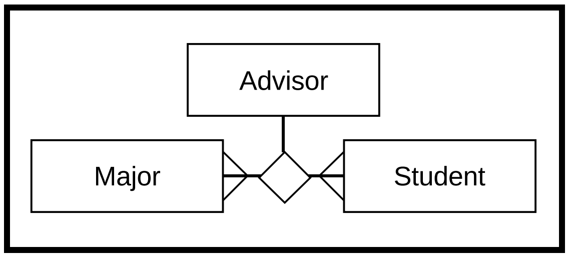
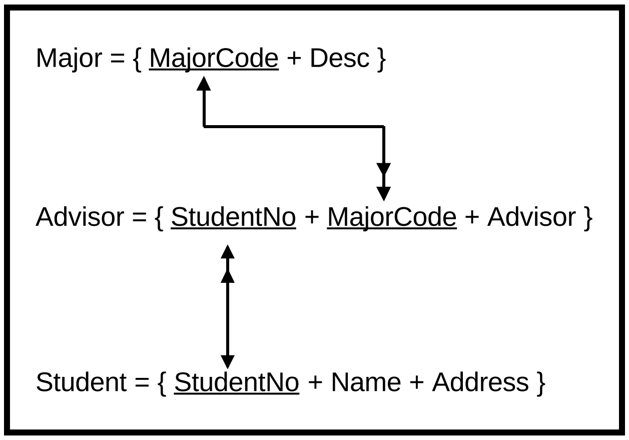
Guide To Data Modeling




Major = { MajorCode + Desc }
Advisor = { StudentNo + MajorCode + Advisor }
Student = { StudentNo + Name + Address }

Major Table

MajorCode	Desc
BA	Business
ENG	Engineering
PHY	Physics

Student Table

StudentNo	Name	Address
6634413	Burrows	123 Main
8743232	Brown	710 Terry
8851000	Welch	4610 17th
9243255	Jones	1701 46th
9659844	Smith	833 Talbert

Advisor Table

StudentNo	MajorCode	Advisor
6634413	BA	Franklin
6634413	PHY	Leung
8743232	BA	Franklin
9659844	BA	Adams
9659844	ENG	Quale
9659844	PHY	Olsen

William E. Burrows

Guide To Data Modeling

Table of Contents

Entity-Relationship (ER) Diagrams

An Entity-Relationship (ER) diagram provides a graphical model of the things that the organization deals with (entities) and how these things are related to one another (relationships). An ER diagram is a high-level, logical model used by both end users and database designers to document the data requirements of an organization. The model is classified as "high-level" because it does not require detailed information about the data. It is called a "logical model" because it provides a conceptual understanding of the data and as opposed to actually defining the way the data will be stored in a database (which is referred to as the "physical" model).

Entities and Attributes

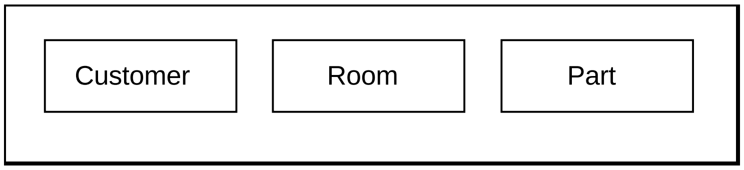
Entities are things. They can be tangible things, like a classroom or a part, or they can be intangible things, like a purchase or a meeting. Attributes (or fields) represent facts about an entity. For example, an Employee entity might be described by social security number, name, and address attributes.

Entities are described by at least two attributes. One attribute is a unique identifier (often referred to as the key field). The value of this attribute must be unique for each occurrence of an entity. For example, if there were dozens of employees, then there would be no duplicate values of the key field for any of the employees. In addition to the key field, there must be at least one other attribute that provides a fact about an entity. That is, an entity would never be described solely by a key field.

As an example, we might have an entity called Customer. Attributes describing Customer might include a customer number as the key field (to allow us to uniquely identify a customer) plus other facts about a customer such as name, address, and age.

We use a rectangle enclosing the entity name to graphically represent an entity on an ER diagram. Figure 1 shows three entities.

Figure 1
Entities are shown as named rectangles on an ER diagram.



Note that the symbol shows only the name of the entity and does not include the attributes. Some data modeling methodologies also include the names of attributes but we will not use that convention here. Also be aware that an entity represents a many of the actual thing, e.g., Customer represents many different actual customers (sometimes referred to as instances).

Relationships

Different entities can be related to one another. For example, in a university, the Student and the Course entities are related because students "enroll in" courses. A database must store not only information about the Student and Course entities, but it must also store the relationship between Student and Course. For example, the user of the database should be able to ask for a list of courses taken by a specific student or ask for a list of students currently enrolled in a specific course.

The relationship between a Student and Course is called a binary relationship because it relates to two entities. A relationship between three entities is called a ternary relationship; when four or more entities participate in a relationship, it is generally called an n-ary relationship. We restrict our discussion here to binary relationships but you should be aware that more complex relationships exist.

Cardinality

When performing data modeling in preparation for designing a database, knowing that two entities are related to each other is not sufficient. In addition to knowing that they are related, the cardinality of the relationship must also be documented. Cardinality is the numerical mapping between entities. This describes "how many" of one entity are related to "how many" of another entity. For example, we can say that a Student entity is related to many Course entities and a Course entity is related to many Student entities.

Notice that cardinality has a "directional" meaning. That is, when we say that a Student entity is related to many Course entities, we know nothing about the relationship in the other direction (Course to Student). For this reason, an ER diagram must document (and be read in) both directions.

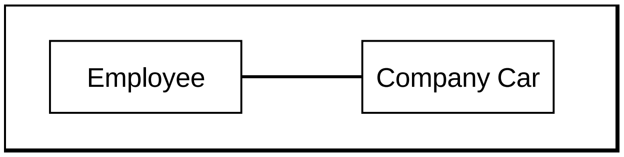
We will investigate three types of cardinality—one-to-one, one-to-many, and many-to-many. Each is discussed in detail below.

One-to-One (1:1)

An example of a one-to-one relationship might be the relationship between a Company Car entity and an Employee entity. An employee is assigned one company car and a company car is assigned to one employee. Note that this relationship might not be true in all companies, that is, in some companies an employee might be assigned to several company cars and/or a company car could be assigned to several employees. It is important to model whatever relationships and cardinality are true for the particular organization under study.

We show a one-to-one relationship on an ER diagram by connecting the two entities with a straight line. Figure 2 shows an example for the Employee and Company Car relationship.

Figure 2
A one-to-one relationship between two entities.



You read this diagram in both directions. Starting with Employee, you say that "an Employee is related to one Company Car" and starting at Company Car, you say that "a Company Car is related to one Employee."

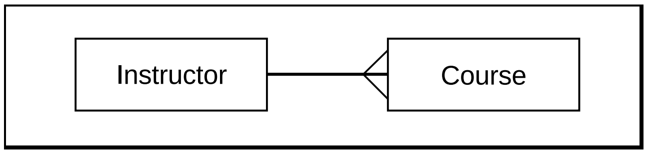
One-to-Many (1:N)

One-to-many relationships are very common. For example, we might have a relationship between an Instructor entity and a Course entity. We might find that an instructor is related to (teaches) several courses while each course has a single instructor. All that is necessary for the relationship to be "many" is for an entity to be related to more than one of another entity.

Again note that this relationship—an instructor is related to several courses, while each course has a single instructor—may not be true at all schools. For example, the relationship does not support team teaching where several instructors teach a course. Again, the relationship that is modeled should reflect the reality of the organization.

We show a one-to-many relationship on an ER diagram by connecting the two entities with a straight line and place a "fork" at the "many" end. Figure 3 shows an example for the Instructor and Course relationship.

Figure 3
A one-to-many relationship between two entities.



You read this diagram by starting at either entity and following the line to the other entity. As you encounter your destination entity, you note the cardinality. Thus, starting at the Instructor entity, you follow the line to the Course entity and note the "many" fork symbol. Thus you read: "Instructor is related to many Courses". Starting with the Course entity and going the other direction, you encounter the Instructor entity and observe the lack of a fork (meaning "one"). You read: "Course is related to one Instructor."

As a standard convention in ER diagrams, always write the singular form of the entity and let the relationship's cardinality attach the plural interpretation if appropriate.

Many-to-Many (N:M)

Using the Instructor-Course example, what if we really wanted to model the situation where several instructors taught a course? We would need to model a many-to-many relationship to do this. Thus, a course could be taught by many instructors and an instructor could teach many courses.

Figure 4 shows how to construct the Instructor-Course many-to-many relationship on an ER diagram.

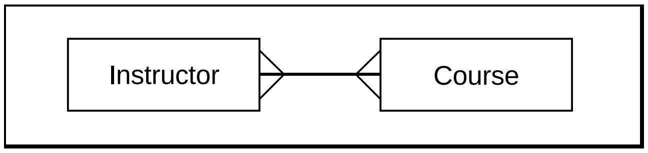


Figure 4
A many-to-many relationship between two entities.

Regardless of which entity you start at, when you encounter your destination entity, you also encounter a "many" fork symbol. Thus, you say that "an Instructor is related to many Courses" and "a Course is related to many Instructors."

Sometimes additional information exists that adds information about a many-to-many relationship. For example, consider the ER diagram in Figure 5.

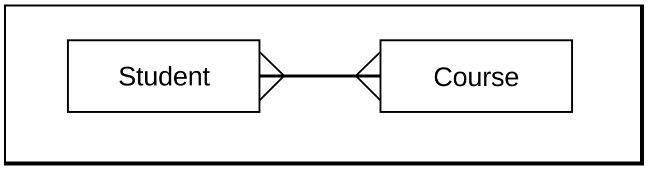


Figure 5
A many-to-many relationship between Student and Course entities.

Here the Student entity is related to many Course entities and vice versa. What if we wanted to store the grade a student received on each course? Where would we store this attribute? It is not really a fact about a student. If I told you the student's name and asked you for the grade, you would ask me to tell you the course identifier for the course I was interested in. For the same reason, the grade is not a fact about a course. If I told you the course identifier, you would ask me which student I was talking about.

The grade is a fact about the relationship of a student completing a course. That is, it requires information on both a specific student and a specific course. This type of situation can only occur in the context of a many-to-many relationship. It results in a new entity called an "associative object."[*] If an associative object exits, it occurs between two entities that are in a many-to-many relationship. It is shown on an ER diagram as an entity connected between the two entities in the many-to-many relationship as depicted in Figure 6.

[*] Shlaer and Mellor, Object-Oriented Systems Analysis: Modeling the World in Data, Yourdon Press, 1988.

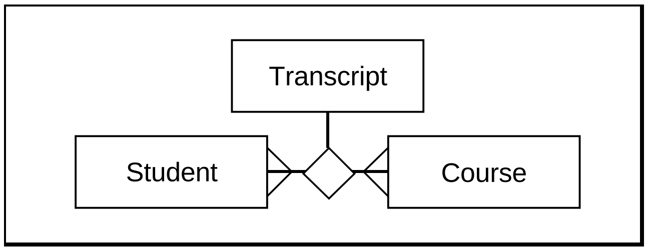


Figure 6
A more complex many-to-many relationship showing an associative object.

In this example, Transcript is an associative object entity. It would include an identifier of the student (the student key field), an identifier of the course (the course key field), and the grade for a specific student and course. If the student could repeat a course more than once with a different grade for each occurrence, then the Transcript associative object would also include the quarter the student took the course. This additional field would be necessary because of the requirement that each entity have a unique key.

Associative objects are somewhat abstract and conceptually difficult for some. We will cover the topic of associative objects in more detail later when we introduce Record Structure Diagrams.

Alternative ER Diagram Symbols

There are a number of different conventions used for ER diagram symbols. The symbols used in this document represent a simplified version of symbols first used by Peter Chen in 1976. You may encounter different symbols in later database classes or in an organization that uses a methodology other that the Chen methodology. However, the basic ideas are still the same regardless of the symbols used in the diagrams.

In addition, some ER diagramming methodologies refine the cardinality rules to include "zero or one" and "zero or more" (both of these mean optional). For example, in a personnel system, an Employee entity could be related to zero or one Spouse entity. These refinements add to the accuracy of the relationships and are important in large-scale database analyses. However, their implications are beyond the scope of this introductory-level guide.

Example ER Diagram

The following narrative describes an architectural firm. Study the narrative as if an analyst had interviewed people in the firm to see how it functions.

