Vilnius Gediminas Technical University

Jelena Mamčenko

Lecture Notes

on

INFORMATION RESOURCES

Part I

Introduction to Dta Modeling and MSAccess

Code **FMITB02004**
Course title **Information Resourses**
Course volume **3,0 cr. (4,50 ECTS cr.)**
Teaching methods (Full-time, daytime studies):
Lectures - **16 h per semestre**
Laboratory works - **32 h per semestre**
Individual work - **72 h per semester**

Course aim
Understandig of models and system of information resourses.

CONTENT

1 Introduction to Data Modeling

An important aspect of most every business is record keeping. In our information society, this has become an important aspect of business, and much of the world's computing power is dedicated to maintaining and using databases.

Databases of all kinds pervade almost every business. All kinds of data, from emails and contact information to financial data and records of sales, are stored in some form of a database. The quest is on for meaningful storage of less-structured information, such as subject knowledge.

This is the first of a two-part article that will provide an introduction to relational databases and the SQL language. This first part describes some of the key elements of the technology with an emphasis on database normalization. The second part will describe a less theoretical approach to database design, as well as provide an introduction to the SQL language.

The concept of relational databases was first described by Edgar Frank Codd (almost exclusively referenced as E. F. Codd in technical literature) in the IBM research report RJ599, dated August 19th, 1969.[1] However, the article that is usually considered the cornerstone of this technology is "A Relational Model of Data for Large Shared Data Banks," published in Communications of the ACM(Vol. 13, No. 6, June 1970, pp. 377-87). Only the first part of the article is available online.

Additional articles by E. F. Codd throughout the 1970s and 80s are still considered gospel for relational database implementations. His famous "Twelve Rules for Relational Databases"[2] were published in two Computerworld articles "Is Your DBMS Really Relational?" and "Does Your DBMS Run By the Rules?" on October 14, 1985, and October 21, 1985, respectively. He has since expanded on the 12 rules, and they now number 333, as published in his book "The Relational Model for Database Management, Version 2" (Addison -Wesley, 1990).

Codd's twelve rules call for a language that can be used to define, manipulate, and query the data in the database, expressed as a string of characters. The language, SQL, was originally developed in the research division of IBM (initially at Yorktown Heights, N.Y., and later at San Jose, Calif., and Raymond Boyce and Donald Chamberlin were the original designers.)[3] and has been adopted by all major relational database vendors. The name SQL originally stood for Structured Query Language. The first commercially available implementation of the language was named SEQUEL (for Sequential English QUEry Language) and was part of IBM's SEQUEL/DS product. The name was later changed for legal reasons. Thus, many long-time database developers use the pronunciation "see-quell."

SQL has been adopted as an ANSI/ISO standard. Although revised in 1999 (usually referenced as SQL99 or SQL3), most vendors are still not fully compliant with the 1992 version of the standard. The 1992 standard is smaller and simpler to reference for a user, and since only some of the 1999-specific requirements are typically implemented at this time, it may be a better starting point for learning the language.

1.1 Data Modeling Overview

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects, and the rules which govern operations on the objects. As the name implies, the data model focuses on what data is required and how it should be organized rather than what operations will be performed on the data. To use a common analogy, the data model is equivalent to an architect's building plans.

A data model is independent of hardware or software constraints. Rather than try to represent the data as a database would see it, the data model focuses on representing the data as the user sees it in the "real world". It serves as a bridge between the concepts that make up real-world events and processes and the physical representation of those concepts in a database.

1.1.1 Methodology

There are two major methodologies used to create a data model: the Entity-Relationship (ER) approach and the Object Model. This document uses the Entity-Relationship approach.

1.1.2 Data Modeling In the Context of Database Design

Database design is defined as: "design the logical and physical structure of one or more databases to accommodate the information needs of the users in an organization for a defined set of applications". The design process roughly follows five steps:

1. planning and analysis
2. conceptual design
3. logical design
4. physical design
5. implementation

The data model is one part of the conceptual design process. The other, typically is the functional model. The data model focuses on what data should be stored in the database while the functional model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries which will access and perform operations on those tables.

1.1.3 Components of A Data Model

The data model gets its inputs from the planning and analysis stage. Here the modeler, along with analysts, collects information about the requirements of the database by reviewing existing documentation and interviewing end-users.

The data model has two outputs. The first is an entity-relationship diagram which represents the data structures in a pictorial form. Because the diagram is easily learned, it is valuable tool to communicate the model to the end-user. The second component is a data document. This document that describes in details the data objects, relationships, and rules required by the database. The dictionary provides the detail required by the database developer to construct the physical database.

1.1.4 Why is Data Modeling Important?

Data modeling is probably the most labor intensive and time consuming part of the development process. Why bother especially if you are pressed for time? A common response by practitioners who write on the subject is that you should no more build a database without a model than you should build a house without blueprints.

The goal of the data model is to make sure that the all data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language, it can be reviewed and verified as correct by the end-users.

The data model is also detailed enough to be used by the database developers to use as a "blueprint" for building the physical database. The information contained in the data model will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers. A poorly designed database will require more time in the long-term. Without careful planning you may create a database that omits data required to create critical reports, produces results that are incorrect or inconsistent, and is unable to accommodate changes in the user's requirements.

1.1.5 Summary

A data model is a plan for building a database. To be effective, it must be simple enough to communicate to the end user the data structure required by the database yet detailed enough for the database design to use to create the physical structure.

The Entity-Relation Model (ER) is the most common method used to build data models for relational databases. The next section provides a brief introduction to the concepts used by the ER Model.

1.2 The Entity-Relationship Model

The Entity-Relationship (ER) model was originally proposed by Peter in 1976 [Chen76] as a way to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram which is used to visually represents data objects. Since Chen wrote his paper the model has been extended and today it is commonly used for database design For the database designer, the utility of the ER model is:

- it maps well to the relational model. The constructs used in the ER model can easily be transformed into relational tables.
- it is simple and easy to understand with a minimum of training. Therefore, the model can be used by the database designer to communicate the design to the end user.
- In addition, the model can be used as a design plan by the database developer to implement a data model in a specific database management software.

1.2.1 Basic Constructs of E-R Modeling

The ER model views the real world as a construct of entities and association between entities.

1.2.2 Entities

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, INVOICES. An entity is analogous to a table in the relational model.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one that does not rely on another for identification. A dependent entity is one that relies on another for identification.

An entity occurrence (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

1.2.3 Special Entity Types

Associative entities (also known as intersection entities) are entities used to associate two or more entities in order to reconcile a many-to-many relationship.

Subtypes entities are used in generalization hierarchies to represent a subset of instances of their parent entity, called the supertype, but which have attributes or relationships that apply only to the subset.

Associative entities and generalization hierarchies are discussed in more detail below.

2 Data Modeling As Part of Database Design

The data model is one part of the conceptual design process. The other is the function model. The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. To put this in the context of the relational database, the data model is used to design the relational tables. The functional model is used to design the queries that will access and perform operations on those tables.

Data modeling is preceeded by planning and analysis. The effort devoted to this stage is proportional to the scope of the database. The planning and analysis of a database intended to serve the needs of an enterprise will require more effort than one intended to serve a small workgroup.

The information needed to build a data model is gathered during the requirements analysis. Although not formally considered part of the data modeling stage by some methodologies, in reality the requirements analysis and the ER diagramming part of the data model are done at the same time.

2.1 Requirements Analysis

The goals of the requirements analysis are:

- to determine the data requirements of the database in terms of primitive objects
- to classify and describe the information about these objects
- to identify and classify the relationships among the objects
- to determine the types of transactions that will be executed on the database and the interactions between the data and the transactions
- to identify rules governing the integrity of the data

The modeler, or modelers, works with the end users of an organization to determine the data requirements of the database. Information needed for the requirements analysis can be gathered in several ways:

- review of existing documents - such documents include existing forms and reports, written guidelines, job descriptions, personal narratives, and memoranda. Paper documentation is a good way to become familiar with the organization or activity you need to model.
- interviews with end users - these can be a combination of individual or group meetings. Try to keep group sessions to under five or six people. If possible, try to have everyone with the same function in one meeting. Use a blackboard, flip charts, or overhead transparencies to record information gathered from the interviews.
- review of existing automated systems - if the organization already has an automated system, review the system design specifications and documentation

The requirements analysis is usually done at the same time as the data modeling. As information is collected, data objects are identified and classified as entities, attributes, or relationship; assigned names; and, defined using terms familiar to the end-users. The objects are then modeled and analyzed using an ER diagram. The diagram can be reviewed by the modeler and the end-users to determine its completeness and accuracy. If the model is not correct, it is modified, which sometimes requires additional information to be collected. The review and edit cycle continues until the model is certified as correct.

Three points to keep in mind during the requirements analysis are:

1. Talk to the end users about their data in "real-world" terms. Users do not think in terms of entities, attributes, and relationships but about the actual people, things, and activities they deal with daily.
2. Take the time to learn the basics about the organization and its activities that you want to model. Having an understanding about the processes will make it easier to build the model.
3. End-users typically think about and view data in different ways according to their function within an organization. Therefore, it is important to interview the largest number of people that time permits.

2.2 Steps In Building the Data Model

While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies, such as IDEFIX, specify a bottom-up development process were the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes, and then the model is finished by adding non-key attributes. Other experts argue that in practice, using a phased approach is impractical because it requires too many meetings with the end-users. The sequence used for this document are:

1. Identification of data objects and relationships
2. Drafting the initial ER diagram with entities and relationships
3. Refining the ER diagram
4. Add key attributes to the diagram
5. Adding non-key attributes
6. Diagramming Generalization Hierarchies
7. Validating the model through normalization
8. Adding business and integrity rules to the Model

In practice, model building is not a strict linear process. As noted above, the requirements analysis and the draft of the initial ER diagram often occur simultaneously. Refining and validating the diagram may uncover problems or missing information which require more information gathering and analysis

2.3 Summary

Data modeling must be preceded by planning and analysis. Planning defines the goals of the database, explains why the goals are important, and sets out the path by which the goals will be reached. Analysis involves determining the requirements of the database. This is typically done by examining existing documentation and interviewing users.

An effective data model completely and accurately represents the data requirements of the end users. It is simple enough to be understood by the end user yet detailed enough to be used by a database designer to build the database. The model eliminates redundant data, it is independent of any hardware and software constraints, and can be adapted to changing requirements with a minimum of effort.

Data modeling is a bottom up process. A basic model, representing entities and relationships, is developed first. Then detail is added to the model by including information about attributes and business rules.

The next section discusses Identifying Data Objects and Relationships.

3 Identifying Data Objects and Relationships

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirements analysis for the purpose of:

- classifying data objects as either entities or attributes
- identifying and defining relationships between entities
- naming and defining identified entities, attributes, and relationships
- documenting this information in the data document

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

Although it is easy to define the basic constructs of the ER model, it is not an easy task to distinguish their roles in building the data model. What makes an object an entity or attribute? For example, given the statement "employees work on projects". Should employees be classified as an entity or attribute? Very often, the correct answer depends upon the requirements of the database. In some cases, employee would be an entity, in some it would be an attribute.

While the definitions of the constructs in the ER Model are simple, the model does not address the fundamental issue of how to identify them. Some commonly given guidelines are:

- entities contain descriptive information
- attributes either identify or describe entities
- relationships are associations between entities

These guidelines are discussed in more detail below.

- Entities
- Attributes
  - Validating Attributes
  - Derived Attributes and Code Values
- Relationships
- Naming Data Objects
- Object Definition
- Recording Information in Design Document

3.1 Entities

There are various definitions of an entity:

"Any distinguishable person, place, thing, event, or concept, about which information is kept"

"A thing which can be distinctly identified"

"Any distinguishable object that is to be represented in a database"

"...anything about which we store information (e.g. supplier, machine tool, employee, utility pole, airline seat, etc.). For each entity type, certain attributes are stored".

These definitions contain common themes about entities:

- an entity is a "thing", "concept" or, object". However, entities can sometimes represent the relationships between two or more objects. This type of entity is known as an associative entity.
- entities are objects which contain descriptive information. If an data object you have identified is described by other objects, then it is an entity. If there is no descriptive information associated with the item, it is not an entity. Whether or not a data object is an entity may depend upon the organization or activity being modeled.
- an entity represents many things which share properties. They are not single things. For example, King Lear and Hamlet are both plays which share common attributes such as name, author, and cast of characters. The entity describing these things would be PLAY, with King Lear and Hamlet being instances of the entity.
- entities which share common properties are candidates for being converted to generalization hierarchies (See below)
- entities should not be used to distinguish between time periods. For example, the entities 1st Quarter Profits, 2nd Quarter Profits, etc. should be collapsed into a single entity called Profits. An attribute specifying the time period would be used to categorize by time
- not every thing the users want to collect information about will be an entity. A complex concept may require more than one entity to represent it. Others "things" users think important may not be entities.

3.2 Attributes

Attributes are data objects that either identify or describe entities. Attributes that identify entities are called key attributes. Attributes that describe an entity are called non-key attributes. Key attributes will be discussed in detail in a latter section.

The process for identifying attributes is similar except now you want to look for and extract those names that appear to be descriptive noun phrases.

3.3 Validating Attributes

Attribute values should be atomic, that is, present a single fact. Having disaggregated data allows simpler programming, greater reusability of data, and easier implementation of changes. Normalization also depends upon the "single fact" rule being followed. Common types of violations include:

- simple aggregation - a common example is Person Name which concatenates first name, middle initial, and last name. Another is Address which concatenates, street address, city, and zip code. When dealing with such attributes, you need to find out if there are good reasons for decomposing them. For example, do the end-users want to use the person's first name in a form letter? Do they want to sort by zip code?
- complex codes - these are attributes whose values are codes composed of concatenated pieces of information. An example is the code attached to automobiles and trucks. The code represents over 10 different pieces of information about the vehicle. Unless part of an industry standard, these codes have no meaning to the end user. They are very difficult to process and update.
- text blocks - these are free-form text fields. While they have a legitimate use, an over reliance on them may indicate that some data requirements are not met by the model.
- mixed domains - this is where a value of an attribute can have different meaning under different conditions

3.4 Derived Attributes and Code Values

Two areas where data modeling experts disagree is whether derived attributes and attributes whose values are codes should be permitted in the data model.

Derived attributes are those created by a formula or by a summary operation on other attributes. Arguments against including derived data are based on the premise that derived data should not be stored in a database and therefore should not be included in the data model. The arguments in favor are:

- derived data is often important to both managers and users and therefore should be included in the data model
- it is just as important, perhaps more so, to document derived attributes just as you would other attributes
- including derived attributes in the data model does not imply how they will be implemented

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female". Those who are against this practice cite that codes have no intuitive meaning to the end-users and add complexity to processing data. Those in favor argue that many organizations have a long history of using coded attributes, that codes save space, and improve flexibility in that values can be easily added or modified by means of look-up tables.

3.5 Relationships

Relationships are associations between entities. Typically, a relationship is indicated by a verb connecting two or more entities. For example:

employees are assigned to projects

As relationships are identified they should be classified in terms of cardinality, optionality, direction, and dependence. As a result of defining the relationships, some relationships may be dropped and new relationships added. Cardinality quantifies the relationships between entities by measuring how many instances of one entity are related to a single instance of another. To determine the cardinality, assume the existence of an instance of one of the entities. Then determine how many specific instances of the second entity could be related to the first. Repeat this analysis reversing the entities. For example:

employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example:

an employee may be assigned to a project

Mandatory relationships, on the other hand, are indicated by words such as must have.

For example:a student must register for at least three course each semester

In the case of the specific relationship form (1:1 and 1:M), there is always a parent entity and a child entity. In one-to-many relationships, the parent is always the entity with the cardinality of one. In one-to-one relationships, the choice of the parent entity must be made in the context of the business being

modeled. If a decision cannot be made, the choice is arbitrary.

3.6 Naming Data Objects

The names should have the following properties:

- unique
- have meaning to the end-user
- contain the minimum number of words needed to uniquely and accurately describe the object

For entities and attributes, names are singular nouns while relationship names are typically verbs. Some authors advise against using abbreviations or acronyms because they might lead to confusion about what they mean. Other believe using abbreviations or acronyms are acceptable provided that they are universally used and understood within the organization.

You should also take care to identify and resolve synonyms for entities and attributes. This can happen in large projects where different departments use different terms for the same thing.

3.7 Object Definition

Complete and accurate definitions are important to make sure that all parties involved in the modeling of the data know exactly what concepts the objects are representing.

Definitions should use terms familiar to the user and should precisely explain what the object represents and the role it plays in the enterprise. Some authors recommend having the end-users provide the definitions. If acronyms, or terms not universally understood are used in the definition, then these should be defined.

While defining objects, the modeler should be careful to resolve any instances where a single entity is actually representing two different concepts (homonyms) or where two different entities are actually representing the same "thing" (synonyms). This situation typically arises because individuals or organizations may think about an event or process in terms of their own function.

An example of a homonym would be a case where the Marketing Department defines the entity MARKET in terms of geographical regions while the Sales Departments thinks of this entity in terms of demographics. Unless resolved, the result would be an entity with two different meanings and properties.

Conversely, an example of a synonym would be the Service Department may have identified an entity called CUSTOMER while the Help Desk has identified the entity CONTACT. In reality, they may mean the same thing, a person who contacts or calls the organization for assistance with a problem. The resolution of synonyms is important in order to avoid redundancy and to avoid possible consistency or integrity problems.

Some examples of definitions are:

Employee	A person who works for and is paid by the organization.
Est_Time	The number of hours a project manager estimates that project will require to complete. Estimated time is critical for scheduling a project and for tracking project time variances.
Assigned	Employees in the organization may be assigned to work on no more than three projects at a time. Every project will have at least two employees assigned to it at any given time.

3.8 Recording Information in Design Document

The design document records detailed information about each object used in the model. As you name, define, and describe objects, this information should be placed in this document. If you are not using an automated design tool, the document can be done on paper or with a word processor. There is no standard for the organization of this document but the document should include information about names, definitions, and, for attributes, domains.

Two documents used in the IDEF1X method of modeling are useful for keeping track of objects. These are the ENTITY-ENTITY matrix and the ENTITY-ATTRIBUTE matrix.

The ENTITY-ENTITY matrix is a two-dimensional array for indicating relationships between entities. The names of all identified entities are listed along both axes. As relationships are first identified, an "X" is placed in the intersecting points where any of the two axes meet to indicate a possible relationship between the entities involved. As the relationship is further classified, the "X" is replaced with the notation indicating cardinality.

The ENTITY-ATTRIBUTE matrix is used to indicate the assignment of attributes to entities. It is similar in form to the ENTITY-ENTITY matrix except attribute names are listed on the rows.

Figure 1 shows examples of an ENTITY-ENTITY matrix and an ENTITY-ATTRIBUTE matrix.

