**zWhat is a Conceptual Model?**

A conceptual model:

• Is called an “Entity Relationship Model”

• Is illustrated using an “Entity Relationship Diagram” (ERD)

• Is the result of completing the Data Modeling process

• Businesses use data to increase sales and/or reduce costs.

• In order to accurately collect this data, a business must create a conceptual model of the data it considers important.

**Conceptual and Physical Models**

• It is the art of planning, developing, and communicating that allows a group of people to work together to achieve a desired outcome.

• Data modeling is the process of capturing the important concepts and rules that shape a business and depicting them visually on a diagram.

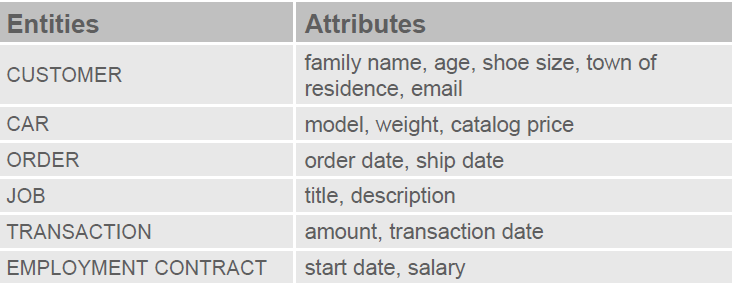
• This diagram becomes the blueprint for designing the physical thing.

• The client’s dream (conceptual model) will become a physical reality (physical model).

**Entities and Instances**



**Attributes**



• Some attributes (such as age) have values that constantly change.

• These are called volatile attributes.

• Other attributes (such as order date) will rarely change, if ever.

• These are nonvolatile attributes.

•Some attributes must contain a value—these are mandatory attributes.

•For example: in most businesses that track personal information, name is required.

•Other attributes may either contain a value or be left null—these are optional attributes.

**An Entity Relationship Model**:

•Is a list of all entities and attributes as well as all relationships between the entities that are of importance.

•Provides background information such as entity descriptions, data types, and constraints.

•Note: The model does not require a diagram, but the diagram is typically a very useful tool.

**Relationships**:

•Represent something of significance or importance to the business

•Show how entities are related to each other

•Exist only between entities (or one entity and itself)

•Are bi-directional

•Are named at both ends

•Have optionality (mandatory or optional)(must / may)

•Have cardinality( one and only one / one or more)

**ER Drawing Conventions**

•Entities are represented by softboxes.

•Entity names go in the softboxes.

•Entity names are always singular and written with all capital letters.

•Attributes are listed under the entity names.

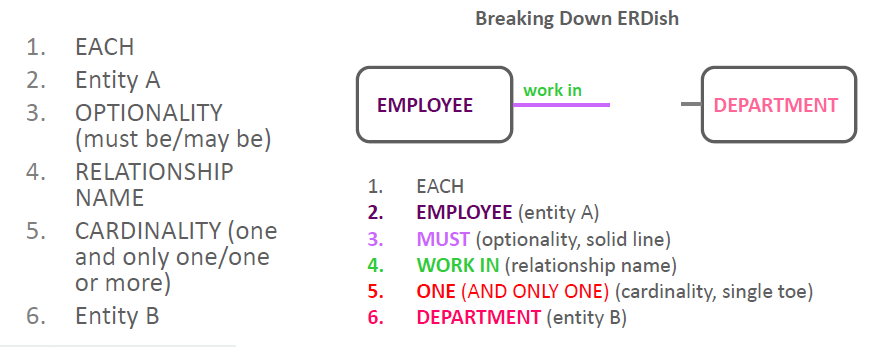
•Mandatory attributes are marked with an asterisk: “\*”

•Optional attributes are marked with a circle: “o”

•Unique identifiers are marked with a hash sign: “#”

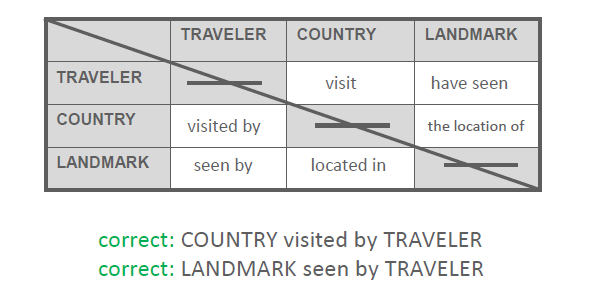
**The Components of ERDish**

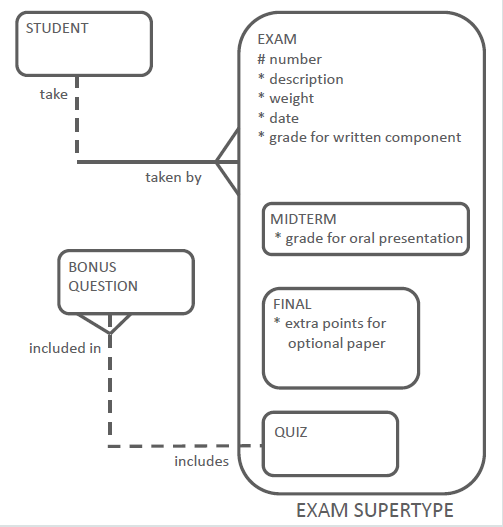
•Since each relationship has two sides, we read the first relationship from left to right (or top to bottom, depending on the ERD layout).



**Matrix Diagrams**

• when we have many relationships we use matrix diagrams, to avoid confusion, be consistent in writing to and reading from the matrix only in one direction.



Supertypes and subtypes:

**A subtype:**

–Inherits all attributes of the supertype

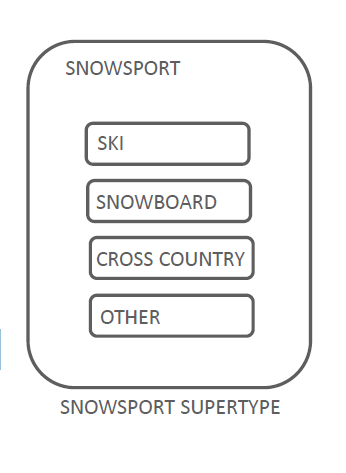
–Inherits all relationships of the supertype

–Usually has its own attributes or relationships

–Is drawn within the supertype

–Never exists alone

–May have subtypes of its own



When modeling supertypes and subtypes, you can   
use three questions to see if the subtype is correctly   
identified:

–Is this subtype a kind of supertype?

–Have I covered all possible cases? (exhaustive)

–Does each instance fit into one and only one subtype? (Mutually exclusive)

**Structural and Procedural Business Rules**

•Structural business rules indicate the types of information to be stored and how the information elements interrelate.

•Procedural rules deal with the prerequisites, steps, processes, or workflow requirements of a business.

