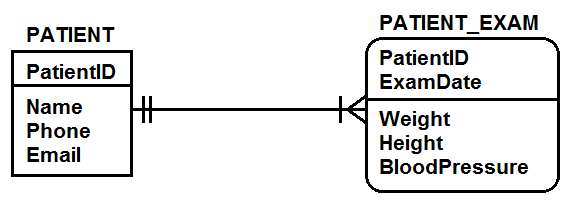
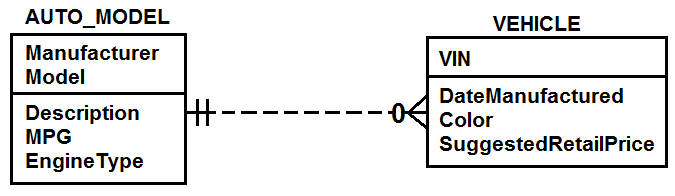
**CS 31 Database Management Programming Lecture 9**

**Identifying Relationship:**



**Nonidentifying Relationship:**



**Strong Entity** – any entity whose existence in the database does not depend on the existence of any other entity

**Weak Entity** – an entity whose logical existence in the database depends on the existence of another entity; the minimum cardinality of the related entity is mandatory

**ID-Dependent Weak Entity** – a weak entity whose identifier contains the identifier of a second entity - designated by drawing box with rounded corners

**NonID-Dependent Weak Entity** – a weak entity whose identifier does not contain the identifier of a second entity

All ID-dependent entities are weak, but not all weak entities are ID-dependent.

The parent entity of a child ID-dependent entity is sometimes referred to as an owner entity.

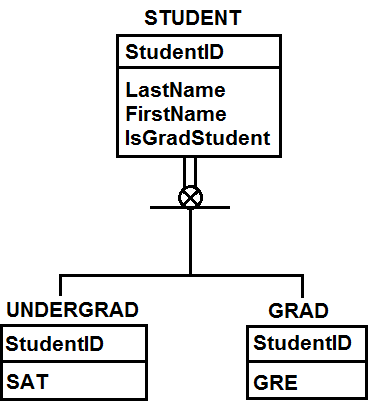
**Identifying Relationship** – a relationship between two entities such that one is ID-dependent on the other; drawn with a solid line

**Nonidentifying Relationship** – a relationship between two entities such that one is not ID-dependent on the other; drawn with a dashed line

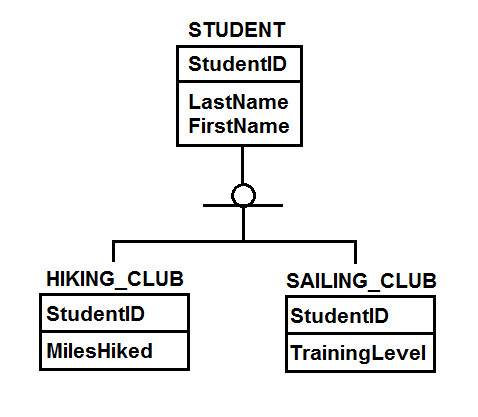
The entities involved in nonidentifying relationships do not need to depend on one another – 2 strong entities can be related. If the nonidentifying relationship is between a weak entity and the strong entity it depends on, then this is noted in the model.

There is some disagreement between texts on what constitutes a weak entity. In some cases, any entity that participates in a relationship with a mandatory minimum cardinality of another entity is weak.

**Exclusive Subtype:**



**Inclusive Subtype:**



**Subtype** – an entity or object that is a subspecies or subcategory of a higher-level type, called a supertype

**Supertype** – an entity or object that logically contains subtypes

A reason for using supertype/subtype in the model is to avoid value-inappropriate nulls. It is an IS-A relationship.

**Discriminator** – an attribute of a supertype entity that determines which subtype pertains to the supertype

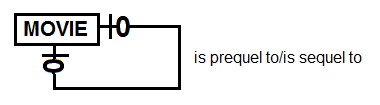
**Exclusive Subtypes** – a subtype in which a supertype instance is related to at most one subtype in a set of possible subtypes; may include discriminator

**Inclusive Subtypes** – a subtype that allows a supertype entity to be associated with more than one subtype; no discriminator

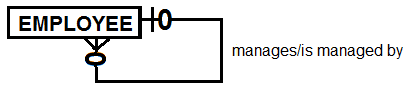
The distinction of partial and total relationships between supertypes and subtypes is sometimes shown in the model. Partial means instances of the supertype are not required to be included in a subtype. Total relationships mean a supertype instance must also be a subtype instance. A hash mark is drawn just below the supertype to designate the total relationship.