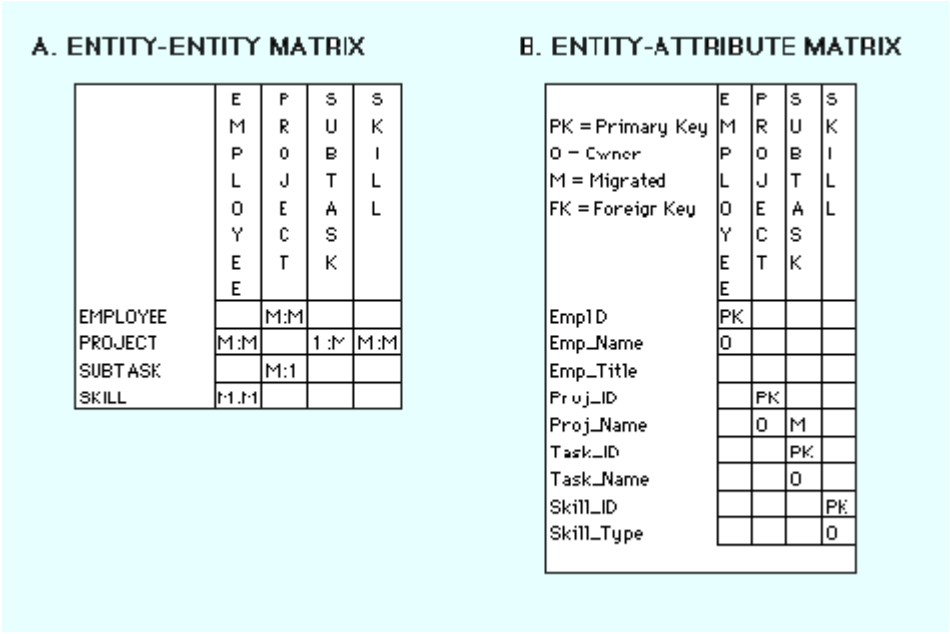


Figure 1. ENTITY-ENTITY matrix and an ENTITY-ATTRIBUTE matrix

3.9 Summary

The first step in creating the data model is to analyze the information gathered during the requirements analysis with the goal of identifying and classifying data objects and relationships. The next step is Developing the Basic Schema.

4 Developing the Basic Schema

Once entities and relationships have been identified and defined, the first draft of the entity relationship diagram can be created. This section introduces the ER diagram by demonstrating how to diagram binary relationships. Recursive relationships are also shown.

4.1 Binary Relationships

Figure 2 shows examples of how to diagram one-to-one, one-to-many, and many-to-many relationships.

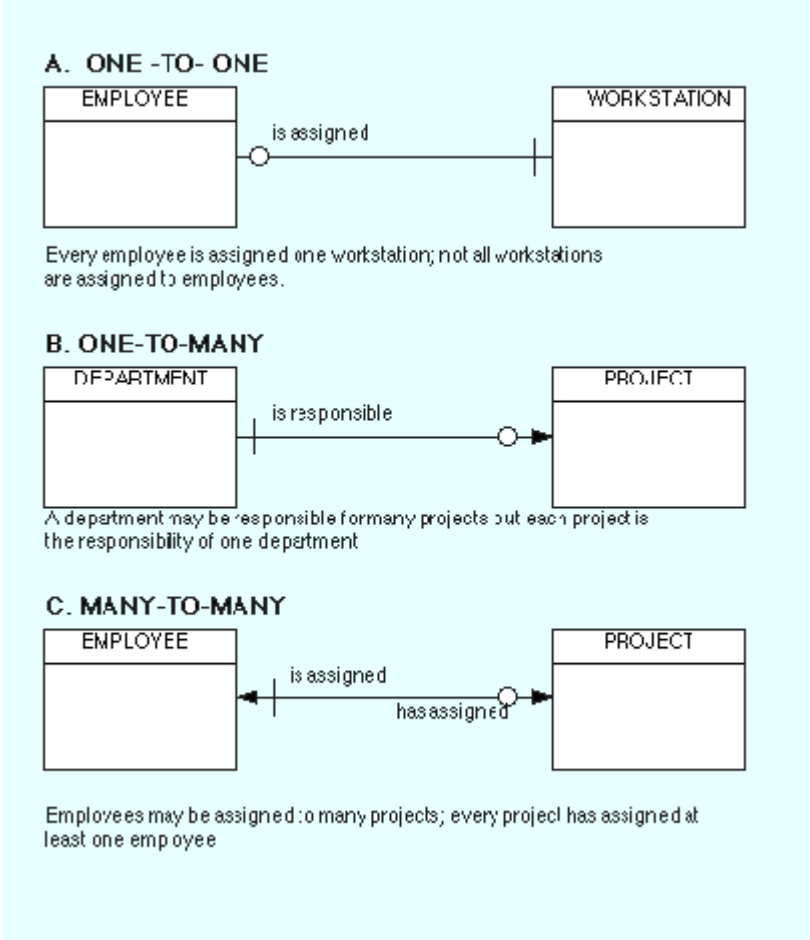


Figure 2. Example of Binary Relationships

4.2 One-To-One

Figure 2A shows an example of a one-to-one diagram. Reading the diagram from left to right represents the relationship every employee is assigned a workstation. Because every employee must have a workstation, the symbol for mandatory existence—in this case the crossbar—is placed next to the WORKSTATION entity. Reading from right to left, the diagram shows that not all workstation are assigned to employees. This condition may reflect that some workstations are kept for spares or for loans. Therefore, we use the symbol for optional existence, the circle, next to EMPLOYEE. The cardinality and existence of a relationship must be derived from the "business rules" of the organization. For example, if all workstations owned by an organization were assigned to employees, then the circle would be replaced

by a crossbar to indicate mandatory existence. One-to-one relationships are rarely seen in "real-world" data models. Some practioners advise that most one-to-one relationships should be collapsed into a single entity or converted to a generalization hierarchy.

4.3 One-To-Many

Figure 2B shows an example of a one-to-many relationship between DEPARTMENT and PROJECT. In this diagram, DEPARTMENT is considered the parent entity while PROJECT is the child. Reading from left to right, the diagram represents departments may be responsible for many projects. The optionality of the relationship reflects the "business rule" that not all departments in the organization will be responsible for managing projects. Reading from right to left, the diagram tells us that every project must be the responsibility of exactly one department.

4.4 Many-To-Many

Figure 2C shows a many-to-many relationship between EMPLOYEE and PROJECT. An employee may be assigned to many projects; each project must have many employee Note that the association between EMPLOYEE and PROJECT is optional because, at a given time, an employee may not be assigned to a project. However, the relationship between PROJECT and EMPLOYEE is mandatory because a project must have at least two employees assigned. Many-To-Many relationships can be used in the initial drafting of the model but eventually must be transformed into two one-to-many relationships. The transformation is required because many-to-many relationships cannot be represented by the relational model. The process for resolving many-to-many relationships is discussed in the next section.

4.5 Recursive relationships

A recursive relationship is an entity is associated with itself. Figure 3 shows an example of the recursive relationship.

An employee may manage many employees and each employee is managed by one employee.

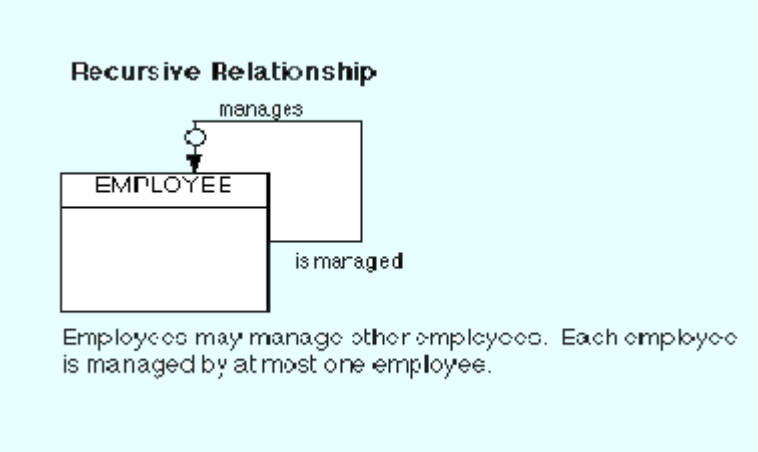


Figure 3. Example of Recursive Relationship

4.6 Summary

The Entity-Relationship diagram provides a pictorial representation of the major data objects, the entities, and the relationships between them. Once the basic diagram is completed, the next step is Refining The Entity-Relationship Diagram.

5 Refining The Entity-Relationship Diagram

This section discusses four basic rules for modeling relationships

5.1 Entities Must Participate In Relationships

Entities cannot be modeled unrelated to any other entity. Otherwise, when the model was transformed to the relational model, there would be no way to navigate to that table. The exception to this rule is a database with a single table.

5.2 Resolve Many-To-Many Relationships

Many-to-many relationships cannot be used in the data model because they cannot be represented by the relational model. Therefore, many-to-many relationships must be resolved early in the modeling process. The strategy for resolving many-to-many relationship is to replace the relationship with an association entity and then relate the two original entities to the association entity. This strategy is demonstrated below Figure 4 shows the many-to-many relationship:

Employees may be assigned to many projects.

Each project must have assigned to it more than one employee.

In addition to the implementation problem, this relationship presents other problems. Suppose we wanted to record information about employee assignments such as who assigned them, the start date of the assignment, and the finish date for the assignment. Given the present relationship, these attributes could not be represented in either EMPLOYEE or PROJECT without repeating information. The first step is to convert the relationship assigned to to a new entity we will call ASSIGNMENT. Then the original entities, EMPLOYEE and PROJECT, are related to this new entity preserving the cardinality and optionality of the original relationships. The solution is shown in Figure 4 (b).

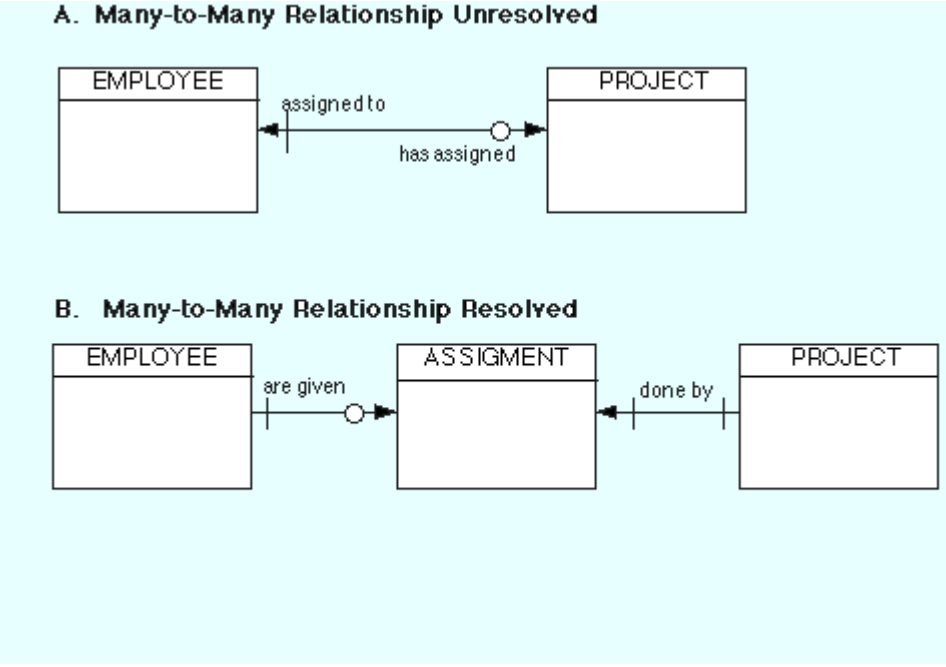


Figure 4. Resolution of a Many-To-Many Relationship

Notice that the schema changes the semantics of the original relation to

employees may be given assignments to projects
and projects must be done by more than one employee assignment.

A many to many recursive relationship is resolved in similar fashion.

5.3 Transform Complex Relationships into Binary Relationships

Complex relationships are classified as ternary, an association among three entities, or n-ary, an association among more than three, where n is the number of entities involved. For example, Figure 5A shows the relationship

Employees can use different skills on any one or more projects.

Each project uses many employees with various skills.

Complex relationships cannot be directly implemented in the relational model so they should be resolved early in the modeling process. The strategy for resolving complex relationships is similar to resolving many-to-many relationships. The complex relationship replaced by an association entity and the original entities are related to this new entity. Entity related through binary relationships to each of the original entities. The solution is shown in Figure 5B below.

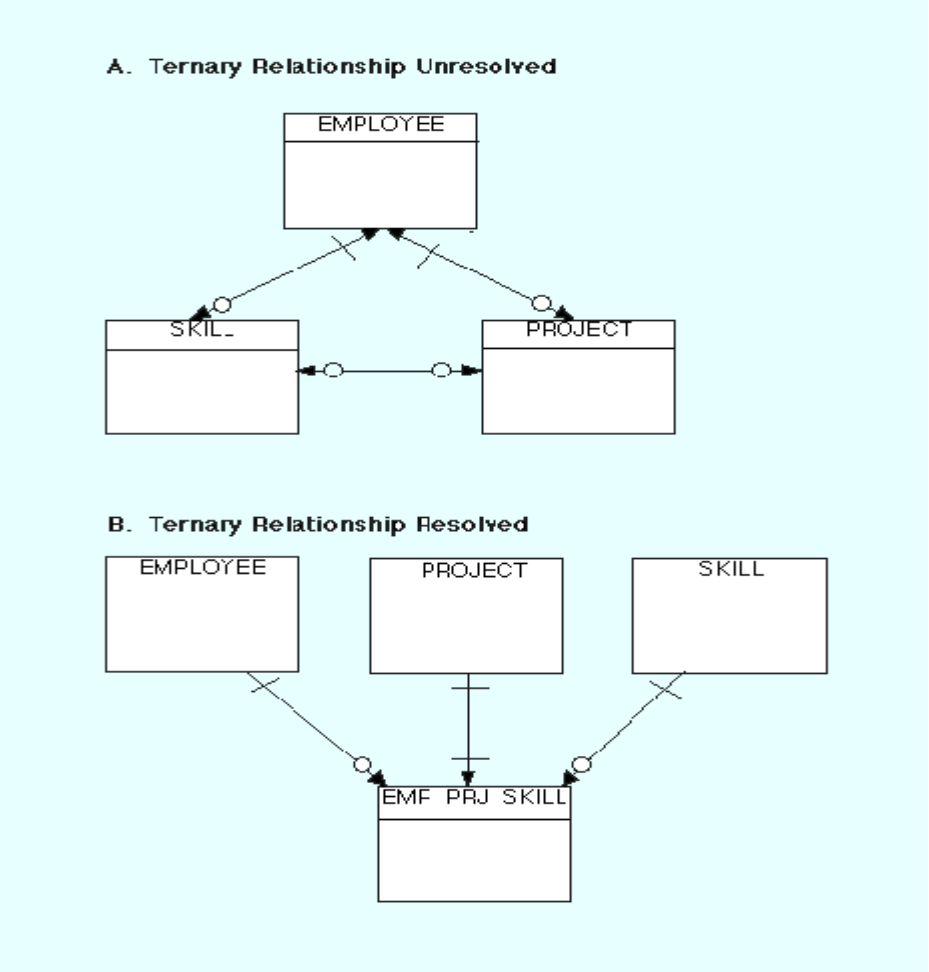


Figure 5. Transforming a Complex Relationship

5.4 Eliminate redundant relationships

A redundant relationship is a relationship between two entities that is equivalent in meaning to another relationship between those same two entities that may pass through an intermediate entity. For example, Figure 6B shows the solution which is to remove the redundant relationship DEPARTMENT assigned WORKSTATIONS.

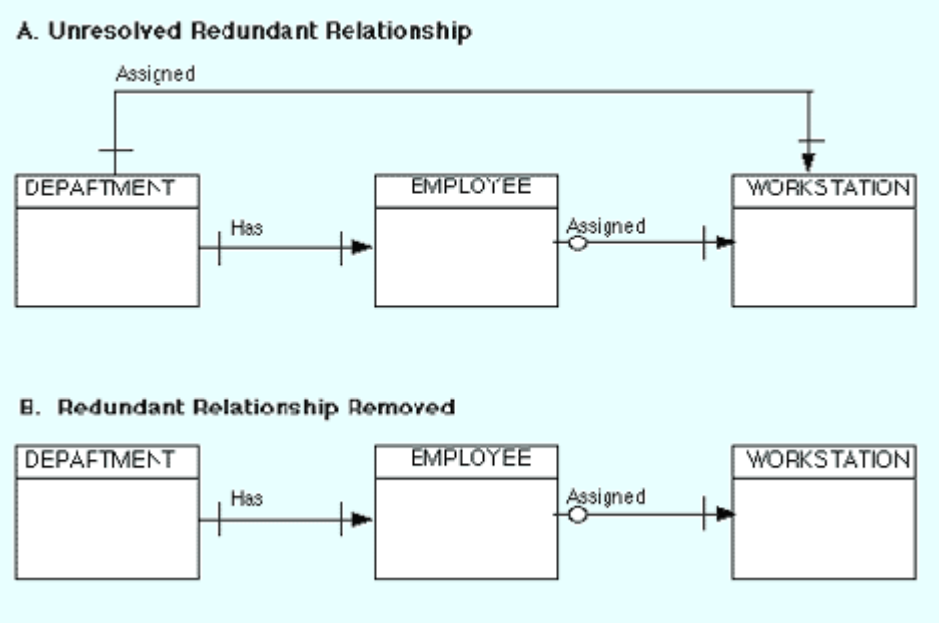


Figure 6. Removing A Redundant Relationship

5.5 Summary

The last two sections have provided a brief overview of the basic constructs in the ER diagram. The next section discusses Primary and Foreign Keys.

6 Primary and Foreign Keys

Primary and foreign keys are the most basic components on which relational theory is based. Primary keys enforce entity integrity by uniquely identifying entity instances. Foreign keys enforce referential integrity by completing an association between two entities. The next step in building the basic data model to

1. identify and define the primary key attributes for each entity
2. validate primary keys and relationships
3. migrate the primary keys to establish foreign keys

6.1 Define Primary Key Attributes

Attributes are data items that describe an entity. An attribute instance is a single value of an attribute for an instance of an entity. For example, Name and hire date are attributes of the entity EMPLOYEE. "Jane Hathaway" and "3 March 1989" are instances of the attributes name and hire date.

The primary key is an attribute or a set of attributes that uniquely identify a specific instance of an entity. Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.

To qualify as a primary key for an entity, an attribute must have the following properties:

- it must have a non-null value for each instance of the entity
- the value must be unique for each instance of an entity
- the values must not change or become null during the life of each entity instance

In some instances, an entity will have more than one attribute that can serve as a primary key. Any key or minimum set of keys that could be a primary key is called a candidate key. Once candidate keys are identified, choose one, and only one, primary key for each entity. Choose the identifier most commonly used by the user as long as it conforms to the properties listed above. Candidate keys which are not chosen as the primary key are known as alternate keys.

An example of an entity that could have several possible primary keys is Employee. Let's assume that for each employee in an organization there are three candidate keys: Employee ID, Social Security Number, and Name.