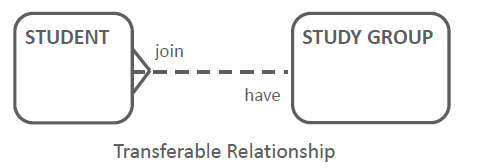
•Many procedural business rules are related to time: event A must happen before event B.

•Structural business rules can nearly always be diagrammed in the ERD.

•Some procedural business rules cannot be diagrammed but must still be documented so that they can be programmed later.

**Relationship Transferability**

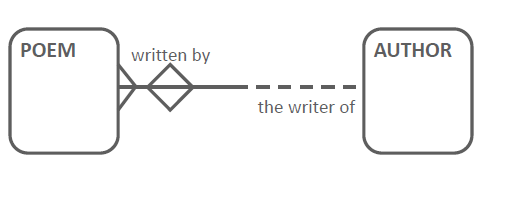
*•Transferable*: A STUDENT being allowed to move from one STUDY GROUP to another.

•There is a relationship between STUDENT and STUDY GROUP that is transferable.

*Nontransferable relationship*:

•Ownership of a POEM belongs with its AUTHOR.

•Authorship is a relationship that cannot be moved to another person.



**Relationship Types :**

•Many-to-many (M:M)

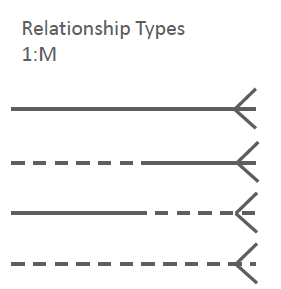
•One-to-many (1:M)

•One-to-one (1:1)

•Redundant

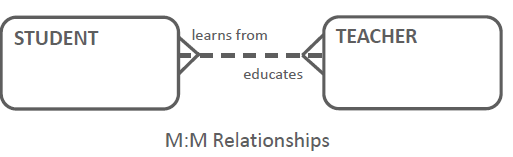
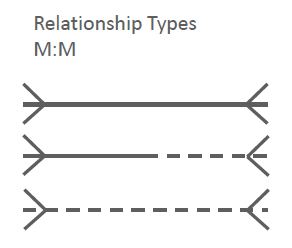
*One-to-Many (1:M) Relationships*

•The various types of 1:M relationships are most common in an ER Model.

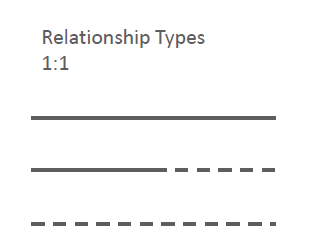
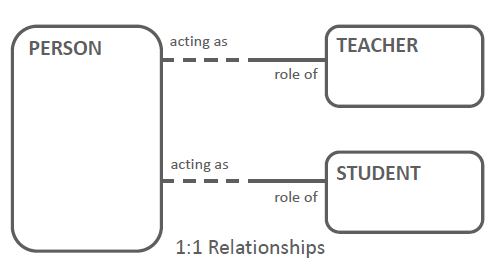
•You have seen several examples already.

*Many-to-Many (M:M) Relationships*

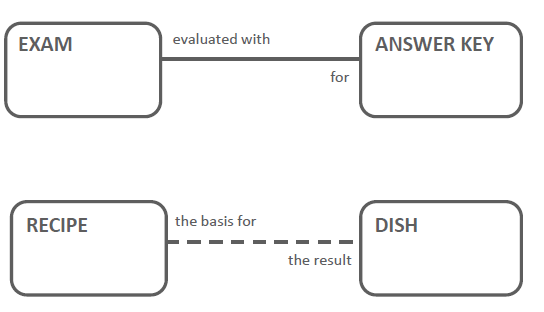
•The various types of M:M relationships are common, particularly in a first version of an ER mo del.

•In later stages of the modeling process, all M:M relationships will be resolved, and disappear.

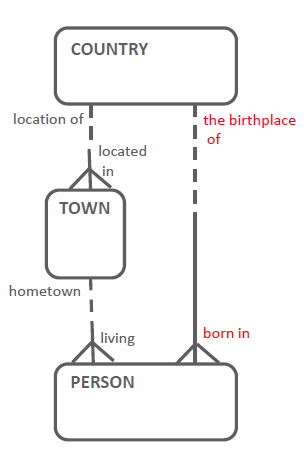
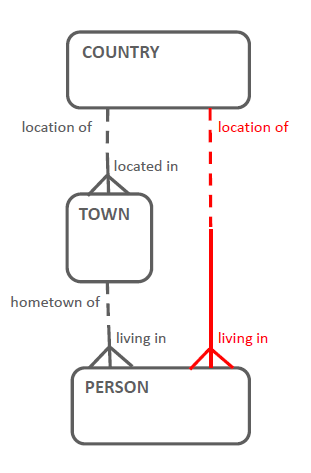
*One-to-One Relationships For Roles*

•Mandatory at one end of the 1 :1 relationship commonly occurs when roles are modeled.

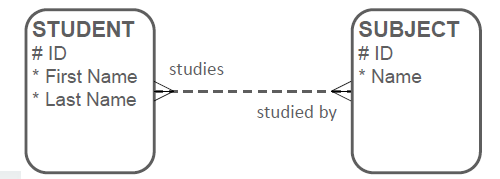
*One-to-One Relationships For Processes*

• 1:1 relationships (of all three variations) also occur when some of the entities represent various stages in a process.

*Redundant Relationships*

In this example, you can derive the relationship from PERSON to COUNTRY from the other two relationships (COUNTRY to TOWN, TOWN to PERSON), so you should remove the direct relationship from COUNTRY to PERSON

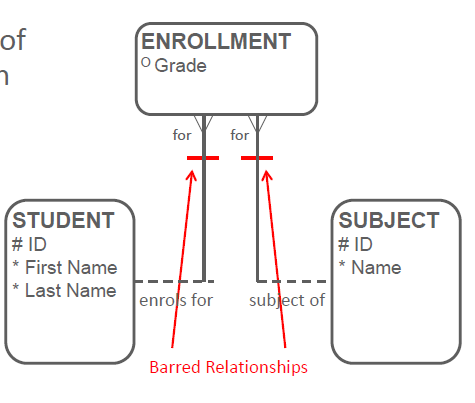
The ERD shown here   
 does not reflect a   
 redundant relationship.

*Many-to-Many (M:M) Relationships*

•Which entity would the attribute “Grade” belong to?

•If we put “Grade” in the STUDENT entity, how would we know which SUBJECT it is for?

•If we put “Grade” in the SUBJECT entity, how would we know which STUDENT got that grade?



The original M:M relationship has become two 1:M relationships.

Note that the relationships have become mandatory at the side of the Intersection entity

Barred relationship: A relationship that participates in an entity's unique identifier

The UID for ENROLLMENT is the combination of STUDENT id and SUBJECT id.

When you draw a barred relationship, you do not add the UID attribute names to the intersection entity, as the bars represent this.