**Recursive Relationship** – a relationship among entities or rows of the same type

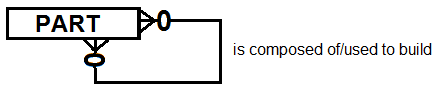
**1:1**



**1:M**

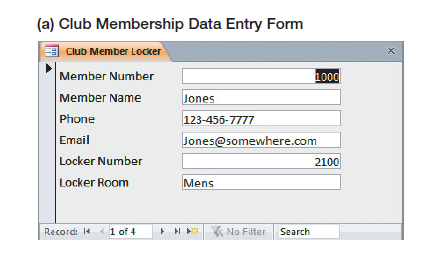


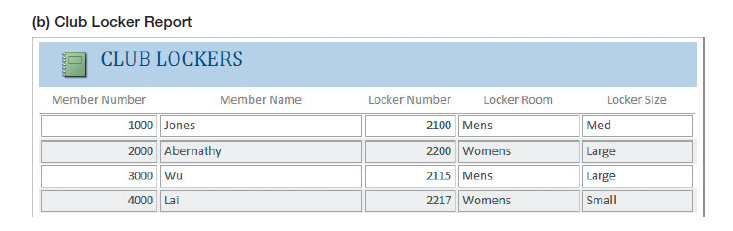
**N:M**



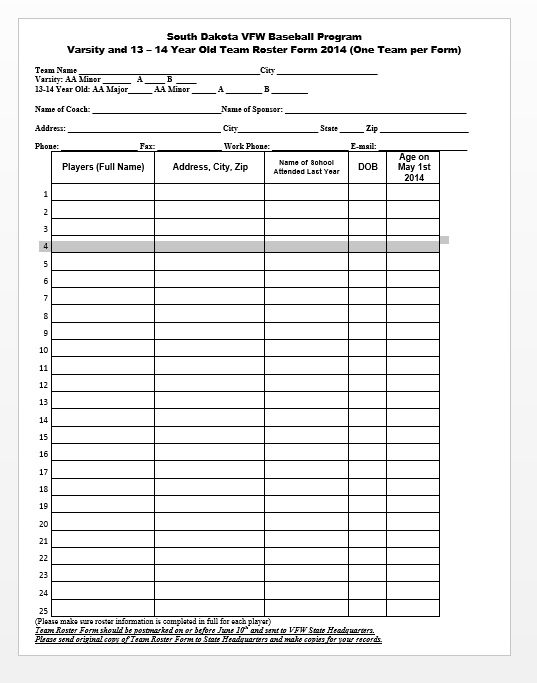
**Form/Report Patterns**

**1:1 Strong Entity Relationships**

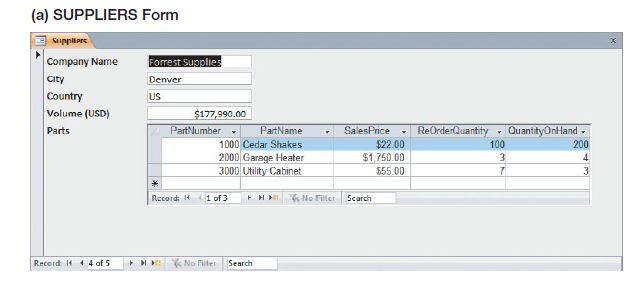


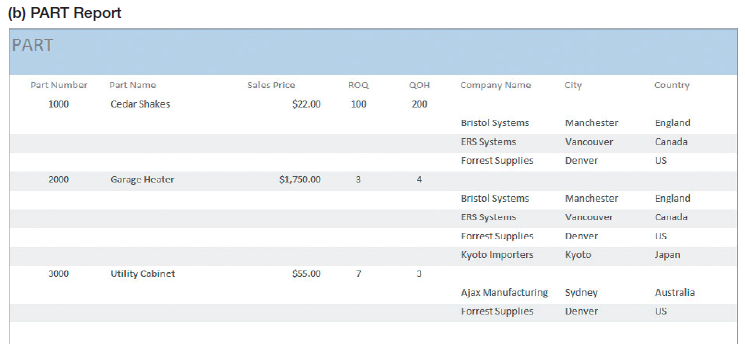


**1:N Strong Entity Relationships**

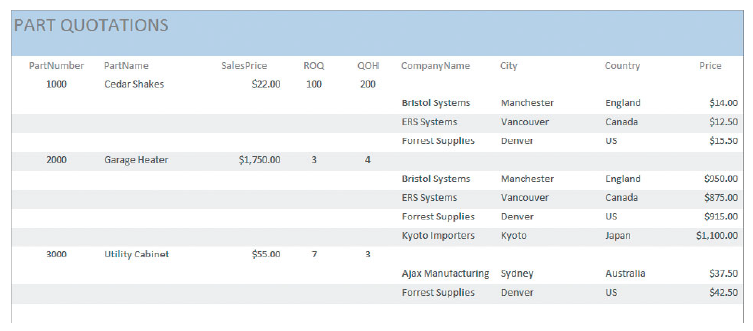


**N:M Strong Entity Relationships**

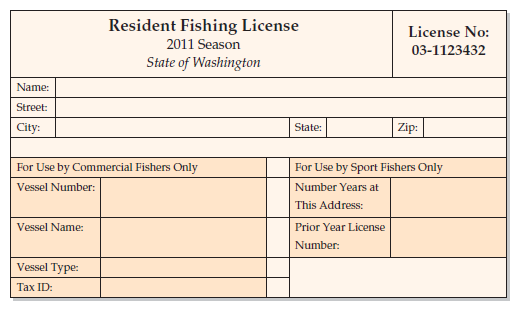




**ID-Dependent Relationships – Association Pattern**



**Supertype/Subtype Relationships – For-Use-By Pattern**



**Lab Lecture 9**

**SQL Join Operation** – The process of combining data rows from two tables by using a relational algebra operation on two relations, X and Y, which produces a third relation Z. A row of X is concatenated with a row of Y to form a new row in Z if the rows in X and Y meet a restriction concerning their value. Normally, the restriction is that one or more columns in X (X1) equal one or more columns in Y (Y1). The join of X with Y in which X1 = Y1 will result in a relation, Z, having the concatenation of rows in X and Y in which the value X1 equals the value of Y1. In theory, restrictions other than equality are allowed (X1 > Y1) but such non-equal joins are not used in practice.

We will be concatenating rows from two or more tables based on the result of evaluating a conditional expression.