Name is the least desirable candidate. While it might work for a small department where it would be unlikely that two people would have exactly the same name, it would not work for a large organization that had hundreds or thousands of employees. Moreover, there is the possibility that an employee's name could change because of marriage. Employee ID would be a good candidate as long as each employee were assigned a unique identifier at the time of hire. Social Security would work best since every employee is required to have one before being hired.

6.2 Composite Keys

Sometimes it requires more than one attribute to uniquely identify an entity. A primary key that made up of more than one attribute is known as a composite key. Figure 7 shows an example of a composite key. Each instance of the entity Work can be uniquely identified only by a composite key composed of Employee ID and Project ID.

WORK

Employee ID	Project ID	Hours_Worked
01	01	200
01	02	120
02	01	50
02	03	120
03	03	100
03	04	200

Figure 7. Example of Composite Key

6.3 Artificial Keys

An artificial keyis one that has no meaning to the business or organization. Artificial keys are permitted when

1) no attribute has all the primary key properties

or

2) the primary key is large and complex.

6.4 Primary Key Migration

Dependent entities, entities that depend on the existence of another entity for their identification, inherit the entire primary key from the parent entity. Every entity within a generalization hierarchy inherits the primary key of the root generic entity.

6.5 Define Key Attributes

Once the keys have been identified for the model, it is time to name and define the attributes that have been used as keys.

There is no standard method for representing primary keys in ER diagrams. For this document, the name of the primary key followed by the notation (PK) is written inside the entity box. An example is shown in Figure 8.

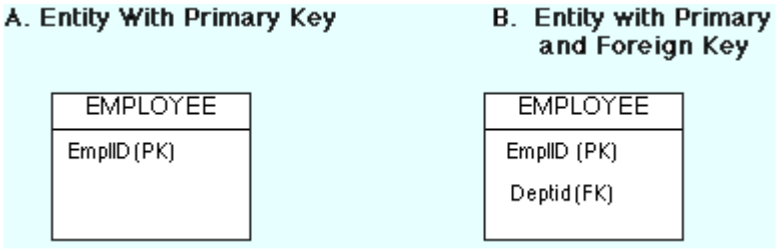


Figure 8. Entities with Key Attributes

6.6 Validate Keys and Relationships

Basic rules governing the identification and migration of primary keys are:

- Every entity in the data model shall have a primary key whose values uniquely identify entity instances.
- The primary key attribute cannot be optional (i.e., have null values).
- The primary key cannot have repeating values. That is, the attribute may not have more than one value at a time for a given entity instance is prohibited. This is known as the No Repeat Rule.
- Entities with compound primary keys cannot be split into multiple entities with simpler primary keys. This is called the Smallest Key Rule.
- Two entities may not have identical primary keys with the exception of entities within generalization hierarchies.
- The entire primary key must migrate from parent entities to child entities and from supertype, generic entities, to subtypes, category entities.

6.7 Foreign Keys

A foreign key is an attribute that completes a relationship by identifying the parent entity. Foreign keys provide a method for maintaining integrity in the data (called referential integrity) and for navigating between different instances of an entity. Every relationship in the model must be supported by a foreign key.

6.8 Identifying Foreign Keys

Every dependent and category (subtype) entity in the model must have a foreign key for each relationship in which it participates. Foreign keys are formed in dependent and subtype entities by migrating the entire primary key from the parent or generic entity. If the primary key is composite, it may not be split.

6.9 Foreign Key Ownership

Foreign key attributes are not considered to be owned by the entities to which they migrate, because they are reflections of attributes in the parent entities. Thus, each attribute in an entity is either owned by that entity or belongs to a foreign key in that entity.

If the primary key of a child entity contains all the attributes in a foreign key, the child entity is said to be "identifier dependent" on the parent entity, and the relationship is called an "identifying relationship." If any attributes in a foreign key do not belong to the child's primary key, the child is not identifier dependent on the parent, and the relationship is called "non identifying."

6.10 Diagramming Foreign Keys

Foreign keys attributes are indicated by the notation (FK) beside them. An example is shown in Figure 8 above.

6.11 Summary

Primary and foreign keys are the most basic components on which relational theory is based. Each entity must have a attribute or attributes, the primary key, whose values uniquely identify each instance of the entity. Every child entity must have an attribute, the foreign key, that completes the association with the parent entity.

The next step in building the model is to Add Attributes to the Model.

7 Adding Attributes to the Model

Non-key attributes describe the entities to which they belong. In this section, we discuss the rules for assigning non-key attributes to entities and how to handle multivalued attributes.

7.1 Relate attributes to entities

Non-key attributes can be in only one entity. Unlike key attributes, non-key attributes never migrate, and exist in only one entity, from parent to child entities.

The process of relating attributes to the entities begins by the modeler, with the assistance of the end-users, placing attributes with the entities that they appear to describe. You should record your decisions in the entity attribute matrix discussed in the previous section. Once this is completed, the assignments are validated by the formal method of normalization.

Before beginning formal normalization, the rule is to place non-key attributes in entities where the value of the primary key determines the values of the attributes. In general, entities with the same primary key should be combined into one entity. Some other guidelines for relating attributes to entities are given below.

7.2 Parent-Child Relationships

- With parent-child relationships, place attributes in the parent entity where it makes sense to do so (as long as the attribute is dependent upon the primary key)
- If a parent entity has no non-key attributes, combine the parent and child entities

7.3 Multivalued Attributes

If an attribute is dependent upon the primary key but is multivalued, has more than one value for a particular value of the key), reclassify the attribute as a new child entity. If the multivalued attribute is unique within the new entity, it becomes the primary key. If not, migrate the primary key from the original, now parent, entity.

For example, assume an entity called PROJECT with the attributes Proj_ID (the key), Proj_Name, Task_ID, Task_Name

PROJECT

Proj_ID	Proj_Name	Task_ID	Task_Name
01	A	01	Analysis
01	A	02	Design
01	A	03	Programming
01	A	04	Tuning
02	B	01	Analysis

Task_ID and Task_Name have multiple values for the key attribute. The solution is to create a new entity, let's call it TASK and make it a child of PROJECT. Move Task_ID and Task_Name from

PROJECT to TASK. Since neither attribute uniquely identifies a task, the final step would be to migrate Proj_ID to TASK.

7.4 Attributes That Describe Relations

In some cases, it appears that an attribute describes a relationship rather than an entity (in the Chen notation of ER diagrams this is permissible). For example,

a MEMBER borrows BOOKS.

Possible attributes are the date the books were checked out and when they are due. Typically, such a situation will occur with a many-to-many relationship and the solution is the same. Reclassify the relationship as a new entity which is a child to both original entities. In some methodologies, the newly created is called an associative entity. See Figure 9 for an example of an converting a relationship into an associative entity.

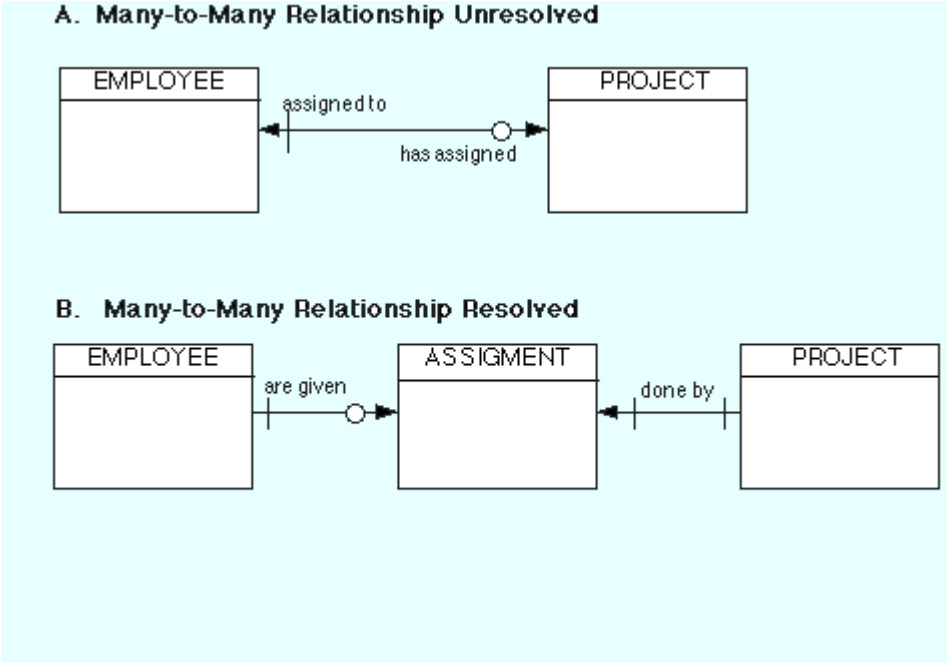


Figure 9. Converting a relationship into an associative entity.

7.5 Derived Attributes and Code Values

Two areas where data modeling experts disagree is whether derived attributes and attributes whose values are codes should be permitted in the data model.

Derived attributes are those created by a formula or by a summary operation on other attributes. Arguments against including derived data are based on the premise that derived data should not be stored in a database and therefore should not be included in the data model. The arguments in favor are:

- derived data is often important to both managers and users and therefore should be included in the data model.
- it is just as important, perhaps more so, to document derived attributes just as you would other attributes
- including derived attributes in the data model does not imply how they will be implemented.

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female". Those who are against this

practice cite that codes have no intuitive meaning to the end-users and add complexity to processing data. Those in favor argue that many organizations have a long history of using coded attributes, that codes save space, and improve flexibility in that values can be easily added or modified by means of look-up tables.

7.6 Including Attributes to the ER Diagram

There is disagreement about whether attributes should be part of the entity-relationship diagram. The IDEF1X standard specifies that attributes should be added. Many experienced practitioners, however, note that adding attributes, especially if there are a large number, clutters the diagram and detracts from its ability to present the end-user with an overview of how the data is structured.

7.7 Summary

By the end of this stage you should have:

1. identified, named, and defined data objects and relationships
2. recorded information about data objects and relationships in the data document
3. created and refined the ER diagram
4. assigned attributes to entities
5. added attributes to ER diagram (optional)

The next section discusses Generalization Hierarchies.

8 Generalization Hierarchies

Up to this point, we have discussed describing an object, the entity, by its shared characteristics, the attributes. For example, we can characterize an employee by their employee id, name, job title, and skill set.

Another method of characterizing entities is by both similarities and differences. For example, suppose an organization categorizes the work it does into internal and external projects. Internal projects are done on behalf of some unit within the organization. External projects are done for entities outside of the organization. We can recognize that both types of projects are similar in that each involves work done by employees of the organization within a given schedule. Yet we also recognize that there are differences between them. External projects have unique attributes, such as a customer identifier and the fee charged to the customer. This process of categorizing entities by their similarities and differences is known as generalization.

8.1 Description

A generalization hierarchy is a structured grouping of entities that share common attributes. It is a powerful and widely used method for representing common characteristics among entities while preserving their differences. It is the relationship between an entity and one or more refined versions. The entity being refined is called the supertype and each refined version is called the subtype. The general form for a generalization hierarchy is shown in Figure 10.

Generalization hierarchies should be used when

1) a large number of entities appear to be of the same type
2) attributes are repeated for multiple entities

or

2) the model is continually evolving.

Generalization hierarchies improve the stability of the model by allowing changes to be made only to those entities germane to the change and simplify the model by reducing the number of entities in the model.

8.2 Creating a Generalization Hierarchy

To construct a generalization hierarchy, all common attributes are assigned to the supertype. The supertype is also assigned an attribute, called a discriminator, whose values identify the categories of the subtypes. Attributes unique to a category, are assigned to the appropriate subtype. Each subtype also inherits the primary key of the supertype. Subtypes that have only a primary key should be eliminated. Subtypes are related to the supertypes through a one-to-one relationship.

8.3 Types of Hierarchies

A generalization hierarchy can either be overlapping or disjoint. In an overlapping hierarchy an entity instance can be part of multiple subtypes. For example, to represent people at a university you have identified the supertype entity PERSON which has three subtypes, FACULTY, STAFF, and STUDENT. It is quite possible for an individual to be in more than one subtype, a staff member who is also registered as a student, for example.In a disjoint hierarchy, an entity instance can be in only one subtype. For example, the entity EMPLOYEE, may have two subtypes, CLASSIFIED and WAGES. An employee may be one type or the other but not both. Figure 10 shows A) overlapping and B) disjoint generalization hierarchy.

8.4 Rules

The primary rule of generalization hierarchies is that each instance of the supertype entity must appear in at least one subtype; likewise, an instance of the subtype must appear in the supertype.

Subtypes can be a part of only one generalization hierarchy. That is, a subtype can not be related to more than one supertype. However, generalization hierarchies may be nested by having the subtype of one hierarchy be the supertype for another.

Subtypes may be the parent entity in a relationship but not the child. If this were allowed, the subtype would inherit two primary keys.

8.5 Summary

Generalization hierarchies are a structure that enables the modeler to represent entities that share common characteristics but also have differences.

The next and final step in the modeling process is to Add Data Integrity Rules.

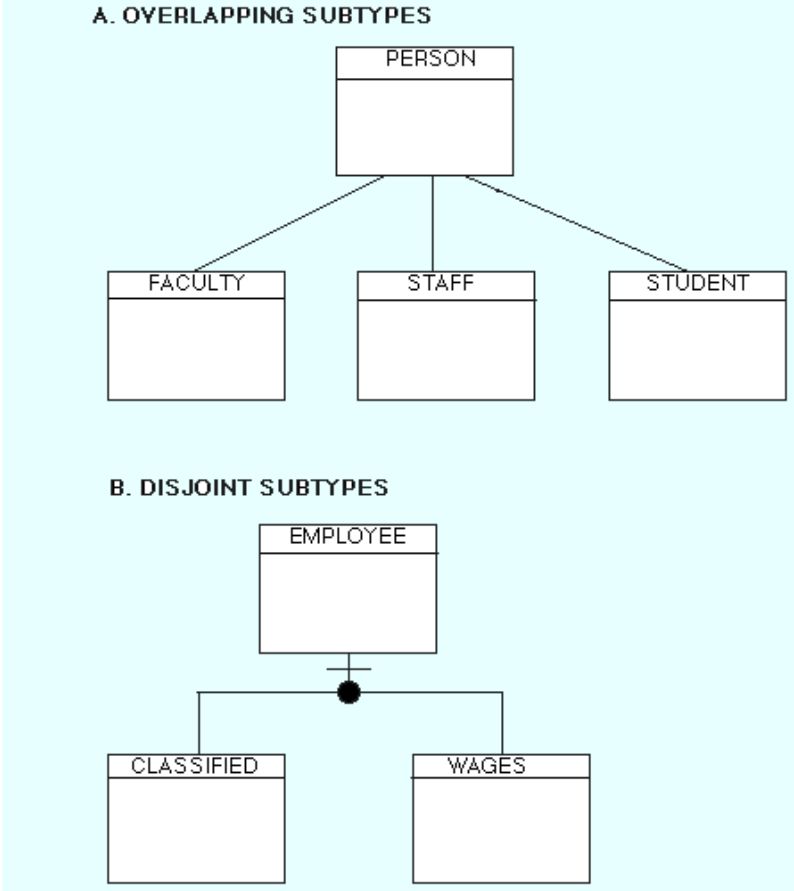


Figure 10. Examples of Generalization Hierarchies

9 Add Data Integrity Rules

Data integrity is one of the cornerstones of the relational model. Simply stated data integrity means that the data values in the database are correct and consistent.

Data integrity is enforced in the relational model by entity and referential integrity rules. Although not part of the relational model, most database software enforces attribute integrity through the use of domain information.

9.1 Entity Integrity

The entity integrity rule states that for every instance of an entity, the value of the primary key must exist, be unique, and cannot be null. Without entity integrity, the primary key could not fulfill its role of uniquely identifying each instance of an entity.

9.2 Referential Integrity

The referential integrity rule states that every foreign key value must match a primary key value in an associated table. Referential integrity ensures that we can correctly navigate between related entities.

9.3 Insert and Delete Rules

A foreign key creates a hierarchical relationship between two associated entities. The entity containing the foreign key is the child, or dependent, and the table containing the primary key from which the foreign key values are obtained is the parent.

In order to maintain referential integrity between the parent and child as data is inserted or deleted from the database certain insert and delete rules must be considered.

9.4 Insert Rules

Insert rules commonly implemented are:

- Dependent. The dependent insert rule permits insertion of child entity instance only if matching parent entity already exists.
- Automatic. The automatic insert rule always permits insertion of child entity instance. If matching parent entity instance does not exist, it is created.
- Nullify. The nullify insert rule always permits the insertion of child entity instance. If a matching parent entity instance does not exist, the foreign key in child is set to null.
- Default. The default insert rule always permits insertion of child entity instance. If a matching parent entity instance does not exist, the foreign key in the child is set to previously defined value.
- Customized. The customized insert rule permits the insertion of child entity instance only if certain customized validity constraints are met.
- No Effect. This rule states that the insertion of child entity instance is always permitted. No matching parent entity instance need exist, and thus no validity checking is done.

9.5 Delete Rules

- Restrict. The restrict delete rule permits deletion of parent entity instance only if there are no matching child entity instances.
- Cascade. The cascade delete rule always permits deletion of a parent entity instance and deletes all matching instances in the child entity.
- Nullify. The nullify delete rules always permits deletion of a parent entity instance. If any matching child entity instances exist, the values of the foreign keys in those instances are set to null.
- Default. The default rule always permits deletion of a parent entity instance. If any matching child entity instances exist, the value of the foreign keys are set to a predefined default value.

- Customized. The customized delete rule permits deletion of a parent entity instance only if certain validity constraints are met.
- No Effect. The no effect delete rule always permits deletion of a parent entity instance. No validity checking is done.

9.6 Delete and Insert Guidelines

The choice of which rule to use is determined by Some basic guidelines for insert and delete rules are given below.

- Avoid use of nullify insert or delete rules. Generally, the parent entity in a parent-child relationship has mandatory existence. Use of the null insert or delete rule would violate this rule.
- Use either automatic or dependent insert rule for generalization hierarchies. Only these rules will keep the rule that all instances in the subtypes must also be in the supertype.
- Use the cascade delete rule for generalization hierarchies. This rule will enforce the rule that only instances in the supertype can appear in the subtypes.

9.7 Domains

A domain is a valid set of values for an attribute which enforce that values from an insert or update make sense. Each attribute in the model should be assigned domain information which includes:

- Data Type—Basic data types are integer, decimal, or character. Most data bases support variants of these plus special data types for date and time.
- Length—This is the number of digits or characters in the value. For example, a value of 5 digits or 40 characters.
- Date Format—The format for date values such as dd/mm/yy or yy/mm/dd
- Range—The range specifies the lower and upper boundaries of the values the attribute may legally have
- Constraints—Are special restrictions on allowable values. For example, the Beginning_Pay_Date for a new employee must always be the first work day of the month of hire.
- Null support—Indicates whether the attribute can have null values
- Default value (if any)—The value an attribute instance will have if a value is not entered.