**UIDs :**

*Simple UIDs vs. Composite UIDs*

A UID that is a single attribute is a simple UID.

However, sometimes a single attribute is not enough to uniquely identify an instance of an entity.

If the UID is a combination of attributes, it is called a composite UID.

Text, application, chat or text message

Description automatically generated

*Artificial UIDs*

Artificial UIDs are those that don’t occur in the natural world but are created for purposes of identification in a system.

People are not born with “numbers,” but a lot of systems assign unique numbers to identify people: student numbers, customer IDs, etc.

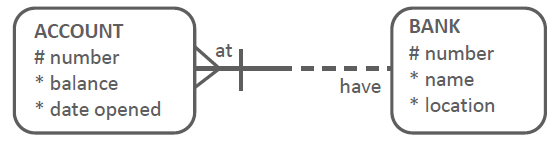
Diagram, text

Description automatically generated

*UIDs from Barred Relationships*

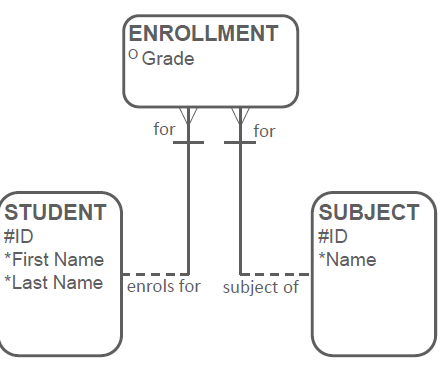
*Two people could have the same bank account number, but at different banks.*

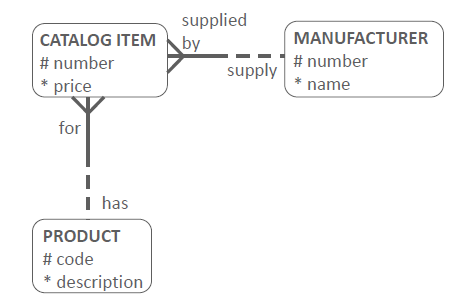
*Bank to bank transfers always need the bank routing number in addition to the bank account number.*

Bank to bank transfers always need the bank routing number in addition to the bank account number.

*UID from Barred Relationship Intersection Entity*

In this example, the UID of ENROLLMENT comes from STUDENT and SUBJECT.

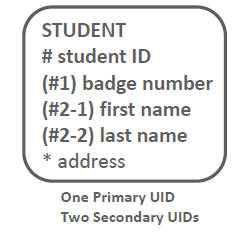
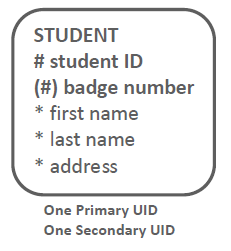


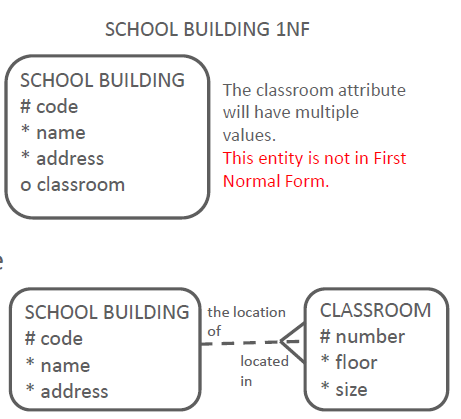
It is possible for an intersection entity to use an artificial  
 attribute as the UID, instead of the barred relationships  
 to the originating entities.

*Candidate UIDs*

•Student ID has been chosen as the primary UID in both of these STUDENT entities.

•The first entity has one secondary UID, while the second has two secondary UIDs (one of which is composite).

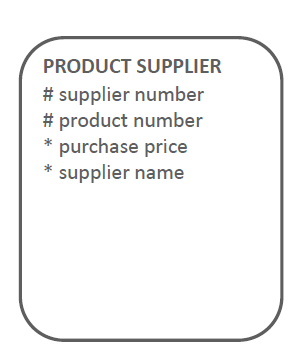


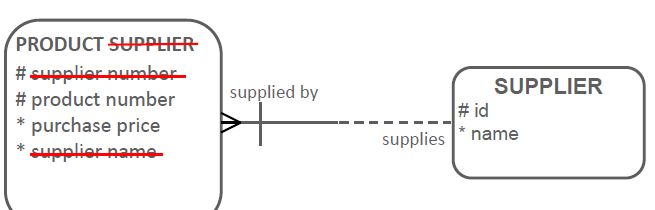
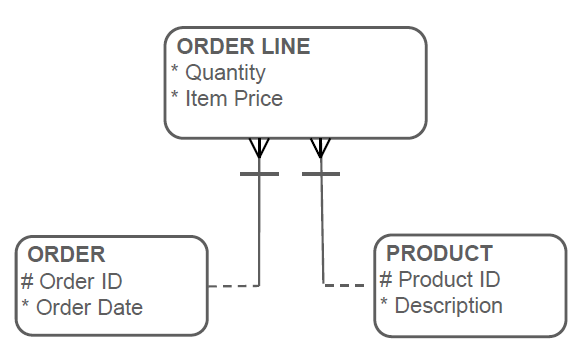
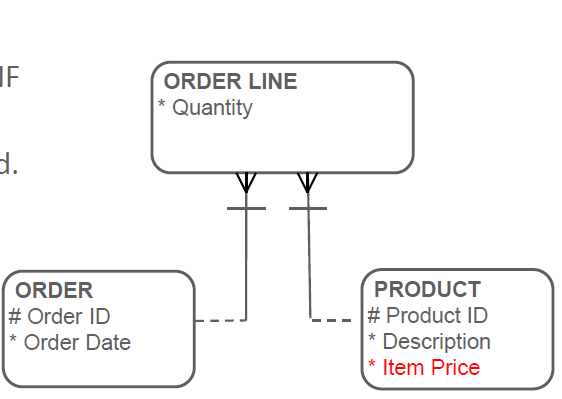
**Normalization**

*First Normal Form (1NF)*

First Normal Form requires that no multi-valued attributes exist.

If an attribute is multi-valued, create an additional entity   
and relate it to the original entity with a 1:M relationship.

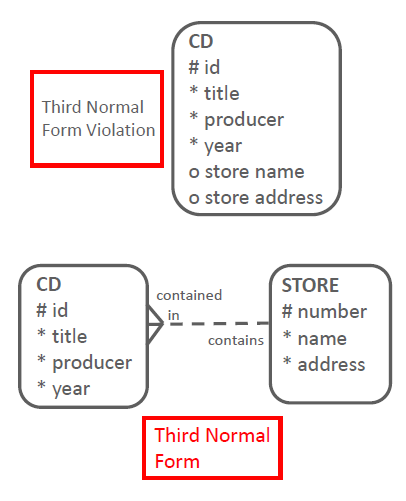
*Second Normal Form (2NF)*

Second Normal Form requires that any non-UID   
attribute be dependent on (be a property of, or a characteristic of)   
the entire UID.