> "An architectural design firm has a number of design projects underway at any point in time. Each of these projects is identified by a unique project identification code. Additional data stored for each project includes a description, a target completion date, and a dollar budget amount.
>
> Design associates staff these projects. These associates include architects, engineers, and computer CAD (Computer-Aided Design) specialists. Each associate is identified by a unique code. Other information about associates includes a field for their name and a field for their specialty. A team of associates who are assembled to provide a unique, competitive group for each specific project staffs projects. It is also common to find any particular associate assigned to several projects. This is particularly true when projects are small. The firm likes to keep track of each of the associate's commitment to a project and they do this by recording the percentage of an associate's time that is allocated to a specific project.
>
> The firm has three partners who manage the projects the firm has under contract. Information about partners includes a unique partner identifier, a field for their name, and a field for their phone number. Since there are only three partners, they each have several projects for which they have responsibility. However, to maintain accountability, each project is managed by a single partner."

Using this information, we want to construct an ER diagram. Thus we need to identify the entities and the relationships that exist between them. The first entity we see is Project. It meets the minimum conditions for an entity because it has a unique identifier (project identification code) and there are additional facts (description, target completion date, and budget) that need to be stored about the entity.

Another entity is Design Associate. It too can be uniquely identified and there are additional facts that need to be stored about it.

There is a clear "staffing" relationship that exists between the Project entity and the Design Associate entity ("Projects are staffed by a team of associates…"). That is, a Project can have many Design Associates. The narrative also states that "It is also common to find any particular associate assigned to several projects." Thus we have a many-to-many relationship between the two entities, Project and Design Associate.

Any time we have a many-to-many relationship, we need to consider whether an associative object is used to support the many-to-many relationship. That is, are there additional facts we need to store that describe the relationship between the Design Associate entity and the Project entity? Consider the following statement: "The firm likes to keep track of each of the associate's commitment to a project and they do this by recording the percentage of an associate's time that is allocated to a specific project". Where do we store this percentage figure? We cannot store it with the Project entity because there are many different percentage values (one for each associate). Similarly, we cannot store this fact with the Design Associate entity because there are also many values (one for each project an associate is assigned to). This percentage figure is a fact about the particular association of a Project entity and a Design Associate entity, and thus, it is an associative object. We will call this associative object "Assignment". The ER diagram for these entities and the relationships is shown in Figure 7.

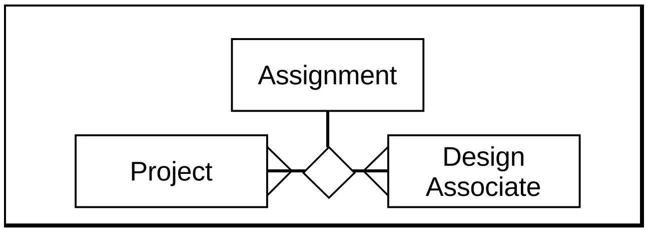
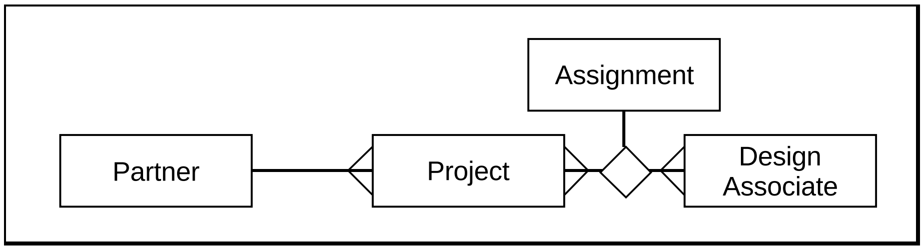


Figure 7
N-M relationship between Project and Design Associate supported by the "Assignment" associative object.

This completes the relationship that exists between the Project entity and the Design Associate entity. We now turn to the Project and Partner entities. We see that a relationship exists because "The firm has three partners who manage the projects…" What is the cardinality of the relationship? A Partner entity is related to many Project entities as indicated by the fact that "they each have several projects for which they have responsibility." On the other hand, a Project entity is related to only one Partner entity because "each project is managed by a single partner." So we have a one-to-many relationship between partners and projects. Since our original ER diagram already shows the Project entity, we simply add this new information as shown in Figure 8.

Figure 8
Final ER diagram.



Converting ER Diagrams to Relational Tables: Record Structure Diagrams (RSD)

The ER diagram is an abstract, logical model of the entities and relationships that are of interest to an organization. This model is independent of the specific details associated with designing a database. However, if one is to build a database, one must be able to somehow convert the information contained in the ER diagram into a specific database design. This section describes the process of converting an ER diagram into a relational database using a tool called a Record Structure Diagram.

Relational Databases

A relational database is made up of a set of tables (files). For example, Figure 9 shows three tables from a hypothetical relational database that supports a many-to-many relationship between students and courses.

Figure 9
Tables in a relational database.

Student Table

StudentNo	Name	Address
6634413	Burrows	123 Main
8743232	Brown	710 Terry
8851000	Welch	4610 17th
9243255	Jones	1701 46th
9659844	Brown	833 Talbert

Transcript Table

StudentNo	CourseNo	Grade
6634413	BA 301	3.2
6634413	BA 400	3.3
8851000	BA 301	3.5
9243255	FIN 350	2.9
9659844	BA 400	3.6
9659844	FIN 350	3.9

Course Table

CourseNo	Desc
BA 301	Intro to Info Sys
BA 400	Marketing
FIN 350	Intro to Finance

Each table is made up of a series of rows (records) and attributes (fields). Key fields, whose values are unique from one row to another, are underlined in the figure.

Notice that there are common attributes in the tables. These attributes are used by the relational database management system (RDBMS) to link or associate the rows from the various tables. For example, assume you wanted to see a list of students and the grades for courses they completed, and you wanted this list to include the student number, student name, course number, course description, and grade attributes. To provide this information, the RDBMS would combine (join) the Student and Transcript tables using the student number attribute which is common to the two tables. The RDBMS would then join the result of the first join to the Course table using the common course number attribute. The appropriate attributes would then be displayed from the resulting combination of the three joined tables.

When converting an ER diagram into a relational database, each entity becomes a separate table. In addition, attributes must be added to the tables and new tables must be created so that the sets of relationships inherent in the ER diagram are maintained. Finally, a key field must be defined for each table.

Identifying Key Fields

An important step in the process of designing a relational database involves the identification of a key field for each table. Sometimes the key field is easy to identify and other times it can be difficult. The following discussion attempts to clarify the process.

Recall that the values of a key field must be unique across all the records in a table. Another way to think about this is to observe that if we specify a value for the key field, then we will find at most one record in the table with a matching value. For example, look at the Student table in Figure 9. If you searched the records of this table trying to find a student number equal to 6634413, then you would find at most one match. If you found more than one match, then the student number field could not be called the key field. If you found no matches, this simply means that the student you were searching for did not exist in the table.

Looking at the Student table in Figure 9 again, search the student name field for the name “Brown”. How many matches did you find? Since you found two matches, the student name field could not be considered the key field. Even if you did not find two names that matched, the fact that you *might have found two that matched* would be sufficient to eliminate the student name field from key field status.

Sometimes there are two or more fields in a table that would be unique. In this case, you could choose any one of them. However, try to select one from the set where its values do not change over time and it always has a value (not empty). For example, considering student number as a potential choice for the key, it is likely that all students would have a student number, and it is unlikely that a university would change a student’s student number after it was assigned. Any field that fails to satisfy these conditions would not make a good key field.

Sometimes none of the individual fields by themselves would be unique. For example, consider the Transcript table in Figure 9. Notice that the student number field is duplicated for each course that a student completed. Similarly, the course number field is duplicated for each student who completed that particular course. However, if we assume that a student does not repeat a course, or, if the course is repeated, only the grade is changed, then we can combine the student number and course number fields and consider them together as the key field. In this case we would have a compound key.

Every table must have a key field, which may be a single attribute or a group of attributes (compound key). What if the assumption that a student does not repeat a course is false? That is, assume that we actually want to keep track of each course a student takes even if it is a repeat. We cannot do this with the data as defined in the Transcript table in Figure 9 because we cannot identify a key field. What is needed is an additional field added to the table so that uniqueness can be achieved. For example, we might add the quarter the course was completed. We would also have to include this field as part of the compound key. Figure 10 shows a revised Transcript table with this additional field.

Figure 10
Transcript table with an additional field and a new key field.

Transcript Table

StudentNo	CourseNo	Quarter	Grade
6634413	BA 301	Aut95	2.4
6634413	BA 400	Aut95	3.3
6634413	BA 301	Win96	3.4
8851000	BA 301	Aut96	3.5
9243255	FIN 350	Spr96	2.9
9659844	BA 400	Aut96	3.6
9659844	FIN 350	Aut96	3.9

If you have difficulty identifying a table's key, then try to image the table with several rows of actual data stored in it. Try combinations of data that do not violate any assumptions as you attempt to find a field or set of fields that are guaranteed to be unique. For example, consider the table in Figure 11. Only the first row of data has been supplied.

Figure 11
A table without the key field identified.

Skill Table

EmployeeNo	MachineNo	SkillLevel
123	23	4

Each of the following assumptions results in a different key field for the same table. Using a pencil and eraser, fill in some data using each assumption to verify that the key field identified in the assumption is correct.

1. *An employee does not operate more than one machine and has a single skill level associated with the operation of that machine.* That is, once you enter a row of data for a specific employee, you'll never enter another row for that same employee. The key field in this case would be the employee number.
2. *An employee can operate several machines but only one employee operates each machine. Also, the employee has a single skill level for any particular machine.* That is, once you enter a row of data for a specific machine, you'll never enter another row for the same machine. The key field would be the machine number.
3. *An employee can operate several machines and several employees can operate a machine. The employee has a single skill level for any particular machine.* That is, the employee number can duplicate (with different machine numbers) and the machine numbers can duplicate (with different employees). However, a specific combination of employee number and machine number will never duplicate because no new information would be communicated if it did. That is, if the same machine number and employee number combination were entered into the table, the third attribute (the skill level) would have to be the same as the original row with the same combination of employee number and machine number. The key field in this case would be the compound key consisting of the employee number and machine number.
4. *An employee can operate several machines and several employees can operate a machine. The employee has several skill levels for any particular machine.* Not only can the employee number and machine number repeat (as in 3 above), but also the same combination of employee and machine number can duplicate with different skill levels. This might be necessary if the company wanted to keep a record of various skill levels an employee attained over time. The key field in this case would be the compound key consisting of the employee number, machine number, and skill level.

Data Notation

In the following discussions, we will use a symbolic notation for defining a table. In this notation, the table name will be followed by an equal sign (read as "is defined as") and a set of attributes enclosed in braces and separated by plus signs (read as "and"). The key field will be underlined. For example, the three tables from Figure 9 would be defined as:

Student = { StudentNo + Name + Address }
Transcript = { StudentNo + CourseNo + Grade }
Course = { CourseNo + Desc }

The first table definition is read "The Student table is defined as the StudentNo key field and the Name field and the Address field." Notice the compound key in the Transcript table.

The following discussion gives a set of rules for converting an ER diagram into a Record Structure Diagram. The rules are straightforward and the only potential complication is the identification of the key field.

ER Diagram With a One-to-One Relationship

Assume that you have the one-to-one relationship shown in the ER diagram in Figure 12.

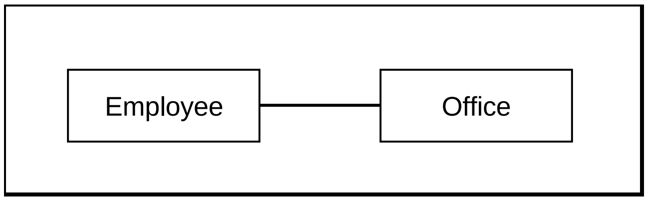


Figure 12
ER diagram shows that an employee has one office and an office has one employee.

Here we assume that each employee has a single office and each office has only one employee. Recall that our definition of an entity requires it to have a unique identifier and at least one additional field. Thus, we know that the Employee entity must have a key field such as employee number and the Office entity must also have a key field such as office number. Each entity also needs additional attributes that provide facts about the entity. With these considerations in mind, we will assume the following definitions exist for the Employee and Office entities:

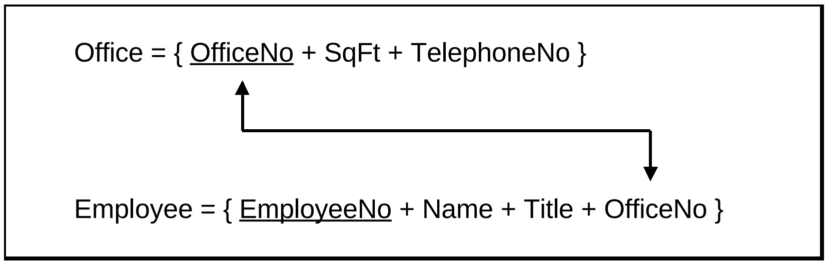
Employee = { EmployeeNo + Name + Title }
Office = { OfficeNo + SqFt + TelephoneNo }

Now that we have defined the entities, we need to set up the relationship. To do this in a one-to-one relationship, take the key field from one (either one) of the entities and make it a nonkey field in the other. A field that is not by itself the key field of a table, but is the key field in another table, is called a foreign key. In this example, we can take the key field from the Office entity and make it a nonkey field (foreign key) in the Employee entity as:

Employee = { EmployeeNo + Name + Title + OfficeNo }

We graphically represent these two tables in the form of a Record Structure Diagram as shown in Figure 13.

