```
CREATE OR REPLACE FUNCTION update_village_production()
RETURNS TRIGGER AS $$
DECLARE
   village_total INT;
BEGIN
   -- Calculate the total production for the specific crop in the village
   SELECT SUM(p.quantity) INTO village_total
   FROM Produces p
   WHERE p.village_no = NEW.village_no
     AND p.district_no = NEW.district_no
     AND p.state_no = NEW.state_no
           AND p.crop_id= NEW.crop_id;

   -- Insert or update the village production for each crop based on the total
   INSERT INTO village_production (crop_id, village_no, district_no, state_no, quantity)
   VALUES (NEW.crop_id, NEW.village_no, NEW.district_no, NEW.state_no, village_total)
   ON CONFLICT (crop_id, village_no, district_no, state_no) -- Assuming these columns
form the unique constraint
   DO UPDATE SET quantity = village_total;  -- Update quantity if the record already exists

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER trg_update_village_production
AFTER INSERT OR UPDATE ON produces  -- Replace 'Land' with your table
FOR EACH ROW
EXECUTE FUNCTION update_village_production();



CREATE OR REPLACE FUNCTION update_district_production_function()
          RETURNS TRIGGER AS $$
DECLARE
   district_total INT;
BEGIN
   -- Calculate the total production for the specific crop in the district
   SELECT SUM(quantity) INTO district_total
   FROM village_production
   WHERE district_no = NEW.district_no
     AND state_no = NEW.state_no
     AND crop_id = NEW.crop_id;

   -- Insert or update the district production for the crop and year
   INSERT INTO district_production (crop_id, district_no, state_no, quantity, year_)
   VALUES (NEW.crop_id, NEW.district_no, NEW.state_no, district_total)
```

```
    ON CONFLICT (crop_id, district_no, state_no)  -- Adjust based on your unique constraints
    DO UPDATE SET quantity = district_total;  -- Update quantity if the record already exists

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION update_state_production_function()
RETURNS TRIGGER AS $$
DECLARE
    state_total INT;
BEGIN
    -- Calculate new total production for the crop in the state
    SELECT SUM(quantity) INTO state_total
    FROM district_production
    WHERE state_no = NEW.state_no
      AND crop_id = NEW.crop_id; -- Use the year from the updated row

    -- Update or insert state production for that crop
    INSERT INTO state_production (crop_id, state_no, quantity)
    VALUES (NEW.crop_id, NEW.state_no, state_total)
    ON CONFLICT (crop_id, state_no)  -- Assuming these columns form the unique constraint
    DO UPDATE SET quantity = state_total;  -- Update quantity if the record already exists

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER update_state_production
AFTER INSERT OR UPDATE ON district_production
FOR EACH ROW
EXECUTE FUNCTION update_state_production_function();

CREATE TRIGGER update_district_production
AFTER INSERT OR UPDATE ON village_production
FOR EACH ROW
EXECUTE FUNCTION update_district_production_function();

--check before delete on village production
CREATE FUNCTION prevent_delete_from_village_production()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if there is any row in Produces with the same crop_id, village_no, district_no, and
state_no
    IF EXISTS (
        SELECT 1
```

```
        FROM Produces
        WHERE crop_id = OLD.crop_id
          AND village_no = OLD.village_no
          AND district_no = OLD.district_no
          AND state_no = OLD.state_no
    ) THEN
        RAISE EXCEPTION 'Cannot delete row from village_production as it is referenced in
Produces';
    END IF;

    RETURN OLD; -- Proceed with deletion if no match is found
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_produces_before_delete_on_village
BEFORE DELETE ON village_production
FOR EACH ROW
EXECUTE FUNCTION prevent_delete_from_village_production();

--check before delete on district production
CREATE FUNCTION prevent_delete_from_district_production()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if there is any row in Produces with the same crop_id, district_no, and state_no
    IF EXISTS (
        SELECT 1
        FROM Produces
        WHERE crop_id = OLD.crop_id
          AND district_no = OLD.district_no
          AND state_no = OLD.state_no
    ) THEN
        RAISE EXCEPTION 'Cannot delete row from district_production as it is referenced in
Produces';
    END IF;

    RETURN OLD; -- Proceed with deletion if no match is found
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_produces_before_delete_on_district
BEFORE DELETE ON district_production
FOR EACH ROW
EXECUTE FUNCTION prevent_delete_from_district_production();

--check before delete for state production
CREATE FUNCTION prevent_delete_from_state_production()
RETURNS TRIGGER AS $$
BEGIN
```

```
    -- Check if there is any row in Produces with the same crop_id and state_no