*Third Normal Form (3NF)*

The rule of Third Normal Form (3NF) states that no non-UID attribute can be dependent on another non-UID attribute.

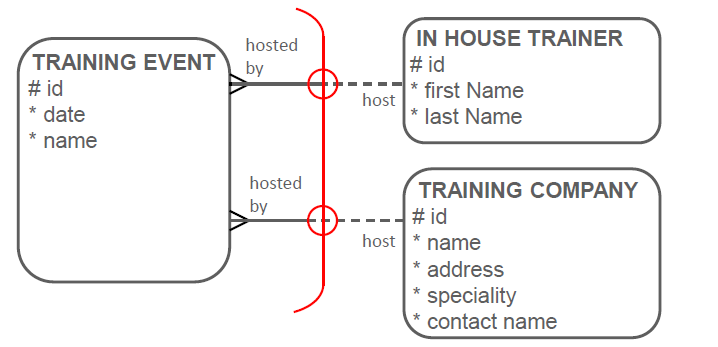
•Third Normal Form prohibits transitive dependencies.



**Arcs**

•An Exclusive OR relationship is a relationship between one entity and two (or more) other entities where only one of the relationships can exist at a time

•In ERDs, we model this type of relationship with an Arc

For example: a TRAINING EVENT can be hosted by either an IN HOUSE TRAINER or an external TRAINING COMPANY.

•An arc is represented on an ERD as a solid line with curved ends.

•A circle is drawn on the arc for every relationship that is part of the arc.

•An arc always belongs to one entity.

•Arcs can include more than two relationships.

•Not all relationships of an entity need to be included in an arc.

•An entity may have several arcs.

•An arc should always consist of relationships of the same optionality.

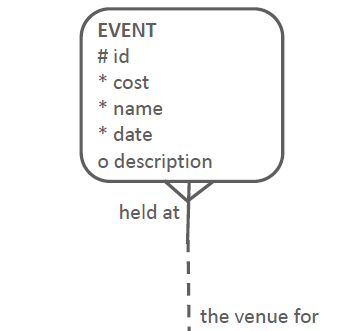
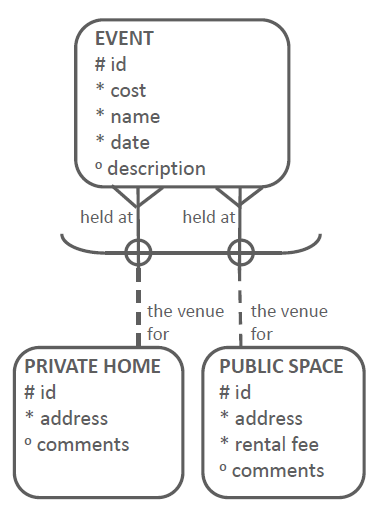
•All relationships in an arc must be mandatory or all must be optional.

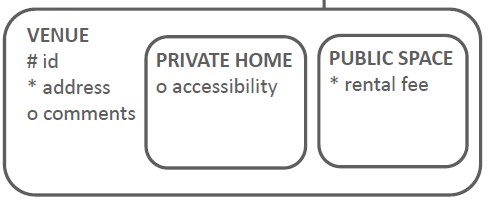
•Relationships in an arc may be of different cardinality, although this is rare.

**Arcs, Supertypes, and Subtypes**

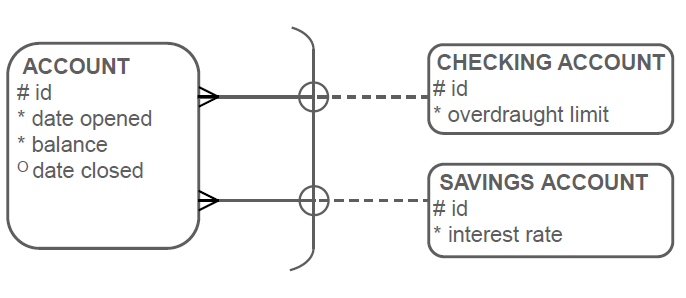
Certain situations are best modeled as an arc, and others as supertype and subtypes.

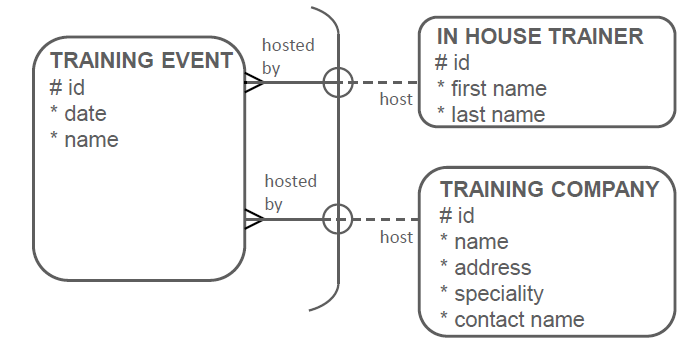
• If the entities that are related through the arc are similar, there may be a case for creating a super/subtype without an arc.



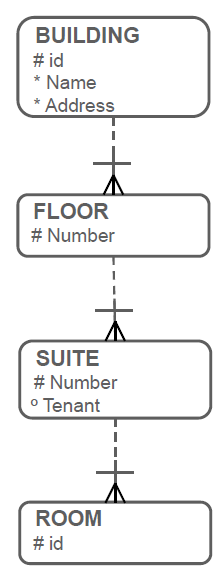


• If the entities that are related through the arc are similar to the arc entity then there may be a case for creating a super/subtype without an arc



• Example 3: IN HOUSE TRAINER and TRAINING COMPANY are NOT types of TRAINING EVENT, and they do not share common attributes. This is best to model with an arc.

**Hierarchies and Recursive Relationships**



*Hierarchy*

•Notice the barred relationships.

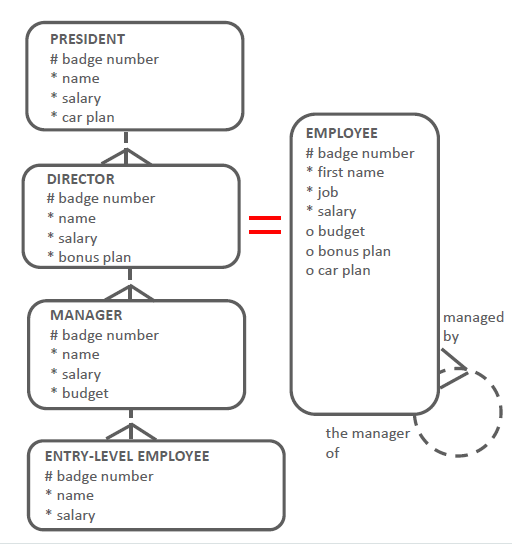
•Here you have a case of the cascading UIDs

•Here each entity’s UID is a combination of its

UID and the UID of the entity above it.

