* 1. (ProjectName, Date) 🡪 EmployeeName

ProjectName 🡪 EmployeeName

* 1. Create two tables PROJECT\_MEETING\_DETAILS and STAFF\_DETAILS

where

PROJECT\_MEETING\_DETAILS (**ProjectName**, **Date**)

Functional Dependencies:

(ProjectName, Date) 🡪 ProjectName

(ProjectName, Date) 🡪 Date

Candidate Keys:

(ProjectName, Date)

STAFF\_DETAILS (**ProjectName**, EmployeeName)

Functional Dependencies:

ProjectName 🡪 EmployeeName

Candidate Keys:

ProjectName

Referential Integrity Constraints:

ProjectName in STAFF\_DETAILS must exist in PROJECT\_MEETING\_DETAILS

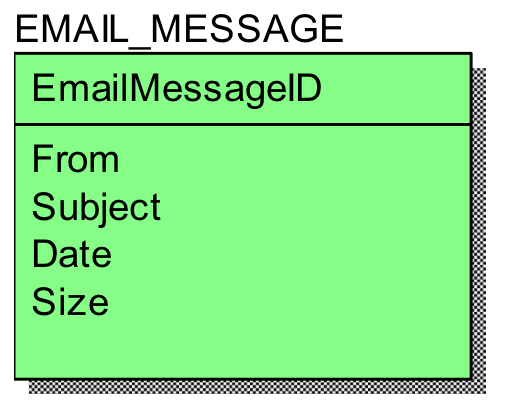
All the determinant are candidate keys hence all the relations are in BCNF.

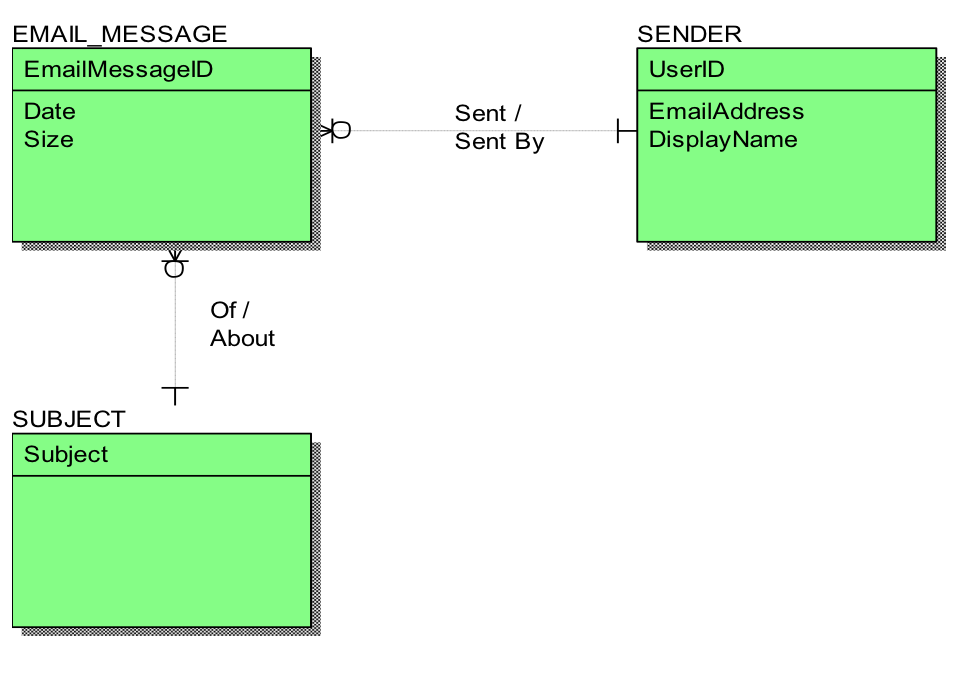
* 1. The modified table is easy to trace the PROJECT details organized and we get better search results/Query result when tables are organized.
     1. ProjectID → EmployeeName 🡺 False
     2. ProjectID → EmployeeSalary 🡺 False
     3. (ProjectID, EmployeeName) → EmployeeSalary 🡺 True but only if EmployeeName 🡪 EmployeeSalary
     4. EmployeeName → EmployeeSalary 🡺 True
     5. EmployeeSalary → ProjectID 🡺 False
     6. EmployeeSalary → (ProjectID, EmployeeName) 🡺 False
  2. (ProjectID, EmployeeName)
  3. No, not all non-key attributes dependent on the primary key. There is one attribute which is EmployeeSalary and it is dependent on EmployeeName.
  4. PROJECT is in 1NF.
  5. INSERTION: First we need to add employee to project then only we can add an employee’s salary in table.