Figure 13
RSD showing the relationship between the two database tables.


Office = { OfficeNo + SqFt + TelephoneNo }
Employee = { EmployeeNo + Name + Title + OfficeNo }

The single arrows at each end of the connecting line mean that you should be able to find one matching record in the corresponding tables using these fields. Look at the sample data in Figure 14.

Figure 14
Sample data shows the existence of the OfficeNo attribute in each of the two tables.

Office Table

<u>OfficeNo</u>	SqFt	TelephoneNo
148	300	543-7122
324	200	543-4474
270	250	543-6566

Employee Table

<u>EmployeeNo</u>	Name	Title	OfficeNo
12106	Burrows	Sr. Lecturer	148
19460	Brown	Chairman	324
87621	Welch	Professor	270

If we look at row 1 in the Employee table, we find an employee who is assigned to office number 148. Using this information, we would expect to find only one row to match office number 148 in the Office table. Using the OfficeNo attribute, you should verify that each row in the Employee table is related to only one row in the Office table and vice versa.

Note that an alternative solution would have been to take the key from the Employee table (EmployeeNo) and make it a nonkey attribute (foreign key) in the Office table.

ER Diagram With a One-to-Many Relationship

One-to-many relationships are created in relational tables using foreign keys similar to one-to-one relationships. Consider the ER diagram in Figure 15.

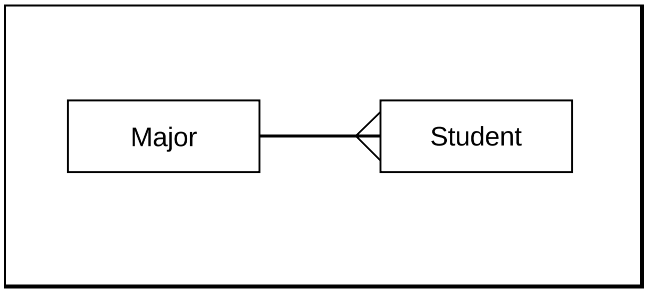


Figure 15
ER diagram showing a Student entity associated with one Major entity and a Major entity associated with many Student entities.

In this diagram, the Student entity is related to only one major (no double majors) while the Major entity is related to many students. Assume that each entity is defined as:

Student = { <u>StudentNo</u> + Name + Address }
Major = { <u>MajorCode</u> + Desc }

The rule for a one-to-many relationship says to take the key field from the "one" entity and make it a nonkey field (foreign key) in the "many" entity. Applying this rule we take the key field from the Major entity (the "one" in the ER diagram) and make it a nonkey field in the Student entity (the "many" entity in the ER diagram).

Student = { <u>StudentNo</u> + Name + Address + MajorCode }

The corresponding Record Structure Diagram is shown in Figure 16.

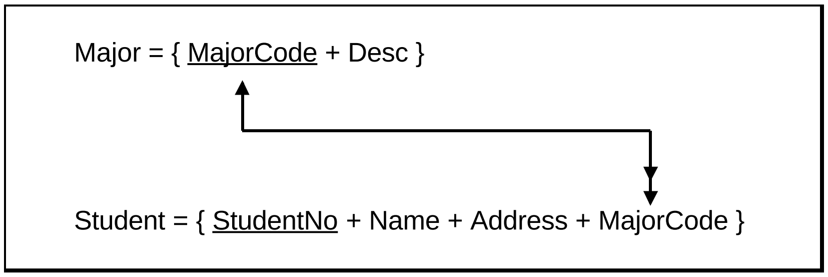

Major = { MajorCode + Desc }
Student = { StudentNo + Name + Address + MajorCode }

Figure 16
RSD showing that a major can be related to many students.

The two arrows pointing to MajorCode in the Student table mean that you expect to find many (more than one) matching records in that table given a specific MajorCode value. Looking at a small set of sample data in Figure 17 demonstrates this.

Figure 17
Data records that show the duplication of MajorCode values in the Student table.

Major Table

<u>MajorCode</u>	Desc
BA	Business
ENG	Engineering
PHY	Physics

Student Table

<u>StudentNo</u>	Name	Address	MajorCode
6634413	Burrows	123 Main	BA
8743232	Brown	710 Terry	ENG
8851000	Welch	4610 17th	ENG
9243255	Jones	1701 46th	ENG
9659844	Smith	833 Talbert	PHY

If you start at MajorCode "ENG" in the Major table (row 2) and search the Student table, you find three matching rows. On the other hand, if you start at StudentNo 8851000 in the Student table, find that student's MajorCode, and then search the Major table using this MajorCode, you find only one matching row.

In a Record Structure Diagram, whenever you point to the entire key field (like MajorCode in the Major table in Figure 16), you will always have a single arrow. Why is this true? Since the values of the key field are unique from row to row, you would never expect to find two or more rows that had the same value for the key field. However, foreign keys are not key fields and can easily be duplicated in two or more rows.

ER Diagram With a Many-to-Many Relationship

Many-to-many relationships are more complex than either one-to-one or one-to-many. Many-to-many relationships do not add foreign keys to an existing table but add new tables to the database instead. There are two different (but related) types of tables—correlation tables and associative objects.

Correlation Tables

Whenever you need to build a relational database to support a many-to-many relationship, you will need to create a new table. Consider the many-to-many relationship in Figure 18.

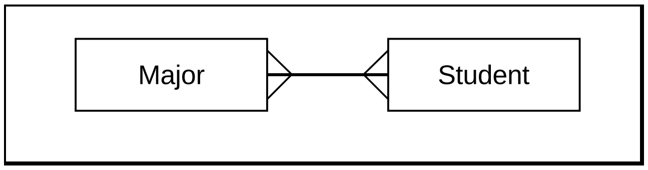


Figure 18
ER diagram showing a student related to many majors.

In this situation, we have relaxed the constraint that a student can have only one major.

Again, assume that the Student and Major entities are defined as:

Student = { <u>StudentNo</u> + Name + Address }
Major = { <u>MajorCode</u> + Desc }

To create a many-to-many relationship, take the key field from each of the two entities and create a new table composed of these fields. Typically these two fields together are used to create a compound key. However, there are situations where only one of them is sufficient to be the key.

In this example we need both fields to produce a unique key, so we take StudentNo and MajorCode and create a new table with the compound key of StudentNo and MajorCode. We now have a new table as:

NewTable = { <u>StudentNo + MajorCode</u> }

This new table is called a “correlation table”. It provides a correlation between the original two tables. The Record Structure Diagram corresponding to this is shown in Figure 19.

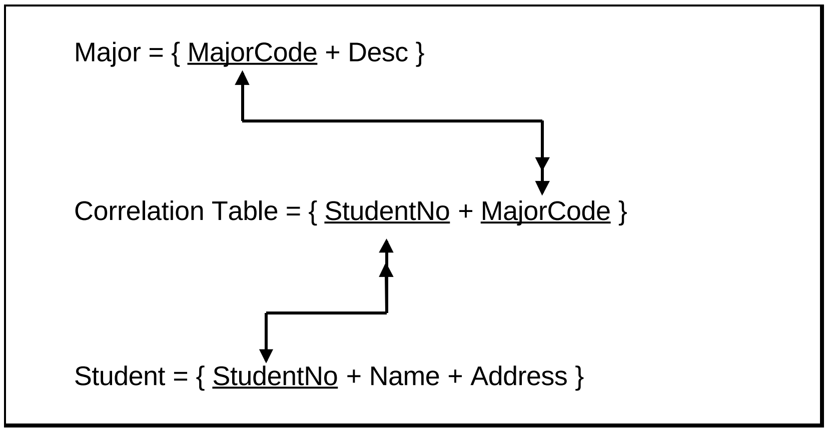

Major = { MajorCode + Desc }
Correlation Table = { StudentNo + MajorCode }
Student = { StudentNo + Name + Address }

Figure 19
RSD showing the correlation table used to support the N-M relationship.

The double (many) arrows point to subsets of the compound key—that is why they can repeat. Again look at some sample data in Figure 20.

Figure 20
Data showing how the correlation table is used to go between the Student and Major tables.

Major Table

MajorCode	Desc
BA	Business
ENG	Engineering
PHY	Physics

Correlation Table

StudentNo	MajorCode
6634413	BA
6634413	PHY
8743232	BA
9659844	BA
9659844	ENG
9659844	PHY

Student Table

StudentNo	Name	Address
6634413	Burrows	123 Main
8743232	Brown	710 Terry
8851000	Welch	4610 17th
9243255	Jones	1701 46th
9659844	Smith	833 Talbert

If you start with StudentNo 6634413 in the Student table, and search the Correlation table for matches of that student number, you will find two matches (rows 1 and 2). The first match has a MajorCode "BA" which has one match in the Major table. The second match has a MajorCode "PHY" which also has one match in the Major table. Through this indirect process, StudentNo 6634413 has been associated with two rows in the Major table (Student has many Majors). Starting with a specific row in the Major table and going through the Correlation table would generate several matches in the Student table (a Major can have many Students).

Notice that all the values of the compound key in the correlation table are unique, e.g., there is only one entry for "6634413 + BA". However, subsets of the compound key, such as StudentNo 6634413, are not unique in the correlation table.

Also note that the correlation table is not shown in Figure 18. Correlation tables are never shown on an ER Diagram because they provide no new information. That is, a correlation table always supports a many-to-many relationship.

Associative Objects

An associative object starts out as a correlation table. That is, it supports a many-to-many relationship between two entities and includes the two key fields of those entities.

However, an associative object also includes at least one additional field beyond the two keys. This additional field may or may not be part of the key of the new table.

Assume there is a many-to-many relationship between the Student and Course entities. We now know that this relationship would be supported with a correlation table that consisted of the keys for the Student and Course tables as follows:

Student/Course Correlation Table = { <u>StudNo + CourseNo</u> }

However, if we want to record a grade for a specific student/course combination, we would add the Grade field to this table changing it from a correlation table to an associative object:

Student/Course Associative Obj = { <u>StudNo + CourseNo</u> + Grade }

Here we see that the additional field is not part of the key.

If we want to allow a student to repeat a course and we want to record a separate grade for each repetition, then we would need to add an additional field such as the quarter the course was taken:

Student/Course Assoc Obj = { <u>StudNo + CourseNo + Qtr</u> + Grade }

Note that in this case, we need to include the quarter as part of the key. Regardless of whether the addition field(s) are part of the key or not, their existence causes a correlation table to become an associative object.

As a reminder, recall that correlation tables are not shown on the ER diagram while associative objects are shown. This is because an associative object is a correlation table plus additional fields. All many-to-many relationships need correlation tables or the “correlation part” of an associative object. This is why they are not shown on the ER diagram—they are always present. However, associative objects are always shown on the ER diagram to tell the reader that additional fields beside the keys of the two entities need to be modeled and represented in the database. See Figures 6, 7, and 8 for examples of ER diagrams with associative objects.

Comparison of Correlation Tables and Associative Objects

Let us return to the ER diagram in Figure 18, except now we want to store the name of the student’s advisor for each major. We assume that a student may have a different advisor for each major. The modified ER diagram is shown in Figure 21.

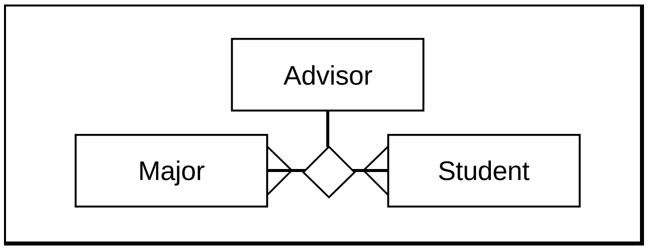


Figure 21
ER diagram showing the student/major relationship with data stored on the advisor.

The resulting Record Structure Diagram is shown in Figure 22. Contrast this with the Record Structure Diagram in Figure 19.

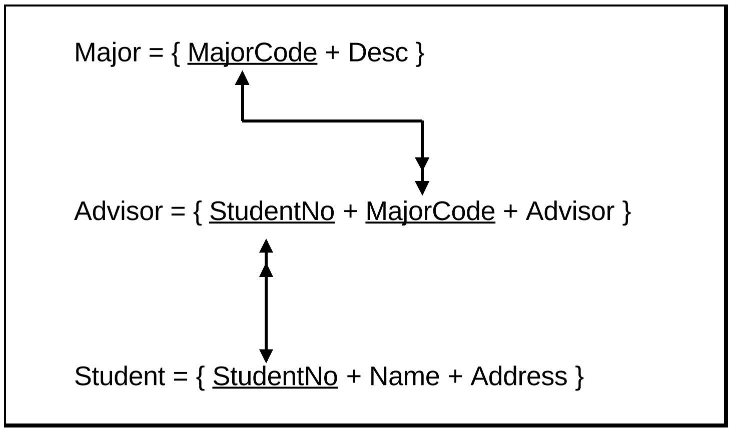

Major = { MajorCode + Desc }
Advisor = { StudentNo + MajorCode + Advisor }
Student = { StudentNo + Name + Address }

Figure 22
RSD showing the Advisor associative object used to support the N-M relationship.