*Recursive Relationship*

•Hierarchical: Hierarchical structures are more explicit

 and are easier for most people to understand because

they are very similar to an organizational chart.

•Each entity can have its own mandatory attributes

and relationships if the business requires this

(Instead of all optional attributes and relationships,

as you would have in a recursive).

•In this way, your data model truly reflects the

business rules.

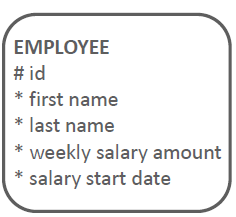
•Recursive: Recursive relationships tend to be simpler

because you are using only one entity.

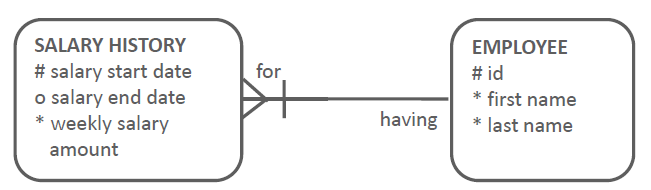
•Your diagram will be less “busy.”

•However, they are less specific –you cannot have

mandatory attributes or relationships unless they are mandatory in all instances of the entity.

**Modeling Historical Data**

•To model salary changes over time, add a SALARY HISTORY entity.



•The UID of the SALARY HISTORY entity is the related EMPLOYEE id and the salary start date.

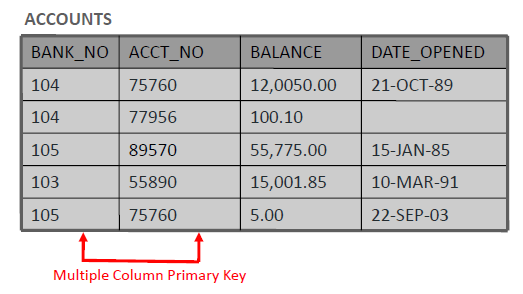
**Primary Key**

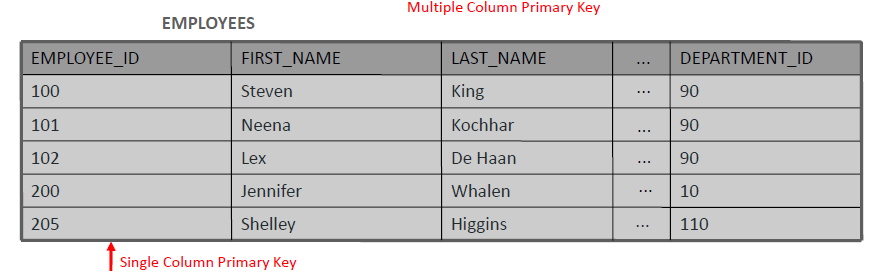
•A primary key (PK) is a column or set of columns that uniquely identifies each row in a table.

•Each table should have a primary key, and a primary key must be unique.

•No part of the primary key can be null.

•A table can have more than one column, or combinations of columns, that could serve as the table’s primary key.

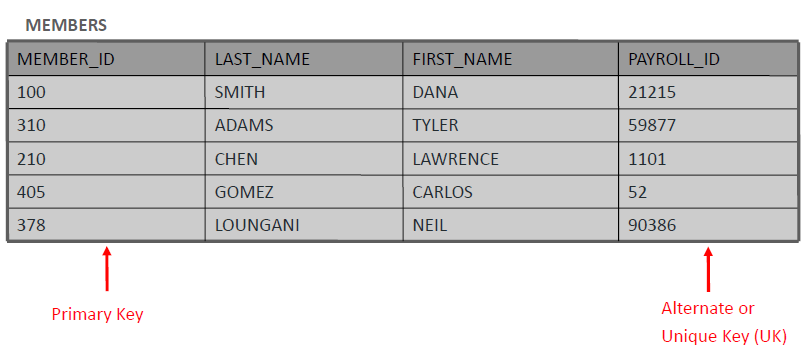




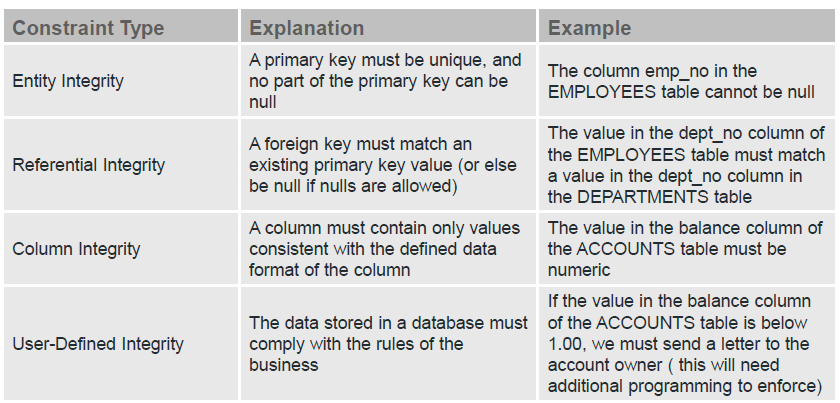
•Each column, or combination of columns, is called a "candidate" key because it could be selected for use as the primary key.

•Select one candidate key to be the primary key for the table.

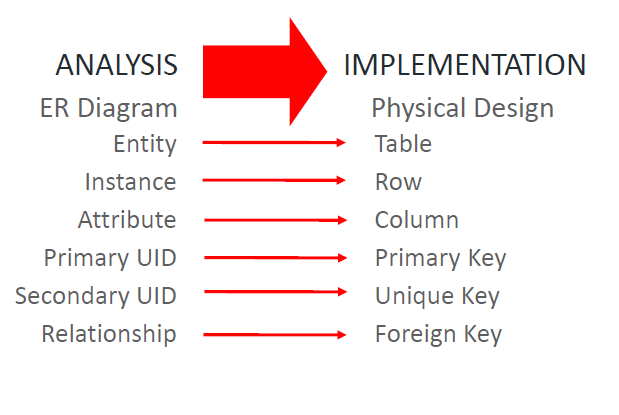
•The other candidates become alternate keys (or unique keys).