MODIFICATION: We need to change EmployeeSalary in all the tables making the whole data unstable.

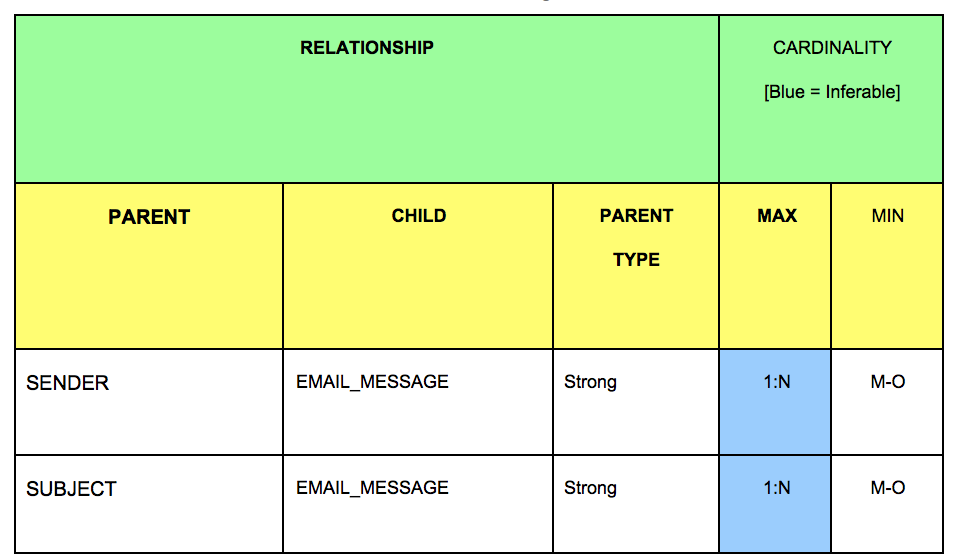
* 1. No, ProjectID is not a determinant.
  2. Yes, EmployeeName is a determinant. The functional dependency of EmployeeName 🡪 EmployeeSalary.
  3. Yes, (ProjectID, EmployeeName) is a determinant. And the functional dependency is (ProjectID, EmployeeName) 🡪 EmployeeSalary
  4. No. Looking at the table, it seems like EmployeeSalary is a determinant. But, when the data is huge then it won’t apply and so, EmployeeSalary is not a determinant.
  5. No, the relation doesn’t contain a transitive dependency.
  6. The table will work will if we modify the PROJECT table as follows:
     1. PROJECT\_ EMPLOYEES (ProjectID, EmployeeName)
     2. EMPLOYEE\_SALARIES (EmployeeName, EmployeeSalary)

All the attributes and their combinations are not unique. The primary key is **EmailMessageID**.