    IF EXISTS (
        SELECT 1
        FROM Produces
        WHERE crop_id = OLD.crop_id
          AND state_no = OLD.state_no
    ) THEN
        RAISE EXCEPTION 'Cannot delete row from state_production as it is referenced in
Produces';
    END IF;

    RETURN OLD; -- Proceed with deletion if no match is found
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_produces_before_delete_on_state
BEFORE DELETE ON state_production
FOR EACH ROW
EXECUTE FUNCTION prevent_delete_from_state_production();


--update after delete in produces
CREATE FUNCTION update_on_delete_village_production_quantity()
RETURNS TRIGGER AS $$
BEGIN
    -- Update the quantity in village_production by summing the remaining quantities in
Produces
    UPDATE village_production
    SET quantity = COALESCE(
        (SELECT SUM(quantity)
         FROM Produces
         WHERE village_no = OLD.village_no
           AND district_no = OLD.district_no
           AND state_no = OLD.state_no),
        0) -- Set quantity to 0 if no matching rows remain in Produces
    WHERE village_no = OLD.village_no
      AND district_no = OLD.district_no
      AND state_no = OLD.state_no;

    -- Delete the row from village_production if the updated quantity is zero
    DELETE FROM village_production
    WHERE village_no = OLD.village_no
      AND district_no = OLD.district_no
      AND state_no = OLD.state_no
      AND quantity = 0;

    RETURN NULL; -- No need to return a row for AFTER DELETE trigger
END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_quantity_after_delete
AFTER DELETE ON Produces
FOR EACH ROW
EXECUTE FUNCTION update_on_delete_village_production_quantity();

--to delete any row that has quantity equal to zero
CREATE FUNCTION delete_district_production_if_zero()
RETURNS TRIGGER AS $$
BEGIN
   -- Check if the quantity is zero
   IF NEW.quantity = 0 THEN
      -- Delete the row from district_production if quantity is zero
      DELETE FROM district_production
      WHERE crop_id = NEW.crop_id
        AND district_no = NEW.district_no
        AND state_no = NEW.state_no;

      -- Prevent the INSERT or UPDATE by returning NULL
      RETURN NULL;
   END IF;

   -- Allow the operation to proceed if quantity is not zero
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_if_quantity_zero_on_district
BEFORE INSERT OR UPDATE ON district_production
FOR EACH ROW
EXECUTE FUNCTION delete_district_production_if_zero();


--to delete any row that has quantity equal to zero
CREATE FUNCTION delete_state_production_if_zero()
RETURNS TRIGGER AS $$
BEGIN
   -- Check if the quantity is zero
   IF NEW.quantity = 0 THEN
      -- Delete the row from state_production if quantity is zero
      DELETE FROM state_production
      WHERE crop_id = NEW.crop_id
        AND state_no = NEW.state_no;

      -- Prevent the INSERT or UPDATE by returning NULL
      RETURN NULL;
   END IF;
```

```
    -- Allow the operation to proceed if quantity is not zero
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_if_quantity_zero_on_state
BEFORE INSERT OR UPDATE ON state_production
FOR EACH ROW
EXECUTE FUNCTION delete_state_production_if_zero();


--to automatically insert buyer with land that satisfy his requirement in wants table
CREATE OR REPLACE FUNCTION insert_into_wants_for_new_buyer()
RETURNS TRIGGER AS $$
BEGIN
    -- Insert the buyer-specific data into WANTS
    INSERT INTO WANTS (land_no, village_no, district_no, state_no, buyer_contact_no)
    SELECT DISTINCT L.land_no, L.village_no, L.district_no, L.state_no,
NEW.buyer_contact_no
    FROM Land L
    JOIN Buyer_village_requirement BVR
        ON L.village_no = BVR.required_village_no
        AND L.district_no = BVR.required_district_no
        AND L.state_no = BVR.required_state_no
    WHERE BVR.buyer_contact_no = NEW.buyer_contact_no

    UNION

    SELECT DISTINCT L.land_no, L.village_no, L.district_no, L.state_no,
NEW.buyer_contact_no
    FROM Land L
    JOIN Buyer B
        ON L.soil_type = B.required_soil_type
    WHERE B.buyer_contact_no = NEW.buyer_contact_no

    UNION

    SELECT DISTINCT L.land_no, L.village_no, L.district_no, L.state_no,
NEW.buyer_contact_no
    FROM Land L
    JOIN Buyer B
        ON L.area BETWEEN B.required_land_area - 10 AND B.required_land_area + 10
    WHERE B.buyer_contact_no = NEW.buyer_contact_no

    UNION
```

```
    SELECT DISTINCT L.land_no, L.village_no, L.district_no, L.state_no,
BC.buyer_contact_no
    FROM Land L
    JOIN Grows_on GO
        ON L.soil_type = GO.soil_id
    JOIN Buyer_crops BC
        ON GO.crop_id = BC.crops_offered
    WHERE BC.buyer_contact_no = NEW.buyer_contact_no;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER buyer_insert_trigger
AFTER INSERT ON Buyer
FOR EACH ROW
EXECUTE FUNCTION insert_into_wants_for_new_buyer();
```