RSD and Relational Databases

The Record Structure Diagram provides the basis for an exact definition of a relational database. Each record definition becomes a table in the database. This includes each entity table plus any correlation tables and associative objects. The foreign keys, correlation tables, and associative objects of the RSD provide the links between tables that support the relationships from the original ER diagram.

Figure 23 shows the relational tables and some sample data for the RSD defined previously in Figure 22. The tables shown in Figure 23 are exactly what you would have to define if you implemented the database using a relational system such as Microsoft Access.

Figure 23
Database tables from the RSD shown in Figure 22.

Major Table

MajorCode	Desc
BA	Business
ENG	Engineering
PHY	Physics

Advisor Table

StudentNo	MajorCode	Advisor
6634413	BA	Franklin
6634413	PHY	Leung
8743232	BA	Franklin
9659844	BA	Adams
9659844	ENG	Quale
9659844	PHY	Olsen

Student Table

StudentNo	Name	Address
6634413	Burrows	123 Main
8743232	Brown	710 Terry
8851000	Welch	4610 17th
9243255	Jones	1701 46th
9659844	Smith	833 Talbert

Example Conversion of An ER Diagram

We will now convert the ER diagram relating to the architectural design firm example we created earlier into a Record Structure Diagram. The ER diagram is repeated in Figure 24 below.

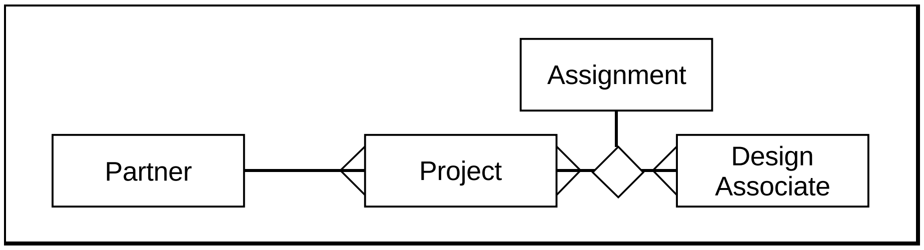


Figure 24
ER diagram from Figure 8.

If we review the narrative given earlier, we can define the fields that exist for the Partner, Project, and Design Associate as follows:

Partner = { <u>PartnerId</u> + Name + PhoneNo }
Project = { <u>ProjectId</u> + Desc + CompleteDate + Budget }
Design Associate = { <u>AssociateId</u> + Name + Specialty }

These definitions store all the attributes but do not store or support the relationships. We need to use the rules for constructing a Record Structure Diagram to set up the relationships.

We begin with the one-to-many relationship between the Partner and Project entities. Recall that the rule says to “take the key field from the ‘one’ entity and make it a nonkey field in the ‘many’ entity.” That is, take the PartnerId field from the Partner entity and make it a nonkey field in the Project entity. The result is:

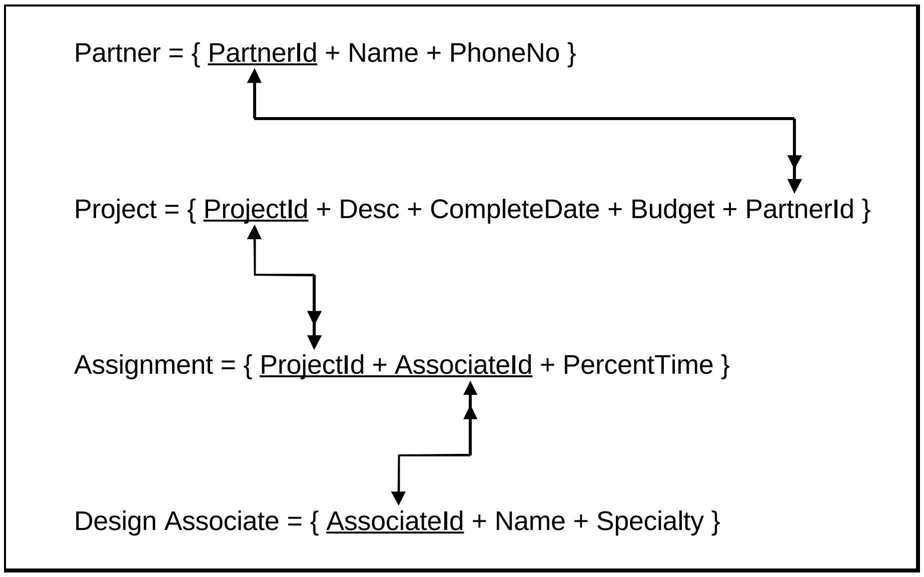
Project = { <u>ProjectId</u> + Desc + CompleteDate + Budget + PartnerId }

The many-to-many relationship between the Project and Design Associate entities requires the creation of a new correlation table that combines the key fields from the Project and Design Associate entities. Since we’ve already determined that we also want to store the percentage of time each design associate is assigned to a project, we store this field along with the keys in the correlation table. Adding this additional field changes the correlation table into an associative object. The definition of the Assignment associative object is:

Assignment = { <u>ProjectId + AssociateId</u> + PercentTime }

Combining the Partner, Project, Design Associate, and Assignment tables in one diagram with appropriate arrows creates the resulting Record Structure Diagram. This diagram is shown in Figure 25.

Figure 25
RSD for the architectural firm's project data.


Partner = { PartnerId + Name + PhoneNo }
Project = { ProjectId + Desc + CompleteDate + Budget + PartnerId }
Assignment = { ProjectId + AssociateId + PercentTime }
Design Associate = { AssociateId + Name + Specialty }

Analyzing RSDs for Problems

Overview

The rules given earlier for converting an ER diagram into a Record Structure Diagram show how to set up the relationships between entities in a relational database. However, it is possible to generate an incorrect relational database even if you use the rules correctly.

The problem relates to the fact that an entity, as defined by the analyst or user, might actually include data on more than one entity. Recall that we said each entity should end up as a separate table in the relational database. If we identify an entity and but it actually contains data on two or more things, then the database will have problems.

Therefore, after the first draft of the Record Structure Diagram has been created, it is important to analyze it for potential problems. If any problems are found, this means that the ER diagram that created the RSD might also be in error. You may to have go back and possibly correct the ER diagram and then derive a new RSD from the revised ER diagram.

The following introduces the concept of functional dependencies between the attributes in a table. The existence of some types of functional dependencies, specifically partial and transitive dependencies, indicates that there is a problem in the relational database design. You need to know how to find and eliminate these types of functional dependencies.

Functional Dependencies

A functional dependency exists between two attributes in a table when the value of one attribute implies or determines the value of another attribute. For example, if I tell you the value of a Social Security Number (SSN), you should be able to identify the name of the person who is assigned that SSN. This is because there is a functional dependency between the value of the SSN attribute and the value of the Name attribute. We can document this dependency using the notation SSN ⇨ Name. We say that “SSN determines Name” or “Name is dependent on SSN.”

Does Name ⇨ SSN? That is, if I tell you a name, can you tell me the person’s SSN? Since many people have the same name, knowing a name does not give you enough information to guarantee that you know the SSN. If the values of a specific attribute in a table are not unique from row to row (like names), then other attributes cannot be dependent on that attribute (it cannot determine them).

Full and Partial Dependencies

If a table has a compound key and at least one nonkey field, then you must analyze it for the possible existence of full and partial dependencies. A full dependency exists if all the fields of the compound key determine a nonkey field in the table. For example, consider the following Transcript table:

Transcript = { <u>StudNo + CourseNo + Qtr</u> + Grade }

This table has a compound key and one nonkey field. In this case, you must know the value all three attributes in the compound key in order to determine the value of the Grade attribute. That is, you must know who you are referring to (StudNo), the course they took (CourseNo), and, assuming that they could take the course more than once, the quarter they took the course (Qtr). This is an example of a full dependency—the entire (full) compound key determines the nonkey Grade field. The existence of a full dependency is a sign of good data design.

Consider the following variation on the transcript example:

Transcript = { <u>StudNo + CourseNo + Qtr</u> + CourseDesc + Grade }

When there is more than one nonkey field, then each one must be evaluated independently for full dependency. Since we've already evaluated the Grade attribute, we'll now look at the CourseDesc attribute.

If we assume that course descriptions do not change over time, then all we need to know to determine course description is the course number, that is, CourseNo ⇨ CourseDesc. On the other hand, if we assume that a course description might change from quarter to quarter, then we would need to know both the course number and quarter, that is, CourseNo + Qtr ⇨ CourseDesc. In either case, we do not need to know the value of the StudNo attribute.

This is a partial dependency, when a nonkey attribute is determined by less than all (part of) the fields of the compound key. Partial dependencies are a sign that there is a problem in the database design.

Let's look at the example in more detail and make the assumption that course descriptions do not change over time, that is, we'll assume that CourseNo ⇨ CourseDesc. We know that the attribute CourseNo must have unique values or else it would not have worked as part of the compound key in the original definition. This means that we have an attribute that uniquely identifies a course (a key field) and an additional attribute describing a fact about the course (its description). What we have here is an entity (a Course entity).

You cannot combine two entities in the same table in a properly formed relational database. Thus, you must fix the problem. The way to do this is to remove the partial dependency from the original table and create a separate table with the key(s) that determined the partial dependency plus the nonkey(s) determined by them. You keep the original table with its compound key and any attributes that are fully dependent on that compound key.

In this example, we would break the original table into two tables as follows:

Transcript = { <u>StudNo + CourseNo + Qtr</u> + Grade }
Course = { <u>CourseNo</u> + CourseDesc }

The original compound key and the fully dependent Grade attribute go into one table and the partial dependency, CourseNo ⇨ CourseDesc, goes into a separate table called Course.

Notice that Course-No remains in the Transcript table and is also the key in the Course table. In the Transcript table, it is part of the compound key and also a foreign key supporting the relationship with the Course table.

You should then go back to the ER diagram that was used to generate the original Transcript table and see what is wrong there. That is, the ER diagram should have shown the Course entity. You would need to go back and fix this problem.

Let's look at another example of a partial dependency. Assume you have an ER diagram that generates the following table:

Client Assignment = { <u>ClientNo + SalespersonNo</u> + SalespersonName }

We assume that a client can be serviced by several salespersons and a salesperson can provide service to many clients.

Since we have the necessary conditions for a partial dependency (a compound key and at least one nonkey field), we need to analyze the table to see if one exists. Do we need to know the client number (part of the compound key) to determine the salesperson name or is the salesperson number sufficient? All we need to know is the salesperson number so we have a partial dependency, SalespersonNo ⇨ SalespersonName.

To solve the problem, we break the original Client table into two tables:

Client Assignment = { <u>ClientNo + SalespersonNo</u> }
Sales Person = { <u>SalespersonNo</u> + SalespersonName }

We kept the original table with its compound key and any attributes that were fully dependent on the compound key. In this case, there were no attributes fully dependent on the compound key. We also created a separate table consisting of the attribute that determined the partial dependency plus the nonkey attribute determined by it.

Like the previous example, we would have to go back to the ER diagram that was used to create the original table and fix it to reflect the fact that two entities (Client and Sales Person) need to be shown. Note also that the original Client table was an associative object but the correction has reduced it to a correlation table. This would also have to be corrected on the ER diagram.

Transitive Dependencies

A transitive dependency occurs when the value of a nonkey attribute is functionally dependent on the value of another nonkey attribute. In order for a transitive dependency to exist, a table must have at least two nonkey attributes.

Consider the following table (FareCode is a code like “C” for “Coach” or “FC” for “First Class”):

Passenger = { <u>PassengerId</u> + Name + FareCode + FarePrice }

Here we have three nonkey attributes so we might have a transitive dependency.

We need to examine the nonkey attributes to see if any functional dependencies exist. Start with passenger name and fare code. If we know a passenger’s name, do we know the fare code? If the answer is yes, we have found a transitive dependency. However, in this case the answer is no. There might be two people on a plane with the same name but one is flying first class and the other is flying coach. The specific value of FareCode is dependent on the unique identifier of Passenger (PassengerId).

Next we’ll check fare code and fare price. If we know the fare code, do we know the fare price? Assuming that there is only one price for each code, then the answer is yes. That is, FareCode ⇨ FarePrice. This is an example of a transitive dependency.