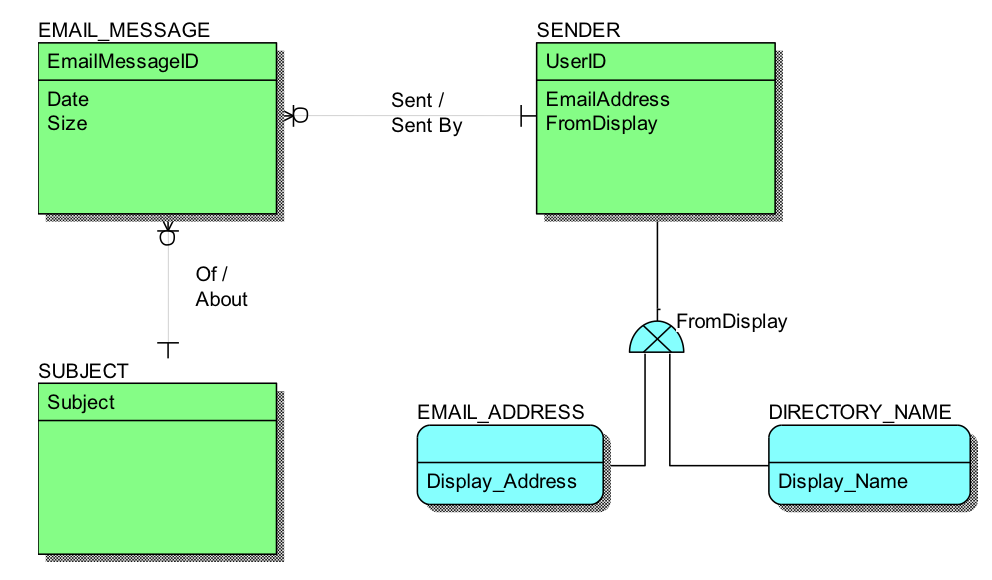


The below figure represents the entity-relation in Crow’s Foot Model of the model shown above:



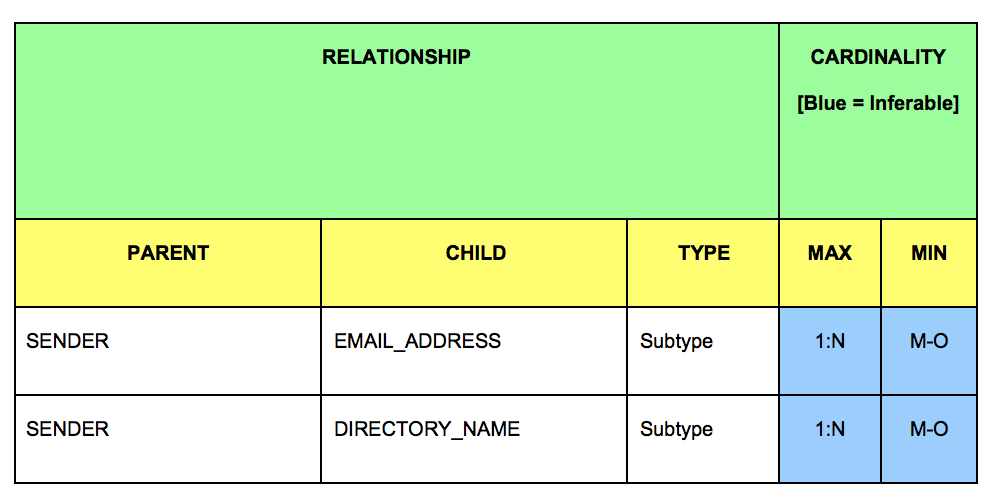
We can see that:

* There is a 1:M relation between EMAIL\_MESSAGE and SENDER. That is, one sender can have many emails. But, all the emails have only one sender each.
* There is a 1:M relation between EMAIL\_MESSAGE and SUBJECT. That is, one subject can have many emails. But, all emails have only one subject each.



The Entity-Relation in Crow’s Foot Model of the model shown above is same as the one in part B, but has some additional subtypes and attributes. Such as, the SENDER entity is assigned two subtypes EMAIL\_ADDRESS and DIRECTORY\_NAME and also an additional attribute as a discriminator is added to SENDER entity named FromDisplay.

The table below represents the information regarding the supertype/ subtype relationships:



We can see that:

* The supertype/subtype relationships contribute to the 1:M relationship between (SENDER and EMAIL\_ADDRESS) and (SENDER and DISPLAY\_NAME)
* The supertype/subtype relationships also contribute to the M:0 relationship between (SENDER and EMAIL\_ADDRESS) and (SENDER and DISPLAY\_NAME).
  1. I’m going to use three entities for implementing this data, MOVIE, THEATER, SHOW\_TIME
     1. MOVIE (**Movie\_Name,** MovieDescription)
     2. SHOWTIMES(**DayTime**)
     3. THEATER (**Theater\_Name**, Distance, Street, City, Phone)

****

* 1. We need to create another entity SUBURB and DISTANCE where,
     1. SUBURB(**Area\_Name**)
     2. DISTANCE(**Distance**)

Note that Distance is not an attribute of THEATER anymore.

