Steps for transforming a Data model into a database design

1- Create a table for each entity

Transforming a Data Model into a Database Design
1. Create a table for each entity:
– Specify the primary key (consider surrogate keys, as appropriate)
– Specify alternate keys
– Specify properties for each column:
• Null status
• Data type
• Default value (if any)
• Data constraints (if any)
– Verify normalization

2- Create relationships by placing the foreign keys

2. Create relationships by placing foreign keys
– Relationships between strong entities (1:1, 1:N, N:M)
– Identifying relationships with ID-dependent entities (intersection tables, association patterns, multivalued attributes, archetype/instance patterns)
– Relationships between a strong entity and a weak but non-ID-dependent entity (1:1, 1:N, N:M)
– Mixed relationships
– Relationships between supertype/subtype entities
– Recursive relationships (1:1, 1:N, N:M)

3- Specify logic enforcing minimum cardinality

3. Specify logic for enforcing minimum cardinality:
– O-O relationships
– M-O relationships
– O-M relationships
– M-M relationships

Create a table for each entity

When you create a table for each entity you need to put a data constraint if there:

- Domain constraints: it means that the attributes must be the same type ex. (age) must be in the domain of integers ONLY can't be inserted with it a string.

- Range constraints: the values must be in the specific range ex. (age) must be between 18 and 25.

- Intrarelation constraints: attributes will be checked with other attributes with a condition and they are in the same table ex. (start date) must be less than the (end date).

- Interrelation constraints: attributes will be checked with other attributes with a condition and they are in a different table ex. (start date) in project table must be less than the (end date) in redlines table.

2- Creating a relationship with strong entities

1:1 strong entity: place a primary key for the first table to the other table as the foreign key, minimal cardinality consideration may be important O-M will be different than M-O, one of them will be better to choose, IT'S O:O.

1:M strong entity: place the primary key of the parent to the child as a foreign key, the parent is the one 1 side and the child is the Many side, IT'S O:O.

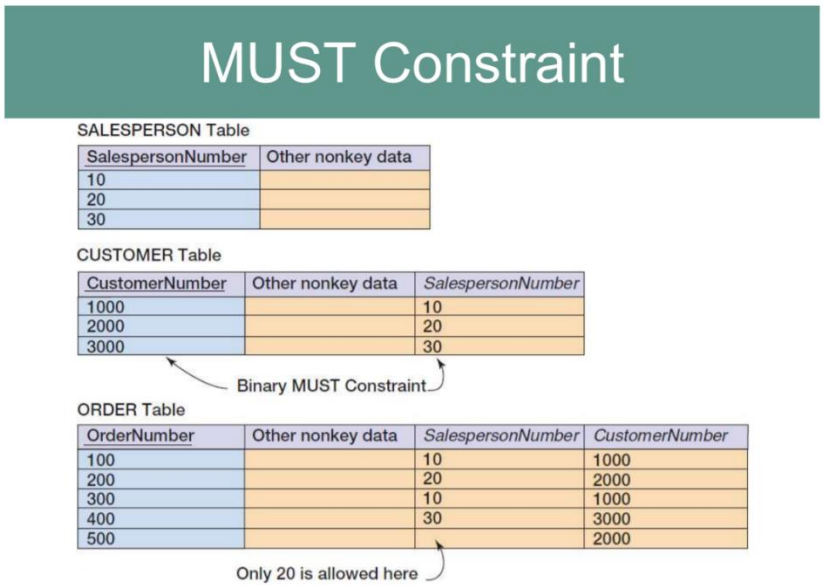
N:M strong entity: we create an intersection table between the two tables by taking the primary key for each table and then put them as a composite key in a third table ex. (company) and (part) creating a new table (company_parts(companyPK, partPK)) each attribute in this table will be a foreign key for his other table.

- An intersection table: it will be used for N:M relationships ex. N:M strong entity like the prev. one.