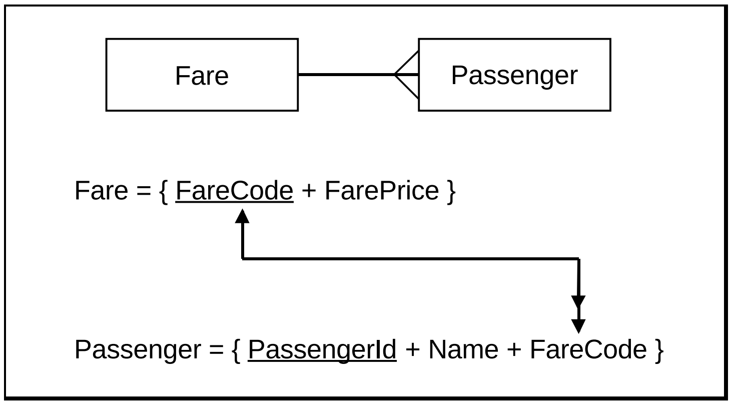
We correct this problem by splitting the original table into two tables. The first is a table with the original key and all values determined directly by it. A separate table is created with the transitive dependency. For our example, we have:

Passenger = { <u>PassengerId</u> + Name + FareCode }
Fare = { <u>FareCode</u> + FarePrice }

As we did with partial dependencies, we have to go back to the ER diagram that was used to create the original table and correct it to show two entities instead of just one.

Also be aware that the correction would have to be reflected in the RSD. The correct ER diagram and the segment of the RSD that reflects the database design without the transitive dependency is shown in Figure 26.

Figure 26
ER diagram and RSD segment for the database without the transitive dependency.


Fare
Passenger
Fare = { FareCode + FarePrice }
Passenger = { PassengerId + Name + FareCode }

You might wonder why the relationship is one-to-many and not one-to-one since both of these cardinality types are supported with foreign keys. The answer is based on common business logic. A one-to-one relationship would mean that a specific fare code could only be associated with one passenger. That is, FC (First Class) could only be associated with one passenger. This does not make business sense.

Transitive dependencies always indicate that there is a problem in the data design. Whenever you have a table with more than one nonkey attribute, you should perform an analysis to determine if any transitive dependencies exist.

Let's look at a second example. Consider the following table definition:

Employee = { EmplNo + Name + ProjectNo + ProjCompleteDate }

Here we have an employee and the project he or she is assigned to. We also have the project's completion date.

Does a transitive dependency exist between any of the nonkey attributes? Does name determine project number? That is, if we know an employee's name, do we know the project they are assigned to? As we have seen before, two employees might have the same name so we do not necessarily know the project number if we know an employee's name.

What do we need to know to determine the project completion date? If we know the project number do we know the project completion date? The answer is probably yes. Project numbers would likely be unique so we know which project we are referring to if we know the project number. Thus, we would know the project completion date. We can conclude that ProjectNo determines ProjCompleteDate.

To remove this transitive dependency, we break the original table into two tables with the original table and all the attributes dependent on the key (EmplNo) and a new table with the transitive dependency. The solution follows:

Employee = { EmplNo + Name + ProjectNo }
Project = { ProjectNo + ProjCompleteDate }

Notice that the ProjectNo attribute stays in the original table. This foreign key provides the link in the database that allows the employee and project information to be joined as needed. Since ProjectNo is a foreign key in the Employee table, the relationship between Project and Employee might be one-to-one or one-to-many. Can you determine which one it is in this case? To answer the question, you need to evaluate the business logic of each alternative. A one-to-one relationship would mean that a specific project could only have one employee. This does not seem to make sense so the relationship is most likely one-to-many.

Repeating Fields

There is an additional potential problem area that might be found in an RSD. Consider the following that defines a student and the courses she has completed:

Student = { StNo + Name + CourseId1 + CourseId2 + CourseId3 }

There are no problems with partial or transitive dependencies, but the definition still has a serious problem—it limits the number of courses to three. This is an example of trying to place fields that repeat for a single entity, e.g., courses, into one row of a table.

If you think about what is actually represented in the table, you'll see that it hides a relationship between Student and Course. Thus there should be at least two tables. One table would define a student as:

Student = { StNo + Name }

A second table would store the relationship between a student and course and might be defined as:

{ <u>StNo + CourseId</u> }

There would likely be a third table that defined non-repeating fields associated with a course such as:

Course = { <u>CourseId</u> + OtherCourseFields }

These three tables represent a many-to-many relationship between Student and Course.

Summary

After you have constructed an RSD from an ER diagram, you should analyze every table for the possibility of partial and/or transitive dependencies.

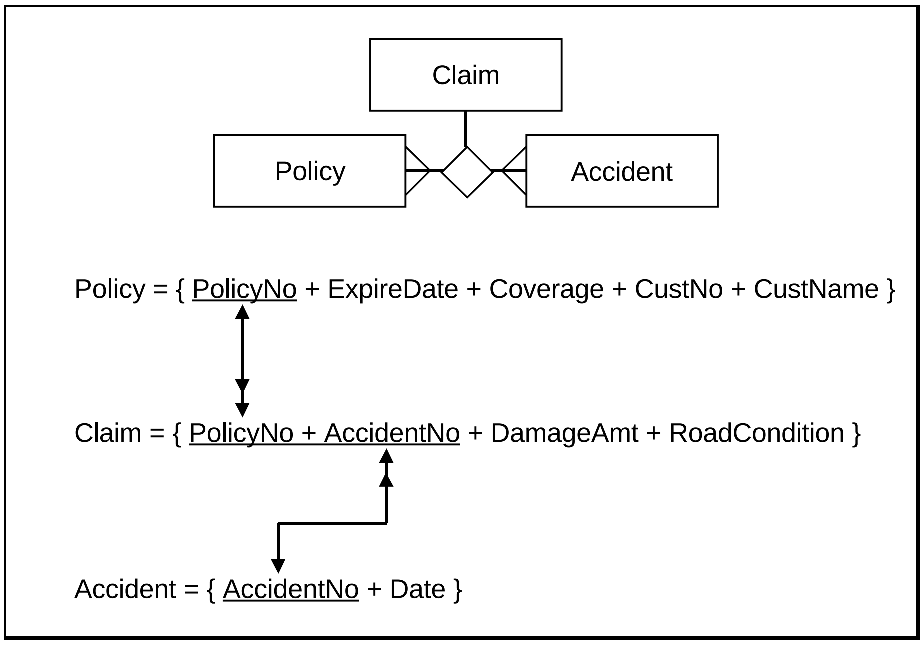
Partial dependencies might exist if a table has a compound key and at least one nonkey attribute. Transitive dependencies might exist if a table has two or more nonkey attributes. Both partial and transitive dependencies indicate that the table represents information on more than a single entity and should be separated into two or more tables. In addition, the ER diagram that was used to create the RSD should be reviewed to see if it reflects the new entities identified as a result of finding partial or transitive dependencies.

Your goal is to have all the tables in the database satisfy the following simple rule: "In each table, every nonkey field should depend on the key field, the whole field, and nothing but the key field". This is another way of saying that no partial or transitive dependencies should exist in the final form of a relational database.

Example RSD With Problems

Figure 27 shows an ER diagram and a RSD that was derived from the ER diagram and other facts. These data are used by an insurance company to keep track of automobile insurance policies and claims made as a result of accidents.

Figure 27
ER diagram and RSD for insurance company accident reporting data-base.


Claim
Policy
Accident
Policy = { PolicyNo + ExpireDate + Coverage + CustNo + CustName }
Claim = { PolicyNo + AccidentNo + DamageAmt + RoadCondition }
Accident = { AccidentNo + Date }

Each of the three tables needs to be analyzed for the existence of partial and/or transitive dependencies. We begin by analyzing the Policy table. Since this table does not have a compound key, then there cannot be partial dependencies. However, it has four nonkey fields so transitive dependencies might exist.

If we know policy expiration date, do we know the coverage amount? Not likely. Does the policy expiration date tell us the customer number or customer name? If it did, then our insurance company might be limited to 365 customers! (Do you see why this is true?)

If we know the coverage amount, do we know the customer number and name? If so, we could only have one customer with a given amount of coverage, which is not reasonable.

If we know the customer number do we know the customer name? The answer is yes and we've just found a transitive dependency. To remove this, we need to break the table into two tables as follows:

Policy = { <u>PolicyNo</u> + ExpireDate + Coverage + CustNo }
Customer = { <u>CustNo</u> + CustName }

We also need to add the Customer entity to the ER diagram and the RSD (which we'll do shortly).

Now let's look at the Claim table. Since this table has a compound key and at least one nonkey attribute, it might contain a partial dependency. To analyze this possibility, we'll look at each nonkey attribute to see if the whole compound key is needed to determine it. Let's start with the damage amount attribute. It does not seem that we need to know the policy number to determine the damage amount since accident numbers are unique (we know this from the key of the Accident table). That is, if we know the unique accident number, this should be sufficient for us to know the damage amount. Thus we have a partial dependency.

The same argument applies to the road condition at the time of the accident. Knowing the accident number is sufficient to determine the road condition. We have another partial dependency.

To remove these two partial dependencies, we keep the original table with its compound key and include all nonkey attributes that are determined by the compound key. In this case, there are none. We also take the partial dependencies and place them in a table of their own. In this case, we take DamageAmt and RoadCondition and place them in a table with AccidentNo as the key field. However, we already have the Accident table with AccidentNo as its key so we just add these attributes to the Accident table. The revised tables are:

Claim = { PolicyNo + AccidentNo }
Accident = { AccidentNo + Date + DamageAmt + RoadCondition }

Notice that the Claim table has been changed from an associative object to a correlation table. Since correlation tables are not shown on ER diagrams, this will require an additional modification to the original ER diagram.

Finally we need to analyze the (revised) Accident table. It does not have a compound key so we don't need to worry about partial dependencies. Also, we already know from our earlier analysis of the Claim table that the key field (AccidentNo) determines the damage amount and road condition attributes. The only possible problem that could remain would be a transitive dependency between date and damage amount or road condition.

It would be hard to argue that by knowing the date, you know the damage amount. If this were true, then every accident that occurred on a given day would result in the same damage amount. If you know the date, do you know the road condition? In a small geographic area this might be true. However, it could rain in the morning and be snowing in the afternoon on the same day. Or it could rain in Seattle and be sunny in Tacoma on the same day. Thus, there is no transitive dependency between the date and the road condition.

Our corrected ER diagram and RSD are shown in Figure 28.

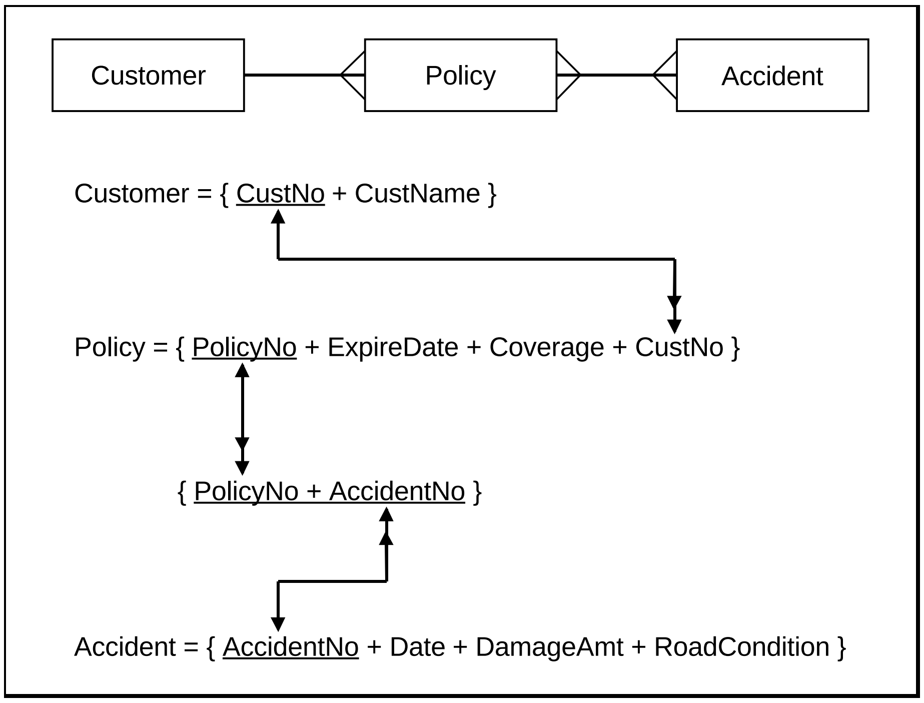

Customer
Policy
Accident
Customer = { CustNo + CustName }
Policy = { PolicyNo + ExpireDate + Coverage + CustNo }
{ PolicyNo + AccidentNo }
Accident = { AccidentNo + Date + DamageAmt + RoadCondition }

Figure 28
Corrected ER diagram and RSD after removing partial and transitive dependencies.

Finding and correcting errors like these is a difficult but important part of designing a database. Just remember that each entity needs to be related to a single thing.