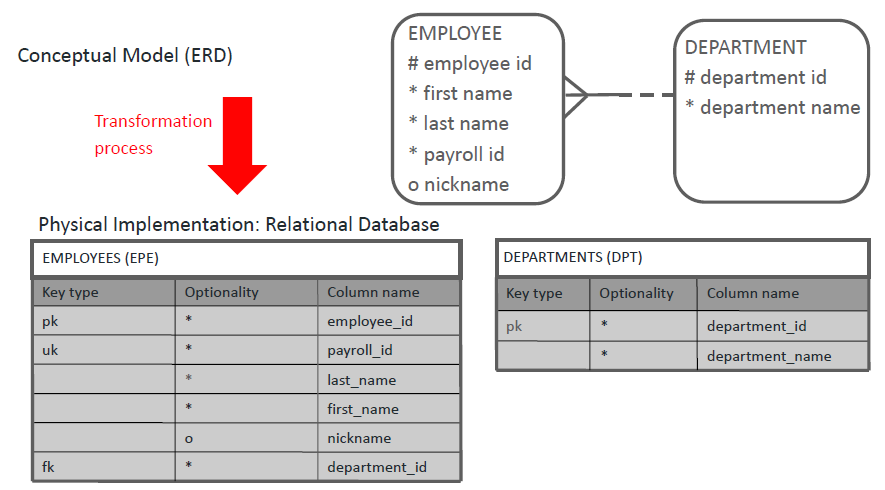


**Summary of Data-Integrity Rules**



**Transforming Conceptual To Physical**





**Naming Restrictions with Oracle**

Table and column names:

•Must start with a letter

•Can contain up to 30 alphanumeric characters

•Cannot contain spaces or special characters such as “!,” but “$,” “#,” and “\_” are permitted.

•Table names must be unique within one user account in the Oracle database.

•Column names must be unique within a table.

•Some words have a special meaning in the Oracle database and in the SQL programming language.

•These are called “reserved” words.

•It is best to avoid using these as names for your tables and columns.

•Some common examples of Oracle reserved words are:

–TABLE

–NUMBER

–SEQUENCE

–ORDER

–VALUES

–LEVEL

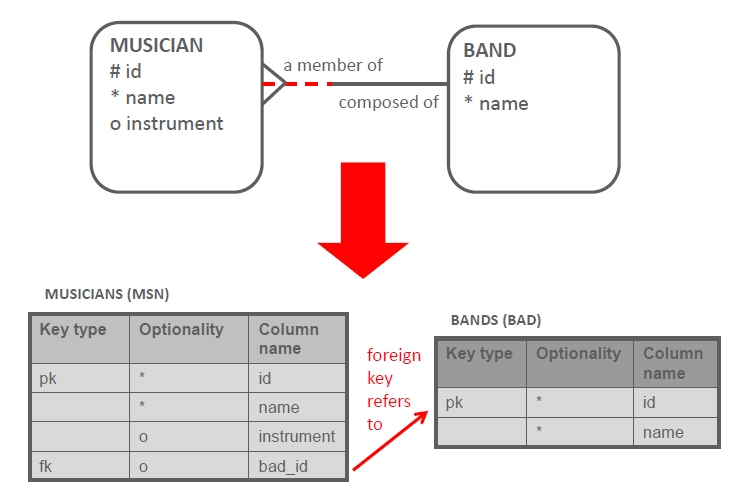
–TYPE

•A complete list can be found on otn.oracle.com.

**Mapping of Relationships**

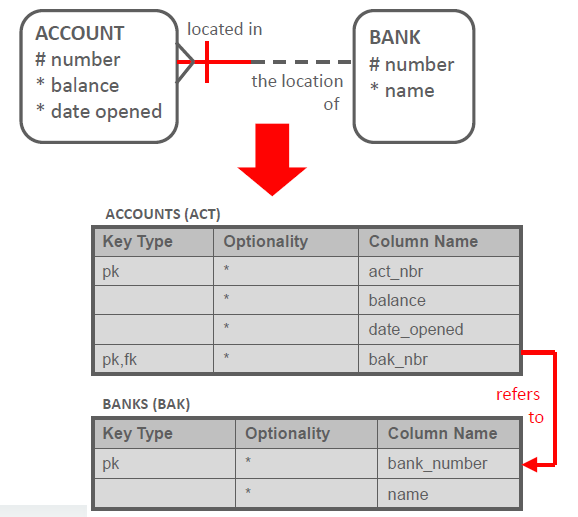
*In general :*

•A relationship creates one or more foreign-key columns in the table on the many side of the relationship.

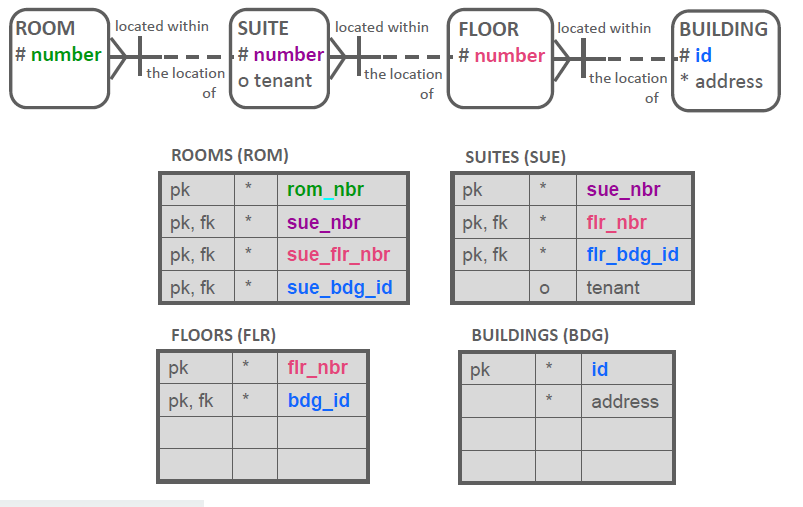
•If the relation is optional on the many side then the fk is optional , if the relation is mandatory on the many side then the fk is mandatory .

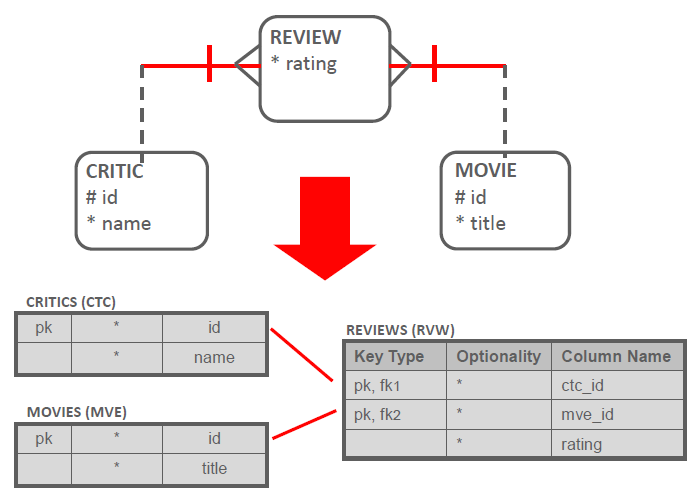
•A barred relationship is mapped to a foreign-key column on the many side, just like any other 1:M relationship.

•In this case, the foreign-key column plays a double role because it is also part of the primary key.



