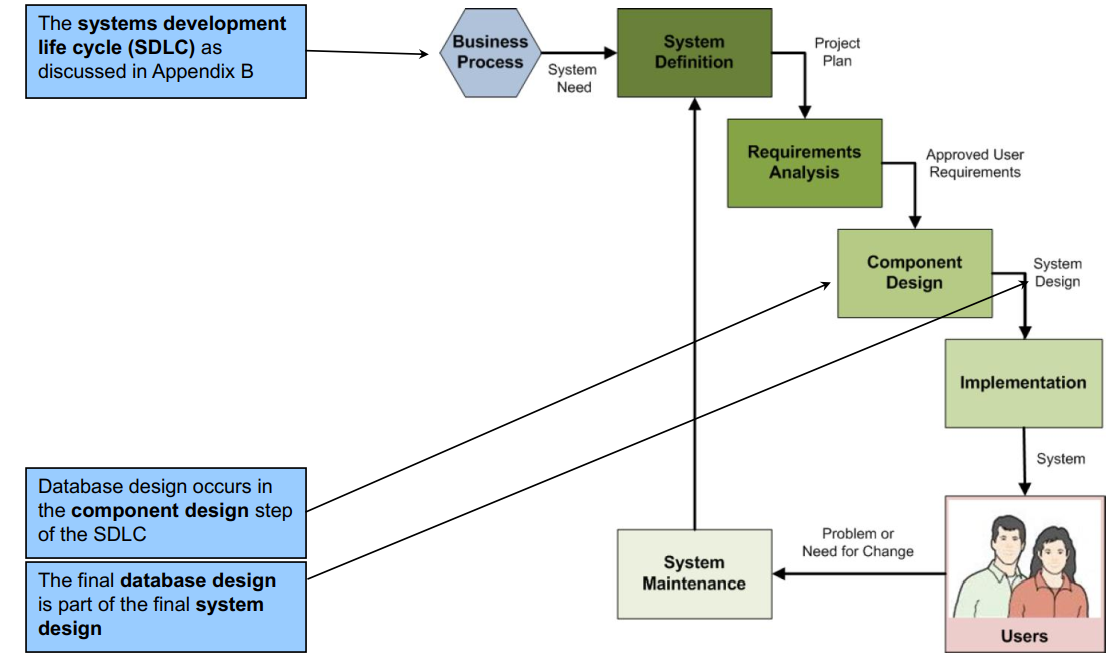
# Chapter 6

• To understand how to transform data models into database designs  
• To be able to identify primary keys and understand when to use a surrogate key  
• To understand the use of referential integrity constraints  
• To understand the use of referential integrity actions  
• To be able to represent ID-dependent, 1:1, 1:N, and N:M relationships as tables  
• To be able to represent weak entities as tables

• To be able to represent supertypes/subtypes as tables  
• To be able to represent recursive relationships as tables  
• To be able to represent ternary relationships as tables  
• To be able to implement referential integrity actions required by minimum cardinalities

A **data model** is transformed into a **database design**.  
• A **database design** is a set of database specifications that can actually be implemented in a *specific DBMS product*.  
– Therefore, a database design for a database in Microsoft SQL Server 2014 will differ from a database design for the same database in Oracle Database or MySQL 5.6

**Database Design in the SDLC**



**Database Design = Logical Design + Some Physical Design**

Books on **systems analysis and design** often identify three design stages:  
– **Conceptual design (conceptual schema)**– **Logical design (logical schema)**– **Physical design (physical schema)**• The **database design** we are discussing is equivalent to the **logical design plus some physical design elements** (data types) as defined in these books.

**Steps for Transforming a Data Model into a Database Design**

**Step 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

**Step 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)

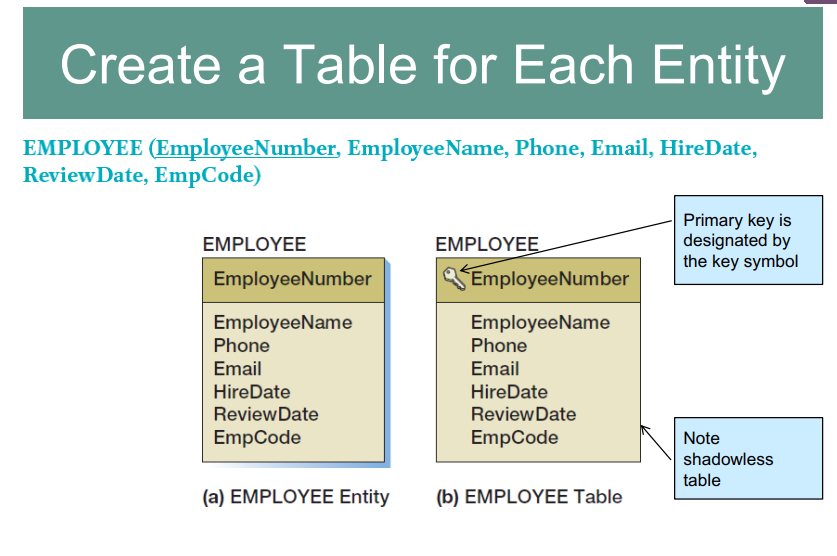
-Mixed Relationships

-Relationships between supertype/subtype entities

-Recursive relationship (1:1, 1:N, N:M)