Problem Set

Problem 1

A recent MBA graduate went to work in his family's business. As his first job, he was placed in charge of the shoe department at one of the larger stores owned by the family. He immediately found that the information about sales and inventory, which was manually recorded, was inaccurate and out of date. He decided to build a small database to improve the quality of information available within the department. Using the following description, construct an ER diagram that provides an accurate overview of the department's data.

Shoes are identified by a unique product code. Each shoe style/size/color combination is considered unique. For example, a pair of women's size 6 black Espirit shoes has product code 56756-6. Size 7 of the same style/color has a different product code. For each shoe (style/size/color), the department stocks several identical pairs. Thus, it must keep track of the quantity of each specific shoe in stock.

Shoes are supplied by a variety of suppliers. However, a specific shoe (style/size/color) will be purchased from a single supplier. Suppliers are identified by a unique supplier code. It is not uncommon for a particular supplier to supply many different types of shoes to the store.

There is a need to keep track of customer sales regardless of the type of transaction (cash, credit, etc.). A unique customer number identifies each customer. Additional information, such as name, address, and phone number is also stored for each customer.

Different customers have different buying habits. Some customers purchase only a single pair of shoes. Others purchase many different pairs while still others purchase several pair of the identical shoe. All of these types of purchases must be recorded. Additional information about a purchase includes the date of purchase and the quantity purchased.

A unique employee number identifies each salesperson. Each sale is credited to a single salesperson. This information is necessary because part of the salesperson's compensation is based on a sales commission and the degree to which the salesperson met their sales quota.

Problem 2

Using the following description, construct an ER diagram.

Information needs to be stored about books. Its ISBN (International Standard Book Number) uniquely identifies each book. Other information about a book includes its title and publication date. In addition to book information, there is also information stored about the book's publisher. This includes a unique publisher identifier, publisher name, and publisher address. A single publisher can only publish a book.

Information on the authors of a book is also stored. This information includes the author's social security number, name, and address. Either a single author or several authors can write any single book.

When the book is printed, it is sent to a printer. Information about the printer includes a unique printer identifier, printer name, and address. A contract is written that indicates the number of books the printer will print and the printing deadline the printer needs to meet. At times, a single book might be contracted to several printers if the quantity required to be printed exceeds the printer's production capacity.

Problem 3

Use the following scenario to construct an ER diagram and then convert the ER diagram into a Record Structure Diagram.

A local social services organization acts as an agent to help female clients in a variety of ways. The organization stores information for each client which includes their social security number, name, address, and phone number. In addition, because some of the services provided to clients deals with the client's dependents, information on dependents is also stored. This includes the dependent's social security number, name, and age.

Clients are provided services that are in turn provided by various service providers. For example, a client might be provided job training (a service) that is provided by an agency in state government (the actual service provider). The organization helps match clients with available services. It is not unusual for a single client to be eligible for several services. Information about services includes a unique service identifier and a description. When a client is provided with a service, the date that this happens also needs to be recorded.

Information on service providers includes a unique provider identifier, name, address, and contact phone number. It is the policy of this organization to deal with as many service providers as possible for a single service. For example, even though they use a state agency to provide job training, they might also use a different private agency for job training.

Problem 4

Use the following scenario to construct an ER diagram.

Customers of the video store are assigned a unique customer number when they make their first rental. In additional to the customer number, other information such as name and address is also collected. Each videocassette that the store owns is identified by a unique code. Thus, if the store owns several copies of the same video, each copy has a unique identification code. Other information about a video includes the date of purchase and number of times the video was rented

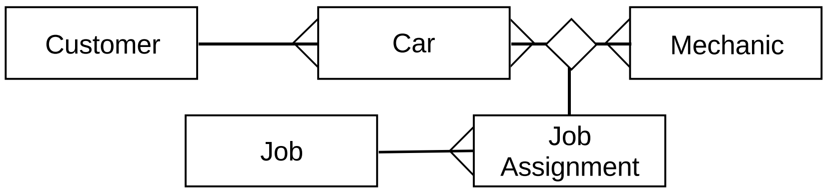
When a customer selects a video to rent, the store needs to record this transaction including the date and time the video was rented. It is not unusual for a customer to rent several videos when they visit the store.

The store assigns a unique identifier to each movie title. For example, the James Bond movie "Goldfinger" is assigned the identifier ADV234. The store may have several cassettes for this movie title. Other information on movies includes a title and year made.

Each movie title is associated with a list of actors and one or more directors. The store has a unique internal code they use to identify each actor. In addition, the store has a different set of internal codes it uses to identify each director. In addition to the actor and director identification codes, other biographic information on actors and directors is stored. Using this information, the store can easily find out all the movies a specific actor has appeared in. Similar information is available for directors.

Problem 5

The following ER diagram documents entities and relationships that exist in an automobile repair shop:



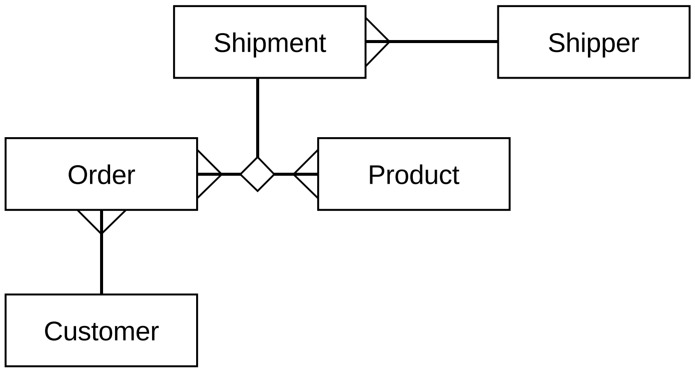
Customers are identified by a unique Cust-No, cars are identified by a unique Vehicle-ID, mechanics are identified by a unique Empl-No, and jobs are identified by a unique Job-Code.

A "Job-Assignment" includes a date field that is necessary in case the same mechanic and car get assigned on different trips to the repair shop. It is also possible that a given mechanic might be assigned to the same car on the same day to do more than one job, e.g., do a lubrication and a tune up.

Using this ER diagram and the additional information above, create a RSD. Explicitly show the key fields described above and any foreign keys and other required fields. Show non-key and other unimportant fields using the general term "other fields".

Problem 6

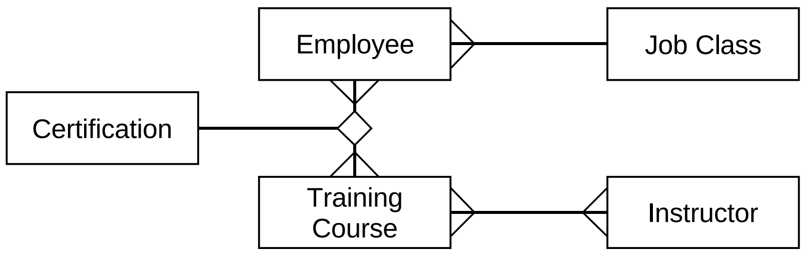
You are given the following ER diagram (note that the ER diagram assumes that products ordered on the same order may be shipped by different shippers):



Convert this ER diagram into a RSD. Explicitly show key fields and foreign keys (make up the names yourself). Show other necessary fields using the general term "other".

Problem 7

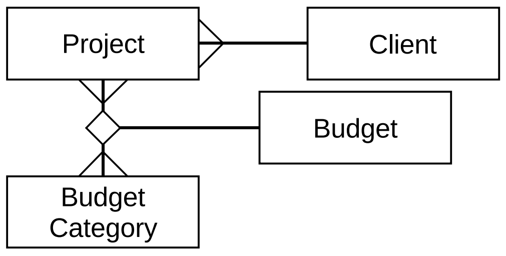
You are given the following ER diagram:



Convert this ER diagram into a RSD. Explicitly show key fields and foreign keys (make up the names yourself). Show other necessary fields using the general term "other".

Problem 8

You are given the following ER diagram that documents projects and their relationship to clients and budget categories.



Assume that data for each project requires 200 bytes of storage which includes a 10–byte key field, data for each Client requires 100 bytes of storage which includes a 10–byte key field, and data for each Budget Category requires 50 bytes of storage which includes a 5-byte key field.

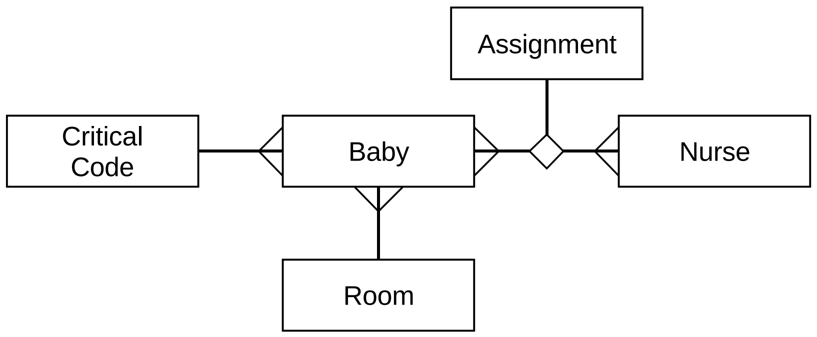
A "budget category" defines an activity that requires funding and expenditures. Examples include travel, outside services, maintenance, etc. For control purposes, each budget category for a specific project is assigned an upper limit. In addition, an actual amount spent is also recorded (a project is within budget if the amount spent is less than the upper limit). These two attributes require 6 bytes of storage each.

Assume that there are 750 projects that the firm needs to store data about. The firm has 500 clients and 35 different budget categories. An average project will be assigned 25 of the 35 budget categories.

Given this information, estimate the database size. Show every table that would be in the database and estimate its total size. Do not add any additional storage for factors such as overhead or potential growth. Do not assume that the sizes given above include foreign keys.

Problem 9

You are given the following ER diagram that documents infants in an intensive care nursing facility:



You are also given the following estimates regarding the size (in bytes) and the number of entities:

Entity	Size of Key	Size of "other"	Total Size*	Number of Entities
Critical Code	5	95	100	50
Baby	10	90	100	80
Room	5	45	50	4
Nurse	10	140	150	20

**Does not include foreign keys*

Assume that when a nurse is assigned to a baby for care, there is a need to store a 6-byte date field and a 4-byte shift field (1st shift, 2nd shift, etc.).

Assuming that each nurse is assigned on average 5 babies to care for, compute the size (in bytes) of the database. Show your calculations for each entity. Do not make any other assumptions (such as growth projections).

Problem 10

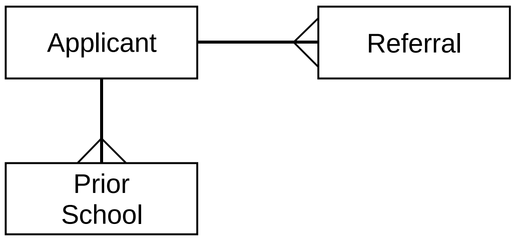
Convert the ER diagram in Problem 8 into a RSD. Create your own names for the key fields. Show all key fields and foreign keys. Use the term “other” for any other fields that are necessary in a table.

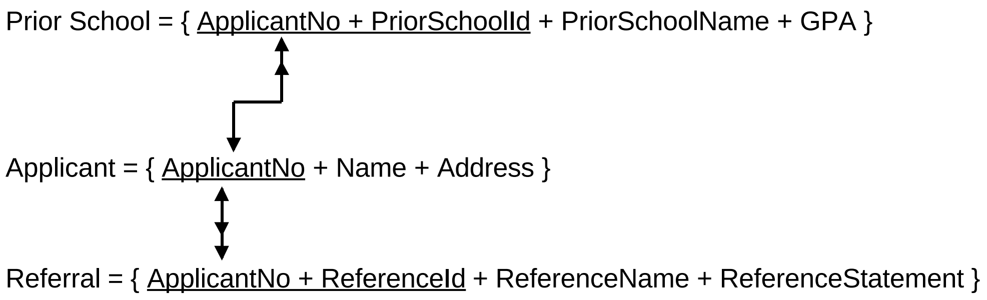
Problem 11

Convert the ER diagram in Problem 9 into a RSD. Create your own names for the key fields. Show all key fields and foreign keys. Use the term “other” for any other fields that are necessary in a table.

Problem 12

The following ER diagram and RSD describe information stored about people applying to a graduate school:





Prior School = { ApplicantNo + PriorSchoolId + PriorSchoolName + GPA }

Applicant = { ApplicantNo + Name + Address }

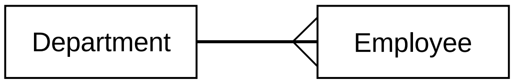
Referral = { ApplicantNo + ReferenceId + ReferenceName + ReferenceStatement }

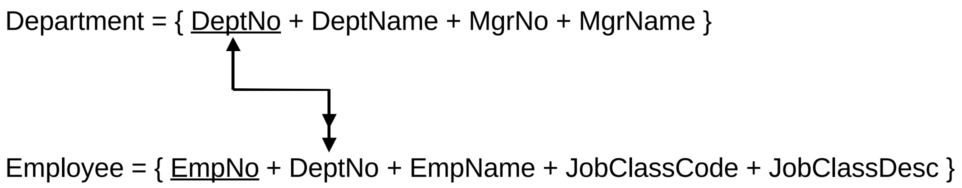
Assume that an applicant can only have one GPA at a prior school and a reference can write only one reference statement per applicant.

