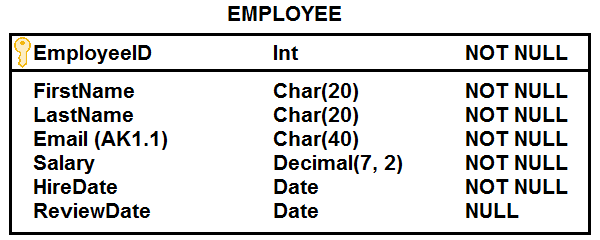
**CS 31 Database Management Programming Lecture 10**

**Transforming a Conceptual Model into a Logical Model**

**Step 1: Create a table for each entity**

* Specify the primary key
  + An ideal primary key is short, numeric, and contains one column – Consider using a surrogate key if these conditions do not exist
  + Institutions will often have policies regarding the use of surrogate keys – Pros include simplifying minimum cardinality requirements (complexity in general) – Cons include meaningless to user
* Specify alternate keys or column(s) that have a unique value – will use a unique constraint to enforce within DBMS
* Specify properties for each column
  + Null status
  + Data type
  + Default value (if any)
  + Data constraints (if any)
    - Domain constraints, range constraints, intrarelation constraints, interrelation constraints
* Verify normalization – Data models usually result in normalized tables when constructed using business requirements, forms, and reports

Data dictionaries will include a list of tables like EMPLOYEE below.



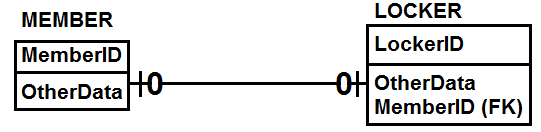
**Step 2: Create relationships by placing foreign keys**

**NOTE:** Identify strong and weak entities in the context of their relationship. A weak entity in one relationship may be a strong entity in another.

* **Relationships between strong entities (1:1, 1:N, N:M)**

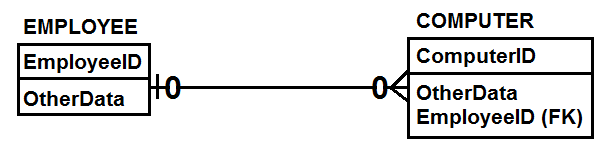
**1:1**

These should be rare. The key of one entity is placed in the other entity as a foreign key. Designate foreign key as unique. If both have optional minimum cardinality, the use by application programs will help identify the best option for placing the foreign key.



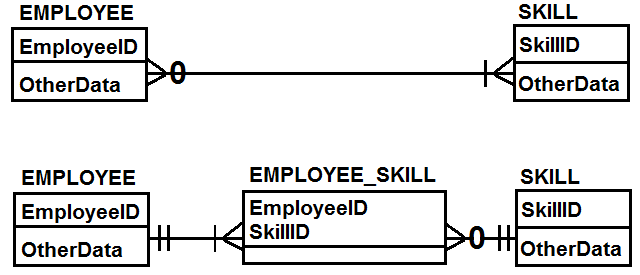
**1:N**

Place the primary key of the parent (one side) in child (many side) as a foreign key.



**N:M**

Create a new table (an intersection table) by placing the primary keys of the two original tables as foreign keys. The primary key of the intersection table will be the composite key consisting of the foreign keys from the original tables. If there are non-key attributes in the intersection table, it is referred to as an association table. The intersection/association table is ID-dependent on the two original tables. The cardinality between each original table and the intersection/association table is 1:N.



* **Relationships involving a non-ID-dependent weak entity**

For 1:N and N:M relationships between a strong and non-ID-dependent weak entity, use the explanation for relationships between strong entities.

If there is a 1:1 relationship between a strong and non-ID-dependent weak entity, it is likely the weak entity can be removed and attributes added to the strong entity.

* **Relationships involving an ID-dependent weak entity**

The foreign key placement should already be done. By definition, an ID-dependent weak entity contains the identifier, or primary key, of the strong entity as part of its primary key.

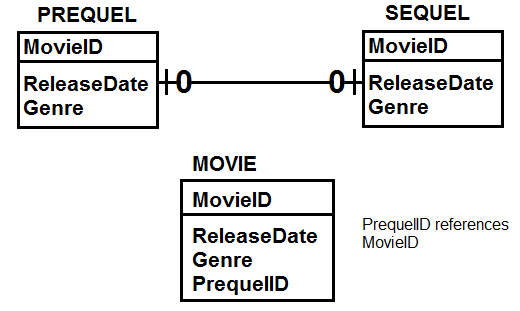
* **Relationships between supertypes/subtypes**

The foreign key placement should already be done. The identifier, or primary key, of the supertype will be the primary key of the subtype.