**Explicit Join** – a SQL join statement that used the SQL **JOIN ON** syntax

**Implicit Join** – a SQL join statement that does not use the SQL **JOIN ON** syntax

Since we are joining two tables and there is a possibility of having the same column name(s) from different tables. We use the **TABLE\_NAME.ColumnName** to distinguish between two columns with the same name but from different tables.

**Cross Join (Cartesian Product)** – Every row from TABLE1 (n rows) is concatenated to every row of TABLE2 (m rows). The result table will have n m rows.

The relational operator used in this class will always be equality (equi join). This is not required. Other relational operators are allowed but rarely used.

**JOIN ON syntax**

For two tables, **JOIN** replaces the comma and the **ON** replaces the **WHERE**. The **WHERE** clause can still be used to restrict rows from displaying in resultant table.

We have been writing inner joins. We can also write **INNER JOIN** instead of just **JOIN**.

SELECT retail\_order.OrderNumber, OrderTotal,

order\_item.OrderNumber, ExtendedPrice

FROM retail\_order, order\_item

WHERE retail\_order.OrderNumber = order\_item.OrderNumber

ORDER BY retail\_order.OrderNumber;

SELECT retail\_order.OrderNumber, OrderTotal,

order\_item.OrderNumber, ExtendedPrice

FROM retail\_order JOIN order\_item

ON retail\_order.OrderNumber = order\_item.OrderNumber

ORDER BY retail\_order.OrderNumber;