Given this information, analyze the ER diagram and RSD for problems. Identify the problems and then correct the ER diagram and RSD.

Problem 13

The following ER diagram and RSD describe information about departments, their managers, and employees assigned to the departments.





Department = { DeptNo + DeptName + MgrNo + MgrName }

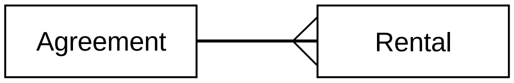
Employee = { EmpNo + DeptNo + EmpName + JobClassCode + JobClassDesc }

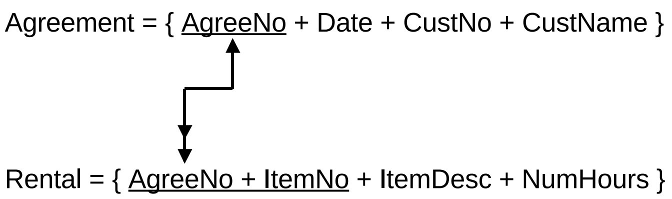
Assume that a department has only one manager, a manager manages only one department, and many employees can share the same job class.

Given this information, analyze the ER diagram and RSD for problems. Identify the problems and then correct the ER diagram and RSD.

Problem 14

The following ER diagram and RSD define rental agreement data that might be collected by a company that rents items to its customers.





Agreement = { AgreeNo + Date + CustNo + CustName }

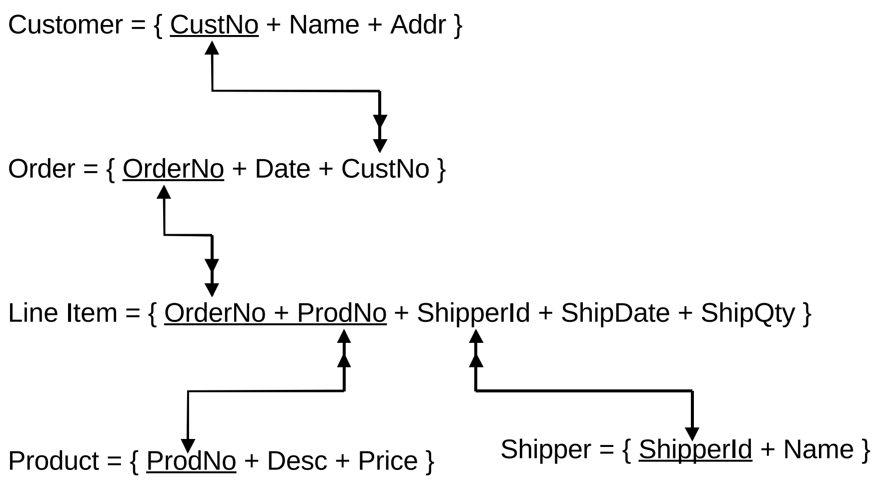
Rental = { AgreeNo + ItemNo + ItemDesc + NumHours }

Assume that an agreement can include several items and a customer can execute many agreements.

Given this information, analyze the ER diagram and RSD for problems. Identify the problems and then correct the ER diagram and RSD.

Problem 15

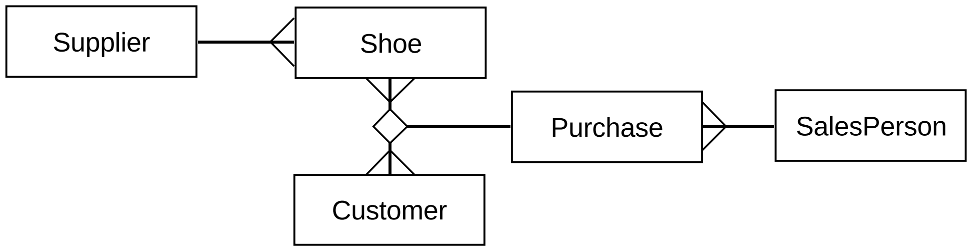
You are given the following RSD:


Customer = { CustNo + Name + Addr }
Order = { OrderNo + Date + CustNo }
Line Item = { OrderNo + ProdNo + ShipperId + ShipDate + ShipQty }
Product = { ProdNo + Desc + Price }
Shipper = { ShipperId + Name }

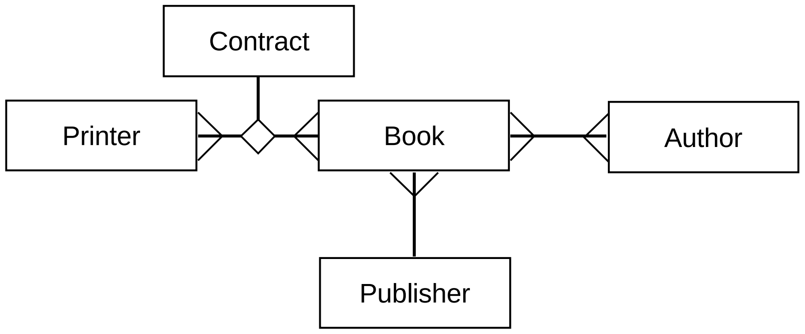
Convert this RSD into an equivalent ER diagram.

Problem-Set Solutions

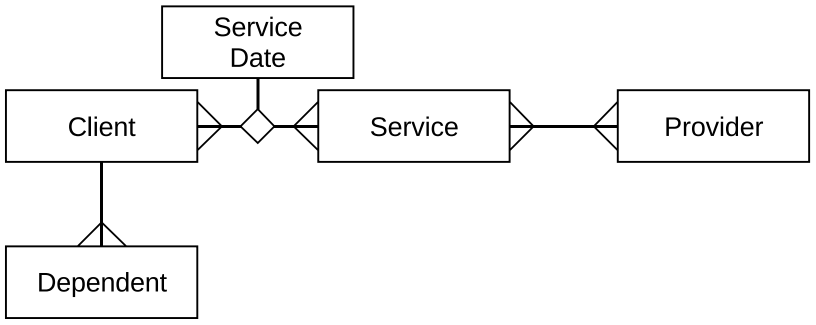
Problem 1



Problem 2

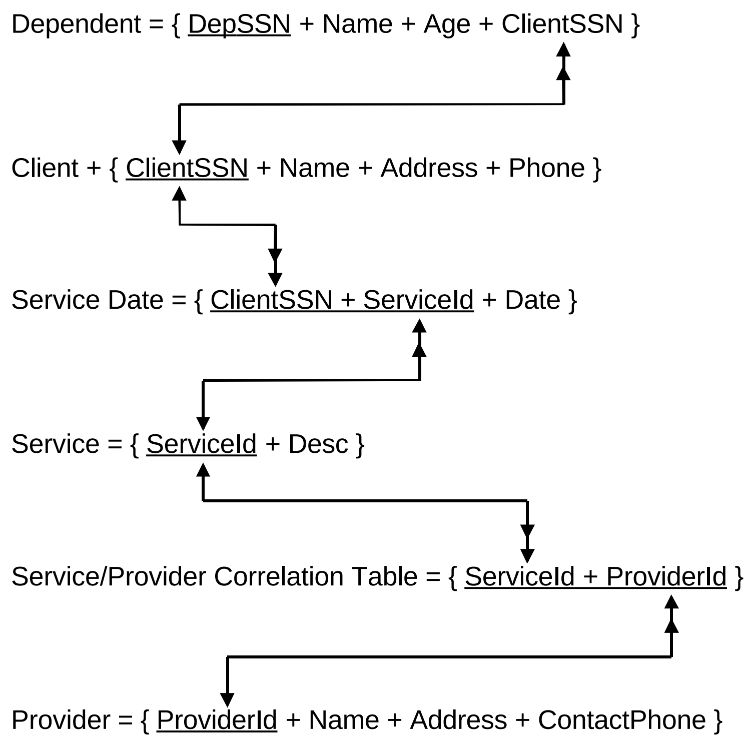


Problem 3



RSD on next page

Problem 3 continued



Dependent = { DepSSN + Name + Age + ClientSSN }

Client + { ClientSSN + Name + Address + Phone }

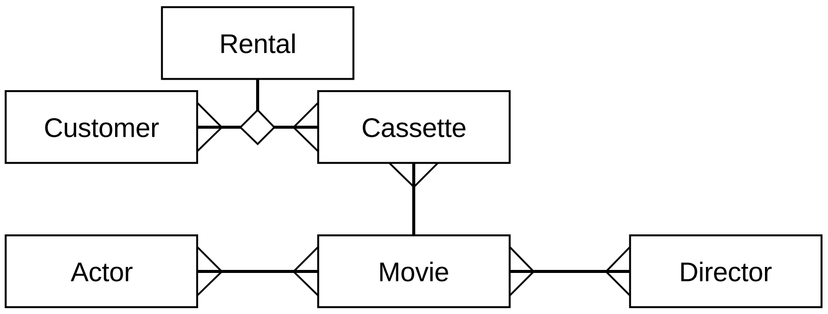
Service Date = { ClientSSN + ServiceId + Date }

Service = { ServiceId + Desc }

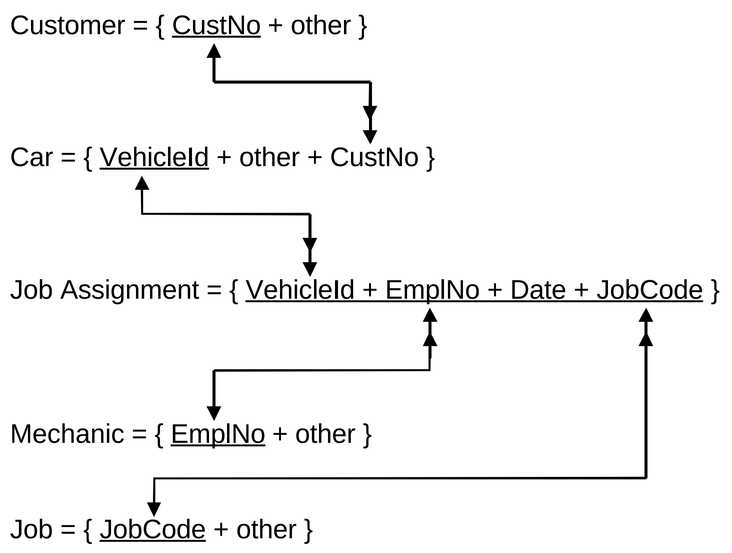
Service/Provider Correlation Table = { ServiceId + ProviderId }

