Task 1: Build all six tables with constraints

Explanation:
This task creates the database structure with proper constraints:

- **PRIMARY KEY** ensures each record has a unique identifier
- **FOREIGN KEY** establishes relationships between tables
- **NOT NULL** prevents missing critical data
- **CHECK** constraints validate data (e.g., positive values, valid statuses)
- **DEFAULT** values provide sensible initial values

Key constraints:

- Field size must be positive
- Crop status must be one of predefined values
- Harvest date cannot be before planting date
- Quantities and costs must be positive

Task 2: Apply CASCADE DELETE (Already implemented in Task 1)

Explanation:
CASCADE DELETE ensures data integrity when deleting records:

- When a **Crop** is deleted, all related **Harvest** records are automatically deleted
- When a **Harvest** record is deleted, all related **Sale** records are automatically deleted
- This prevents orphaned records and maintains referential integrity
- **Note:** Already implemented in Task 1 table definitions

Task 3: Insert sample data

Explanation:
This populates the database with realistic test data:

- **3 fields** with different sizes and soil types
- **5 crops** across different fields with various planting/harvest dates

- **5 workers** with different roles and wages
- **Fertilizer applications** showing input costs
- **Harvest records** tracking yield quantities and quality grades
- **Sales transactions** recording actual sales data

This data will allow us to test all subsequent queries effectively.

```
74  -- Insert 3 fields
75  INSERT INTO Field (FieldID, FieldName, SizeHectares, Location, SoilType) VALUES
76  (1, 'North Field', 5.0, 'Northern Section', 'Loamy'),
77  (2, 'South Field', 3.5, 'Southern Section', 'Clay'),
78  (3, 'East Field', 4.2, 'Eastern Section', 'Sandy Loam');
79
80  SELECT * FROM Field
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of

	fieldid [PK] integer	fieldname character varying (100)	sizehectares numeric (10,2)	location character varying (100)	soiltype character varying (50)
1	1	North Field	5.00	Northern Section	Loamy
2	2	South Field	3.50	Southern Section	Clay
3	3	East Field	4.20	Eastern Section	Sandy Loam

```sql
-- Insert 5 crops
INSERT INTO Crop (CropID, FieldID, CropName, PlantingDate, HarvestDate, Status) VALUES
(1, 1, 'Maize', '2024-03-15', '2024-07-20', 'Harvested'),
(2, 1, 'Beans', '2024-04-01', '2024-06-30', 'Sold'),
(3, 2, 'Wheat', '2024-03-20', '2024-07-15', 'Harvested'),
(4, 2, 'Tomatoes', '2024-04-10', '2024-07-05', 'Ready'),
(5, 3, 'Potatoes', '2024-03-25', '2024-07-25', 'Growing');

SELECT * FROM Crop
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of

	cropid [PK] integer	fieldid integer	cropname character varying (100)	plantingdate date	harvestdate date	status character varying (20)
1	1	1	Maize	2024-03-15	2024-07-20	Harvested
2	2	1	Beans	2024-04-01	2024-06-30	Sold
3	3	2	Wheat	2024-03-20	2024-07-15	Harvested
4	4	2	Tomatoes	2024-04-10	2024-07-05	Ready
5	5	3	Potatoes	2024-03-25	2024-07-25	Growing

```sql
-- Insert 5 workers
INSERT INTO Worker (WorkerID, FullName, Role, Contact, DailyWage) VALUES
(1, 'John Kamau', 'Harvester', '0712345678', 1200.00),
(2, 'Mary Wanjiku', 'Supervisor', '0723456789', 2000.00),
(3, 'Peter Omondi', 'Planter', '0734567890', 1000.00),
(4, 'Grace Achieng', 'Harvester', '0745678901', 1200.00),
(5, 'James Mwangi', 'Irrigator', '0756789012', 1100.00);

SELECT * FROM Worker
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	workerid [PK] integer	fullname character varying (100)	role character varying (50)	contact character varying (15)	dailywage numeric (10,2)
1	1	John Kamau	Harvester	0712345678	1200.00
2	2	Mary Wanjiku	Supervisor	0723456789	2000.00
3	3	Peter Omondi	Planter	0734567890	1000.00
4	4	Grace Achieng	Harvester	0745678901	1200.00
5	5	James Mwangi	Irrigator	0756789012	1100.00

```sql
-- Insert sample fertilizers
INSERT INTO Fertilizer (FertilizerID, CropID, Name, QuantityUsed, Cost, DateApplied) VALUES
(1, 1, 'NPK 17:17:17', 50.0, 2500.00, '2024-04-01'),
(2, 1, 'Urea', 25.0, 1500.00, '2024-05-15'),
(3, 2, 'DAP', 30.0, 1800.00, '2024-04-10'),
(4, 3, 'NPK 20:20:20', 40.0, 2200.00, '2024-04-05'),
(5, 4, 'Organic Compost', 60.0, 3000.00, '2024-04-20');