****

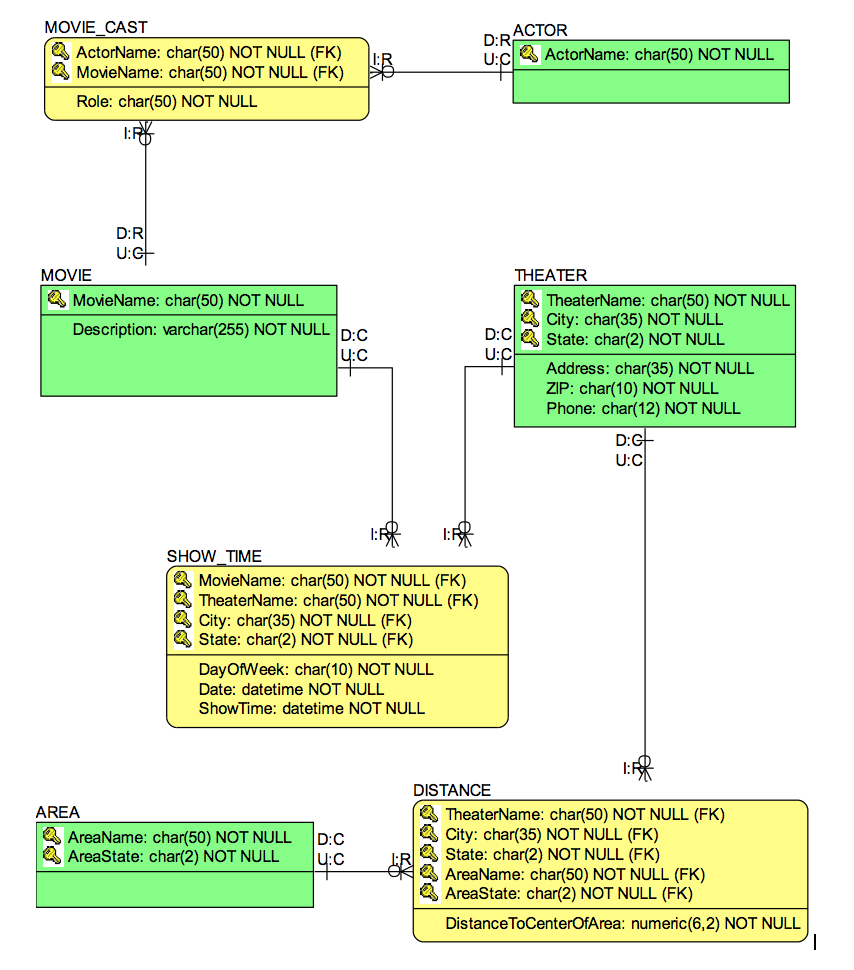
* 1. If you want to make this data model national, we need to change our answer to B so that it can be used for other metropolitan areas. The given diagram below shows the changes and its implementation. We added an attribute **STATE\_NAME** in SUBURB entity to work this out that is the only change made.



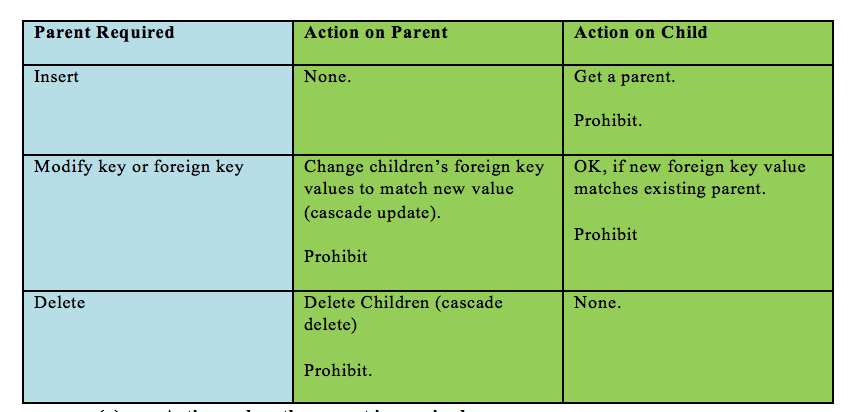
* 1. To implement the given question, we add a new entity LEADINGCASTS and an identifier **Cast\_Name.** The cardinality between LEADINGCASTS and MOVIE is M:M as casts can be in many movies and movies can have many cast members.

