Project Response Questions

Problem 1) Provide documentation on why you have the constraints that you specified.

Table: Farmworker

Attribute Name	Constraint(s)	Reasoning
Position	❏ Not Null Constraint ❏ Domain Constraint	Position has a not null constraint because logically, you cannot have an employee without an assigned position. Position also has a domain constraint which ensures that an employee has a 'valid' position on the farm. An example of an invalid position would be a nuclear engineer, since nuclear engineers do not work on farms.
Employee_ID	❏ Uniqueness Constraint ❏ Primary Key Constraint ❏ Entity Integrity Constraint	Employee_ID has a uniqueness constraint because Employee is a primary key, and each primary key must be unique. Each primary key constraint in sql also automatically attaches the not null constraint, as there cannot be a null primary key. This attached not null constraint is called the entity integrity constraint. Employee_ID identifies the specific value in this entity, and as a result is the primary key.
Salary	❏ Check Constraint	Salary has a check constraint to ensure no employee is being paid a negative salary.
Experience	❏ Not Null Constraint ❏ Check Constraint	Experience has a not null constraint because everyone has some sort of experience when working on the farm; Let it be none or 5+ years. It's due to this logic, that Experience also has a check constraint to ensure that it has a value greater than or equal to 0.
RDIF_ Assignment	❏ Check Constraint	RFID_Assignment has a check constraint since no RFID in Beehive can be assigned a negative value.

Table: Livestock

Attribute Name	Constraint(s)	Reasoning
Feed	❏ Not Null Constraint ❏ Domain Constraint	Feed has a not null constraint because logically livestock must eat something. The domain constraint limits input to acceptable forms of feed: “grass”,”seeds.” since not all livestock animals can eat everything.
Type	❏ Not Null Constraint	Type has a not null constraint because logically, you cannot have a “null” animal in real life.
ID_Number	❏ Uniqueness Constraint ❏ Primary Key Constraint ❏ Entity Integrity Constraint	ID_Number has a uniqueness constraint because ID_Number is a primary key, and each primary key must be unique. Each primary key constraint in sql also automatically attaches the not null constraint, as there cannot be a null primary key. This attached not null constraint is called the entity integrity constraint. ID_Number identifies the specific value in this entity, and as a result is the primary key.
Caretaker_ID	❏ Not Null Constraint ❏ Foreign Key Constraint	Caretaker_ID has a foreign constraint because Caretaker points to the primary key of Employee_ID. Since Caretaker_ID is a foreign key, it also cannot be null, resulting in the not null constraint.

Table: Beehive

Attribute Name	Constraint(s)	Reasoning
Indisposition	❏ Not Null Constraint	Indisposition has a not null constraint because logically, a beehive is either “healthy” or it’s not.
Produce	❏ Not Null Constraint ❏ Domain Constraint	Produce has a not null constraint because logically, every beehive creates or contributes to produce. Since the beehive can create or contribute, there is a domain check to limit what a beehive can produce: since logically, a beehive cannot produce steel.
RFID	❏ Uniqueness Constraint ❏ Primary Key Constraint ❏ Entity Integrity Constraint ❏ Not Null Constraint ❏ Check Constraint	RFID has a uniqueness constraint because RFID is a primary key, and each primary key must be unique. Each primary key constraint in sql also automatically attaches the not null constraint, as there cannot be a null primary key. This attached not null constraint is called the entity integrity constraint. RFID identifies the specific value in this entity, and as a result is the primary key. No RFID has a negative value, therefore it cannot be assigned a negative RFID. As a result, there is a check constraint to ensure that RFID is greater than or equal to 0.
Frame	❏ Not Null Constraint	Frame has a not null constraint because logically, bees make their own frames within a ‘super:’ both inside and outside of captivity.
Bee	❏ Not Null Constraint	Bee has a not null constraint since every bee has a specific type: from honey bees to wax bees.
Super	❏ Not Null Constraint	Super contains a not null constraint because logically, bees will make supers inside and outside of captivity. These supers will be in some form of condition.