Provider = { ProviderId + Name + Address + ContactPhone }

Problem 4



Problem 5



Customer = { CustNo + other }

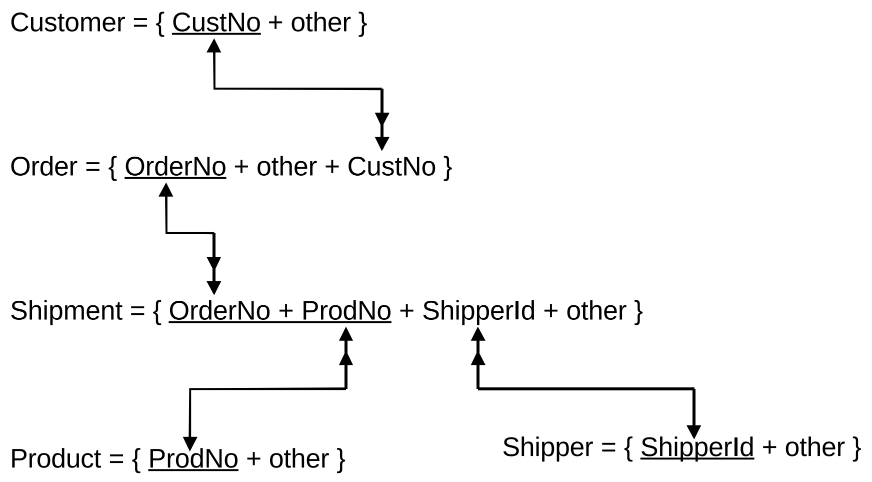
Car = { VehicleId + other + CustNo }

Job Assignment = { VehicleId + EmplNo + Date + JobCode }

Mechanic = { EmplNo + other }

Job = { JobCode + other }

Problem 6



Customer = { CustNo + other }

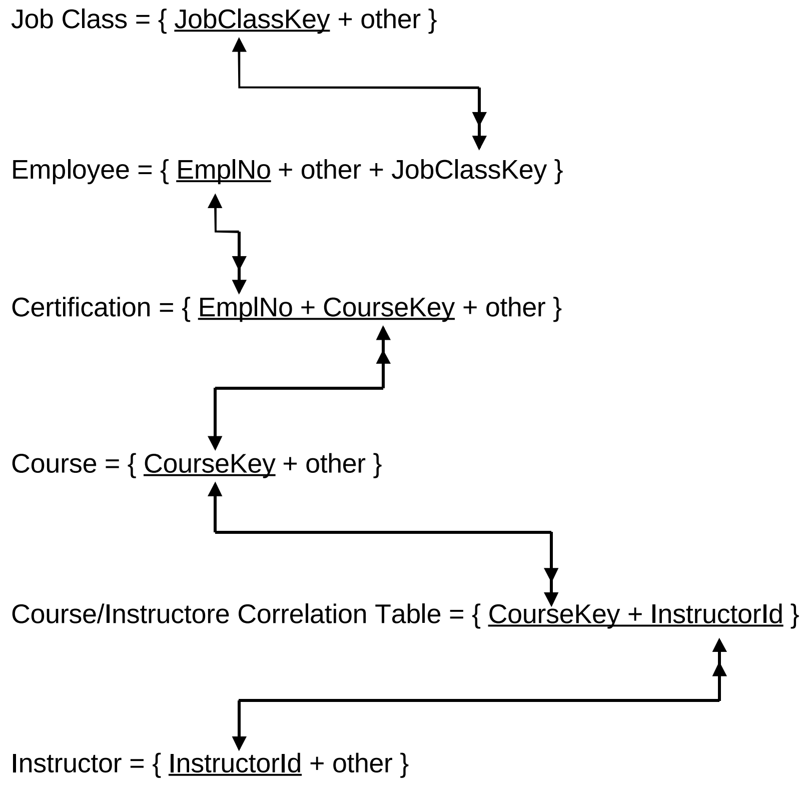
Order = { OrderNo + other + CustNo }

Shipment = { OrderNo + ProdNo + ShipperId + other }

Product = { ProdNo + other }

Shipper = { ShipperId + other }

Problem 7



Job Class = { JobClassKey + other }

Employee = { EmplNo + other + JobClassKey }

Certification = { EmplNo + CourseKey + other }

Course = { CourseKey + other }

Course/Instructore Correlation Table = { CourseKey + InstructorId }

Instructor = { InstructorId + other }

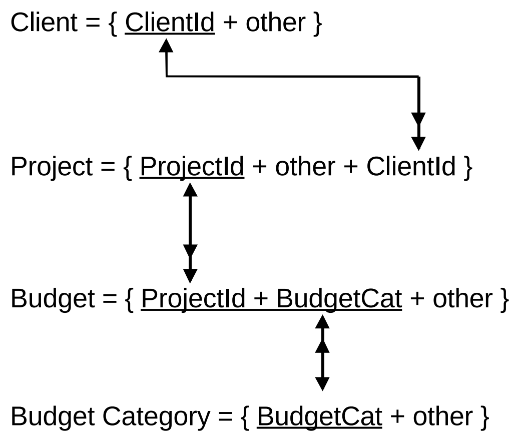
Problem 8

Client:	500 records x 100 bytes/rec = 50,000 bytes
Project:	750 records x 210 bytes/rec = 157,500 bytes
Budget Category:	35 records x 50 bytes/rec = 1,750 bytes
Budget:	750 projects x 25 records/project = 18,750 records 18,750 records x (10+5+6+6) bytes/rec = 506,250 bytes

Problem 9

Critical Code:	100 bytes/code x 50 codes = 5,000 bytes
Room:	50 bytes/room x 4 rooms = 200 bytes
Nurse:	150 bytes/nurse x 20 nurses = 3,000 bytes
Baby:	100 bytes for baby plus 5 bytes (critical code foreign key) + 5 bytes (room foreign key) = 110 bytes/baby x 80 babies = 8,800 bytes
Assignment:	10 bytes (baby key) + 10 bytes (nurse key) 6 bytes (date) + 4 bytes (shift) = 30 bytes/assignment. 20 nurses x 5 assignments per nurse = 100 assignments 30 bytes/assignment x 100 assignments = 3,000 bytes

Problem 10



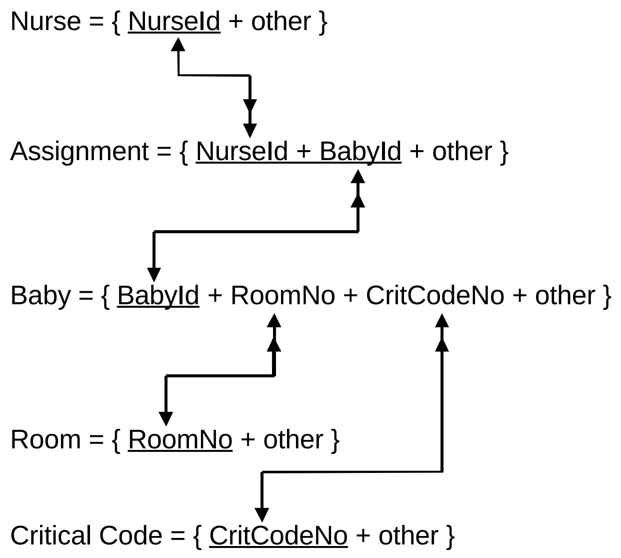
Client = { ClientId + other }

Project = { ProjectId + other + ClientId }

Budget = { ProjectId + BudgetCat + other }

Budget Category = { BudgetCat + other }

Problem 11



Nurse = { NurseId + other }

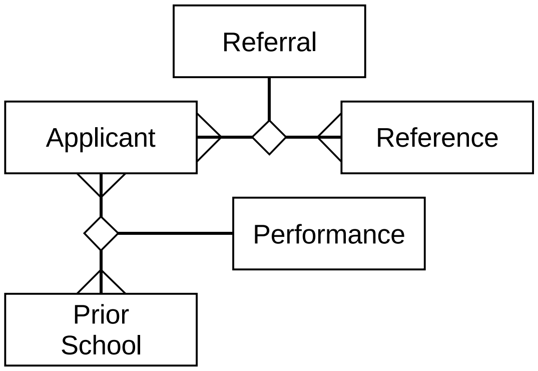
Assignment = { NurseId + BabyId + other }

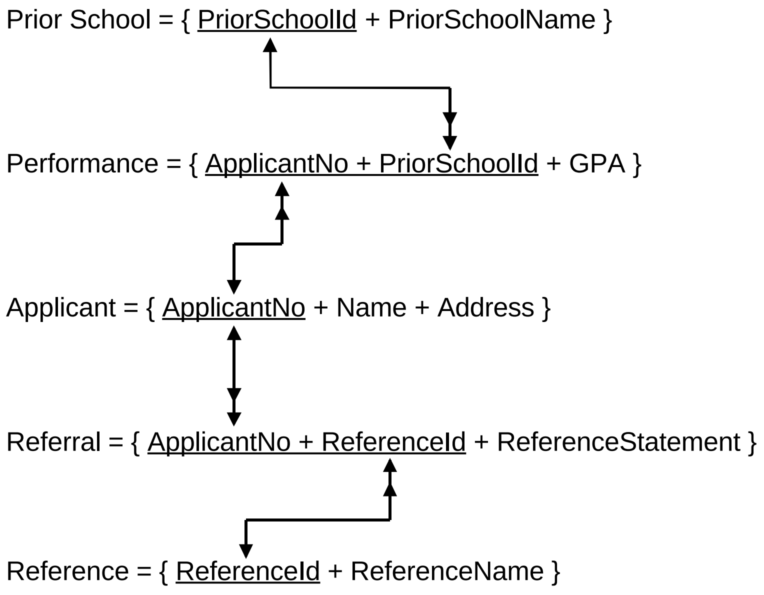
Baby = { BabyId + RoomNo + CritCodeNo + other }

Room = { RoomNo + other }

Critical Code = { CritCodeNo + other }

Problem 12





Prior School = { PriorSchoolId + PriorSchoolName }

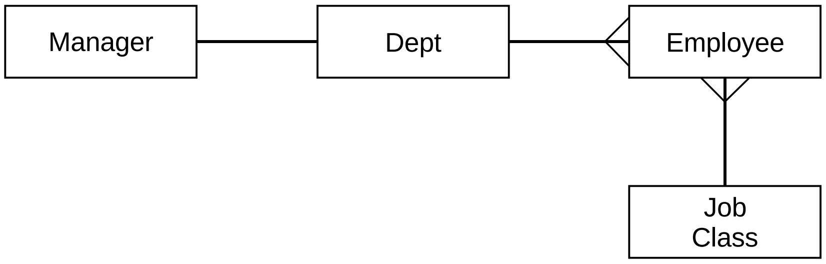
Performance = { ApplicantNo + PriorSchoolId + GPA }

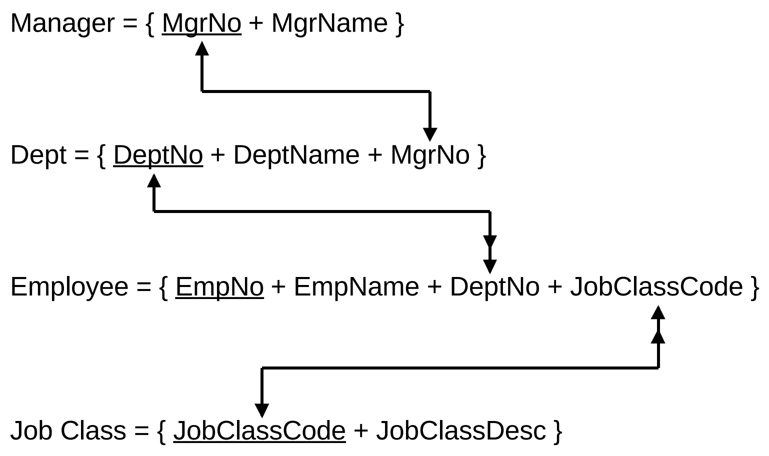
Applicant = { ApplicantNo + Name + Address }

Referral = { ApplicantNo + ReferenceId + ReferenceStatement }

Reference = { ReferenceId + ReferenceName }

Problem 13





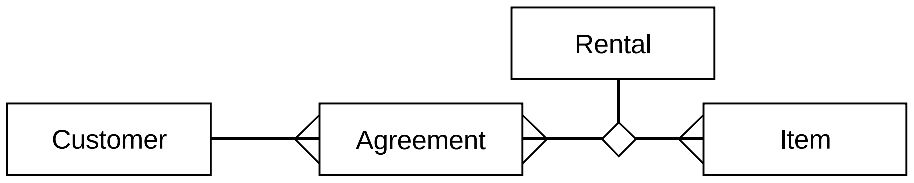
Manager = { MgrNo + MgrName }

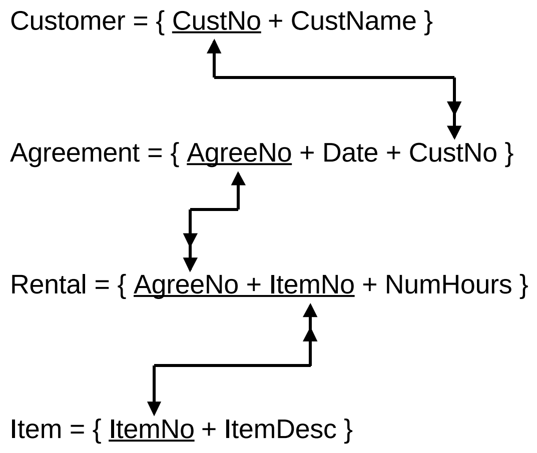
Dept = { DeptNo + DeptName + MgrNo }

Employee = { EmpNo + EmpName + DeptNo + JobClassCode }

Job Class = { JobClassCode + JobClassDesc }

Problem 14





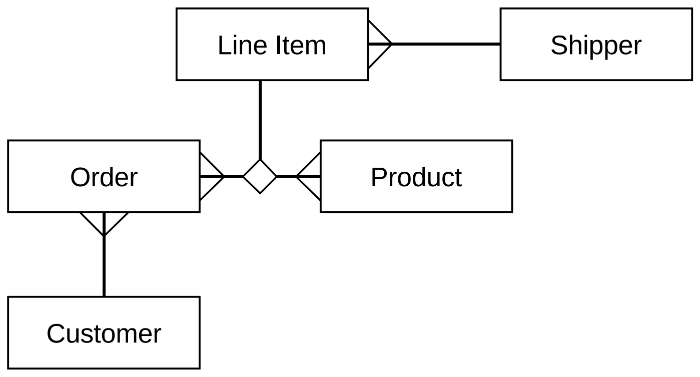
Customer = { CustNo + CustName }

Agreement = { AgreeNo + Date + CustNo }

Rental = { AgreeNo + ItemNo + NumHours }

Item = { ItemNo + ItemDesc }

Problem 15


Line Item
Shipper
Order
Product
Customer