9.8 Primary Key Domains

The values of primary keys must be unique and nulls are not allowed.

9.9 Foreign Key Domains

The data type, length, and format of primary keys must be the same as the corresponding primary key. The uniqueness property must be consistent with relationship type. A one-to-one relationship implies a unique foreign key; a one-to-many relationship implies a non-unique foreign key.

10 Overview of the Relational Model

The relational model was formally introduced by Dr. E. F. Codd in 1970 and has evolved since then, through a series of writings. The model provides a simple, yet rigorously defined, concept of how users perceive data. The relational model represents data in the form of two-dimension tables. Each table represents some real-world person, place, thing, or event about which information is collected. A relational database is a collection of two-dimensional tables. The organization of data into relational tables is known as the logical view of the database. That is, the form in which a relational database presents data to the user and the programmer. The way the database software physically stores the data on a computer disk system is called the internal view. The internal view differs from product to product and does not concern us here.

A basic understanding of the relational model is necessary to effectively use relational database software such as Oracle, Microsoft SQL Server, or even personal database systems such as Access or Fox, which are based on the relational model.

This document is an informal introduction to relational concepts, especially as they relate to relational database design issues. It is not a complete description of relational theory.

This section discusses the basic concepts—data structures, relationships, and data integrity—that are the basis of the relational model.

- Data Structure and Terminology
- Notation
- Properties of Relational Tables
- Relationships and Keys
- Data Integrity
- Relational Data Manipulation
- Normalization
- Advanced Normalization

11 Data Structure and Terminology

In the relational model, a database is a collection of relational tables. A relational table is a flat file composed of a set of named columns and an arbitrary number of unnamed rows. The columns of the tables contain information about the table. The rows of the table represent occurrences of the "thing" represented by the table. A data value is stored in the intersection of a row and column. Each named column has a domain, which is the set of values that may appear in that column. Figure 11 shows the relational tables for a simple bibliographic database that stores information about book title, authors, and publishers.

A Relational Data Base

AUTHOR

au_id	au_lname	au_fname	address	city	state
172-32-1176	White	Johnson	10932 Bigge Rd.	Menlo Park	CA
213-46-8915	Green	Marjorie	309 63rd St. #411	Oakland	CA
238-95-7766	Carson	Cheryl	589 Darwin Ln.	Berkeley	CA
267-41-2394	O'Leary	Michael	22 Cleveland Av. #14	San Jose	CA
274-80-9391	Straight	Dean	5420 College Av.	Oakland	CA
341-22-1782	Smith	Meander	10 Mississippi Dr.	Lawrence	KS
409-56-7008	Bennet	Abraham	6223 Bateman St.	Berkeley	CA
427-17-2319	Dull	Ann	3410 Blonde St.	Palo Alto	CA
472-27-2349	Gringlesby	Burt	PO Box 792	Covelo	CA
486-29-1786	Locksley	Charlene	18 Broadway Av.	San Francisco	CA

TITLE

title_id	title	type	price	pub_id
BU1032	The Busy Executive's Database Guide	business	19.99	1389
BU1111	Cooking with Computers	business	11.95	1389
BU2075	You Can Combat Computer Stress!	business	2.99	736
BU7832	Straight Talk About Computers	business	19.99	1389
MC2222	Silicon Valley Gastronomic Treats	mod_cook	19.99	877
MC3021	The Gourmet Microwave	mod_cook	2.99	877
MC3026	The Psychology of Computer Cooking	UNDECIDED		877
PC1035	But Is It User Friendly?	popular_comp	22.95	1389
PC8888	Secrets of Silicon Valley	popular_comp	20	1389
PC9999	Net Etiquette	popular_comp		1389
PS2091	Is Anger the Enemy?	psychology	10.95	736

PUBLISHER

pub_id	pub_name	city
736	New Moon Books	Boston
877	Binnet & Hardley	Washington
1389	Algodata Infosystems	Berkeley
1622	Five Lakes Publishing	Chicago
1756	Ramona Publishers	Dallas
9901	GGG&G	München
9952	Scootney Books	New York
9999	Lucerne Publishing	Paris

AUTHOR_TITLE

au_id	title_id
172-32-1176	PS3333
213-46-8915	BU1032
213-46-8915	BU2075
238-95-7766	PC1035
267-41-2394	BU1111
267-41-2394	TC7777
274-80-9391	BU7832
409-56-7008	BU1032
427-17-2319	PC8888
472-27-2349	TC7777

Figure 11. Relational tables

There are alternate names used to describe relational tables. Some manuals use the terms tables, fields, and records to describe relational tables, columns, and rows, respectively. The formal literature tends to use the mathematical terms, relations, attributes, and tuples. Figure 12 summarizes these naming conventions.

In This Document	Formal Terms	Many Database Manuals
Relational Table	Relation	Table
Column	Attribute	Field
Row	Tuple	Record

Figure 12. Terminology

Notation

Relational tables can be expressed concisely by eliminating the sample data and showing just the table name and the column names. For example,

AUTHOR	(au_id, au_lname, au_fname, address, city, state, zip)
TITLE	(title_id, title, type, price, pub_id)
PUBLISHER	(pub_id, pub_name, city)
AUTHOR_TITLE	(au_id, title_id)

12 Properties of Relational Tables

Relational tables have six properties:

1. Values are atomic.
2. Column values are of the same kind.
3. Each row is unique.
4. The sequence of columns is insignificant.
5. The sequence of rows is insignificant.
6. Each column must have a unique name.

Values Are Atomic

This property implies that columns in a relational table are not repeating group or arrays. Such tables are referred to as being in the "first normal form" (1NF). The atomic value property of relational tables is important because it is one of the cornerstones of the relational model.

The key benefit of the one value property is that it simplifies data manipulation logic.

Column Values Are of the Same Kind

In relational terms this means that all values in a column come from the same domain. A domain is a set of values which a column may have. For example, a Monthly_Salary column contains only specific monthly salaries. It never contains other information such as comments, status flags, or even weekly salary.

This property simplifies data access because developers and users can be certain of the type of data contained in a given column. It also simplifies data validation. Because all values are from the same domain, the domain can be defined and enforced with the Data Definition Language (DDL) of the database software.

Each Row is Unique

This property ensures that no two rows in a relational table are identical; there is at least one column, or set of columns, the values of which uniquely identify each row in the table. Such columns are called primary keys and are discussed in more detail in Relationships and Keys.

This property guarantees that every row in a relational table is meaningful and that a specific row can be identified by specifying the primary key value.

The Sequence of Columns is Insignificant

This property states that the ordering of the columns in the relational table has no meaning. Columns can be retrieved in any order and in various sequences. The benefit of this property is that it enables many users to share the same table without concern of how the table is organized. It also permits the physical structure of the database to change without affecting the relational tables.

The Sequence of Rows is Insignificant

This property is analogous the one above but applies to rows instead of columns. The main benefit is that the rows of a relational table can be retrieved in different order and sequences. Adding information to a relational table is simplified and does not affect existing queries.

Each Column Has a Unique Name

Because the sequence of columns is insignificant, columns must be referenced by name and not by position. In general, a column name need not be unique within an entire database but only within the table to which it belongs.

13 Relationships and Keys

A relationship is an association between two or more tables. Relationships are expressed in the data values of the primary and foreign keys.

A primary key is a column or columns in a table whose values uniquely identify each row in a table. A foreign key is a column or columns whose values are the same as the primary key of another table. You can think of a foreign key as a copy of primary key from another relational table. The relationship is made between two relational tables by matching the values of the foreign key in one table with the values of the primary key in another.

Keys are fundamental to the concept of relational databases because they enable tables in the database to be related with each other. Navigation around a relational database depends on the ability of the primary key to unambiguously identify specific rows of a table. Navigating between tables requires that the foreign key is able to correctly and consistently reference the values of the primary keys of a related table. For example, the figure below shows how the keys in the relational tables are used to navigate from AUTHOR to TITLE to PUBLISHER. AUTHOR_TITLE is an all key table used to link AUTHOR and TITLE. This relational table is required because AUTHOR and TITLE have a many-to-many relationship.

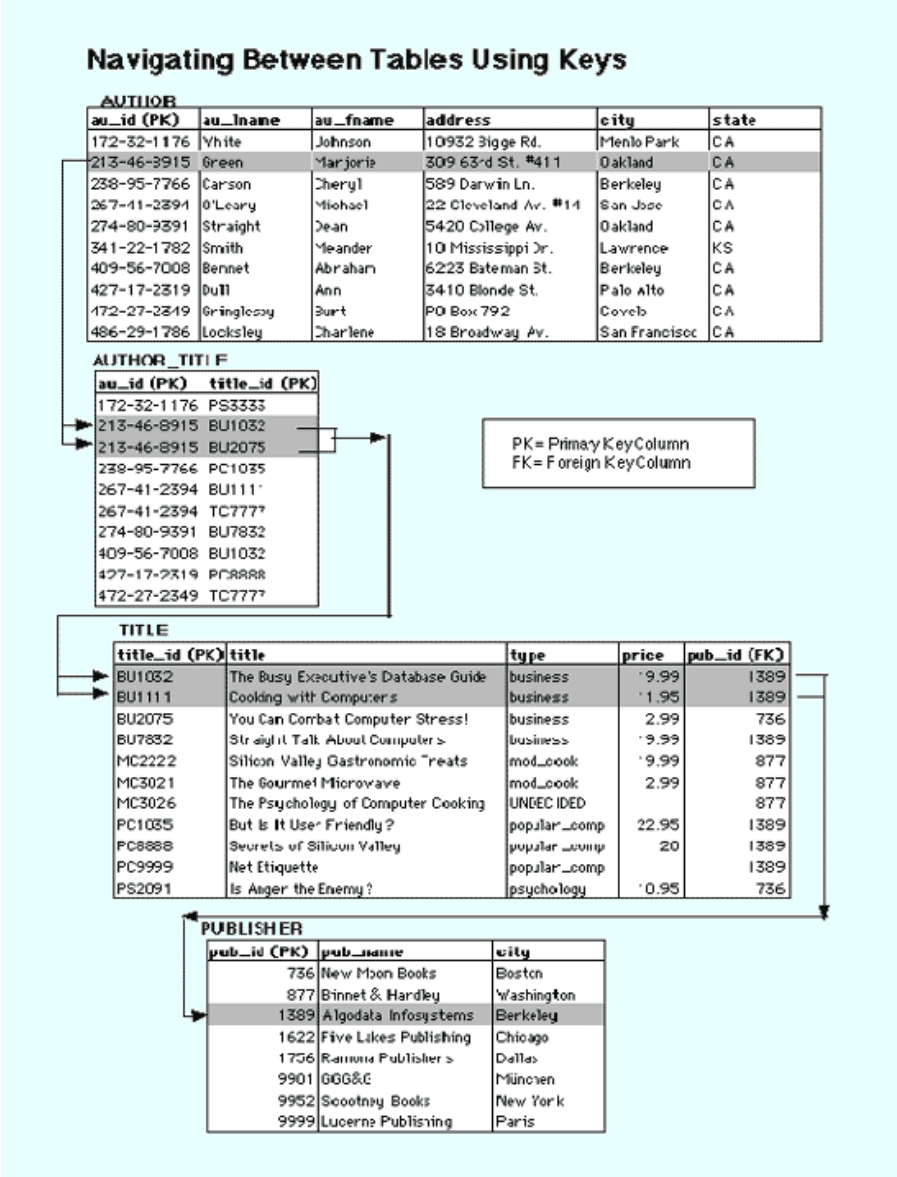


Figure 13. How the keys are used to navigate Data Integrity

Data integrity means, in part, that you can correctly and consistently navigate and manipulate the tables in the database. There are two basic rules to ensure data integrity; entity integrity and referential integrity.

The entity integrity rule states that the value of the primary key can never be a null value (a null value is one that has no value and is not the same as a blank). Because a primary key is used to identify a unique row in a relational table, its value must always be specified and should never be unknown. The integrity rule requires that insert, update, and delete operations maintain the uniqueness and existence of all primary keys.

The referential integrity rule states that if a relational table has a foreign key, then every value of the foreign key must either be null or match the values in the relational table in which that foreign key is a primary key.

14 Relational Data Manipulation

Relational tables are sets. The rows of the tables can be considered as elements of the set. Operations that can be performed on sets can be done on relational tables. The eight relational operations are:

14.1 Union

The union operation of two relational tables is formed by appending rows from one table to those of a second table to produce a third. Duplicate rows are eliminated. The notation for the union of Tables A and B is A UNION B.

The relational tables used in the union operation must be union compatible. Tables that are union compatible must have the same number of columns and corresponding columns must come from the same domain. Figure 14 shows the union of A and B.

Note that the duplicate row [1, A, 2] has been removed.

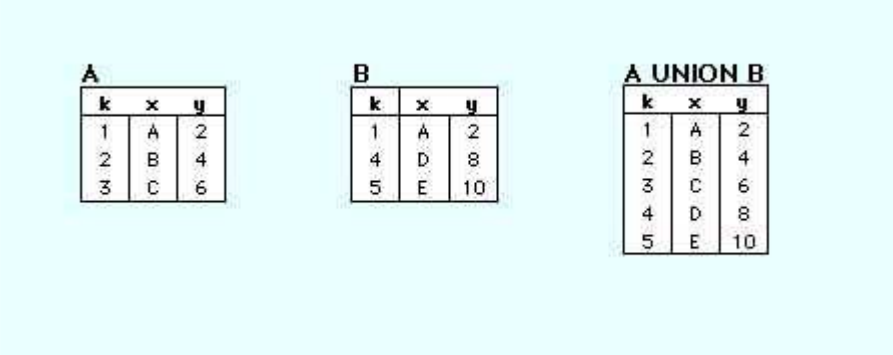


Figure 14. A UNION B

14.2 Difference

The difference of two relational tables is a third that contains those rows that occur in the first table but not in the second. The Difference operation requires that the tables be union compatible. As with arithmetic, the order of subtraction matters. That is, A - B is not the same as B - A. Figure 15 shows the different results.

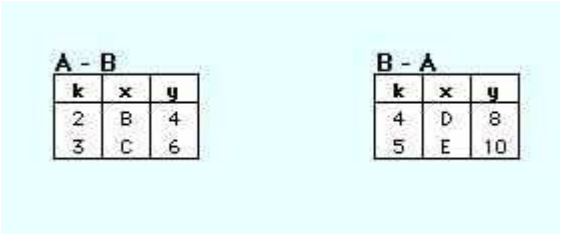


Figure 15. The Difference Operator

14.3 Intersection

The intersection of two relational tables is a third table that contains common rows. Both tables must be union compatible. The notation for the intersection of A and B is A [intersection] B = C or A INTERSECT B. Figure 16 shows the single row [1, A, 2] appears in both A and B.

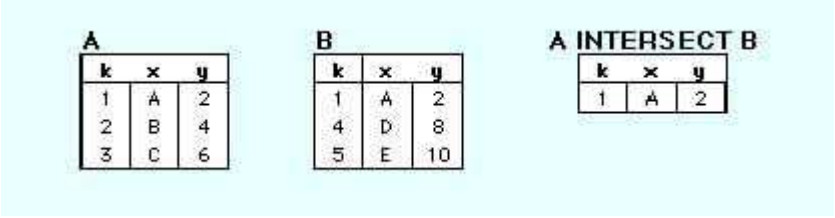
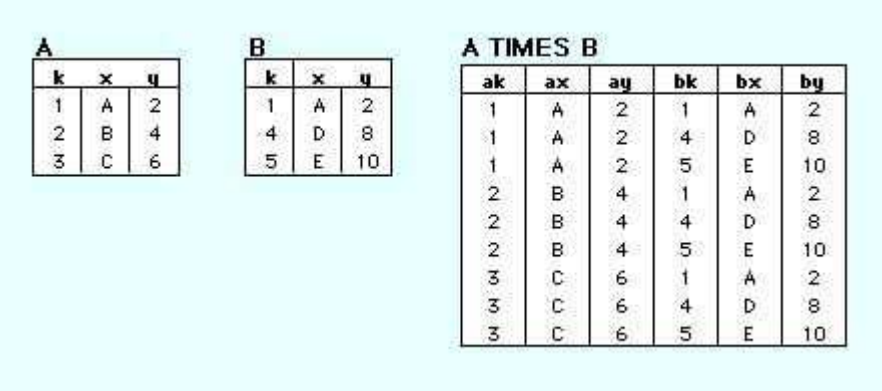


Figure 16. Intersection

14.4 Product

The product of two relational tables, also called the Cartesian Product, is the concatenation of every row in one table with every row in the second. The product of table A (having m rows) and table B (having n rows) is the table C (having m x n rows). In the Figure 17 the product is denoted as A X B or A TIMES B.



A

k	x	y
1	A	2
2	B	4
3	C	6

B

k	x	y
1	A	2
4	D	8
5	E	10

A TIMES B

ak	ax	ay	bk	bx	by
1	A	2	1	A	2
1	A	2	4	D	8
1	A	2	5	E	10
2	B	4	1	A	2
2	B	4	4	D	8
2	B	4	5	E	10
3	C	6	1	A	2
3	C	6	4	D	8
3	C	6	5	E	10

Figure 17. Product

The product operation is by itself not very useful. However, it is often used as an intermediate process in a Join.

14.5 Projection

The project operator retrieves a subset of columns from a table, removing duplicate rows from the result.

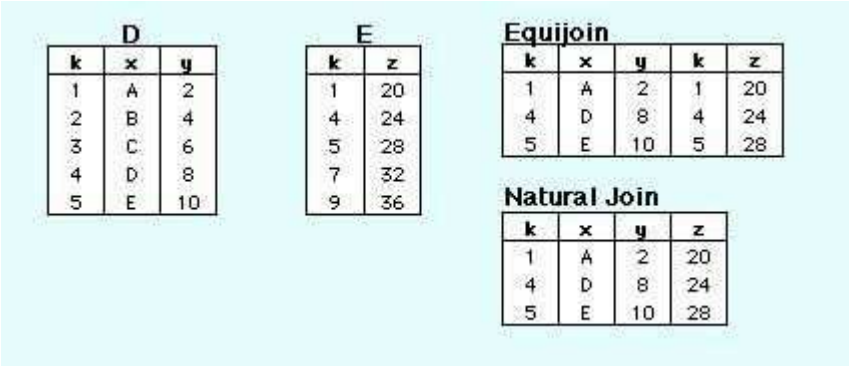
14.6 Selection

The select operator, sometimes called restrict to prevent confusion with the SQL SELECT command, retrieves subsets of rows from a relational table based on a value(s) in a column or columns.

14.7 Join

A join operation combines the product, selection, and, possibly, projection. The join operator horizontally combines (concatenates) data from one row of a table with rows from another or the same table when certain criteria are met. The criteria involve a relationship among the columns in the join relational table. If the join criterion is based on equality of column value, the result is called an equijoin. A natural join is an equijoin with redundant columns removed.

Figure 18 illustrates a join operation. Tables D and E are joined based on the equality of k in both tables. The first result is an equijoin. Note that there are two columns named k; the second result is a natural join with the redundant column removed.