Table: Produce

Attribute Name	Constraint(s)	Reasoning
Price	❏ Not Null Constraint ❏ Check Constraint	Price has a not null constraint because logically, all produce 'costs' something to produce. As a result, produce has a check constraint which checks to see that the price is greater than or equal to zero. You cannot have negatively priced produce.
Type	❏ Not Null Constraint	Type has a not null constraint because logically, all produce is sorted and sold. An example of this would be that you do not go to the supermarket and buy milk from the fruit baskets, milk has a dairy type, and would be found in the dairy aisle.
Bar_Code	❏ Uniqueness Constraint ❏ Primary Key Constraint ❏ Entity Integrity Constraint	Bar_Code has a uniqueness constraint because Bar_Code is a primary key, and each primary key must be unique. Each primary key constraint in sql also automatically attaches the not null constraint, as there cannot be a null primary key. This attached not null constraint is called the entity integrity constraint. Bar_Code identifies the specific value in this entity, and as a result is the primary key.
Inventory	❏ Not Null Constraint ❏ Check Constraint	Inventory has a not null constraint because logically, a farm will have x amount of produce. A farm can have 5 of a produce or 0, but a farm cannot have -2 of an item. As a result, there is a check constraint, which checks to make sure there is at least 0 inventory.
Buyer	❏ Forign Key Constraint	Buyer has a forign key constraint which points to Customer's social security number. This indicates who purchased the item.
Harvest	❏ Not Null Constraint ❏ Check Constraint	Harvest has a not null constraint as you must collect, butcher, or 'harvest' items on a specific day. There is a check constraint on this attribute since you cannot sell items you have not yet harvested. This check constraint checks the date against the current day.
HarvestedBy	❏ Forign Key Constraint ❏ Not Null Constraint	HarvestedBy has a forign key which points to the employee_ID that harvested the items. Since someone had to harvest the item, logically this attribute has a not null constraint.

Table: Customer

Attribute Name	Constraint(s)	Reasoning
Social_Security_Number	❏ Uniqueness Constraint ❏ Primary Key Constraint ❏ Entity Integrity Constraint	Social_Security_Number has a uniqueness constraint because Social_Security_Number is a primary key, and each primary key must be unique. Each primary key constraint in sql also automatically attaches the not null constraint, as there cannot be a null primary key. This attached not null constraint is called the entity integrity constraint. Social_Security_Number identifies the specific value in this entity, and as a result is the primary key.
Receipt	❏ Check Constraint	Receipt has a check constraint to ensure that the value is greater than or equal to zero because if a customer has a receipt: that means they purchased some set of items, and the transaction would need to be stored.
Name	❏ Not Null Constraint	Name has a not null constraint because logically, customers must have a name.
Sex	❏ Check Constraint	Sex has a check constraint to see that values are ‘’m’ for male, or ‘f’ for female. This determines the sex of the customer.
Birthday	❏ Not Null Constraint ❏ Check Constraint	Birthday has a not null constraint because logically you must have a birthday. There is a check constraint to ensure that you do not have a birthday which would result in the customer being “dead” (prior to 1900.)

Problem 2) Discussion about possibly having to use update as we did when we created the company database's tables.

We would possibly need to use update() when we would have to update a value within an entity inside the database. An example of this would be when a customer changes their address, and an address value would need to be added / changed to their address list. Another example would be when the indisposition of a BeeHive changes.

Problem 3) Any problems you encountered when you did inserts as pertains to constraint violations and explanations of why they were violations and how you solved the constraint violations problems. If you did not have any problems with constraint violations, then give 4 examples of your inserts and explain how you avoided constraint violations. Note that the constraint violations we have discussed have the following names: domain constraint; key constraint; entity integrity constraint; referential integrity constraint.

An issue I came across was when I tried to create table "Customer_Address" with multiple primary keys. This is a violation because there can only be one primary key, as that key is the identifier of the entity. You cannot have multiple identifying factors for one specific entity, otherwise this would result with an integrity constraint violation. I resolved this issue by merging my Customer_Address table into Customer. I separated multivalued values with a comma to specify that they were multivalued.

Problem 4) Your dia diagram from Project Two. If this diagram has changed in any way, please redo it before submitting it.

Dia Diagram is attached to project email as both .dia as well as .png.