**Step 3: Specify logic for enforcing minimum cardinality**

* O-O relationships
* M-O relationships
* O-M relationships
* M-M relationships

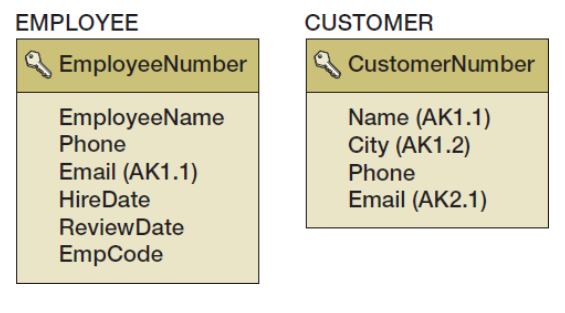


The **ideal primary key** is short, numeric, and fixed.

**Surrogate keys** meet the ideal, but have no meaning to users.

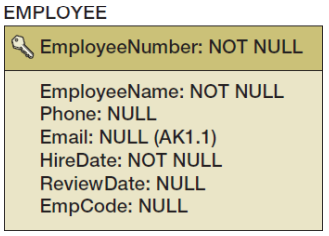
**Specify Candidate (Alternate) Keys**

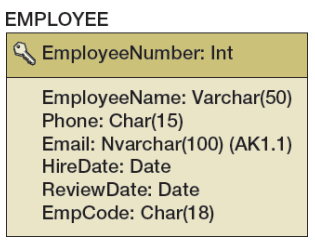
The terms **candidate key** and **alternate key** are synonymous.  
• **Candidate keys** are alternate identifiers of unique rows in a table.  
• Will use **AK*n.m*** notation, where ***n*** is the number of the alternate key, and ***m*** is the column number in that alternate key.



**Specify Column Properties: Null Status**

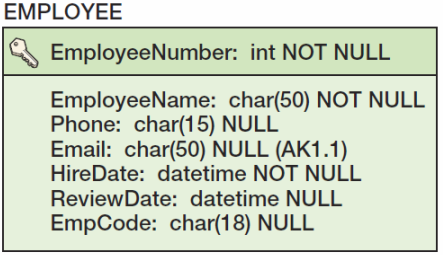
**Null status** indicates whether or not the value of the column can be NULL.



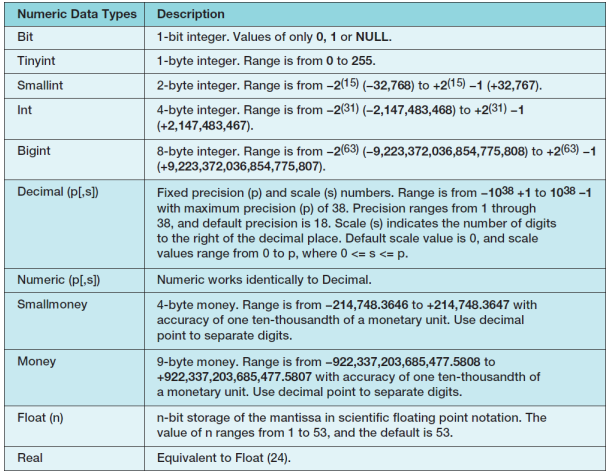
**Specify Column Properties: Data Type**

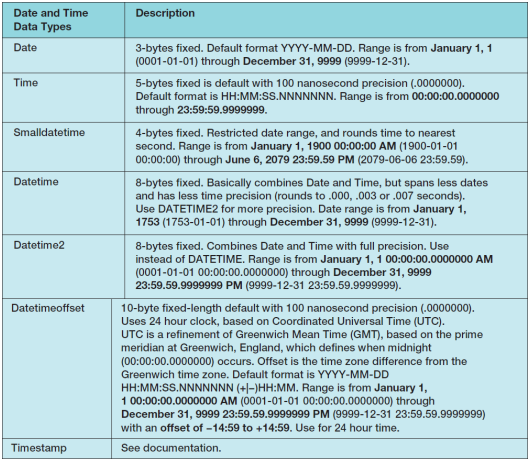
Generic data types:  
– Char(n), Nchar(n)  
– Varchar(n), Nvarchar(n)  
– Date  
– Time  
– Integer  
– Decimal(m,n)  
– Numeric(m,n)  
– Money(m,n)