****

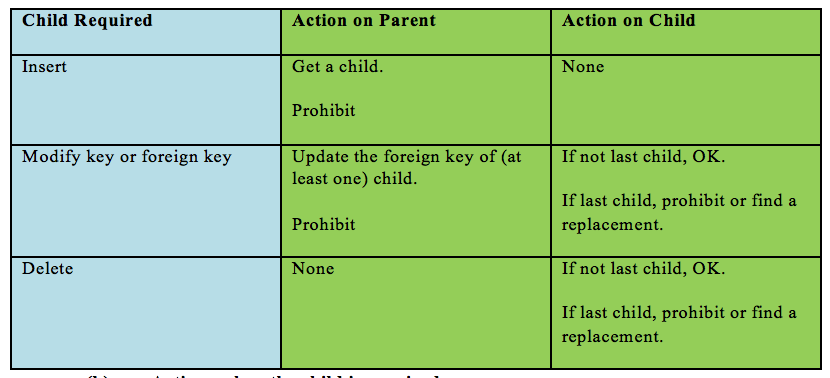
* 1. To implement the given roles in the data we are just going to add a new attribute to LEADINGCASTS named ROLES and rest all are the same.



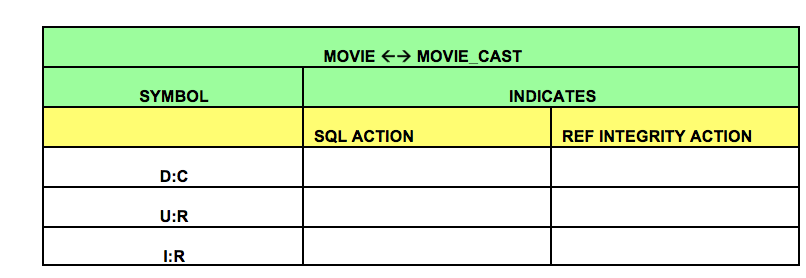
The required actions that need to performed when the parent is required.

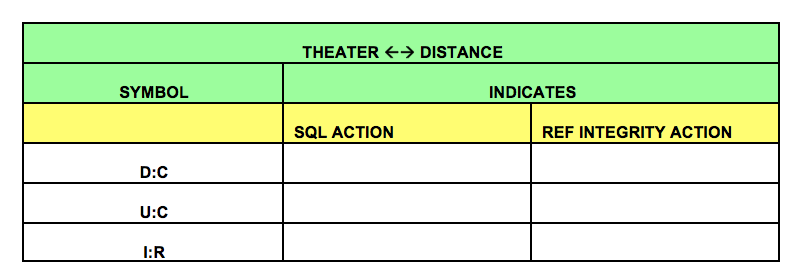
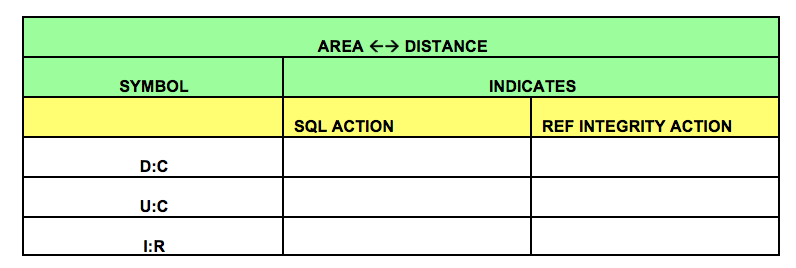
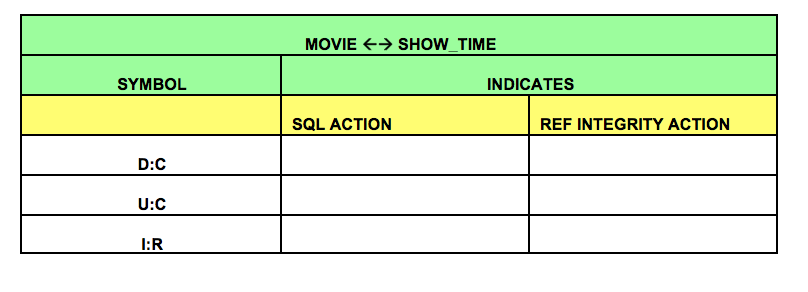