D

k	x	y
1	A	2
2	B	4
3	C	6
4	D	8
5	E	10

E

k	z
1	20
4	24
5	28
7	32
9	36

Equijoin

k	x	y	k	z
1	A	2	1	20
4	D	8	4	24
5	E	10	5	28

Natural Join

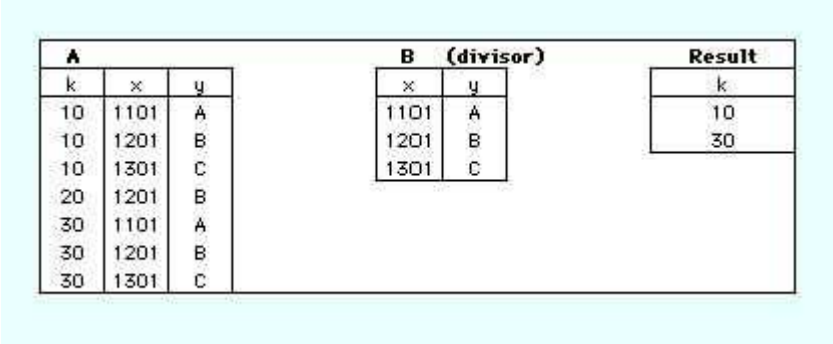
k	x	y	z
1	A	2	20
4	D	8	24
5	E	10	28

Figure 18. Join

Joins can also be done on criteria other than equality.

14.8 Division

In the Figure 19 the division operator results in columns values in one table for which there are other matching column values corresponding to every row in another table.



A

k	x	y
10	1101	A
10	1201	B
10	1301	C
20	1201	B
30	1101	A
30	1201	B
30	1301	C

B (divisor)

x	y
1101	A
1201	B
1301	C

Result

k
10
30

Figure 19. Division

15 Relationships

A Relationship represents an association between two or more entities. An example of a relationship would be:

employees are assigned to projects
projects have subtasks
departments manage one or more projects

Relationships are classified in terms of degree, connectivity, cardinality, and existence. These concepts will be discussed below.

15.1 Attributes

Attributes describe the entity of which they are associated. A particular instance of an attribute is a value. For example, "Jane R. Hathaway" is one value of the attribute Name. The domain of an attribute is the collection of all possible values an attribute can have. The domain of Name is a character string.

Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

15.2 Classifying Relationships

Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence. Not all modeling methodologies use all these classifications.

15.3 Degree of a Relationship

The degree of a relationship is the number of entities associated with the relationship. The n-ary relationship is the general form for degree n. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities is the most common type in the real world. A recursive binary relationship occurs when an entity is related to itself. An example might be "some employees are married to other employees".

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

15.4 Connectivity and Cardinality

The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

A one-to-one (1:1) relationship is when at most one instance of a entity A is associated with one instance of entity B. For example, "employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.

A one-to-many (1:N) relationships is when for one instance of entity A, there are zero, one, or many instances of entity B, but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

a department has many employees

each employee is assigned to one department

A many-to-many (M:N) relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is:

employees can be assigned to no more than two projects at the same time;
projects must have assigned at least three employees

A single employee can be assigned to many projects; conversely, a single project can have assigned to it many employee. Here the cardinality for the relationship between employees and projects is two and the cardinality between project and employee is three. Many-to-many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

15.5 Direction

The direction of a relationship indicates the originating entity of a binary relationship. The entity from which a relationship originates is the parent entity; the entity where the relationship terminates is the child entity.

The direction of a relationship is determined by its connectivity. In a one-to-one relationship the direction is from the independent entity to a dependent entity. If both entities are independent, the direction is arbitrary. With one-to-many relationships, the entity occurring once is the parent. The direction of many-to-many relationships is arbitrary.

15.6 Type

An identifying relationship is one in which one of the child entities is also a dependent entity. A non-identifying relationship is one in which both entities are independent.

15.7 Existence

Existence denotes whether the existence of an entity instance is dependent upon the existence of another, related, entity instance. The existence of an entity in a relationship is defined as either mandatory or optional. If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory. An example of mandatory existence is the statement "every project must be managed by a single department". If the instance of the entity is not required, it is optional. An example of optional existence is the statement, "employees may be assigned to work on projects".

15.8 Generalization Hierarchies

A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher level entity type called a supertype or generic entity. The lower-level of entities become the subtype, or categories, to the supertype. Subtypes are dependent entities.

Generalization occurs when two or more entities represent categories of the same real-world object. For example, Wages_Employees and Classified_Employees represent categories of the same entity,

Employees. In this example, Employees would be the supertype; Wages_Employees and Classified_Employees would be the subtypes.

Subtypes can be either mutually exclusive (disjoint) or overlapping (inclusive). A mutually exclusive category is when an entity instance can be in only one category. The above example is a mutually exclusive category. An employee can either be wages or classified but not both. An overlapping category is when an entity instance may be in two or more subtypes. An example would be a person who works for a university could also be a student at that same university. The completeness constraint requires that all instances of the subtype be represented in the supertype.

Generalization hierarchies can be nested. That is, a subtype of one hierarchy can be a supertype of another. The level of nesting is limited only by the constraint of simplicity. Subtype entities may be the parent entity in a relationship but not the child.

15.9 ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. The original notation used by Chen is widely used in academics texts and journals but rarely seen in either CASE tools or publications by non-academics. Today, there are a number of notations used, among the more common are Bachman, crow's foot, and IDEFIX.

All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin. The symbols used for the basic ER constructs are:

- entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.
- relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.
- attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.
- cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.
- existence is represented by placing a circle or a perpendicular bar on the line. Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional.

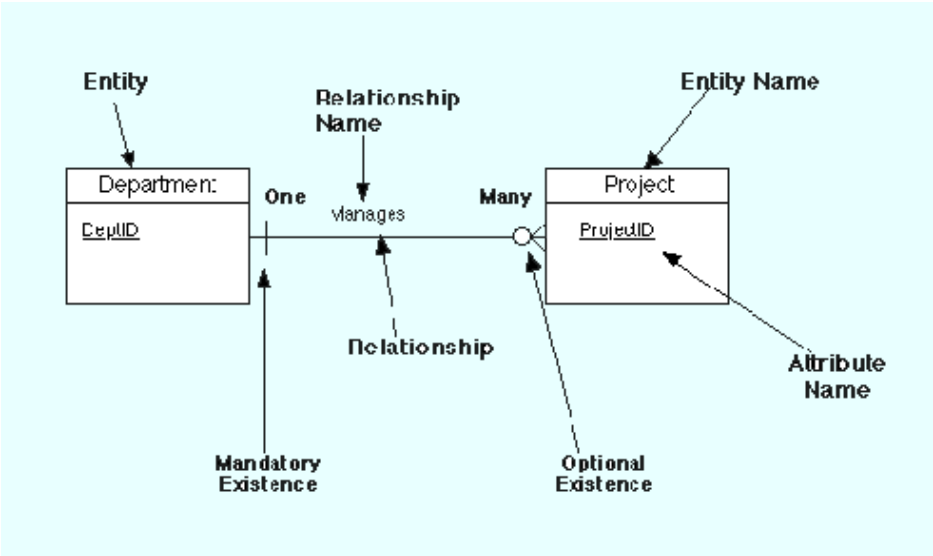


Figure 20. ER Notation

Examples of these symbols are shown in Figure 20.

15.10 Summary

The Entity-Relationship Model is a conceptual data model that views the real world as consisting of entities and relationships. The model visually represents these concepts by the Entity-Relationship diagram. The basic constructs of the ER model are entities, relationships, and attributes. Entities are concepts, real or abstract, about which information is collected. Relationships are associations between the entities. Attributes are properties which describe the entities. Next, we will look at the role of data modeling in the overall database design process and a method for building the data model. To proceed, see Data Modeling As Part of Database Design.

16 Normalization

Normalization is a design technique that is widely used as a guide in designing relational databases. Normalization is essentially a two step process that puts data into tabular form by removing repeating groups and then removes duplicated data from the relational tables.

Normalization theory is based on the concepts of normal forms. A relational table is said to be a particular normal form if it satisfied a certain set of constraints. There are currently five normal forms that have been defined. In this section, we will cover the first three normal forms that were defined by E. F. Codd.

16.1 Basic Concepts

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the third normal form (3NF). A relational table is in 3NF if and only if all non-key columns are (a) mutually independent and (b) fully dependent upon the primary key. Mutual independence means that no non-key column is dependent upon any combination of the other columns. The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF. In order to better understand the 2NF and higher forms, it is necessary to understand the concepts of functional dependencies and lossless decomposition.

16.2 Functional Dependencies

The concept of functional dependencies is the basis for the first three normal forms. A column, Y, of the relational table R is said to be functionally dependent upon column X of R if and only if each value of X in R is associated with precisely one value of Y at any given time. X and Y may be composite. Saying that column Y is functionally dependent upon X is the same as saying the values of column X identify the values of column Y. If column X is a primary key, then all columns in the relational table R must be functionally dependent upon X.

A short-hand notation for describing a functional dependency is:

R.x —>; R.y

which can be read as in the relational table named R, column x functionally determines (identifies) column y.

Full functional dependence applies to tables with composite keys. Column Y in relational table R is fully functional on X of R if it is functionally dependent on X and not functionally dependent upon any subset of X. Full functional dependence means that when a primary key is composite, made of two or more columns, then the other columns must be identified by the entire key and not just some of the columns that make up the key.

16.3 Overview

Simply stated, normalization is the process of removing redundant data from relational tables by decomposing (splitting) a relational table into smaller tables by projection. The goal is to have only primary keys on the left hand side of a functional dependency. In order to be correct, decomposition must

be lossless. That is, the new tables can be recombined by a natural join to recreate the original table without creating any spurious or redundant data.

16.4 Sample Data

Data taken from Date [Date90] is used to illustrate the process of normalization. A company obtains parts from a number of suppliers. Each supplier is located in one city. A city can have more than one supplier located there and each city has a status code associated with it. Each supplier may provide many parts. The company creates a simple relational table to store this information that can be expressed in relational notation as:

FIRST (s#, status, city, p#, qty)

where

s#	supplier identifcation number (this is the primary key)
status	status code assigned to city
city	name of city where supplier is located
p#	part number of part supplied
qty>	quantity of parts supplied to date

In order to uniquely associate quantity supplied (qty) with part (p#) and supplier (s#), a composite primary key composed of s# and p# is used.

16.5 First Normal Form

A relational table, by definition, is in first normal form. All values of the columns are atomic. That is, they contain no repeating values.

Figure 21 shows the table FIRST in 1NF.

FIRST

s#	status	city	p#	qty
s1	20	London	p1	300
s1	20	London	p2	200
s1	20	London	p3	400
s1	20	London	p4	200
s1	20	London	p5	100
s1	20	London	p6	100
s2	10	Paris	p1	300
s2	10	Paris	p2	400
s3	10	Paris	p2	200
s4	20	London	p2	200
s4	20	London	p4	300
s4	20	London	p5	400

Figure 21. Table in 1NF

Although the table FIRST is in 1NF it contains redundant data. For example, information about the supplier's location and the location's status have to be repeated for every part supplied. Redundancy causes

what are called update anomalies. Update anomalies are problems that arise when information is inserted, deleted, or updated. For example, the following anomalies could occur in FIRST:

- INSERT. The fact that a certain supplier (s5) is located in a particular city (Athens) cannot be added until they supplied a part.
- DELETE. If a row is deleted, then not only is the information about quantity and part lost but also information about the supplier.
- UPDATE. If supplier s1 moved from London to New York, then six rows would have to be updated with this new information.

16.6 Second Normal Form

The definition of second normal form states that only tables with composite primary keys can be in 1NF but not in 2NF.

A relational table is in second normal form 2NF if it is in 1NF and every non-key column is fully dependent upon the primary key.

That is, every non-key column must be dependent upon the entire primary key. FIRST is in 1NF but not in 2NF because status and city are functionally dependent upon only on the column s# of the composite key (s#, p#). This can be illustrated by listing the functional dependencies in the table:

s# —> city, status

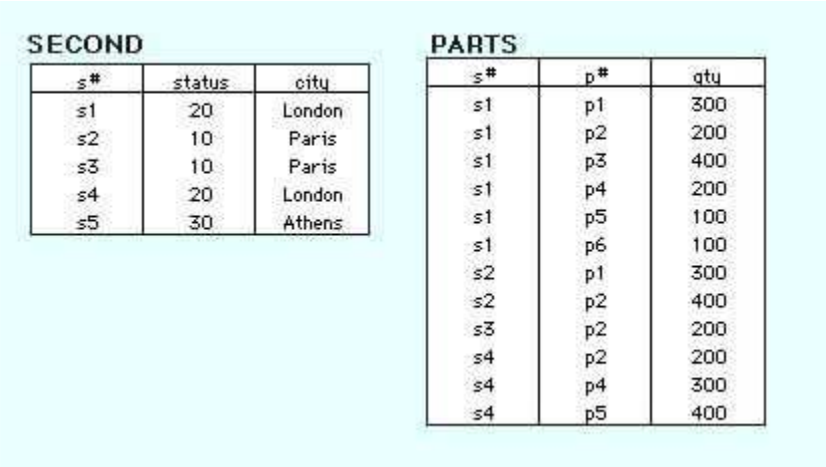
city —> status

(s#,p#) —>qty

The process for transforming a 1NF table to 2NF is:

1. Identify any determinants other than the composite key, and the columns they determine.
2. Create and name a new table for each determinant and the unique columns it determines.
3. Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
4. Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
5. The original table may be renamed to maintain semantic meaning.

To transform FIRST into 2NF we move the columns s#, status, and city to a new table called SECOND. The column s# becomes the primary key of this new table. The results are shown below in Figure 22.



SECOND

s#	status	city
s1	20	London
s2	10	Paris
s3	10	Paris
s4	20	London
s5	30	Athens

PARTS

s#	p#	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

Figure 22. Tables in 2NF

Tables in 2NF but not in 3NF still contain modification anomalies. In the example of SECOND, they are:
INSERT. The fact that a particular city has a certain status (Rome has a status of 50) cannot be inserted until there is a supplier in the city.
DELETE. Deleting any row in SUPPLIER destroys the status information about the city as well as the association between supplier and city.

16.7 Third Normal Form

The third normal form requires that all columns in a relational table are dependent only upon the primary key. A more formal definition is:

A relational table is in third normal form (3NF) if it is already in 2NF and every non-key column is non transitively dependent upon its primary key. In other words, all nonkey attributes are functionally dependent only upon the primary key.

Table PARTS is already in 3NF. The non-key column, qty, is fully dependent upon the primary key (s#, p#). SUPPLIER is in 2NF but not in 3NF because it contains a transitive dependency. A transitive dependency is occurs when a non-key column that is a determinant of the primary key is the determinate of other columns. The concept of a transitive dependency can be illustrated by showing the functional dependencies in SUPPLIER:

SUPPLIER.s# —> SUPPLIER.status
SUPPLIER.s# —> SUPPLIER.city
SUPPLIER.city —> SUPPLIER.status

Note that SUPPLIER.status is determined both by the primary key s# and the non-key column city. The process of transforming a table into 3NF is:

1. Identify any determinants, other the primary key, and the columns they determine.
2. Create and name a new table for each determinant and the unique columns it determines.
3. Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
4. Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
5. The original table may be renamed to maintain semantic meaning.

To transform SUPPLIER into 3NF, we create a new table called CITY_STATUS and move the columns city and status into it. Status is deleted from the original table, city is left behind to serve as a foreign key to CITY_STATUS, and the original table is renamed to SUPPLIER_CITY to reflect its semantic meaning. The results are shown in
Figure 23 below.

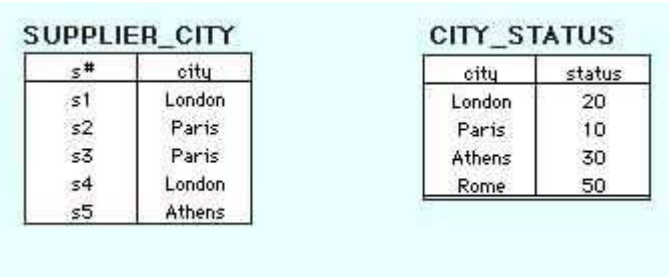


Figure 23. Tables in 3NF

The results of putting the original table into 3NF have created three tables. These can be represented in "psuedo-SQL" as:

PARTS (#s, p#, qty)
Primary Key (s#,#p)
Foreign Key (s#) references SUPPLIER_CITY.s#
SUPPLIER_CITY(s#, city)
Primary Key (s#)
Foreign Key (city) references CITY_STATUS.city
CITY_STATUS (city, status)
Primary Key (city)

17 Advantages of Third Normal Form

The advantage of having relational tables in 3NF is that it eliminates redundant data which in turn saves space and reduces manipulation anomalies. For example, the improvements to our sample database are:

INSERT. Facts about the status of a city, Rome has a status of 50, can be added even though there is not supplier in that city. Likewise, facts about new suppliers can be added even though they have not yet supplied parts.

DELETE. Information about parts supplied can be deleted without destroying information about a supplier or a city. UPDATE. Changing the location of a supplier or the status of a city requires modifying only one row.

17.1 Advanced Normalization

After 3NF, all normalization problems involve only tables which have three or more columns and all the columns are keys. Many practitioners argue that placing entities in 3NF is generally sufficient because it is rare that entities that are in 3NF are not also in 4NF and 5NF. They further argue that the benefits gained from transforming entities into 4NF and 5NF are so slight that it is not worth the effort. However, advanced normal forms are presented because there are cases where they are required.

17.2 Boyce-Codd Normal Form

Boyce-Codd normal form (BCNF) is a more rigorous version of the 3NF deal with relational tables that had (a) multiple candidate keys, (b) composite candidate keys, and (c) candidate keys that overlapped. BCNF is based on the concept of determinants. A determinant column is one on which some of the columns are fully functionally dependent.

A relational table is in BCNF if and only if every determinant is a candidate key.

17.3 Fourth Normal Form

A relational table is in the fourth normal form (4NF) if it is in BCNF and all multivalued dependencies are also functional dependencies.

Fourth normal form (4NF) is based on the concept of multivalued dependencies (MVD). A Multivalued dependency occurs when in a relational table containing at least three columns, one column has multiple rows whose values match a value of a single row of one of the other columns. A more formal definition given by Date is:

given a relational table R with columns A, B, and C then

R.A —>> R.B (column A multidetermines column B)

is true if and only if the set of B-values matching a given pair of A-values and C-values in R depends only on the A-value and is independent of the C-value.

MVD always occur in pairs. That is R.A —>> R.B holds if and only if R.A —>> R.C also holds.

Suppose that employees can be assigned to multiple projects. Also suppose that employees can have multiple job skills. If we record this information in a single table, all three attributes must be used as the key since no single attribute can uniquely identify an instance.

The relationship between emp# and prj# is a multivalued dependency because for each pair of emp#/skill values in the table, the associated set of prj# values is determined only by emp# and is independent of skill. The relationship between emp# and skill is also a multivalued dependency, since the set of Skill values for an emp#/prj# pair is always dependent upon emp# only.