SELECT * FROM Fertilizer
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	fertilizerid [PK] integer	cropid integer	name character varying (100)	quantityused numeric (10,2)	cost numeric (10,2)	dateapplied date
1	1	1	NPK 17:17:17	50.00	2500.00	2024-04-01
2	2	1	Urea	25.00	1500.00	2024-05-15
3	3	2	DAP	30.00	1800.00	2024-04-10
4	4	3	NPK 20:20:20	40.00	2200.00	2024-04-05
5	5	4	Organic Compost	60.00	3000.00	2024-04-20

```sql
-- Insert sample harvests
INSERT INTO Harvest (HarvestID, CropID, WorkerID, QuantityKG, DateCollected, Grade, Buyer) VALUES
(1, 1, 1, 1500.5, '2024-07-20', 'A', 'GreenMart Ltd'),
(2, 2, 4, 800.75, '2024-06-30', 'B', 'FreshProduce Co'),
(3, 3, 1, 1200.25, '2024-07-15', 'A', 'GrainCorp'),
(4, 4, 4, 600.0, '2024-07-05', 'B', 'Local Market'),
(5, 1, 3, 500.0, '2024-07-25', 'A', 'Export Quality');

SELECT * FROM Harvest
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1 of 1

	harvestid [PK] integer	cropid integer	workerid integer	quantitykg numeric (10,2)	datecollected date	grade character varying (10)	buyer character varying (100)
1	1	1	1	1500.50	2024-07-20	A	GreenMart Ltd
2	2	2	4	800.75	2024-06-30	B	FreshProduce Co
3	3	3	1	1200.25	2024-07-15	A	GrainCorp
4	4	4	4	600.00	2024-07-05	B	Local Market
5	5	1	3	500.00	2024-07-25	A	Export Quality

```sql
-- Insert sample sales
INSERT INTO Sale (SaleID, HarvestID, Buyer, QuantitySold, PricePerKG, SaleDate) VALUES
(1, 1, 'GreenMart Ltd', 1500.5, 45.00, '2024-07-21'),
(2, 2, 'FreshProduce Co', 800.75, 60.00, '2024-07-01'),
(3, 3, 'GrainCorp', 1200.25, 50.00, '2024-07-16'),
(4, 4, 'Local Market', 600.0, 40.00, '2024-07-06'),
(5, 5, 'Export Quality', 500.0, 70.00, '2024-07-26');

SELECT * FROM Sale
```

Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	saleid [PK] integer	harvestid integer	buyer character varying (100)	quantitysold numeric (10,2)	priceperkg numeric (10,2)	saledate date
1	1	1	GreenMart Ltd	1500.50	45.00	2024-07-...
2	2	2	FreshProduce Co	800.75	60.00	2024-07-...
3	3	3	GrainCorp	1200.25	50.00	2024-07-...
4	4	4	Local Market	600.00	40.00	2024-07-...
5	5	5	Export Quality	500.00	70.00	2024-07-...

Task 4: Retrieve harvest yield per field

Explanation:
This query analyzes production efficiency by:

- **SUM(h.QuantityKG)** calculates total harvest weight per field
- **COUNT(DISTINCT c.CropID)** shows crop diversity per field
- **JOIN** operations connect Field → Crop → Harvest tables
- **GROUP BY** aggregates data at field level
- Results help identify most productive fields and optimize resource allocation

```sql
-- Task 4: Retrieve harvest yield per field

SELECT
    f.FieldName,
    f.Location,
    SUM(h.QuantityKG) AS TotalHarvestKG,
    COUNT(DISTINCT c.CropID) AS NumberOfCrops
FROM Field f
JOIN Crop c ON f.FieldID = c.FieldID
JOIN Harvest h ON c.CropID = h.CropID
GROUP BY f.FieldID, f.FieldName, f.Location
ORDER BY TotalHarvestKG DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 2

	fieldname character varying (100)	location character varying (100)	totalharvestkg numeric	numberofcrops bigint
1	North Field	Northern Section	2801.25	2
2	South Field	Southern Section	1800.25	2

Task 5: Update crop status after sale completion

Explanation:
This automates business workflow by:

- Identifying crops that have been sold (have sales records)
- Updating their status from 'Harvested' or 'Ready' to 'Sold'
- Using a **subquery** to find crops with sales transactions
- Maintaining accurate inventory and production status
- Can be scheduled to run automatically after sales are recorded

```sql
-- Task 5: Update crop status after sale completion

-- Update crop status to 'Sold' when sales are completed
UPDATE Crop
SET Status = 'Sold'
WHERE CropID IN (
    SELECT DISTINCT c.CropID
    FROM Crop c
    JOIN Harvest h ON c.CropID = h.CropID
    JOIN Sale s ON h.HarvestID = s.HarvestID
    WHERE c.Status != 'Sold'
);

-- Verify the update
SELECT CropID, CropName, Status FROM Crop;
```

Data Output | Messages | Notifications

	cropid [PK] integer	cropname character varying (100)	status character varying (20)
1	2	Beans	Sold
2	5	Potatoes	Growing
3	1	Maize	Sold
4	3	Wheat	Sold
5	4	Tomatoes	Sold