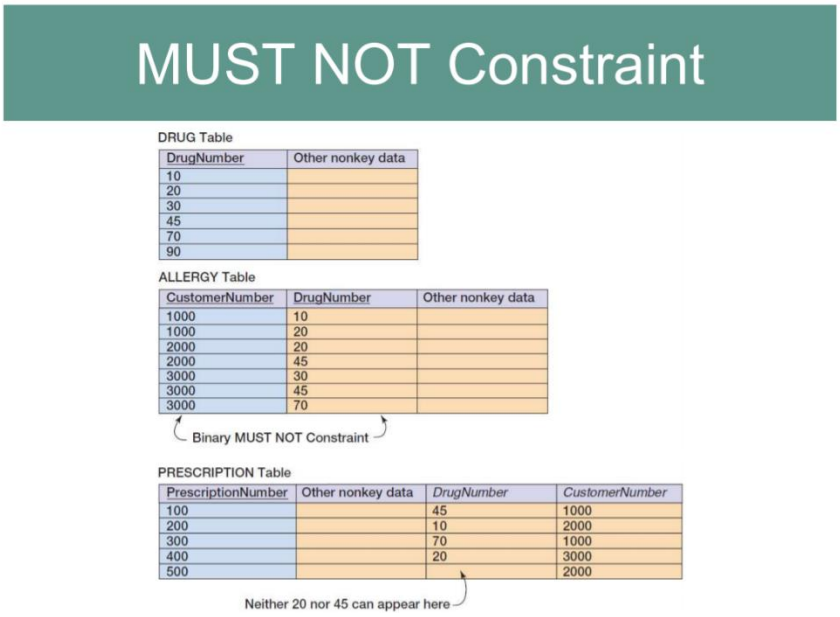
- An association table: is the same as the intersection table but the only difference is that association table will have one more attribute that is connected to the parents table, ex. (company_parts(companyPK, partPK, price))

3-Representing ternary and higher order relationships

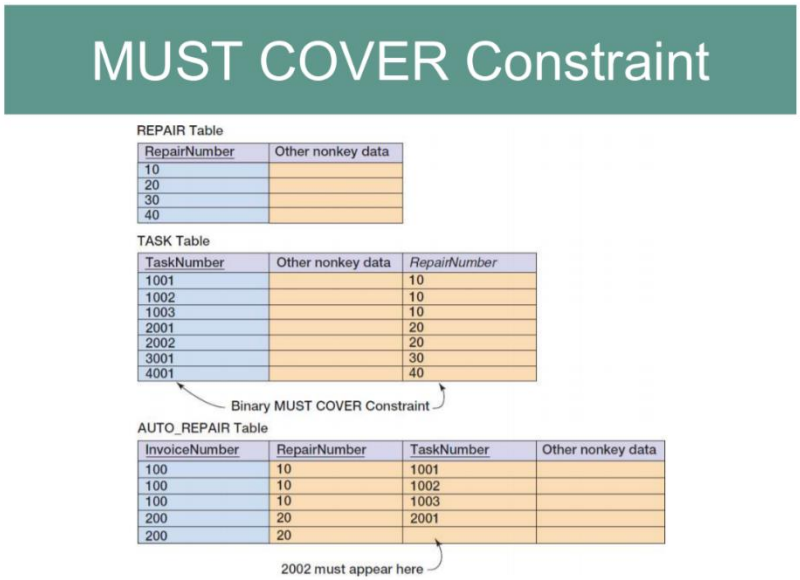
Must constraint: ex.



Must not constraint: ex.



Must cover constraint: ex.



Design for minimum cardinality

O-O parent is optional : child is optional and M-O parent is mandatory : child is optional and so on.

No action needs to be taken in O:O relationships.

- cascade update: occurs when we update a value in the parent table and that action happens to the child (Foreign key).
- Cascade delete: occurs when we delete a value in the parent and that action happens to the child (Foreign key).

-For strong entities they do not cascade on delete.
-For weak entities they do cascade on delete.

Actions when the Parent is required:

Parent Required	Action on Parent	Action on Child
Insert	None.	Get a parent. Prohibit.
Modify key or foreign key	Change children's foreign key values to match new value (**cascade update**). Prohibit.	OK, if new foreign key value matches existing parent. Prohibit.
Delete	Delete children (**cascade delete**). Prohibit.	None.

Actions when the child is required:

Child Required	Action on Parent	Action on Child
Insert	Get a child. Prohibit.	None.
Modify key or foreign key	Update the foreign key of (at least one) child. Prohibit.	If not last child, OK. If last child, prohibit or find a replacement.
Delete	None.	If not last child, OK. If last child, prohibit or find a replacement.

Triggers and stored procedures actions:

When implementing actions for M-O relationships

- Check the actions when the parent is required.
- Make sure that:
    - Every child has a parent.
    - Operations never create orphans.
- The DBMS will do the action if:
    - Referential integrity constraints are probably defined.
    - The foreign column is NOT NULL.

When implementing actions for O-M relationships

- Check the actions when the child is required.
- The DBMS does not provide much help.
- Triggers or other application codes will need to be written

---

When implementing actions for M-M relationships

- The worst of all the other cases.
- Check all the actions (when the parent is required) and (when the child is required).
- Complicated and careful application programming will be needed.