To transform a table with multivalued dependencies into the 4NF move each MVD pair to a new table. The result is shown in Figure 24.

Figure 24. Tables in 4NF

17.4 Fifth Normal Form

A table is in the fifth normal form (5NF) if it cannot have a lossless decomposition into any number of smaller tables.

While the first four normal forms are based on the concept of functional dependence, the fifth normal form is based on the concept of join dependence. Join dependency means that an table, after it has been decomposed into three or more smaller tables, must be capable of being joined again on common keys to form the original table. Stated another way, 5NF indicates when an entity cannot be further decomposed. 5NF is complex and not intuitive. Most experts agree that tables that are in the 4NF are also in 5NF except for "pathological" cases. Teorey suggests that true many-to-many-to-many ternary relations are one such case.

Adding an instance to an table that is not in 5NF creates spurious results when the tables are decomposed and then rejoined. For example, let's suppose that we have an employee who uses design skills on one project and programming skills on another. This information is shown below.

emp#	prj#	skill
1211	11	Design
1211	28	Program

Next we add an employee (1544) who uses programming skills on Project 11.

emp#	prj#	skill
1211	11	Design
1211	28	Program
1544	11	Program

Next, we project this information into three tables as we did above. However, when we rejoin the tables, the recombined table contains spurious results.

emp#	prj#	skill	
1211	11	Design	
1211	11	Program	<<—spurious data
1211	28	Program	
1544	11	Design	<<—spurious data
1544	11	Program	

By adding one new instance to a table not in 5NF, two false assertions were stated:

Assertion 1

- Employee 1211 has been assigned to Project 11.
- Project 11 requires programming skills.
- Therefore, Employee 1211 must use programming skills while assigned to Project 11.

Assertion 2

- Employee 1544 has been assigned to project 11.
- Project 11 needs Design skills.
- Therefore, Employee 1544 must use Design skills in Project 11.

18 Getting Started with MSAccess Database

18.1 A Few Terms

These words are used often in Access so you will want to become familiar with them before using the program and this tutorial.

- A database is a collection of related information.
- An object is a competition in the database such as a table, query, form, or macro.
- A table is a grouping of related data organized in fields (columns) and records (rows) on a datasheet. By using a common field in two tables, the data can be combined. Many tables can be stored in a single database.
- A field is a column on a datasheet and defines a data type for a set of values in a table. For a mailing list table might include fields for first name, last name, address, city, state, zip code, and telephone number.
- A record in a row on a datasheet and is a set of values defined by fields. In a mailing list table, each record would contain the data for one person as specified by the intersecting fields.
- Design View provides the tools for creating fields in a table.
- Datasheet View allows you to update, edit, and delete in formation from a table.

After opening Access, you will be presented with the window shown below. Select one of the first two options if you are creating a new database, or the third if you want to edit an existing database. All three choices are explained in detail below.

Blank Access database

- Unlike Word documents, Excel worksheets, and Power Point presentations, you must save an Access database before you start working on it. After selecting "Blank Access database", you will first be prompted to specify a location and name for the database.

- Find the folder where the database should reside in the Save in drop-down menu.
- Type the name of the database in the File name line and click the Create button.

Access database wizards, pages, and projects

Access' wizards and layout are existing database structures that only need data input. Select a database type and click OK. Name the database on the next screen.

Open an existing database

If the database was opened recently on the computer, it will be listed on the main window. Highlight the database name and click OK. Otherwise, highlight "More Files..." in the list and click OK. From the subsequent window, click the "Look In:" drop-down menu to find the folder where the database is located, highlight the database name in the listing and click OK.

Converting to Access 2000

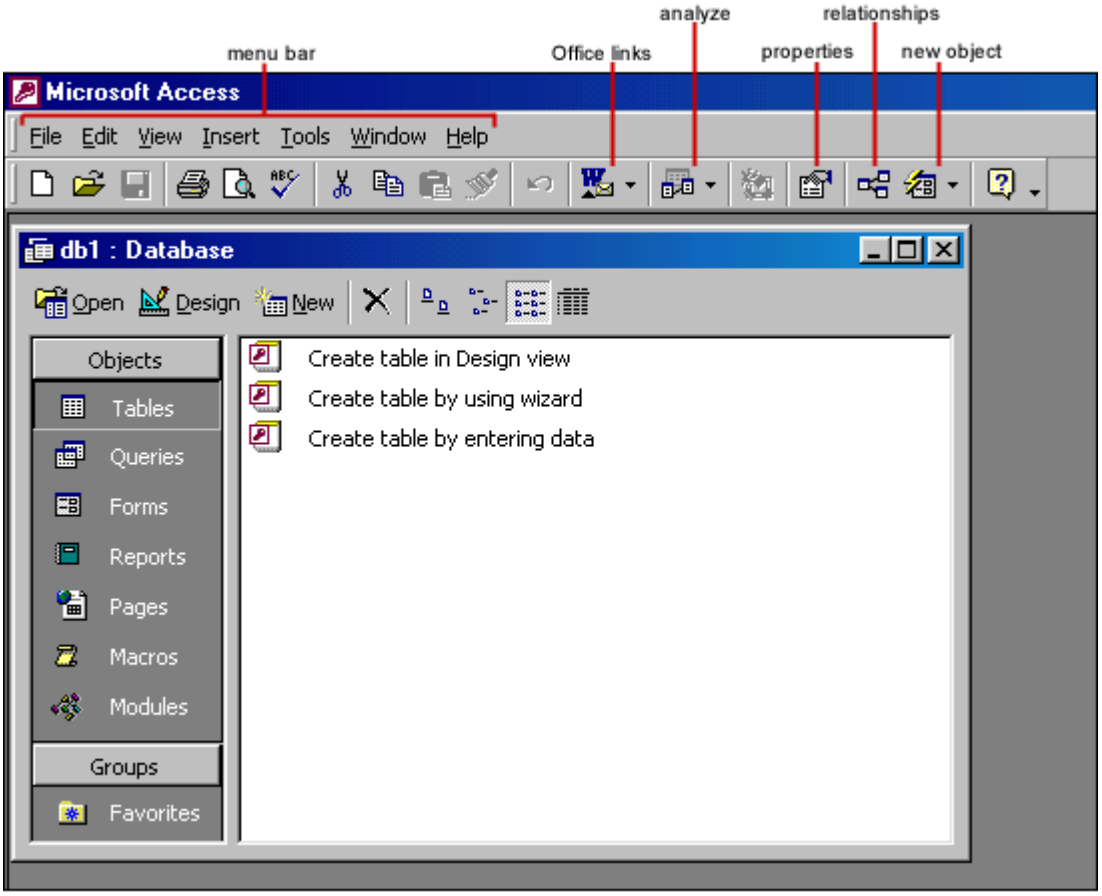
Before opening an existing file that was created in a previous version of Access, it must first be converted to Access 2000 format. Convert a database by following these steps:

- Open Access and select Tools|Database Utilities|Convert Database|To Current Access Database Version from the menu bar.
- Select the database that should be converted and click the Convert button.
- The new version will be a completely separate database and the old one will remain intact so you must then name the new version of the database.

18.2 Screen Layouts

Database Window

The Database Window organizes all of the objects in the database. The default tables listing provides links for creating tables and will list all of the tables in the database when they have been added (Fig. 25).

Design 

View

Figure 25. Design View customizes the fields in the database so that data can be entered.

Datasheet View

The datasheet allows you to enter data into the database.

19 Creating Tables

19.1 Introduction to Tables

Tables are grids that store information in a database similar to the way an Excel worksheet stores information in a workbook. Access provides three ways to create a table for which there are icons in the Database Window. Double-click on the icons to create a table.

- Create table in Design view will allow you to create the fields of the table. This is the most common way of creating a table and is explained in detail below.
- Create table using wizard will step you through the creation of a table.
- Create table by entering data will give you a blank datasheet with unlabelled columns that looks much like an Excel worksheet. Enter data into the cells and click the Save button. You will be prompted to add a primary key field. After the table is saved, the empty cells of the datasheet are trimmed. The fields are given generic names such as "Field1", "Field2", etc. To rename them with more descriptive titles that reflect the content of the fields, select Format|Rename Column from the menu bar or highlight the column, right-click on it with the mouse, and select Rename Column from the shortcut menu.

Create a Table in Design View

Design View will allow you to define the fields in the table before adding any data to the datasheet. The window is divided into two parts: a top pane for entering the field name, data type, and an option description of the field, and a bottom pane for specifying field properties.

- Field Name - This is the name of the field and should represent the contents of the field such as "Name", "Address", "Final Grade", etc. The name can not exceed 64 characters in length and may include spaces.
- Data Type is the type of value that will be entered into the fields.
  - Text - The default type, text type allows any combination of letters and numbers up to a maximum of 255 characters per field record.
  - Memo - A text type that stores up to 64,000 characters.
  - Number - Any number can be stored.
  - Date/Time - A date, time, or combination of both.
  - Currency - Monetary values that can be set up to automatically include a dollar sign ($) and correct decimal and comma positions.
  - AutoNumber - When a new record is created, Access will automatically assign a unique integer to the record in this field. From the General options, select Increment if the numbers should be assigned in order or random if any random number should be chosen. Since every record in a datasheet must include at least one field that distinguishes it from all others, this is a useful data type to use if the existing data will not produce such values.
  - Yes/No - Use this option for True/False, Yes/No, On/Off, or other values that must be only one of two.
  - OLE Object - An OLE (Object Linking and Embedding) object is a sound, picture, or other object such as a Word document or Excel spreadsheet that is created in another program. Use this data type to embed an OLE object or link to the object in the database.
  - Hyperlink - A hyperlink will link to an Internet or Intranet site, or another location in the database. The data consists of up to four parts each separated by the pound sign (#):

DisplayText#Address#SubAddress#ScreenTip. The Address is the only required part of the string. Examples:

```
Internet hyperlink example:     FGCU Home Page#http://www.fgcu.edu#
Database link example:          #c:\My Documents\database.mdb#MyTable
```

- Description (optional) - Enter a brief description of what the contents of the field are.
- Field Properties - Select any pertinent properties for the field from the bottom pane.

Field Properties

Properties for each field are set from the bottom pane of the Design View window.

- Field Size is used to set the number of characters needed in a text or number field. The default field size for the text type is 50 characters. If the records in the field will only have two or three characters, you can change the size of the field to save disk space or prevent entry errors by limiting the number of characters allowed. Likewise, if the field will require more than 50 characters, enter a number up to 255. The field size is set in exact characters for Text type, but options are give for numbers:
  - Byte - Positive integers between 1 and 255
  - Integer - Positive and negative integers between -32,768 and 32,768
  - Long Integer (default) - Larger positive and negative integers between -2 billion and 2 billion.
  - Single - Single-precision floating-point number
  - Double - Double-precision floating-point number
  - Decimal - Allows for Precision and Scale property control
- Format conforms the data in the field to the same format when it is entered into the datasheet. For text and memo fields, this property has two parts that are separated by a semicolon. The first part of the property is used to apply to the field and the second applies to empty fields.

Text and memo format.

Text Format			
Format	Datasheet Entry	Display	Explanation
@@@-@@@@	1234567	123-4567	@ indicates a required character or space
@@@-@@@&	123456	123-456	& indicates an optional character or space
<	HELLO	hello	< converts characters to lowercase
>	hello	HELLO	> converts characters to uppercase
@\!	Hello	Hello!	\ adds characters to the end
@;"No Data Entered"	Hello	Hello	
@;"No Data Entered"	(blank)	No Data Entered	

- Number format. Select one of the preset options from the drop down menu or construct a custom format using symbols explained below:

Number Format			
Format	Datasheet Entry	Display	Explanation
###,##0.00	123456.78	123,456.78	0 is a placeholder that displays a digit or 0 if there is none. # is a placeholder that displays a digit or nothing if there is none.
$###,##0.00	0	$0.00	
###.00%	.123	12.3%	% multiplies the number by 100 and added a percent sign

- Currency format. This formatting consists of four parts separated by semicolons: format for positive numbers; format for negative numbers; format for zero values; format for Null values.

Currency Format	
Format	Explanation
$##0.00;($##0.00)[Red];$0.00;"none"	Positive values will be normal currency format, negative numbers will be red in parentheses, zero is entered for zero values, and “none“ will be written for Null values.

- Date format. In the table below, the value “1/1/01“ is entered into the datasheet, and the following values are displayed as a result of the different assigned formats.

Date Format		
Format	Display	Explanation
dddd","mmmm d","yyyy	Monday, January 1, 2001	dddd, mmmm, and yyyy print the full day name, month name, and year
ddd","mmm ". " d", '"yy	Mon, Jan. 1, '01	ddd, mmm, and yy print the first three day letters, first three month letters, and last two year digits
"Today is " dddd	Today is Monday	
h:n:s: AM/PM	12:00:00 AM	“n“ is used for minutes to avoid confusion with months

- Yes/No fields are displayed as check boxes by default on the datasheet. To change the formatting of these fields, first click the Lookup tab and change the Display Control to a text box. Go back to the General tab choices to make formatting changes. The formatting is designated in three sections separated by semicolons. The first section does not contain anything but the semicolon must be included. The second section specifies formatting for Yes values and the third for No values.

Yes/No Format	
Format	Explanation
;"Yes"[green];"No"[red]	Prints “Yes“ in green or “No“ in red

- Default Value - There may be cases where the value of a field will usually be the same for all records. In this case, a changeable default value can be set to prevent typing the same thing numerous times. Set the Default Value property.

Primary Key

Every record in a table must have a primary key that differentiates it from every other record in the table. In some cases, it is only necessary to designate an existing field as the primary key if you are certain that every record in the table will have a different value for that particular field. A social security number is an example of a record whose values will only appear once in a database table.

Designate the primary key field by right-clicking on the record and selection Primary Key from the shortcut menu or select Edit|Primary Key from the menu bar. The primary key field will be noted with a key image to the left. To remove a primary key, repeat one of these steps.

If none of the existing fields in the table will produce unique values for every record, a separate field must be added. Access will prompt you to create this type of field at the beginning of the table the first time you save the table and a primary key field has not been assigned. The field is named "ID" and the data type is "autonumber". Since this extra field serves no purpose to you as the user, the autonumber type automatically updates whenever a record is added so there is no extra work on your part. You may also choose to hide this column in the datasheet as explained on a later page in this tutorial.

Indexes

Creating indexes allows Access to query and sort records faster. To set an indexed field, select a field that is commonly searched and change the Indexed property to Yes (Duplicates OK) if multiple entries of the same data value are allowed or Yes (No Duplicates) to prevent duplicates.

Field Validation Rules

Validation Rules specify requirements (change word) for the data entered in the worksheet. A customized message can be displayed to the user when data that violates the rule setting is entered. Click the expression builder ("...") button at the end of the Validation Rule box to write the validation rule. Examples of field validation rules include <> 0 to not allow zero values in the record, and ??? to only all data strings three characters in length.

Input Masks

An input mask controls the value of a record and sets it in a specific format. They are similar to the Format property, but instead display the format on the datasheet before the data is entered. For example, a telephone number field can formatted with an input mask to accept ten digits that are automatically formatted as "(555) 123-4567". The blank field would look like (___) ___-____. An an input mask to a field by following these steps:

- In design view, place the cursor in the field that the input mask will be applied to.
- Click in the white space following Input Mask under the General tab.
- Click the "..." button to use the wizard or enter the mask, (@@@) @@@-@@@@, into the field provided. The following symbols can be used to create an input mask from scratch:

Input Mask Symbols	
Symbol	Explanation
A	Letter or digit
0	A digit 0 through 9 without a + or - sign and with blanks displayed as zeros
9	Same as 0 with blanks displayed as spaces
#	Same as 9 with +/- signs
?	Letter
L	Letter A through Z
C or &	Character or space
<	Convert letters to lower case
>	Convert letters to upper case

19.2 Datasheet Records

Adding Records

Add new records to the table in datasheet view by typing in the record beside the asterisk (*) that marks the new record. You can also click the new record button at the bottom of the datasheet to skip to the last empty record.

Editing Records

To edit records, simply place the cursor in the record that is to be edited and make the necessary changes. Use the arrow keys to move through the record grid. The previous, next, first, and last record buttons at the bottom of the datasheet are helpful in maneuvering through the datasheet.

Deleting Records

Delete a record on a datasheet by placing the cursor in any field of the record row and select Edit|Delete Record from the menu bar or click the Delete Record button on the datasheet toolbar.

Adding and Deleting Columns

Although it is best to add new fields (displayed as columns in the datasheet) in design view because more options are available, they can also be quickly added in datasheet view. Highlight the column that the new column should appear to the left of by clicking its label at the top of the datasheet and select Insert|Column from the menu bar.

Entire columns can be deleted by placing the cursor in the column and selecting Edit|Delete Column from the menu bar.

Resizing Rows and Columns

The height of rows on a datasheet can be changed by dragging the gray sizing line between row labels up and down with the mouse. By changing the height on one row, the height of all rows in the datasheet will be changed to the new value.

Column width can be changed in a similar way by dragging the sizing line between columns. Double click on the line to have the column automatically fit to the longest value of the column. Unlike rows, columns on a datasheet can be different widths. More exact values can be assigned by selecting FormatiRow Height or FormatiColumn Width from the menu bar.

Freezing Columns

Similar to freezing panes in Excel, columns on an Access table can be frozen. This is helpful if the datasheet has many columns and relevant data would otherwise not appear on the screen at the same time. Freeze a column by placing the cursor in any record in the column and select FormatiFreeze Columns from the menu bar. Select the same option to unfreeze a single column or select FormatiUnfreeze All Columns.

Hiding Columns

Columns can also be hidden from view on the datasheet although they will not be deleted from the database. To hide a column, place the cursor in any record in the column or highlight multiple adjacent columns by clicking and dragging the mouse along the column headers, and select FormatiHide Columns from the menu bar.

To show columns that have been hidden, select FormatiUnhide Columns from the menu bar. A window displaying all of the fields in the table will be listed with check boxes beside each field name. Check the boxes beside all fields that should be visible on the data table and click the Close button.

Finding Data in a Table

Data in a datasheet can be quickly located by using the Find command.

- Open the table in datasheet view.
- Place the cursor in any record in the field that you want to search and select EditiFind... from the menu bar.
- Enter the value criteria in the Find What: box.
- From the Look In: drop-down menu, define the area of the search by selecting the entire table or just the field in the table you placed your cursor in during step 2.
- Select the matching criteria from Match: to and click the More >> button for additional search parameters.
- When all of the search criteria is set, click the Find Next button. If more than one record meets the criteria, keep clicking Find Next until you reach the correct record.

Replace

The replace function allows you to quickly replace a single occurrence of data with a new value or to replace all occurrences in the entire table.

- Select EditiReplace... from the menu bar (or click the Replace tab if the Find window is already open).
- Follow the steps described in the Find procedure for searching for the data that should be replaced and type the new value of the data in the Replace With: box.
- Click the Find Next button to step through occurrences of the data in the table and click the Replace button to make single replacements. Click Replace All to change all occurrences of the data in one step.