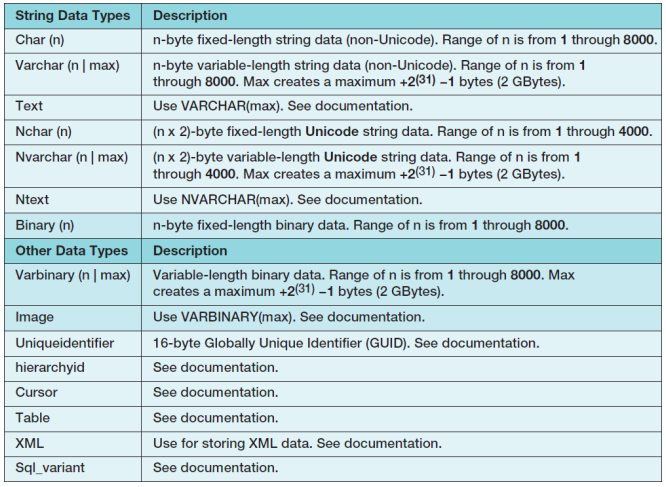
**Specify Colmun Properties: Data Type + Null Status**



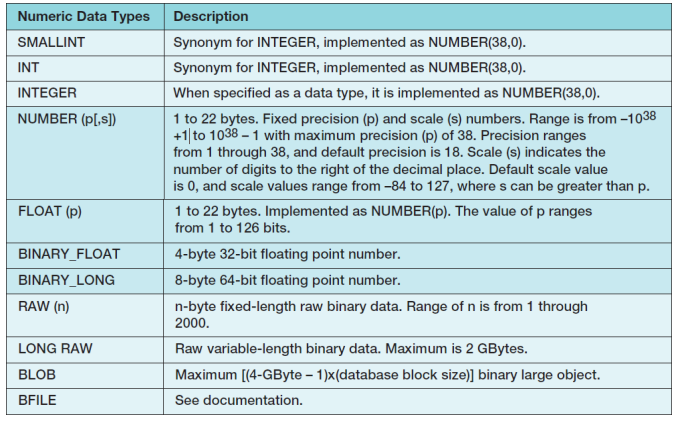
**SQL SERVER 2014 Data Types**

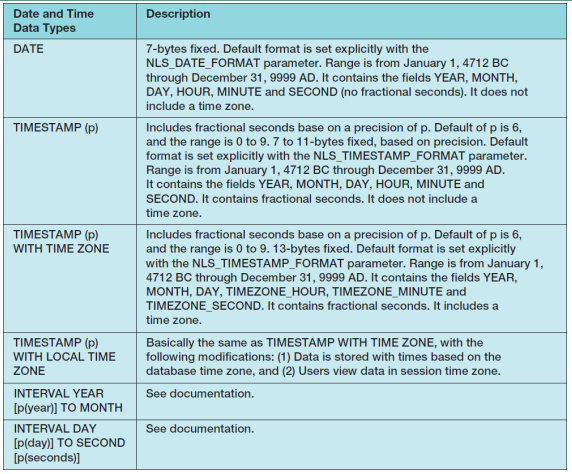


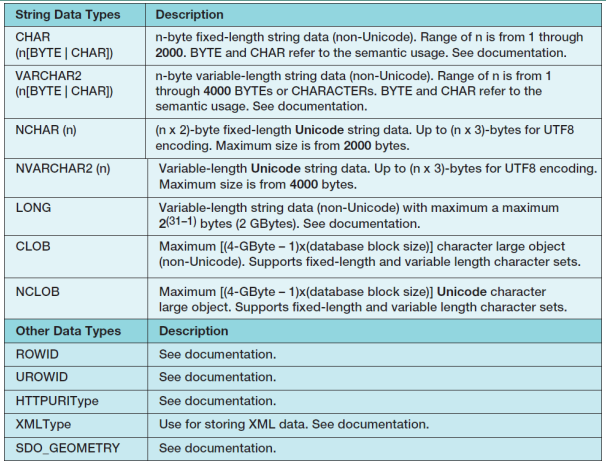




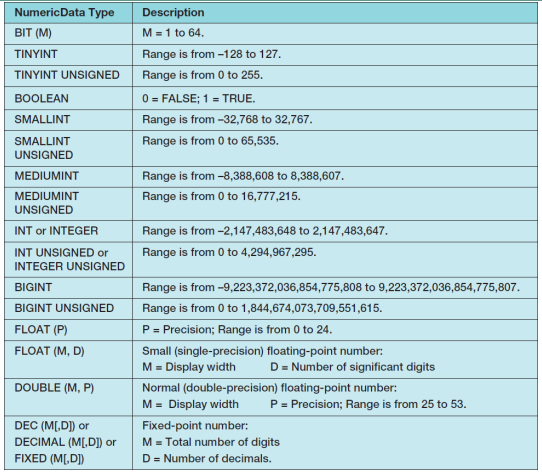
**Oracle Database Types**

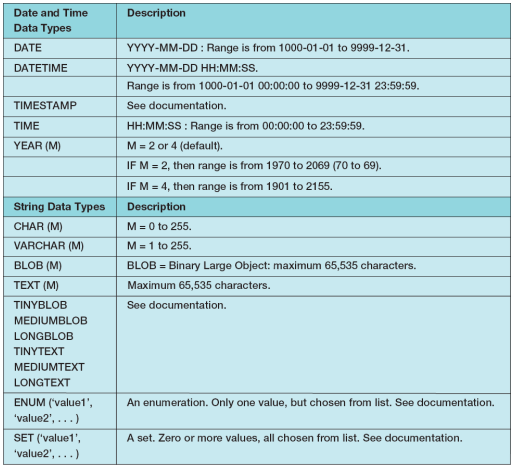






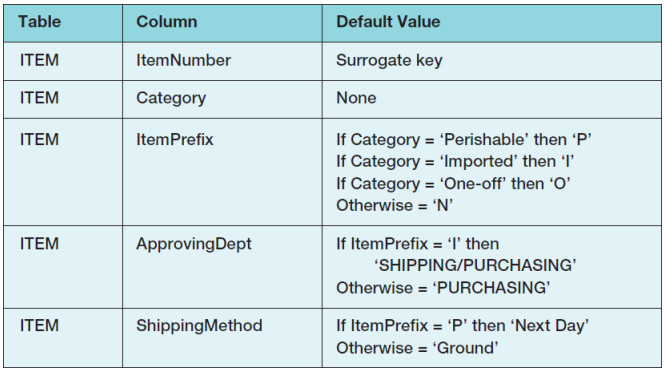
**MYSQL Datatypes**





**Specify Column Properties: Default Value**

A **default value** is the value supplied by the DBMS when a new row is created



**Specify Column Properties: Data Constraints**

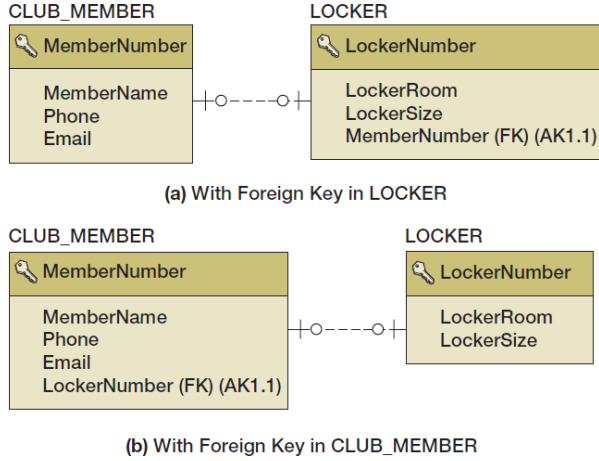
**Data constraints** are limitations on data values:  
– **Domain constraint**—column values must be in a given set of specific values.  
– **Range constraint**—column values must be within a given range of values.  
– **Intrarelation constraint**—column values are limited by comparison to values in other columns in the *same* table.  
