP;


  DELETE FROM "ActiveRent"
  WHERE  ("endDate"::TIMESTAMP AT TIME ZONE 'Asia/Kolkata') < (CURRENT_TIMESTAMP AT
TIME ZONE 'Asia/Kolkata')::TIMESTAMP;

END;
$$ LANGUAGE plpgsql;

-- select * from cron.job_run_details;
-- select * from cron.job;
-- select cron.unschedule(9);
```