Check Spelling and AutoCorrect

The spell checker can be used to flag spelling errors in text and menu fields in a datasheet. Select ToolsiSpelling from the menu bar to activate the spell checker and make corrections just as you would using Word or Excel. The AutoCorrect feature can automatically correct common spelling errors such as two INitial CApitals, capitalizing the first letter of the first word of a sentence, and anything you define. Select ToolsiAutoCorrect to set these features.

Print a Datasheet

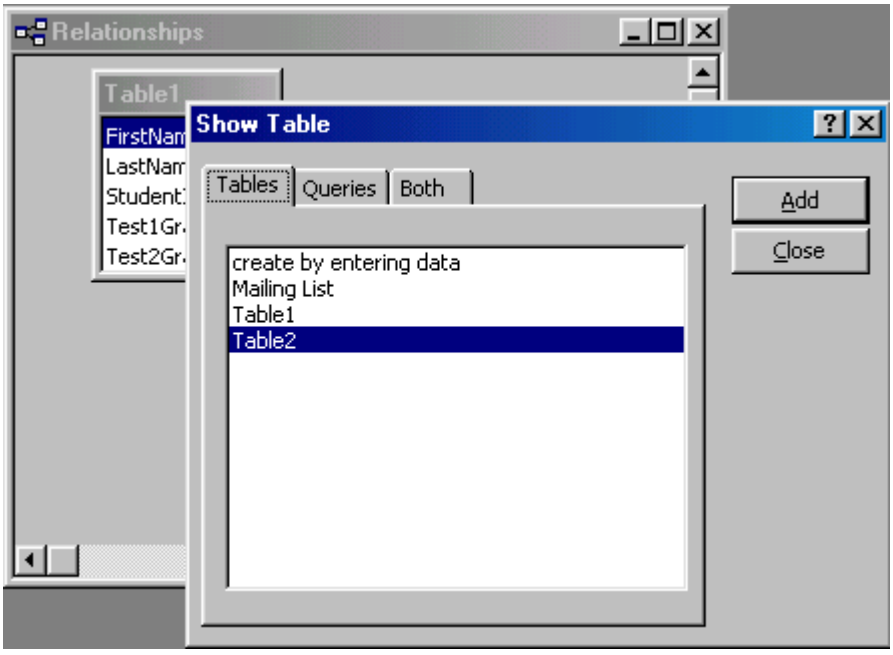
Datasheets can be printed by clicking the Print button on the toolbar or select FileiPrint to set more printing options.

19.3 Table Relationships

Table Relationships

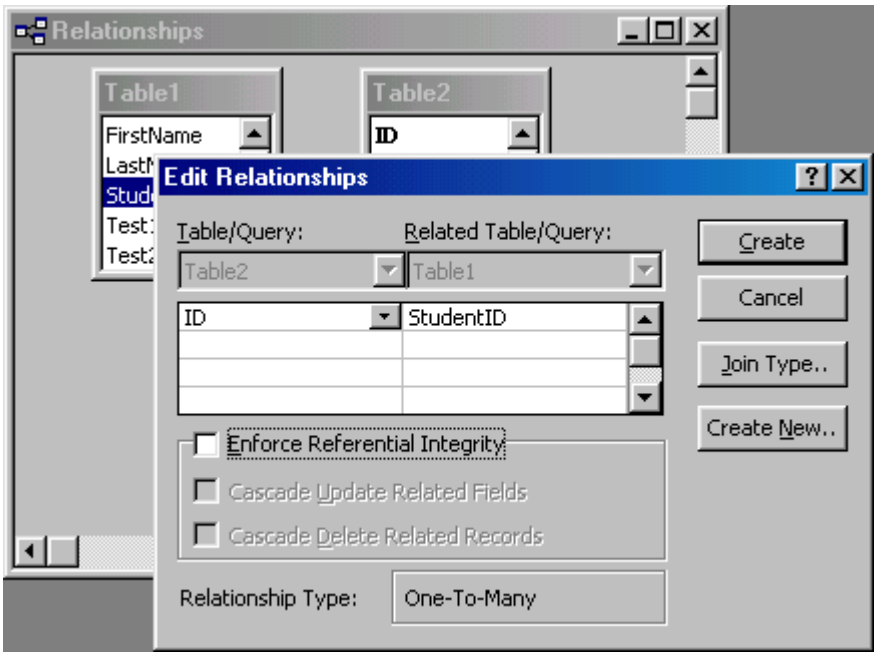
To prevent the duplication of information in a database by repeating fields in more than one table, table relationships can be established to link fields of tables together. Follow the steps below to set up a relational database:

- Click the Relationships button on the toolbar.
- From the Show Table window (click the Show Table button on the toolbar to make it appear), double click on the names of the tables you would like to include in the relationships. When you have finished adding tables, click Close.

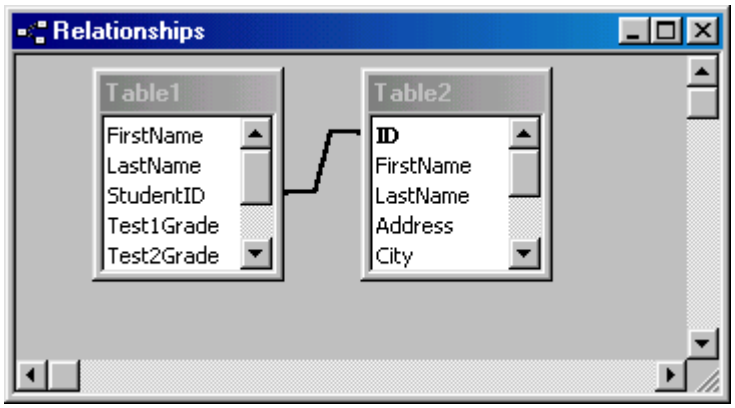


- To link fields in two different tables, click and drag a field from one table to the corresponding field on the other table and release the mouse button. The Edit Relationships window will appear. From this window, select different fields if necessary and select an option from Enforce Referential Integrity if necessary. These options give Access permission to automatically make

changes to referential tables if key records in one of the tales is deleted. Check the Enforce Referential Integrity box to ensure that the relationships are valid and that the data is not accidentally deleted when data is added, edited, or deleted. Click Create to create the link.



- A line now connects the two fields in the Relationships window.



- The datasheet of a relational table will provide expand and collapse indicators to view subdatasheets containing matching information from the other table. In the example below, the student address database and student grade database were related and the two can be shown simultaneously using the expand feature. To expand or collapse all subdatasheets at once, select FormatiSubdatasheetiExpand All or Collapse All from the toolbar.

19.4 Sorting and Filtering

Sorting and filtering allow you to view records in a table in a different way either by reordering all of the records in the table or view only those records in a table that meet certain criteria that you specify.

Sorting

You may want to view the records in a table in a different order than they appear such as sorting by a date or in alphabetical order, for example. Follow these steps to execute a simple sort of records in a table based on the values of one field:

- In table view, place the cursor in the column that you want to sort by.
- Select Records|Sort|Sort Ascending or Records|Sort|Sort Descending from the menu bar or click the Sort Ascending or Sort Descending buttons on the toolbar.

To sort by more than one column (such as sorting by date and then sorting records with the same date alphabetically), highlight the columns by clicking and dragging the mouse over the field labels and select one of the sort methods stated above.

Filter by Selection

This feature will filter records that contain identical data values in a given field such as filtering out all of the records that have the value "Smith" in a name field. To Filter by Selection, place the cursor in the field that you want to filter the other records by and click the Filter by Selection button on the toolbar or select Records|Filter|Filter By Selection from the menu bar. In the example below, the cursor is placed in the City field of the second record that displays the value "Ft. Myers" so the filtered table will show only the records where the city is Ft. Myers.

Filter by Form

If the table is large, it may be difficult to find the record that contains the value you would like to filter by so using Filter by Form may be advantageous instead. This method creates a blank version of the table with drop-down menus for each field that each contain the values found in the records of that field. Under the default Look for tab of the Filter by Form window, click in the field to enter the filter criteria. To specify an alternate criteria if records may contain one of two specified values, click the Or tab at the bottom of the window and select another criteria from the drop-down menu. More Or tabs will appear after one criteria is set to allow you to add more alternate criteria for the filter. After you have selected all of the criteria you want to filter, click the Apply Filter button on the toolbar.

The following methods can be used to select records based on the record selected by that do not have exactly the same value. Type these formats into the field where the drop-down menu appears instead of selecting an absolute value.

Filter by Form	
Format	Explanation
Like "*Street"	Selects all records that end with "Street"
<="G"	Selects all records that begin with the letters A through G
>1/1/00	Selects all dates since 1/1/00
<> 0	Selects all records not equal to zero

Saving A Filter

The filtered contents of a table can be saved as a query by selecting File|Save As Query from the menu bar. Enter a name for the query and click OK. The query is now saved within the database.

Remove a Filter
To view all records in a table again, click the depressed Apply Filter toggle button on the toolbar.

20 Queries

20.1 Introduction to Queries

Queries select records from one or more tables in a database so they can be viewed, analyzed, and sorted on a common datasheet. The resulting collection of records, called a dynaset (short for dynamic subset), is saved as a database object and can therefore be easily used in the future. The query will be updated whenever the original tables are updated. Types of queries are select queries that extract data from tables based on specified values, find duplicate queries that display records with duplicate values for one or more of the specified fields, and find unmatched queries display records from one table that do not have corresponding values in a second table.

20.2 Create a Query in Design View

Follow these steps to create a new query in Design View:

- From the Queries page on the Database Window, click the New button.
- Select Design View and click OK.
- Select tables and existing queries from the Tables and Queries tabs and click the Add button to add each one to the new query.
- Click Close when all of the tables and queries have been selected.
- Add fields from the tables to the new query by double-clicking the field name in the table boxes or selecting the field from the Field: and Table: drop-down menus on the query form. Specify sort orders if necessary.
- Enter the criteria for the query in the Criteria: field. The following table provides examples for some of the wildcard symbols and arithmetic operators that may be used. The Expression Builder can also be used to assist in writing the expressions.

Query Wildcards and Expression Operators	
Wildcard / Operator	Explanation
? Street	The question mark is a wildcard that takes the place of a single letter.
43th *	The asterisk is the wildcard that represents a number of characters.
<100	Value less than 100
>=1	Value greater than or equal to 1
<>"FL"	Not equal to (all states besides Florida)
Between 1 and 10	Numbers between 1 and 10
Is Null Is Not Null	Finds records with no value or all records that have a value
Like "a*"	All words beginning with “a“
>0 And <=10	All numbers greater than 0 and less than 10
"Bob" Or "Jane"	Values are Bob or Jane

- After you have selected all of the fields and tables, click the Run button on the toolbar.
- Save the query by clicking the Save button.

20.3 Query Wizard

Access' Query Wizard will easily assist you to begin creating a select query.

- Click the Create query by using wizard icon in the database window to have Access step you through the process of creating a query.
- From the first window, select fields that will be included in the query by first selecting the table from the drop-down Tables/Queries menu. Select the fields by clicking the > button to move the field from the Available Fields list to Selected Fields. Click the double arrow button >> to move all of the fields to Selected Fields. Select another table or query to choose from more fields and repeat the process of moving them to the Selected Fields box. Click Next > when all of the fields have been selected.
- On the next window, enter the name for the query and click Finish.
- Refer to steps 5-8 of the previous tutorial to add more parameters to the query.

20.4 Find Duplicates Query

This query will filter out records in a single table that contain duplicate values in a field.

- Click the New button on the Queries database window, select Find Duplicates Query Wizard from the New Query window and click OK.
- Select the table or query that the find duplicates query will be applied to from the list provided and click Next >.
  Select the fields that may contain duplicate values by highlighting the names in the Available fields list and clicking the > button to individually move the fields to the Duplicate-value fields list or >> to move all of the fields. Click Next > when all fields have been selected.
- Select the fields that should appear in the new query along with the fields selected on the previous screen and click Next >
- Name the new query and click Finish.

20.5 Delete a Query

To delete a table from the query, click the table's title bar and press the Delete key on the keyboard.

21 Introduction to Expressions

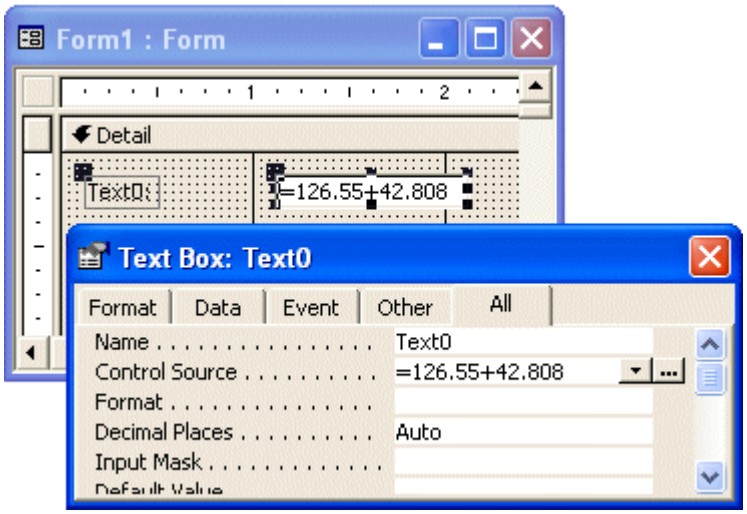
21.1 Overview of Expressions

An expression is a combination of data fields, operators, values, and/or procedures destined to produce a new value. There are various types of expressions you will be using in your applications. When creating an expression, you would ask the database engine to supply data of a specific field following your recommendation. Data you specify is usually not provided by a field on a table. Instead, you can create a field that is a combination of fields from a table, a form, a query, or a report. The data of the expression can also be the result of a combination of dependent fields or values external to any table, form, query, or report.

There are two main things you will use to create your expressions: operators and functions. As we saw in the past, an operator is a symbol (or a character) that is applied to one or more items to produce a new value. As we will learn later on, a function is an assignment that accomplishes a specific and isolated job then gives back a result.

21.2 Algebraic Expressions

To create an expression, you use any combination of the operators we have used so far. Once the expression is ready, you can assign it to a field on a form or report. To do that, while in Design View, you can access the Control Source of the field in its Properties window, type the assignment operator "=", followed by the expression. For example, imagine you want to create an expression as 126.55 + 42.808. To display the result of this expression in the text box of a form, in the Control Source of the text box, you can type = 126.55+42.808 and press Enter



When such a form displays in Form View, the field that holds the expression would display the result (provided it can successfully get the result). Later on, we will see other techniques of creating an expression.

So far, we have seen that, data provided to a form is controlled by the Data Source property. Therefore, when creating an expression for a form's field, you must supply the expression to the Data Source field of the Properties window. Probably the simplest expression you can create is by transferring the value from an existing field to another field. Imagine you have a field from a form and the field is named MailingAddress. If you want the value of the MailingAddress field to be transferred or assigned to a field named ShippingAddress, in the Control Source of the ShippingAddress field, you would type =MailingAddress. You can also include the name of a field in square brackets as we saw when studying

operator. Instead of MailingAddress, it would consider it as [MailingAddress]. If you were assigning a field to another, you would write the assignment as =[MailingAddress].

Suppose you have a field named HourlySalary and another field named WeeklyHours. If you want to calculate the weekly salary of an employee, you would multiply the HourlySalary by the WeeklyHours and assign the result to another field. In the Control Source property of the resulting field, you would type =[HourlySalary]*[WeeklyHours]

22 Windows Controls and Expressions

A person's interaction with the computer is mostly done using Windows controls as these objects hold or present various pieces of information that the person can use. Therefore, Windows controls are highly involved in expressions. The expressions we have used so far were fairly straightforward and did not need any complex algorithm to accomplish their purpose. Unfortunately, as a database developer, you will face various scenarios, some of them might range from difficult to impossible. There are two main tools you need to build these sorts of expressions. First, you should know what is available so you can wisely play with Microsoft Access. Second, you will make use of your ability to logically think and process information.

22.1 Text-Based Controls

A string is an empty space, a character, or a group of characters. Because the character or group of characters that constitute a string is considered "as is", a string must be included in double-quotes whenever it is involved in any expression. Examples of strings are "g", "Gabriel", "Congo". In reality, these examples of strings are referred to as values. The value of a string is usually stored in the name of a field. For example, the last names of employees, such as "Pierre", "Harvey", "Charles", "Hermine", can be commonly stored in a field named FirstName. The ability to store a category of string in a named field allows such a field to be used in an expression, which then would produce a standard result for all records involved in that field. Based on this, you can use operators on names of fields instead of on values of those fields.

The most classic operation you will perform on strings consists of adding strings such as a first name and a last name in order to produce a new string that could represent a full name. Adding two strings is also referred to as concatenating them.

To concatenate two or more strings, you can use the addition operator "+". An example would be FirstName + LastName. This would produce a string as FirstNameLastName. Instead of having the first and last names tied, you may need to include an empty space between them. For this reason, you can add three strings such as FirstName + " " + LastName to get a new string.

Although you can use the addition operator on strings, you might start thinking that any of the other algebraic operations can be used on strings, not at all. The addition operator was especially written (in computer programming, we say that it was overloaded) to be applied to strings. None of the other arithmetic operations (subtraction, multiplication, division, and remainder) can be applied to strings and it would not make sense. For this reason, Microsoft Access (and the (Visual) Basic language) provide an alternate and appropriate operator to add strings.

Besides, or instead of, the addition operator, you can also add strings using the & (called ampersand) operator. To add two strings or fields named FirstName and LastName, you can use the concatenation operator as FirstName & LastName. Like the addition operator, this operation appends the second string to the end or right of the first one. To produce a more readable string, you can add an empty string in the middle. The operation would become FirstName & " " & LastName.

23 The Series-Based Functions

A series or collection-based function is one that considers a particular column and performs an operations on all of its cells. For example, if you have a particular column in which users enter a string, you may want to count the number of strings that have been entered in the cells under that column. In the same way, suppose you have a column under whose cells users most enter numbers. Using a series-based function, you can get the total of the values entered in the cells of that column.

The general syntax of series-based functions is:

FunctionName(Series)

The FunctionName is one of those we will see shortly. Each of these functions takes one argument, which is usually the name of the column whose cells you want to consider the operation.

Sum: To perform the addition on various values of a column, you can use the Sum() function. This function is highly valuable as it helps to perform the sum of values in various transactions.

Count: The Count() function is used to count the number of values entered in the cells of a column.

Average: The Avg() function calculates the sum of values of a series and divides it by the count to get an average.

Minimum: Once a series of values have been entered in cells of a column, to get the lowest value in those cells, you can call the Min() function.

Maximum: As opposed to the Min() function, the Max() function gets the highest value of a series.

24 The Domain-Based Functions

A domain-based function is used to get a value from another object and deliver it to the object in which it is being used or called. The general syntax of these functions is:

FunctionName(WhatValue, FromWhatObject, WhatCriteria)

To perform its operation, a domain-based function needs three pieces of information, two of which are required (the first two arguments) and one is optional (the third argument).
when calling one of these functions, you must specify the value of the column you want to retrieve. This is provided as the WhatValue in our syntax. This argument is passed as a string.