– **Interrelation constraint**—column values are limited by comparison to values in other columns in *other*tables (referential integrity constraints on foreign keys).

**Verify Normalization**

The tables should be normalized based on the data model.  
• Verify that all tables are in either:  
– **BCNF**– **4NF**

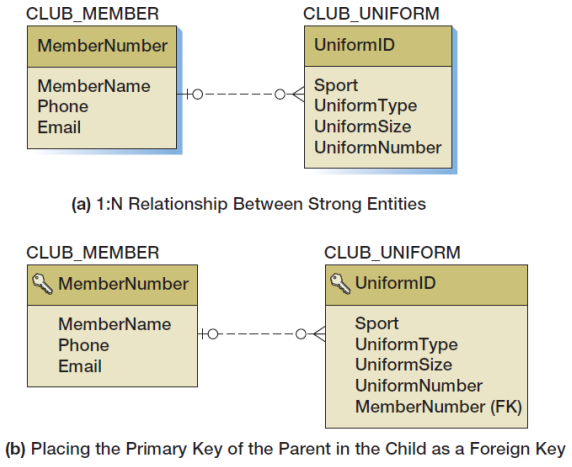
**Create Relationships: 1:1 Strong Entity Relationships**

Place the **primary key** of one entity in the other entity as a **foreign key**.  
– Either design will work—no parent, no child  
– Minimum cardinality considerations may be important.  
• O-M will require a different design than M-O.  
• One design will be very preferable.



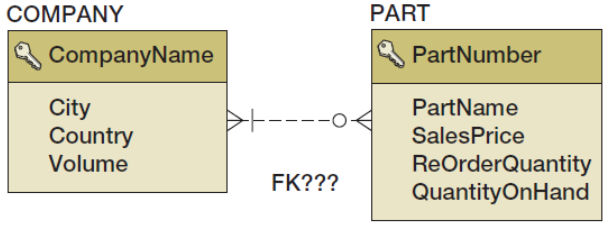
**Create Relationships: 1:N Strong Entity Relationships**

• Place the **primary key** of the table on the *one side* of the relationship into the table on the *many side* of the relationship as the **foreign key**.  
• The *one* side is the **parent** table and the *many* side is the **child** table, so “place the key of the parent in the child.