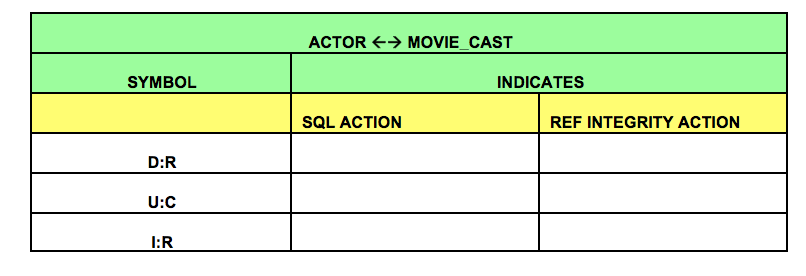
The required actions that need to performed when the child is required.



The entitiy relation diagram below shows the actions for referential integrity which are needed for the parent.







* 1. (ProjectName, Date) 🡪 EmployeeName

ProjectName 🡪 EmployeeName

* 1. Create two tables PROJECT\_MEETING\_DETAILS and STAFF\_DETAILS

where

PROJECT\_MEETING\_DETAILS (**ProjectName**, Date)

Functional Dependencies:

(ProjectName, Date) 🡪 ProjectName

(ProjectName, Date) 🡪 Date

Candidate Keys:

(ProjectName, Date)

STAFF\_DETAILS (**ProjectName**, EmployeeName)

Functional Dependencies:

ProjectName 🡪 EmployeeName

Candidate Keys:

ProjectName

Referential Integrity Constraints:

ProjectName in STAFF\_DETAILS must exist in PROJECT\_MEETING\_DETAILS

All the determinant are candidate keys hence all the relations are in BCNF.

* 1. The modified table is easy to trace the Project details when we have the database organized and we get better search results/Query result when tables are organized.
  2. SELECT \* FROM CUSTOMER;
  3. SELECT LastName, FirstName, Phone FROM CUSTOMER;
  4. SELECT LastName, FirstName, Phone

FROM CUSTOMER

WHERE FirstName = ‘John’;

* 1. SELECT LastName, FirstName, Phone

FROM CUSTOMER

WHERE FirstName LIKE ‘ D% ’;

* 1. SELECT MAX(TOTAL), MIN(TOTAL)

FROM SALE;

* 1. SELECT AVG(TOTAL) FROM SALE;
  2. SELECT COUNT (CustomerID) FROM CUSTOMER;
  3. SELECT LastName, FirstName

FROM CUSTOMER

GROUP BY LastName, FirstName;

* 1. SELECT COUNT (DISTINCT CONCAT (LastName, FirstName))

FROM CUSTOMER;

* 1. SELECT LastName, FirstName, Phone

FROM CUSTOMER

WHERE CustomerID IN

(SELECT CustomerID FROM SALE

WHERE TOTAL> 100.00)

ORDER BY LastName ASC, FirstName DESC;

* 1. SELECT LastName, FirstName, Phone

FROM CUSTOMER

WHERE CustomerID IN

(SELECT CustomerID FROM SALE

WHERE SaleID IN

(SELECT SaleID FROM SALE\_ITEM

WHERE ItemID IN

(SELECT ItemID FROM ITEM

WHERE ItemDescription = ‘Desk Lamp’)))

ORDER BY LastName ASC, FirstName DESC;

L. SELECT LastName, FirstName, Phone

FROM CUSTOMER, SALE

WHERE CUSTOMER.CustomerID= SALE.CustomerID

AND

SaleID IN

(SELECT SaleID FROM SALE\_ITEM

WHERE ItemID IN

(SELECT ItemID FROM ITEM

WHERE ItemDescription = ‘Desk Lamp’))

ORDER BY LastName ASC, FirstName DESC;