•Cascade Barred Relationships

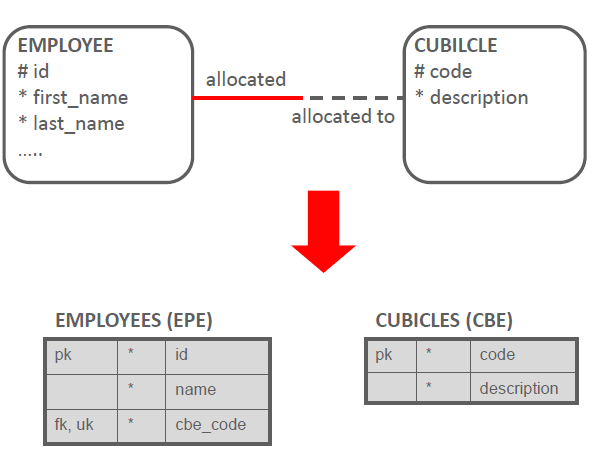


•Many-to-Many Relationships

One-to-One Relationships

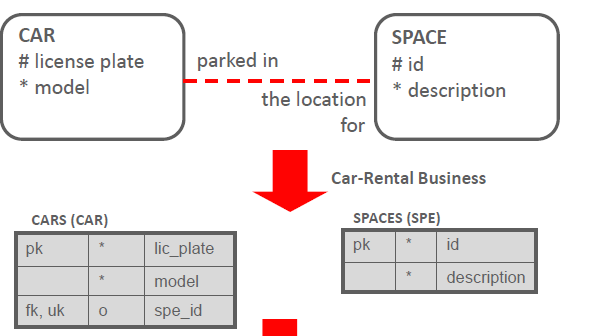
•When transforming a 1:1 relationship, you create a foreign key and a unique key.

•All columns of this foreign key are also part of the unique key.

•If the relationship is mandatory on one side, the foreign key is created in the corresponding table.

Optional One-to-One

•If the relationship is optional on both sides, you can choose which table gets the foreign key.



•If the relationship is mandatory at both ends, you have the same limitation in the database as a 1:M relationship that is mandatory at the one end.

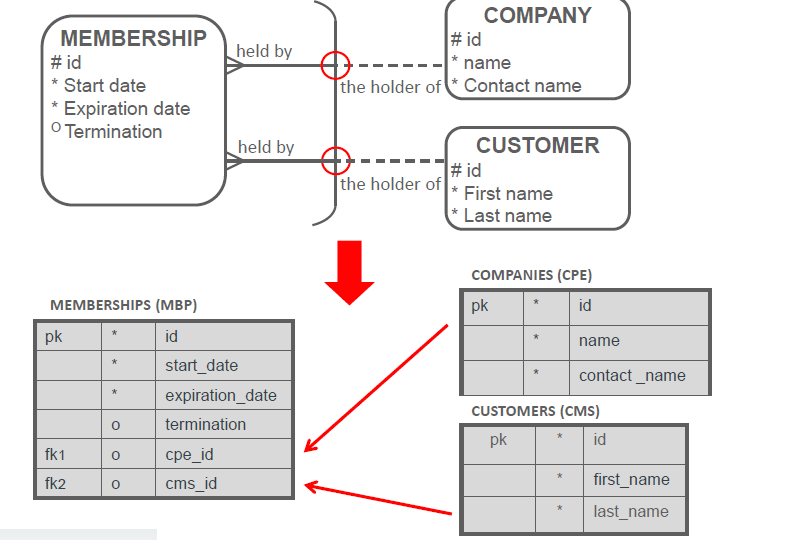
•Therefore, you would need to write additional code to enforce it.

Mapping arcs

•The entity that has the arc will map to a table that contains foreign keys from the tables on the “one” end of the relationships.

•Note that even if the relationships in the arc are mandatory on the many side, the resulting foreign keys have to be optional (because one of them will always be blank).

•Since the arc represents exclusive relationships, additional code is needed to enforce that only one of the foreign keys has a value for every row in the table.



•A nontransferable relationship in the conceptual model means that the foreign-key column in the database table cannot be updated.

•The foreign-key constraint by itself cannot enforce this in the database.

•Additional programming will be needed to make sure that the database follows this business rule.

**Supertype Implementation: Single Table**

•This choice produces a single table for the implementation of the supertype entity and its subtypes.

**Rules:**

–Tables: Only one table is created, regardless of the number of subtypes.

–Columns: The single table gets one column for each attribute of the supertype, along with the original optionality of the attribute.

–The table also gets a column for each attribute belonging to the subtype, but the columns all become optional.

–Additionally, a mandatory column should be created to act as a discriminator column to distinguish between the different subtypes of the entity.

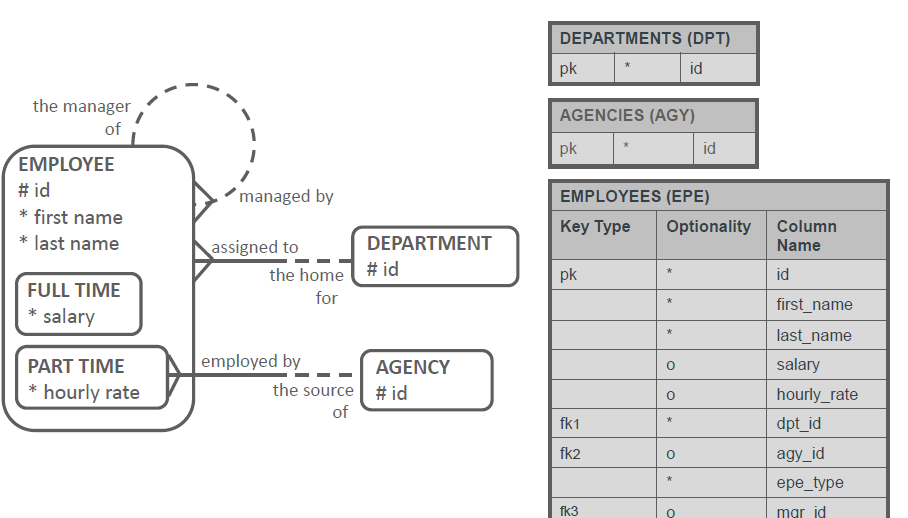
–The value it can take is from the set of all the subtype short names (FTE, PTE, OTR in the example).

–This discriminator column is usually called <table\_short\_name>\_type, which would be epe\_typein the example.

–Identifiers: Unique identifiers transform into primary and unique keys.

–Relationships: Relationships at the supertypelevel transform as usual. Relationships at the subtype level are implemented as optional foreign-key columns.

–Integrity constraints: A check constraint is needed to ensure that for each particular subtype, all columns that come from mandatory attributes are not null.



*When Do You Choose the Single Table/Supertype Implementation?*

•The single-table implementation is a common and flexible implementation.

•It is the one you are likely to consider first and is especially appropriate where:

–Most of the attributes are at the super type level.

–Most of the relationships are at the super type level.