The FromWhatObject is the name of the object that holds the value. It is usually the name of a form. This argument also is passed as a string.

The third argument, WhatCriteria in our syntax, specifies the criterion that will be used to filter the WhatValue value. It follows the normal rules of setting a criterion.
Domain First: If you want to find out what was the first value entered in the cells of a certain column of an external form or report, you can call the DFirst() function.
Domain Last: The DLast() function does the opposite of the DFirst() function: It retrieves the last value entered in a column of a form or report.

Domain Sum: To get the addition of values that are stored in a column of another form or report, you can use the DSum() function.

Domain Count: The DCount() function is used to count the number of values entered in the cells of a column of a table.

Domain Average: The DAvg() function calculates the sum of values of a series and divides it by the count of cells on the same external form or report to get an average.

Domain Minimum: The DMin() function is used to retrieve the minimum value of the cells in a column of an external form or report.

Domain Maximum: As opposed to the DMin() function, the DMax() function gets the highest value of a series of cells in the column of an external form or report.

25 Forms

Forms are used as an alternative way to enter data into a database table.

Create Form by Using Wizard

To create a form using the assistance of the wizard, follow these steps:

- Click the Create form by using wizard option on the database window.
- From the Tables/Queries drop-down menu, select the table or query whose datasheet the form will modify. Then, select the fields that will be included on the form by highlighting each one the Available Fields window and clicking the single right arrow button > to move the field to the Selected Fields window. To move all of the fields to Select Fields, click the double right arrow button >>. If you make a mistake and would like to remove a field or all of the fields from the Selected Fields window, click the left arrow < or left double arrow << buttons. After the proper fields have been selected, click the Next > button to move on to the next screen.
- On the second screen, select the layout of the form.
  - Columnar - A single record is displayed at one time with labels and form fields listed side-by-side in columns
  - Justified - A single record is displayed with labels and form fields are listed across the screen
  - Tabular - Multiple records are listed on the page at a time with fields in columns and records in rows
  - Datasheet - Multiple records are displayed in Datasheet View
  - Click the Next > button to move on to the next screen.
- Select a visual style for the form from the next set of options and click Next >.
- On the final screen, name the form in the space provided. Select “Open the form to view or enter information“ to open the form in Form View or “Modify the form's design“ to open it in Design View. Click Finish to create the form.

Create Form in Design View

To create a form from scratch without the wizard, follow these steps:

- Click the New button on the form database window.
- Select "Design View" and choose the table or query the form will be associated with the form from the drop-down menu.
- Select ViewiToolbox from the menu bar to view the floating toolbar with additional options.
- Add controls to the form by clicking and dragging the field names from the Field List floating window. Access creates a text box for the value and label for the field name when this action is accomplished. To add controls for all of the fields in the Field List, double-click the Field List window's title bar and drag all of the highlighted fields to the form.

Adding Records Using A Form

Input data into the table by filling out the fields of the form. Press the Tab key to move from field to field and create a new record by clicking Tab after the last field of the last record. A new record can also be created at any time by clicking the New Record button at the bottom of the form window. Records are automatically saved as they are entered so no additional manual saving needs to be executed.

Editing Forms

The follow points may be helpful when modifying forms in Design View.

- Grid lines - By default, a series of lines and dots underlay the form in Design View so form elements can be easily aligned. To toggle this feature on and off select ViewiGrid from the menu bar.
- Snap to Grid - Select FormatiSnap to Grid to align form objects with the grid to allow easy alignment of form objects or uncheck this feature to allow objects to float freely between the grid lines and dots.
- Resizing Objects - Form objects can be resized by clicking and dragging the handles on the edges and corners of the element with the mouse.
- Change form object type - To easily change the type of form object without having to create a new one, right click on the object with the mouse and select Change To and select an available object type from the list.
- Label/object alignment - Each form object and its corresponding label are bounded and will move together when either one is moved with the mouse. However, to change the position of the object and label in relation to each other (to move the label closer to a text box, for example), click and drag the large handle at the top, left corner of the object or label.
- Tab order - Alter the tab order of the objects on the form by selecting ViewiTab Order... from the menu bar. Click the gray box before the row you would like to change in the tab order, drag it to a new location, and release the mouse button.
- Form Appearance - Change the background color of the form by clicking the Fill/Back Color button on the formatting toolbar and click one of the color swatches on the palette. Change the color of individual form objects by highlighting one and selecting a color from the Font/Fore Color palette on the formatting toolbar. The font and size, font effect, font alignment, border around each object, the border width, and a special effect can also be modified using the formatting toolbar:

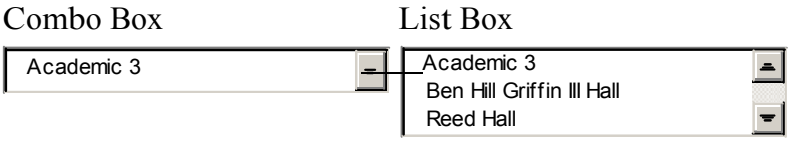
- Page Header and Footer - Headers and footers added to a form will only appear when it is printed. Access these sections by selecting ViewiPage Header/Footer on the menu bar. Page numbers can also be added to these sections by selecting InsertiPage Numbers. A date and time can be added from InsertiDate and Time.... Select ViewiPage Header/Footer again to hide these sections from view in Design View.

Form Controls

This page explains the uses for other types of form controls including lists, combo boxes, checkboxes, option groups, and command buttons.

List and Combo Boxes

If there are small, finite number of values for a certain field on a form, using combo or list boxes may be a quicker and easier way of entering data. These two control types differ in the number of values they display. List values are all displayed while the combo box values are not displayed until the arrow button is clicked to open it as shown in these examples:



By using a combo or list box, the name of the academic building does not need to be typed for every record. Instead, it simply needs to be selected from the list. Follow these steps to add a list or combo box to a form:

- Open the form in Design View.
- Select ViewiToolbox to view the toolbox and make sure the “Control Wizards“ button is pressed in.
- Click the list or combo box tool button and draw the outline on the form. The combo box wizard dialog box will appear.
- Select the source type for the list or combo box values and click Next >.
- Depending on your choice in the first dialog box, the next options will vary. If you chose to look up values from a table or query, the following box will be displayed. Select the table or query from which the values of the combo box will come from. Click Next > and choose fields from the table or query that was selected. Click Next > to proceed.
- On the next dialog box, set the width of the combo box by clicking and dragging the right edge of the column. Click Next >.
- The next dialog box allows tells Access what to do with the value that is selected. Choose “Remember the value for later use“ to use the value in a macro or procedure (the value is discarded when the form is closed), or select the field that the value should be stored in. Click Next > to proceed to the final screen.
- Type the name that will appear on the box’s label and click Finish.

Check Boxes and Option Buttons

Use check boxes and option buttons to display yes/no, true/false, or on/off values. Only one value from a group of option buttons can be selected while any or all values from a check box group can be

chosen. Typically, these controls should be used when five or less options are available. Combo boxes or lists should be used for long lists of options. To add a checkbox or option group:

- Click the Option Group tool on the toolbox and draw the area where the group will be placed on the form with the mouse. The option group wizard dialog box will appear.
- On the first window, enter labels for the options and click the tab key to enter additional labels. Click Next > when finished typing labels.
    - On the next window, select a default value if there is any and click Next >.
- Select values for the options and click Next >.
    - Choose what should be done with the value and click Next >.
- Choose the type and style of the option group and click Next >.
- Type the caption for the option group and click Finish.

Command Buttons

In this example, a command button beside each record is used to open another form.

- Open the form in Design View and ensure that the Control Wizard button on the toolbox is pressed in.
- Click the command button icon on the toolbox and draw the button on the form. The Command Button Wizard will then appear.
    - On the first dialog window, action categories are displayed in the left list while the right list displays the actions in each category. Select an action for the command button and click Next >.
- The next few pages of options will vary based on the action you selected. Continue selecting options for the command button.
    - Choose the appearance of the button by entering caption text or selecting a picture. Check the Show All Pictures box to view the full list of available images. Click Next >.
- Enter a name for the command button and click Finish to create the button.

26 Subforms

What Is A Subform?

A subform is a form that is placed in a parent form, called the main form. Subforms are particularly useful to display data from tables and queries that have one-to-many relationships. For example, in the sample below, data on the main form is drawn from an item information table while the subform contains all of the orders for that item. The item record is the "one" part of this one-to-many relationship while the orders are the "many" side of the relationship since many orders can be placed for the one item.

The remainder of this page explains three methods for creating subforms and they assume that the data tables and/or queries have already been created.

Create a Form and Subform at Once

Use this method if neither form has already been created. A main form and subform can be created automatically using the form wizard if table relationships are set properly or if a query involving multiple tables is selected. For example, a relationship can be set between a table containing customer information and one listing customer orders so the orders for each customer are displayed together using a main form and subform. Follow these steps to create a subform within a form:

- Double-click Create form by using wizard on the database window.

- From the Tables/Queries drop-down menu, select the first table or query from which the main form will display its data. Select the fields that should appear on the form by highlighting the field names in the Available Fields list on the left and clicking the single arrow > button or click the double arrows >> to choose all of the fields.
- From the same window, select another table or query from the Tables/Queries drop-down menu and choose the fields that should appear on the form. Click Next to continue after all fields have been selected.
- Choose an arrangement for the forms by selecting form with subform(s) if the forms should appear on the same page or Linked forms if there are many controls on the main form and a subform will not fit. Click Next to proceed to the next page of options.
- Select a tabular or datasheet layout for the form and click Next.
- Select a style for the form and click Next.
- Enter the names for the main form and subform. Click Finish to create the forms.
- New records can be added to both tables or queries at once by using the new combination form.

Subform Wizard

If the main form or both forms already exist, the Subform Wizard can be used to combine the forms. Follow these steps to use the Subform Wizard:

- Open the main form in Design View and make sure the Control Wizard button on the toolbox is pressed in.
- Click the Subform/Subreport icon on the toolbox and draw the outline of the subform on the main form. The Subform Wizard dialog box will appear when the mouse button is released.
- If the subform has not been created yet, select "Use existing Tables and Queries". Otherwise, select the existing form that will become the subform. Click Next to continue.
- The next dialog window will display table relationships assumed by Access. Select one of these relationships or define your own and click Next.
- On the final dialog box, enter the name of the subform and click Finish.

Drag-and-Drop Method

Use this method to create subforms from two forms that already exist. Make sure that the table relationships have already been set before proceeding with these steps.

- Open the main form in Design View and select WindowiTile Vertically to display both the database window and the form side-by- side.
- Drag the form icon beside the name of the subform onto the detail section of the main form design.

26.1 More Forms

Multiple-Page Forms Using Tabs

Tab controls allow you to easily create multi-page forms. Create a tab control by following these steps:

- Click the Tab Control icon on the toolbox and draw the control on the form.
- Add new controls to each tab page the same way that controls are added to regular form pages and click the tabs to change pages. Existing form controls cannot be added to the tab page by dragging and dropping. Instead, right-click on the control and select Cut from the shortcut menu. Then right-click on the tab control and select Paste. The controls can then be repositioned on the tab control.

- Add new tabs or delete tabs by right-clicking in the tab area and choosing Insert Page or Delete Page from the shortcut menu.
- Reorder the tabs by right-clicking on the tab control and selecting Page Order.
- Rename tabs by double-clicking on a tab and changing the Name property under the Other tab.

Conditional Formatting

Special formatting that depends on the control's value can be added to text boxes, lists, and combo boxes. A default value can set along with up to three conditional formats. To add conditional formatting to a control element, follow these steps:

- Select the control that the formatting should be applied to and select FormatiConditional Formatting from the menu bar.
- Under Condition 1, select one of the following condition types:
  - Field Value Is applies formatting based upon the value of the control. Select a comparison type from the second drop-down menu and enter a value in the final text box.
  - Expression Is applies formatting if the expression is true. Enter a value in the text box and the formatting will be added if the value matches the expression.
  - Field Has Focus will apply the formatting as soon as the field has focus.
- Add additional conditions by clicking the Add >> button and delete conditions by clicking Delete... and checking the conditions to erase.

Password Text Fields

To modify a text box so each character appears as an asterisk as the user types in the information, select the text field in Design View and click Properties. Under the Data tab, click in the Input Mask field and then click the button [...] that appears. Choose "Password" from the list of input masks and click Finish. Although the user will only see asterisks for each character that is typed, the actual characters will be saved in the database.

Change Control Type

If you decide the type of a control needs to be changed, this can be done without deleting the existing control and creating a new one although not every control type can be converted and those that can have a limited number of types they can be converted to. To change the control type, select the control on the form in Design View and choose FormatiChange To from the menu bar. Select one of the control types that is not grayed out.

Multiple Primary Keys

To select two fields for the composite primary key, move the mouse over the gray column next to the field names and note that it becomes an arrow. Click the mouse, hold it down, and drag it over all fields that should be primary keys and release the button. With the multiple fields highlighted, click the primary key button.

27 Reports

Reports will organize and group the information in a table or query and provide a way to print the data in a database.

Using the Wizard

Create a report using Access' wizard by following these steps:

- Double-click the "Create report by using wizard" option on the Reports Database Window.
- Select the information source for the report by selecting a table or query from the Tables/Queries drop-down menu. Then, select the fields that should be displayed in the report by transferring them from the Available Fields menu to the Selected Fields window using the single right arrow button > to move fields one at a time or the double arrow button >> to move all of the fields at once. Click the Next > button to move to the next screen.
- Select fields from the list that the records should be grouped by and click the right arrow button > to add those fields to the diagram. Use the Priority buttons to change the order of the grouped fields if more than one field is selected. Click Next > to continue.
- If the records should be sorted, identify a sort order here. Select the first field that records should be sorted by and click the A-Z sort button to choose from ascending or descending order. Click Next > to continue.
    - Select a layout and page orientation for the report and click Next >.
    - Select a color and graphics style for the report and click Next >.
- On the final screen, name the report and select to open it in either Print Preview or Design View mode. Click the Finish button to create the report.

Create in Design View

To create a report from scratch, select Design View from the Reports Database Window.

- Click the New button on the Reports Database Window. Highlight "Design View" and choose the data source of the report from the drop-down menu and click OK.
- You will be presented with a blank grid with a Field Box and form element toolbar that looks similar to the Design View for forms. Design the report in much the same way you would create a form. For example, double-click the title bar of the Field Box to add all of the fields to the report at once. Then, use the handles on the elements to resize them, move them to different locations, and modify the look of the report by using options on the formatting toolbar. Click the Print View button at the top, left corner of the screen to preview the report.

Printing Reports

Select File¡Page Setup to modify the page margins, size, orientation, and column setup. After all changes have been made, print the report by selecting File¡Print from the menu bar or click the Print button on the toolbar.

28 Importing, Exporting, and Linking

Importing

Importing objects from another database will create a complete copy of a table, query, or any other database object that you select. Import a database object by following these steps:

- Open the destination database.
- Select File|Get External|Import from the menu bar.
- Choose the database the object is located in a click the Import button.
- From the Import Objects window, click on the object tabs to find the object you want to import into the database. Click the Options >> button to view more options. Under Import Tables, select "Definition and Data" if the entire table should be copied or "Definition Only" if the table structure should be copied but not the data. Under Import Queries, select "As Tables" if the queries should appear as regular tables in the destination database. Highlight the object name, and click OK.
- The new object will now appear with the existing objects in the database.

Exporting

The effect of importing can also be achieved using the opposite method of exporting.

- Open the database containing an object that will be copied (exported) to another database.
- Find the object in the Database Window and highlight it. Then, select File|Export... from the menu bar.
- Select the destination database from the window and click Save.
- You will be prompted to name the new object and may also be given other options, such as whether to copy the structure or data and structure of a table. Click OK to complete the export procedure.

Linking

Unlike importing, linking objects from another database will create a link to an object in another database while not copying the table to the current database. Create a link by following these steps:

- Open the destination database.
- Select File|Get External|Link Tables... from the menu bar.
- Choose the database that the table is located in and click the Link button.
- A window listing the tables in the database will then appear. Highlight the table or tables that should be linked and click OK. A link to the table will appear in the Database Window as a small table icon preceded by a small right arrow.

Keyboard shortcuts can save time and the effort of constantly switching from the keyboard to the mouse to execute simple commands. Print this list of Access keyboard shortcuts and keep it by your computer for a quick reference.

Note: A plus sign indicates that the keys need to be pressed at the same time.

Action	Keystroke

Database actions	
Open existing database	CTRL+O
Open a new database	CTRL+N
Save	CTRL+S
Save record	SHIFT+ENTER
Print	CTRL+P
Display database window	F11
Find and Replace	CTRL+F
Copy	CTRL+C
Cut	CTRL+X
Paste	CTRL+V
Undo	CTRL+Z
Help	F1
Toggle between Form and Design view	F5

Other	
Insert line break in a memo field	CTRL+ENTER
Insert current date	CTRL+;
Insert current time	CTRL+:
Copy data from previous record	CTRL+'
Add a record	CTRL++
Delete a record	CTRL+-

Action	Keystroke

Editing	
Select all	CTRL+A
Copy	CTRL+C
Cut	CTRL+X
Paste	CTRL+V
Undo	CTRL+Z
Redo	CTRL+Y
Find	CTRL+F
Replace	CTRL+H
Spell checker	F7
Toggle between Edit mode and Navigation mode	F2
Open window for editing large content fields	SHIFT+F2
Switch from current field to current record	ESC

Navigating Through a datasheet	
Next field	TAB
Previous field	SHIFT+TAB
First field of record	HOME
Last field of record	END
Next record	DOWN ARROW
Previous record	UP ARROW
First field of first record	CTRL+HOME
Last field of last record	CTRL+END

29 References:

1. Batini, C., S. Ceri, S. Kant, and B. Navathe. Conceptual Database Design: An Entity Relational
2. Approach. The Benjamin/Cummings Publishing Company, 1991.
3. Date, C. J. An Introduction to Database Systems, 5th ed. Addison-Wesley, 1990.
4. Fleming, Candace C. and Barbara von Halle. Handbook of Relational Database Design. Addison-Wesley,1989.
5. Kroenke, David. Database Processing, 2nd ed. Science Research Associates, 1983. Martin, James. Information Engineering. Prentice-Hall, 1989.
6. Reingruber, Michael C. and William W. Gregory. The Data Modeling Handbook: A Best-Practice
7. Approach to Building Quality Data Models. John Wiley & Sons, Inc., 1994.
8. Simsion, Graeme. Data Modeling Essentials: Analysis, Design, and Innovation. International ThompsonComputer Press, 1994.
9. Teory, Toby J. Database Modeling & Design: The Basic Principles, 2nd ed.. Morgan KaufmannPublishers, Inc., 1994. http://www.fgcu.edu/support/office2000/access/index.html.
10. http://www.functionx.com/access/.
11. http://www.functionx.com/access/.