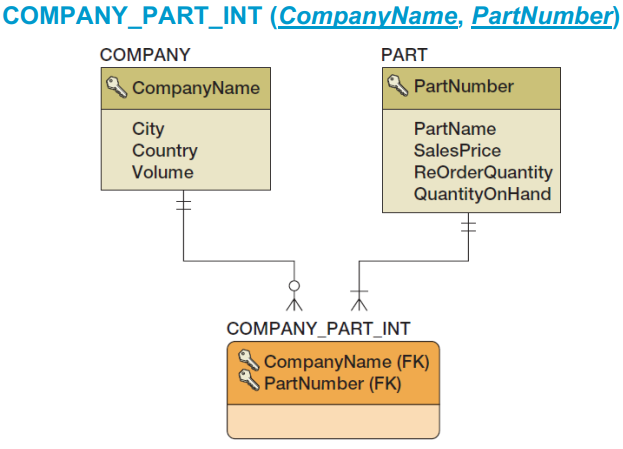


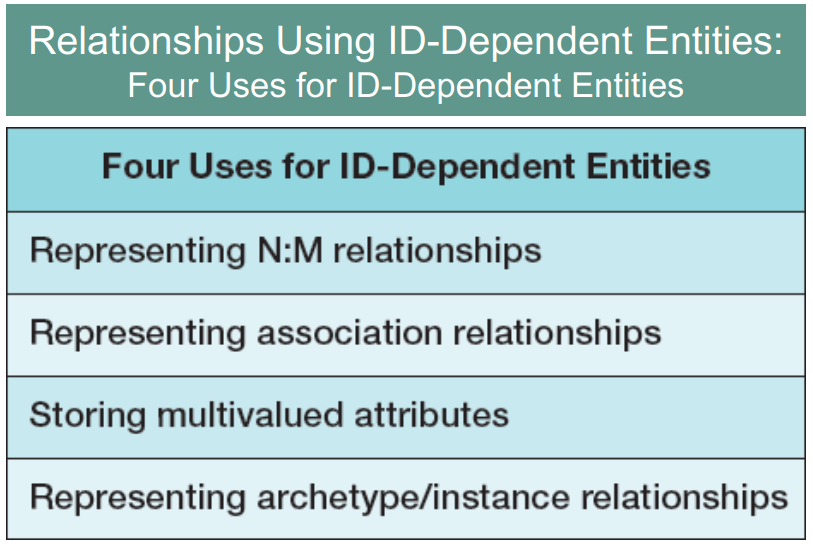
**Create Relationships: N:M Strong Entity Relationships**

• In an N:M strong entity relationship there is ***no place for the foreign key in either table*.**– A COMPANY may supply many PARTs.  
– A PART may be supplied by many COMPANYs



• The solution is to create an **intersection table** that stores data about the corresponding rows  
from each entity.  
• The intersection table consists only of the ***primary keys of each table* which form a *composite primary key*.**  
• Each table’s primary key becomes a ***foreign key***linking back to that table.  
**COMPANY\_PART\_INT (*CompanyName*, *PartNumber*)**





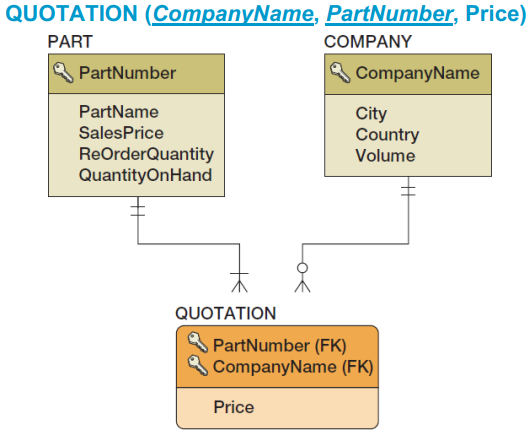
**Relationships Using ID-Dependent Entities: Association Relationships**

An **intersection** table  
– Holds the relationships between two strong entities in an N:M relationship  
– Contains *only* the primary keys of the two entities:

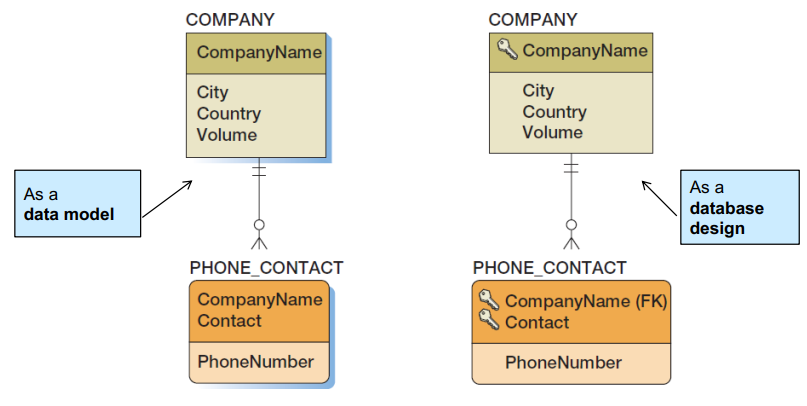
• As a composite primary key

• As foreign keys

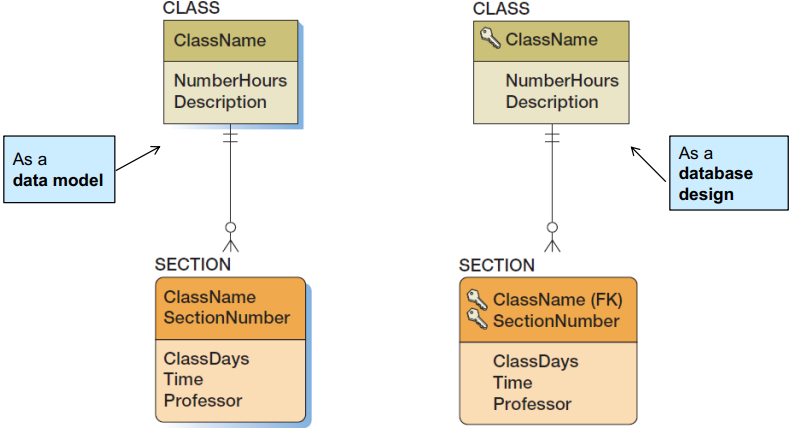
• An **association table**– Has all the characteristics of an intersection table  
– **PLUS** it has one or more columns of attributes specific to the associations of the other two entities



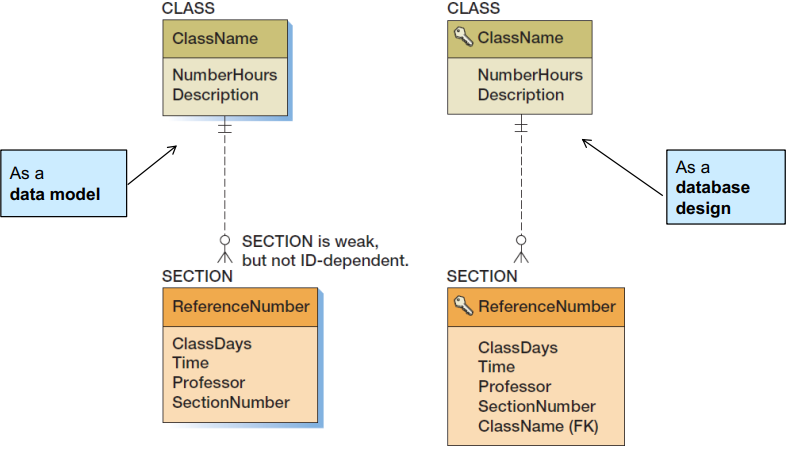
**Relationships Using ID-Dependent Entities: Multivalued Attributes**



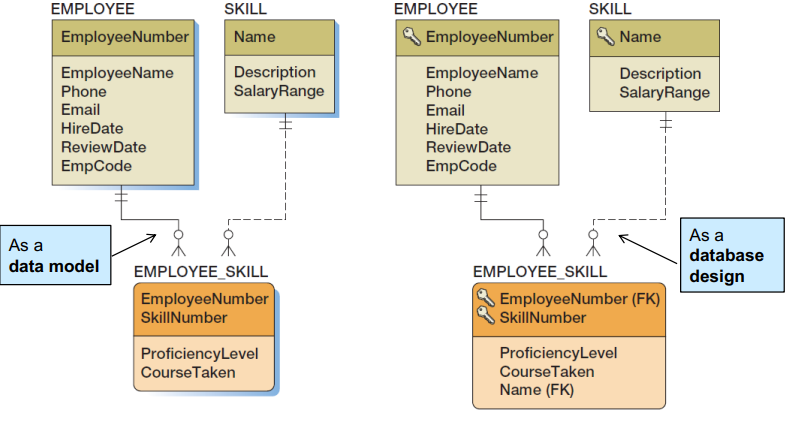
**Relationships Using ID-Dependent Entities: Archetype/Instance Pattern**



**Relationships Using Weak Entities: Archetype/Instance Pattern**



**Mixed Entity Relationships**



|  |  |
| --- | --- |
| **Mixed Entity Relationships: The SALES\_ORDER PATTERN** |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Subtype Relationships** | | **Recursive Relationships:**  **1:1 Recursive relationships** |
|  | |  |
| **Recursive Relationships:**  **1:N Recursive Relationships** | **Recursive Relationships:**  **N:M Recursive Relationships** | |
|  |  | |

**Representing Ternary and Higher-Order Relationships**

Ternary and higher-order relationships may be constrained by the binary relationship that comprise them.  
– **MUST constraint**—requires that one entity must be combined with another entity in the ternary (or higherorder) relationship.  
– **MUST NOT constraint**—requires that certain combinations of two entities are not allowed to occur in the ternary (or higher-order) relationship.  
– **MUST COVER constraint**—a binary relationship specifies all combinations of two entities that must appear in the ternary (or higher-order) relationship.