–Business rules are globally the same for the subtypes.

**Subtype Implementation: Two Table**

•You create a table for each of the subtypes.

•So, in reality, you could have more than two tables, if you had more than two subtypes.

**Rules :**

–Tables: One table per first-level subtype.

–Columns: Each table gets one column for each attribute of the supertypealong with its original optionality.

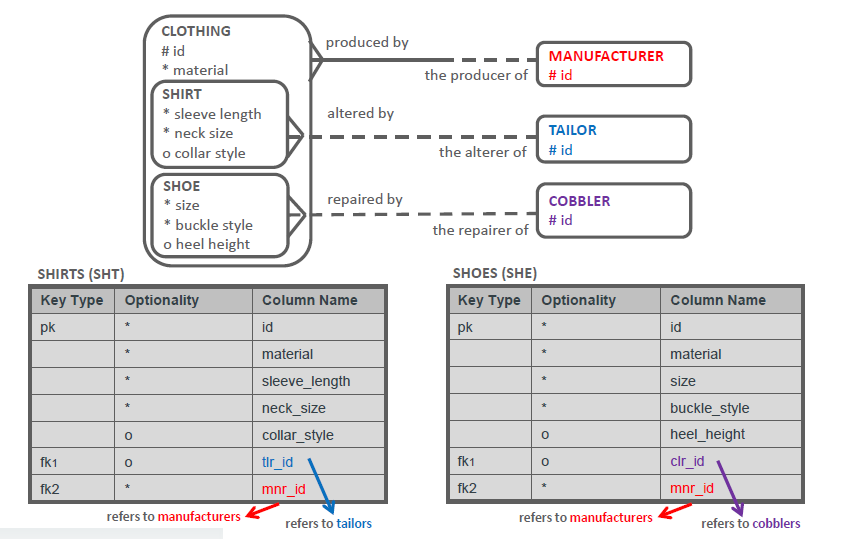
–Each table also gets one column for each attribute belonging to the subtype along with its original optionality.

–Identifiers: The primary UID at the supertypelevel creates a primary key for each table. Secondary UIDs of the supertypebecome unique keys in each table.

–Relationships: All tables get a foreign key for a relationship at the supertypelevel, with the original optionality.

•For relationships at the subtype levels, the foreign key is implemented in the table it is mapped to.

•Original optionality is retained.



*When to Consider Subtype Implementation*

Subtype implementation may be appropriate when:

•Subtypes have very little in common. There are few attributes at the supertypelevel and several at the subtype level.

•Most of the relationships are at the subtype level.

•Business rules and functionality are quite different between subtypes.

•How tables are used is different --for example, one table is being queried while the other is being updated.

**Supertypeand Subtype (Arc) Implementation**

•This choice produces one table for every entity.

•The supertype table has a foreign key for each subtype table.

•These foreign keys represent exclusive relationships.

•They are optional because only one of them can have a value for each row in the table**.**

**Rules:**

–Tables: As many tables are created as there are subtypes, as well as one for the supertype.

–Columns: Each table gets a column for all attributes of the entity it is based on, with the original optionality.

•identifiers: The primary UID of the supertype level creates a primary key for each of the tables.

–All other unique identifiers become unique keys in their corresponding tables.

•Relationships: All tables get a foreign key for a relevant relationship at the entity level, with the original optionality.

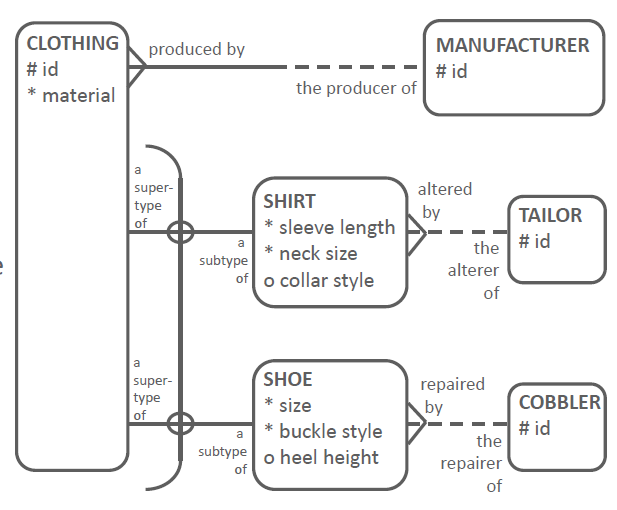
•Integrity constraints: Two additional columns are created in the table based on the supertype.

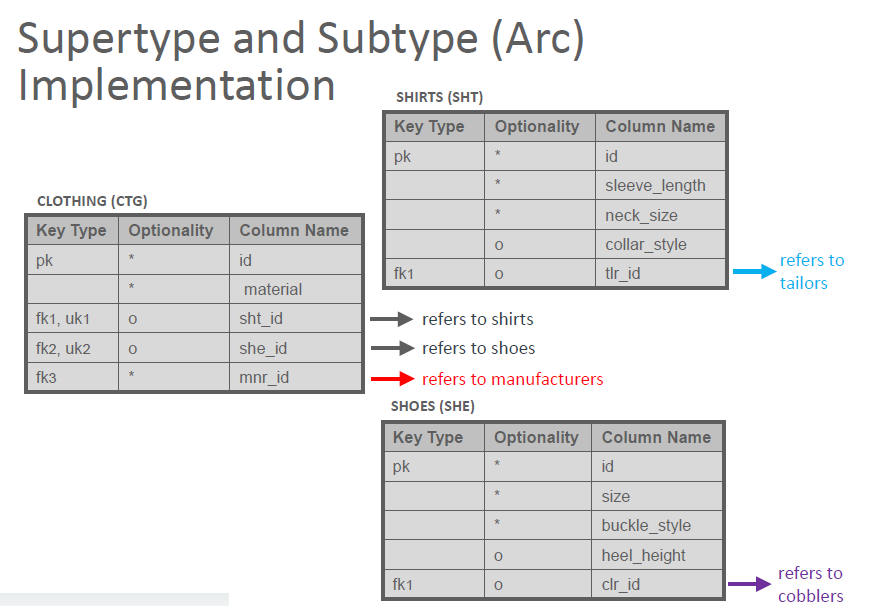
•They are foreign-key columns referring to the tables that implement the subtypes.

•The columns are optional because the foreign keys are in an arc.

•An additional check constraint is needed to implement the arc.

•The foreign-key columns are also unique keys because they implement a mandatory 1:1 relationship.





*When to Consider Both a Supertype and Subtype (Arc) Implementation*

•This implementation is rarely used, but it could be appropriate when:

–Subtypes have very little in common and each table represents information that can be used independently.

–For example, when the CLOTHING table gives all global information, and both SHOES and SHIRTS give specific information, and the combination of global and specific information is hardly ever needed.

–Business rules and functionality are quite different between all types.

–How tables are used is different.