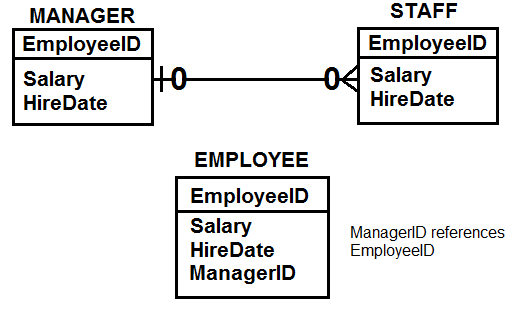
* **Recursive relationships**

Use the explanation for relationships between strong entities with self-referencing foreign keys.

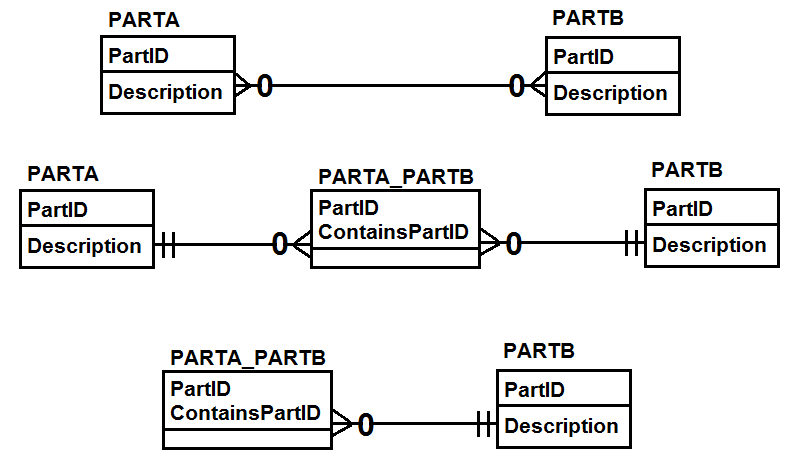
**1:1 Recursive**



**1:N Recursive**

****

**N:M Recursive**

****

**Enforcing Minimum Cardinality**

Enforcing minimum cardinality of the parent can be done using basic SQL DDL statements.

|  |  |  |
| --- | --- | --- |
| **6A** |  |  |
| **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. |
|  |  |  |
| **6B** |  |  |
| **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. |

**Cascading Update** – a referential integrity action specifying that when the key of a parent row is updated, the foreign keys of matching child rows should be updated as well

**Cascading Deletion** - a referential integrity action specifying that when a parent row is deleted, related child rows should be deleted as well

**1:N Relationships**

* Place primary key of parent (one side) in child table (many side) as a foreign key.
* Create referential integrity constraint.
* If parent is mandatory, foreign key cannot be NULL.
* If parent is optional, foreign key can be NULL.
* If parent is mandatory, implement actions in 6A.
* If child is mandatory, implement actions in 6B.

**1:1 Relationships**

* These should be rare in relations database design.
* If both min. cardinalities are optional you can put primary key of either table in the other table as a foreign key. Designate foreign key as unique. NULL will be allowed.
* If one cardinality is optional, treat the mandatory cardinality as the parent in the 1:N case and implement actions in 6A. Keep in mind there can be only one child. Designate FK as unique. NULL not allowed.
* If both cardinalities are mandatory, implement actions in 6A and 6B. Keep in mind there can be only one child. Designate FK as unique. NULL not allowed.

**N:M Relationships**

* Create a separate relation (an intersection table) with the primary key of each table as composite primary key of intersection table.
* If there are other attributes in intersection table, it is referred to as an association table.
* The relationship between each entity and the intersection table will be 1:N. Default to the explanation for 1:N strong entities.

**Lab Lecture 11**

**OUTER JOIN** –A join in which all of the rows of a table appear in the join result, regardless of whether they have a match in the join condition. In a left outer join, all the rows in the left-hand relation appear; in a right outer join, all the rows in the right-hand relation appear.

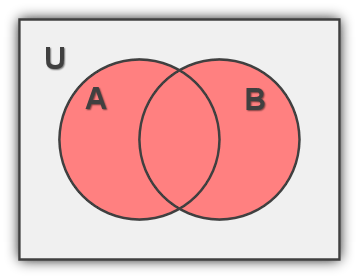
**RIGHT JOIN** and **RIGHT OUTER JOIN** are equivalent

**LEFT JOIN** and **LEFT OUTER JOIN** are equivalent

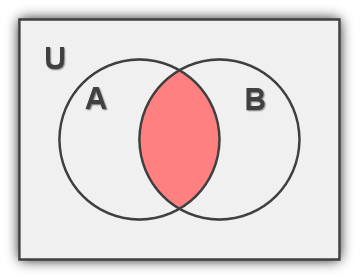
Set operations require the same number of columns and columns must be the same data types.

**SQL Set Operations**

**UNION** – The result is all rows in one or both tables ().



**INTERSECT** – The result is all rows common to both tables (



**EXCEPT/MINUS (SET DIFFERENCE)** – The result is all rows in the first table but not the second ().