**Design for Minimum Cardinality**

Relationships can have the following types of minimum cardinality:  
– **O-O**: parent optional and child optional  
– **M-O**: parent mandatory and child optional  
– **O-M**: parent optional and child mandatory  
– **M-M**: parent mandatory and child mandatory  
• We will use the term ***action*** to mean a **minimum cardinality enforcement action**.  
• *No* action needs to be taken for *O-O relationships*.

**Cascading Updates and Deletes**

• A **cascading update** occurs when a change to the parent’s primary key is applied to the child’s foreign key.  
– Surrogate keys never change and there is no need for cascading updates when using them.  
• A **cascading delete** occurs when associated child rows are deleted along with the deletion of a parent row.  
– For strong entities, generally do *not* cascade deletes  
– For weak entities, generally do cascade deletes

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 (casecade 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. |

**Application Programming: Triggers**

• Application programming uses SQL embedded in triggers, stored procedures, and other program code to  
accomplish a specific task.  
• A trigger is a stored program that is executed by the DBMS whenever a specified event occurs on a specified  
table or view (defined in Chapter Seven).  
• Triggers are used to enforce specific minimum cardinality enforcement actions not otherwise programmed into the DBMS.  
• Triggers will be discussed in detail in Chapters 7, 10A (Microsoft SQL Server 2014), 10B (Oracle Database),and 10C (MySQL 5.6).

**Actions To Apply to Enforce Minimum Cardinality**

|  |  |  |
| --- | --- | --- |
| Relationships Minimum Cardinality | Action to Apply | Remarks |
| O-O | Nothing |  |
| M-O | Parent-required actions | Easily enforced by DBMS; define referential integrity constraint and make foreign key NOT NULL |
| O-M | Child-required actions | Difficult to enforce. Requires use of triggers or other application code. |
| M-M | Parent-required actions and child-required actions | Very difficult to enforce. Requires a combination of complex triggers. Triggers can lock each other out. Many problems! |

**Implementing Actions for M-O relationships**

• Make sure that:  
– Every child has a parent.  
– Operations never create orphans.  
• The DBMS will enforce the action as long as:  
– Referential integrity constraints are properly defined.  
– The foreign key column is NOT NULL.

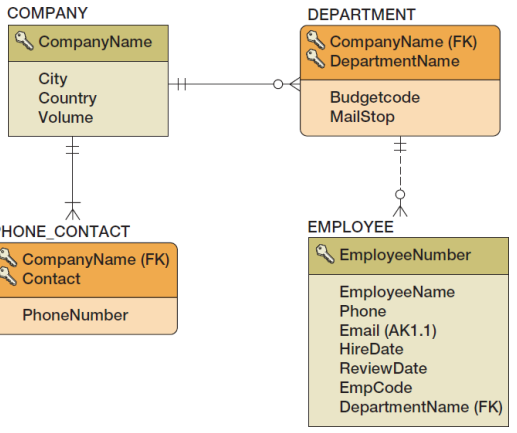
**Implementing Actions for O-M Relationships**

• The DBMS does not provide much help.  
• Triggers or other application codes will need to be written.

**Implementing Actions for M-M Relationships**

• The worst of all possible worlds:  
– Especially in strong entity relationships.  
– In relationships between strong and weak entities the problem is often easier when all transactions are initiated from the strong entity side.  
• All actions in both Figure 6-29(a) and Figure 6-29(b) must be applied simultaneously.  
• Complicated and careful application programming will be needed.

**Implementing Actions for M-O Relationships: Department and Employee**



• DEPARMENT is parent—EMPLOYEE is child.  
• Actions on parent:

– DEPARTMENT rows can be created.  
– DEPARTMENT primary key—cascade updates if not surrogate key.  
– IF a DEPARTMENT is deleted, do we delete the associate EMPLOYEEs?

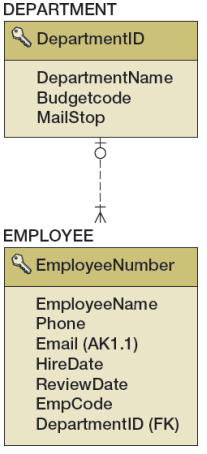
• IF YES—cascade deletes.  
• IF NO—prohibit associate employees

• Actions on child  
– Set referential integrity constraint and set foreign key to NOT NULL.

• A new EMPLOYEE must have a valid DEPARTMENT or disallow the insert.  
• EMPLOYEEs can be reassigned to a different DEPARTMENT if a valid DEPARTMENT or disallow the update.