Task 6: Identify the most profitable crop of the season

Explanation:
This performs cost-benefit analysis by:

- Calculating **Total Revenue** (Quantity Sold × Price per KG)
- Calculating **Total Fertilizer Cost** from fertilizer applications
- Computing **Net Profit** (Revenue - Fertilizer Cost)
- Using **subquery** to aggregate fertilizer costs per crop
- **ORDER BY NetProfit DESC** ranks crops by profitability
- **LIMIT 1** returns only the most profitable crop

```sql
-- Task 6: Identify the most profitable crop of the season

SELECT
    c.CropName,
    f.FieldName,
    SUM(s.QuantitySold * s.PricePerKG) AS TotalRevenue,
    SUM(fert.TotalFertilizerCost) AS TotalFertilizerCost,
    (SUM(s.QuantitySold * s.PricePerKG) - SUM(fert.TotalFertilizerCost)) AS NetProfit
FROM Crop c
JOIN Field f ON c.FieldID = f.FieldID
JOIN Harvest h ON c.CropID = h.CropID
JOIN Sale s ON h.HarvestID = s.HarvestID
JOIN (
    SELECT CropID, SUM(Cost) AS TotalFertilizerCost
    FROM Fertilizer
    GROUP BY CropID
) fert ON c.CropID = fert.CropID
GROUP BY c.CropID, c.CropName, f.FieldName
ORDER BY NetProfit DESC
LIMIT 1;
```

Data Output | Messages | Notifications

	cropname character varying (100)	fieldname character varying (100)	totalrevenue numeric	totalfertilizercost numeric	netprofit numeric
1	Maize	North Field	102522.5000	8000.00	94522.5000

Task 7: Create a view summarizing total fertilizer cost per crop

Explanation:
This creates a reusable summary view that:

- **Aggregates fertilizer data** per crop (total cost, quantity, applications)
- **Joins with field information** for context
- Provides **pre-calculated metrics** like average cost per application
- Can be used in multiple reports without rewriting complex queries
- Updates automatically when underlying data changes

```sql
-- Task 7: Create a view summarizing total fertilizer cost per crop

CREATE VIEW CropFertilizerCostSummary AS
SELECT
    c.CropID,
    c.CropName,
    f.FieldName,
    COUNT(fert.FertilizerID) AS NumberOfApplications,
    SUM(fert.QuantityUsed) AS TotalQuantityUsed,
    SUM(fert.Cost) AS TotalFertilizerCost,
    AVG(fert.Cost) AS AverageCostPerApplication
FROM Crop c
JOIN Field f ON c.FieldID = f.FieldID
LEFT JOIN Fertilizer fert ON c.CropID = fert.CropID
GROUP BY c.CropID, c.CropName, f.FieldName;

-- Query the view
SELECT * FROM CropFertilizerCostSummary ORDER BY TotalFertilizerCost DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5

	cropid integer	cropname character varying (100)	fieldname character varying (100)	numberofapplications bigint	totalquantityused numeric	totalfertilizercost numeric	averagecostperapplication numeric
1	5	Potatoes	East Field	0	[null]	[null]	[null]
2	1	Maize	North Field	2	75.00	4000.00	2000.0000000000000000
3	4	Tomatoes	South Field	1	60.00	3000.00	3000.0000000000000000
4	3	Wheat	South Field	1	40.00	2200.00	2200.0000000000000000
5	2	Beans	North Field	1	30.00	1800.00	1800.0000000000000000

Task 8: Implement a trigger preventing fertilizer application before planting date

Explanation:
This ensures data quality by:

- **BEFORE INSERT trigger** validates data before saving
- **Checks if fertilizer application date** is before planting date
- **Raises an error** if validation fails using SIGNAL SQLSTATE
- Prevents illogical data (can't fertilize before planting)
- Maintains temporal consistency in farm operations

```sql
-- Task 8: Implement a trigger preventing fertilizer application before planting date

CREATE OR REPLACE FUNCTION check_fertilizer_date()
RETURNS TRIGGER AS $$
DECLARE
    plant_date DATE;
BEGIN
    -- Get the planting date for the crop
    SELECT PlantingDate INTO plant_date
    FROM Crop
    WHERE CropID = NEW.CropID;

    -- Check if fertilizer application date is before planting date
    IF NEW.DateApplied < plant_date THEN
        RAISE EXCEPTION 'Fertilizer cannot be applied before planting date';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create the trigger
CREATE TRIGGER check_fertilizer_date_trigger
BEFORE INSERT ON Fertilizer
FOR EACH ROW
EXECUTE FUNCTION check_fertilizer_date();

-- Test the trigger (this should fail)
INSERT INTO Fertilizer (FertilizerID, CropID, Name, QuantityUsed, Cost, DateApplied)
VALUES (6, 1, 'Test Fertilizer', 10.0, 500.00, '2024-03-01');
```

Data Output | Messages | Notifications

```
ERROR:  Fertilizer cannot be applied before planting date
CONTEXT:  PL/pgSQL function check_fertilizer_date() line 12 at RAISE
```