– EMPLOYEEs can be deleted.

**Implementing Actions for O-M Relationships: Department and Employee**



• DEPARTMENT is parent—EMPLOYEE is child.  
• There must be at least one child row for each parent at all times.  
• Actions on parent:

– DEPARTMENT rows can only be created when a relationship is created to a child row—REQUIRES A TRIGGER.  
– DEPARTMENT primary keys can only be updated if at least one EMPLOYEE foreign key is also updated  
—REQUIRES A TRIGGER.  
– Can a DEPARTMENT be deleted?

• YES—it is the EMPLOYEE who is required.

• Actions on child

– OK to insert a new EMPLOYEE.

– There must be one EMPLOYEE for each department.

• Cannot change EMPLOYEE foreign key (DEPARTMENT) if last EMPLOYEE in the DEPARTMENT.

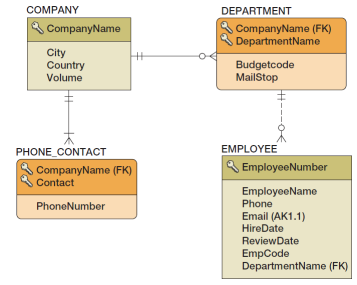
• Cannot delete an EMPLOYEE if last EMPLOYEE in the DEPARTMENT.

**Implementing Actions for M-M Relationships: DEPARTMENT AND EMPLOYEE**

• DEPARMENT is parent—EMPLOYEE is child.  
• All of the previous (M-O and O-M) apply at the same time!  
• This creates conflicts that require careful programming to avoid or fix problems such as:  
– A new DEPARTMENT insert will run a trigger that tries to create a new EMPLOYEE, but the EMPLOYEE row is checked by the DBMS for a valid DEPARTMENT before the transaction is completed.  
– If we try to delete a DEPARTMENT with any EMPLOYEEs we will find the trigger on EMPLOYEE  
delete will not let us delete the last EMPLOYEE, so we can’t delete the DEPARMENT.

**Documenting the Minimum Cardinality Design: Documenting Required Parents**

• COMPANY is parent, DEPARTMENT is child.  
• The relationship is M-O.   
• This can often be done in the database design tools.

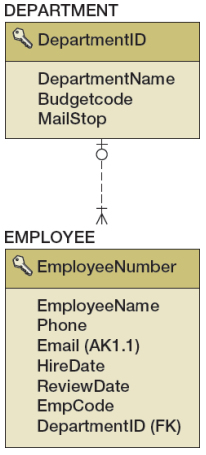


**Documenting the Minimum Cardinality Design: Documenting Required Children**

• Needs written documentation  
• Can use Figure 6-29(b) as a “boilerplate document” and fill it out for each specific situation

**Documenting the Minimum Cardinality Design: Documenting Required Children**

• DEPARTMENT is parent, EMPLOYEE is child.  
• The relationship is O-M.  
• Use documentation based on Figure 6-29(b)—see the next slide.



**Documenting the Minimum Cardinality Design: Documenting Required Children**

|  |  |  |
| --- | --- | --- |
| **EMPLOYEE Is Required Child** | **Action on DEPARTMENT** | **Action on EMPLOYEE** |
| Insert | Trigger to create row in EMPLOYEE when inserting DEPARTMENT. Disallow DEPARTMENT insert if EMPLOYEE data are not available | None |
| Modify key or foreign key | Not possible, surrogate key | Trigger needed:  If not last EMPLOYEE, OK.  If last EMPLOYEE, prohibit or assign another EMPLOYEE |

**Summary of Minimum Cardinality Design**

|  |  |  |
| --- | --- | --- |
| Relationship Minimum Cardinality | Design Decisions to Be Made | Design Documentation |
| M-O | Update cascade or prohibit?  Delete cascade or prohibit  Policy for obtaining parent on insert of child | Referential integrity actions plus documentation for policy on obtaining parent for child insert. |
| O-M | Policy for obtaining child on insert or parent  Primary key update cascade or prohibit  Policy for update of child foreign key  Policy for deletion |  |
| M-M | Al decisions for M-O and O-M above, plus how to process trigger conflict on insertion of first instance of parent/child and deletion of last instance of parent/child. | For mandatory parent, RI actions plus documentation for policy on obtaining parent for child insert. For mandator child, as a boilerplate. Add documentation on how to process trigger conflict. |

**VRG Database Design**

• Surrogate keys are needed for:  
– CUSTOMER  
– WORK  
– TRANS  
• We can also use a surrogate key for ARTIST.  
• This will change the identifying relationships to nonidentifying relationships.  
• WORK and TRANS become weak, non-ID-dependent entities.  
• Foreign keys:  
– TRANS.CustomerID is NULL to allow acquisitions without an immediate sale to a CUSTOMER.  
– All other foreign keys are NOT NULL.